FIRST EDITION

<packt>

In color

The Embedded Linux Security Handbook

Fortify your embedded Linux systems from design to deployment

MATT ST. ONGE

Foreword by Rama Krishnan, Senior Director of Engineering, Veritas Technologies

The Embedded Linux Security Handbook

Portfolio Director: Rohit Rajkumar

Relationship Lead: Kaustubh Manglurkar

Content Engineer: Shreya Sarkar

Program Manager: Sandip Tadge

Growth Lead: Namita Velgekar

Copy Editor: Safis Editing

Technical Editor: Nithik Cheruvakodan

Proofreader: Shreya Sarkar

Indexer: Hemangini Bari

Production Designer: Gokul Raj S.T

Marketing Owner: Nivedita Pandey

First published: April 2025

Production reference: 1210325

Published by Packt Publishing Ltd.

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB, UK.

ISBN 978-1-83588-564-2

www.packtpub.com

To my loving partner, Nicolle, for pushing me when I needed it and for joining me on this amazing life journey.

[illegible]

- *Matt*

Foreword

Security matters.

In today's world with the technology landscape evolving rapidly and the ever-increasing interdependence of the systems within the digital infrastructure, security is no longer a luxury but an essential bedrock of every system, from the smallest of embedded devices to the largest data center supercomputers.

Recent reports, such as the Verizon *Data Breach Investigations Report (DBIR)* and IBM's *Cost of a Data Breach Report*, underscore a surge in threats ranging from sophisticated malware to simpler insider threats and social engineering attacks. The Verizon 2023 *DBIR* states that over 74% of breaches involved human elements, underscoring the need for robust security practices at both the hardware and software levels. Additionally, IBM's 2023 report found that the global average cost of each data breach had risen to $4.45 million, with critical industries like healthcare becoming especially vulnerable. According to Cybersecurity Ventures (https://cybersecurityventures.com/), the global annual cost of cybercrime was predicted to reach $9.5 trillion in 2024 and will reach $10.5 trillion in 2025. Further, numerous authors and reports have featured how cyber attackers don't even have to hack into computer systems anymore. They simply log in.

Today, no system administrator or security professional can afford to overlook the importance of a well-secured Linux environment.

In the face of growing cybersecurity threats, this book fills a critical gap by providing a comprehensive guide to Linux security tailored for those who build and maintain embedded Linux systems or appliances.

Matt has over 30 years of experience in the world of Linux systems, both appliance and other server systems, helping Linux customers, partners, and the community build secure embedded Linux servers. He has acquired deep insights into the strengths and vulnerabilities of these systems, and his knowledge spans from design stages to challenges of safeguarding hardware and software environments.

This makes Matt uniquely qualified to write this book.

In this book, Matt focuses on topics that every system and security professional needs to understand to prevent security incidents, beginning with an introduction to the cybersecurity landscape, underlining the importance of starting from the requirements and design stage, guiding you through securing the hardware and OS, and exploring concepts and components like Trusted Platform Modules, disk encryption, and OS immutability.

Matt doesn't stop at merely securing hardware and software sides. He also explores the frequently overlooked areas like the BIOS, firmware security, and the boot process ensuring that the system is protected even before the software comes into play. His approach includes safeguarding the system from potential threats posed by both the end-users and the environment, emphasizing a holistic approach to Linux security for appliances.

What makes this book stand out is Matt's ability to draw on his experience to offer actionable advice and best practices to continuously improve appliance security through their entire lifecycle and navigate the complex economic and regulatory environment. This book goes beyond immediate threats, addressing the longer-term challenges of designing and keeping systems safe and secure, and provides the knowledge and tools you need to protect your systems in today's complex security threat environment.

As you embark on this journey through Linux security, you are in the capable hands of someone who has spent his career securing open-source systems. Matt's experience, coupled with his strong commitment to the Linux community, ensures that this book will serve as both a foundational text and a practical guide to securing Linux environments at every level.

It is with great pleasure that I introduce you to *The Embedded Linux Security Handbook* by Matt St. Onge. I am confident that you will find this book not only informative but also essential to your work in securing Linux systems.

Rama Krishnan,

Senior Director of Engineering, Veritas Technologies

Author's Note

Before we dive into things, I'd like to tell you how I initially got involved with embedded Linux systems appliances and why it's been my passion.

My Linux journey began in the mid-1990s; however, my journey with embedded Linux systems began about a decade ago, when I took on the role of senior solutions architect at Red Hat. This is where I began working with Mike Zitomer, who soon became not just a leader to me but also a great mentor and friend. Mike built Red Hat's Embedded Partner Program and was building a team to expand it.

I remember this like it was just yesterday. Mike brought me into a meeting with my first embedded systems partner, a telecommunications systems provider (name withheld due to NDA) way back in 2015. By that time, I had already been working with Linux and open source software for about twenty years. The concepts of appliance building fascinated me. I was hooked. It wasn't much longer before I joined Mike's team as the lead technical resource. The wild ride begins. There's never been a dull moment.

In our careers, it's rare but sometimes we have those who help shape the direction we travel and guide us toward success; for me that mentor was Mike. I cannot thank Mike enough for all he has taught me over the years. I am forever grateful.

Over the past nine-plus years, I've spent my days working alongside the product managers, architects, developers, and product support staff of these Red Hat partners. My focus was never selling anything. If I helped my partners, good things would surely come. Enablement, empowerment, consultation, and education are all my focus. I've created countless lab exercises, presentations, reference materials, and so on all in support of building better, secure Linux appliances.

Through these activities, I have assisted my partners in prototyping, designing, building, and supporting such a vast array of solutions. Some save lives. Many improve creature comforts. Some are purely for fun. Others help defend our nation. This role has left me feeling like I have truly helped make the world a better place… somewhat at least.

So here we are. I hope the lessons I share with you here can ultimately assist you (metaphorically, of course) to "build a better mousetrap." I am humbled and happy you've chosen to join me on this journey into applying security into a system.

Also, let me clearly state now that Red Hat® has no involvement in this book. None whatsoever. This work is of my mind and opinion, not that of any employer, past or present. This project is for the benefit of anyone building an appliance. I'm not focused on any specific distribution of Linux;

however, I may have my own opinions. I intend to share them. The good, the bad, and the ugly. This is all in the pursuit of enabling you not to feel the pains that many others have already endured.

Contributors

About the author

Matt St. Onge is an Associate Principal Solutions Architect at Red Hat, focused on providing enablement through their Embedded Systems partner program. Since 2015, his activities at Red Hat have ranged from best practices sessions to design reviews and even leading rapid prototyping workshops. Matt has assisted hundreds of product teams over the past decade in building quality, secure Linux appliances.

Matt has been active in the open source and Linux communities for over 30 years by contributing to solutions via GitHub, creating new projects for the benefit of the embedded systems community, and hosting webinars and user group meetings.

When not building solutions in the lab, Matt can be found hiking with his dog in the mountains, golfing, or cooking elaborate meals on the grill for his extended family.

I want to sincerely thank the people who have been close to me and supported me, especially my partner, Nicolle Langone, for her patience and love. I also want to express my gratitude for the guidance, mentorship, and advice from some amazing leaders in my life: Mike Zitomer, Ron Pacheco, Kevin Miller, Chuck Mattern, Matt Davis, and John Tietjen.

Finally, I'd like to thank many others who have helped me with research, data, review, or other tasks related to this work: OnLogic, Derek Fanton, Adam Rael, Herve Lemaitre, Mark Thacker, Mark Russell, Matt Micene, Dan Pacek, Rama Krishnan, Dawn Marini, Prashant Divate, Richard Rios, Paul Armstrong, Tony Mazon, Kevin Holmes, Phil Avery, Rick Stahovec, Kamlesh Gurudasani, Oreoluwa Oluwafemi, Ahmed Elbanna, and to all those who helped and didn't know it.

Thanks to all of you!

About the reviewers

Prashant Divate is a passionate technologist, open-source evangelist, and embedded systems expert with over 6 years of experience. He specializes in embedded Linux systems, the Yocto Project, hardware bring-up, and crafting cutting-edge solutions for real-world challenges.

A maintainer of the meta-sirius Yocto Board Support Package (BSP) layer, Prashant excels at optimizing Linux-based systems and pushing the boundaries of embedded development. Beyond tech, he is a drawing artist and is a general-class ham radio operator (VU2OWC), licensed for worldwide communication.

With a keen eye for detail and a love for knowledge sharing, Prashant is dedicated to refining technical content to ensure clarity and impact.

Dawn Marini has been in the IT field for over two decades in a variety of roles, from help desk to systems administration. She is currently a Specialist Solutions Architect for Virtualization Infrastructure at Red Hat, Inc. Largely self-taught, she started writing programs in Basic in fourth grade on her grandfather's Commodore 64. Dawn is also active in Toastmasters, has a second-degree black belt in Isshin-ryu karate, and is learning Italian.

Kamlesh Gurudasani is the security lead at Texas Instruments and a member of the Trusted Firmware Technical Steering Committee and CIP Security working group, representing Texas Instruments (TI). He specializes in embedded security, cryptography, Linux Direct Rendering Manager (DRM), and power management for Arm architectures.

He contributes to Trusted Firmware-A (TF-A), OP-TEE, and Linux kernel cryptography, focusing on runtime security, secure boot, and vulnerability management. Kamlesh was also a speaker at EOSS24, presenting on Arm System Control and Management Interface (SCMI).

A big fan of open source, he actively supports its growth and enhances security in long-term industrial systems.

Oreoluwa Oluwafemi is a seasoned embedded systems engineer with a wealth of experience designing and developing IoT-based systems from the ground up. He earned a bachelor's degree in Computer Engineering from Covenant University, which laid the foundation for his career. He has expertly led the design and development of Industrial Internet of Things (IIoT) systems utilizing wireless standards such as Zigbee, Thread, Wi-Fi, Bluetooth Low Energy (BLE), and Long Range Wide Area

Network (LoRaWAN), and has deployed these solutions in conjunction with SCADA systems on offshore assets for process monitoring. Oreoluwa has also led IoT-based micro-mobility product teams that developed software solutions enabling EV-based transportation via mobile apps—built on Linux-based server systems and other major cloud computing solutions. His expertise is further enhanced by his work with Linux using RHEL at Huawei Technologies, where he focuses on process, network, and disk management. Oreoluwa currently applies his deep technical knowledge at Cors System Technologies.

Ahmed Elbanna is an Embedded Software Engineer with extensive experience in Embedded Linux, Automotive Cybersecurity, and various automotive domains, including C++, Model-Based Design, and Automotive Connectivity. Passionate about technology and literature, Ahmed enjoys delving into diverse genres and sharing his thoughtful, well-rounded book reviews.

Table of Contents

Preface

Welcome to *The Embedded Linux Security Handbook*. Together, we'll be embarking on a journey of knowledge – a deep exploration of what is considered (by many) tribal knowledge. This is the kind of stuff you cannot find online. My purpose for writing this book is to share my vast knowledge of applying security and best practices to the creation of Linux appliances.

Security doesn't end with a few configurations. It's perpetual knowledge and evolution mixed with a healthy dose of preparation and vigilance.

This book will take you on a hands-on journey of how and when to apply best practices and access public and private security resources, as well as the often-complex application of security measures throughout the myriad tasks required to create a secure and supportable Linux-based appliance solution.

By the end of this journey, you'll be best positioned to design, prototype, build, and support embedded Linux systems (and products) like never before. So, grab your favorite beverage and a notebook, and let's get moving.

Who this book is for

Although this book should appeal to a wide range of technologists, those who will get the most out of what I am presenting here are product teams, embedded software engineers, security professionals, and architects – more specifically, those who are directly or indirectly involved with the scoping, design, creation, and supporting of embedded Linux systems appliances.

What this book covers

Chapter 1, *Welcome to the Cyber Security Landscape*, introduces the reasons why we are all here on this journey and why security matters.

Chapter 2, *Security Starts at the Design Table*, as we start our journey together, introduces you to the idea that security is not a feature but rather is built into everything we do by default.

Chapter 3, *Applying Design Requirements Criteria – Hardware Selection*, reviews the advantages, pitfalls, and reasons for or against the selection of known standard hardware platforms.

Chapter 4, *Applying Design Requirements Criteria – the Operating System*, continuing on from defining your hardware requirements, identifies existing limitations and features within certain distributions that will also impact your hardware and operating system pairings – much like a fine wine to a great meal.

Chapter 5, *Basic Needs in My Build Chain*, reviews the many additional components that you will need to leverage in securing your product.

Chapter 6, *Disk Encryption*, dives deep, with hands-on exercises, into ways to automate the secure encryption of your solutions data.

Chapter 7, *The Trusted Platform Module*, delves, with hands-on exercises, into leveraging your system's TPM module to store cryptographic keys and passphrases.

Chapter 8, *Boot, BIOS, and Firmware Security*, is a hands-on deep dive into securing your BIOS and how your systems boot.

Chapter 9, *Image-Based Deployments*, explores image-based operating systems and how they can make your solution more secure.

Chapter 10, *Childproofing the Solution: Protection from the End-User and Their Environment*, reviews methodologies on how to protect your appliance from its end-users while creating a rich, positive user experience.

Chapter 11, *Knowing the Threat Landscape – Staying Informed*, reviews the treasure trove of resources available to you to keep informed and educated on the ever-changing threat landscape.

Chapter 12, *Are My Devices' Communications and Interactions Secure?*, explores how secure devices attached to your Linux system actually are. We'll also do a hands-on deep dive into ways of securing and encrypting your network communications properly.

Chapter 13, *Applying Government Security Standards – System Hardening*, does a hands-on deep dive into applying government security standards to our systems.

Chapter 14, *Customer and Community Feedback Loops*, reviews putting it all together and how to involve your customers, users, and partners in the continuous improvement chain.

To get the most out of this book

In order to get the most out of this book, it is assumed that the reader has some prior experience in Linux systems administration or experience as a user of Linux systems.

Many of the concepts in this book are extremely advanced. The lessons in this book combined with your existing knowledge should catapult you to a level of mastery of not only understanding the internals of Linux systems but also building products based upon Linux.

There are numerous hands-on exercises in this book to enhance the learning activities. Should you choose to follow along with these exercises, the author recommends having at least two PC-grade systems with a minimum of 16 GB of RAM and 1 TB of storage each for best results.

Additionally, this book contains several screenshots. These have been captured to provide an overview of key processes. As a result, the text in these images may appear small at 100% zoom. However, you can view clearer versions of these images at this link: https://packt.link/gbp/9781835885642.

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook. If there's an update to the code, it will be updated in the GitHub repository. Check here often for updates and new content related to the book, cheat sheets, reference materials, and new exercises.

We also have other code bundles from our rich catalog of books and videos available at https://github.com/PacktPublishing/. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: “To ensure the service runs even when the `$USER` is not logged in, we’ll need to use `linger`.”

A block of code is set as follows:

```
# we will use these registries only
[registries.search]
registries = ['registry.redhat.io','quay.io']
```

Any command-line input or output is written as follows:

```
$ systemctl -user start myapplication.service
$ systemctl -user enable myapplication.service
```

When we wish to draw your attention to a command-line input or output in a set of steps, the relevant lines or items are set in bold:

1. Run a test scan.

```
$  oscap-ssh <username>@<hostname> <port> oval eval --report <scan-report.html>
<path to rhel-9.oval.xml>
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: “Go to **Applications**, then select **SCAP Workbench**.”

TIPS OR IMPORTANT NOTES

Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at customercare@packtpub.com and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at copyright@packt.com with a link to the material.

Part 1:Introduction to Embedded Systems and Secure Design

In this part, we will introduce you to the threats that impact your decision processes, hardware selection, operating system selection, and all the strengths and vulnerabilities that may alter your design criteria.

This part has the following chapters:

1
Welcome to the Cyber Security Landscape

Cue up the music. *Welcome to the Jungle*. The cyber security landscape is ever-changing and evolving. One could easily compare it to the mercurial New England weather patterns in the Northeast of the United States. Such weather can be dangerous for a hiker; similarly, such changes in cyber threats can be devastating for a technology solution.

In the coming chapters, we will dive deeper into the realm of **Linux-embedded systems** (or embedded Linux systems, also known as Linux appliances, or simply appliances) and exactly how you can apply this new knowledge to secure your solutions. We'll make this journey together, and at the end of the tunnel, you will be best positioned to build a better appliance.

This chapter is here just to set the foundation of Linux-embedded systems before we dive deeper. In this chapter, we will cover the following topics:

- What is a Linux-embedded system?
- How are Linux-embedded systems used?
- Why is securing Linux-embedded systems so important?

Let's get started.

What is a Linux-embedded system?

It's probably safe to assume that if you're reading this book, you may have some knowledge of what your company defines as a Linux-embedded system. To truly have a rich understanding of this subject, let's review the **appliance model** for creating sellable solutions.

The **appliance model** provides solutions and services that an organization (or individual) cannot easily build for themselves. This also addresses another limitation; oftentimes, users of these appliances have little to no access to any technologists who would take on ownership of the care of said solution. The appliance model is most prevalent in home consumer electronics but touches every known vertical in the industry.

Most appliances leverage purpose-built hardware and have unique form factors that best enable their usage. These solutions are all around us every day. Some easily identifiable consumer appliance examples would be your home's Wi-Fi router, cable TV box, or smart laundry washing machine. A resounding example would be the smartphones that we all carry and cannot imagine living without.

Let's affectionately pick a generic home Wi-Fi router to dive deeper into the appliance model concepts. Virtually anyone can use its services without understanding how the underlying technology works or is implemented.

The router has a custom form factor for far more than aesthetics. The housing contains all the electronic components needed to successfully provide the services for which it's intended. Inside, there's a small computer running a Linux operating system. I bet you can see where I am going with this. This is how we came to call these offerings *Linux-embedded systems*, as they contain a custom computer and a Linux operating system, along with custom software to provide a unique set of services to the intended end-user.

Defining an appliance doesn't stop there. It must provide a service, or services, which can be easily consumed by its end-users. In an appliance, this is achieved by the implementation of complex software and the appropriate hardware. In that home Wi-Fi router example, the manufacturer has built it to provide a simple-to-use web-based interface in order to *manage* the services provided. In this case, this is how the end-users allow people to access wireless internet.

Another key facet of the appliance model that must be built in is the ability to accept and process updates to its configuration or software, which absolutely includes its Linux operating system. It has to be simple for the end-user; one must assume the end-user is not technical. If the end-users actually possessed the technical capabilities, they could, in theory, build this themselves and you'd be out of business. Staying with the Wi-Fi router example, it has built into its web interface the simple function for self-updating. The end-user clicks on a button in the interface and the magic happens behind the scenes. Programmatically, the appliance gets an update package from a secure repository provided by the solutions builder; it downloads the package; and then finally processes all those application and Linux updates without any intervention by the end-user. The process notifies the end-user of its completion status (and sometimes requires a restart of the appliance).

Not every embedded Linux system (appliance) is on dedicated hardware. The advent of virtualization and, more recently, the great push to move things into the cloud have unleashed a whole new delivery method: the **virtual appliance**.

Virtual appliances are rapidly gaining traction in multiple cloud marketplaces. Why? It's simple. Companies that, for many years, have produced on-premises appliance solutions have had to evolve in order to retain their customer base. Because of this, many solutions builders now offer an on-premises hardware offering, a virtual appliance for an on-premises offering, and a cloud provider-based virtual appliance offering.

Generally, each of these solutions can be built from the same code base, however, depending on what exactly the appliance provides the end-user, it may need to be tailored and refactored differently for each virtualization platform's networking and security requirements. This is because the cloud providers (often referred to as **hyperscalers**) each have their own proprietary methods and APIs for networking, security, and end-user access. What I mean by this specifically is the application stack may be unified upon all delivery methods; however, the virtual machine image format, networking, and other facets will be different. For example, Azure and Google handle their network stack quite differently. Storage is also quite different. Hence, the **user interface** (**UI**), especially for the appliance's setup, needs to have these differences in virtualization platforms or hyperscalers' requirements taken into consideration.

Next, we'll review how embedded Linux systems are utilized.

How are Linux-embedded systems used?

Linux-embedded systems are everywhere. Most people don't give it a second thought. I venture the guess that you reach for one of these things when you first wake up in the morning and engage with it countless times every day. Your smartphone is a Linux-embedded system and, in my opinion, is an awesome example of one.

Embedded Linux systems surround us; some with stealth, others with pure elegance and entertainment, with all the beauty and grace their builders have instilled in their design. These appliance solutions run a massive range of use cases. Some are a complete solution and others are but a simple component to a greater ecosystem or solution.

Most, when successfully designed and implemented, provide services to their user community seamlessly without the user even having to think about what is happening under the covers. Much of this must be credited to how resilient, lightweight, and flexible Linux solutions can be. A Microsoft Windows-embedded system is far more reliant on its GUI components than Linux. It's far easier to deliver a complete solution on Linux with a GUI than its Windows-based competition.

Here, I will try to provide a generalized list of where Linux is leveraged today (regardless of distribution or hardware platform) in providing services. I'll throw in my disclaimer as well – this list is general at best and may not include all use cases currently deployed. I present this in no particular order and will try not to leave anything out but it would be impossible for me to list everywhere Linux-embedded systems are in use. So, let's look at the following list:

- **Artificial intelligence (AI)**: Linux is at the heart of all AI advancements in recent years. It is the preferred operating system of choice due to its flexibility and vast developer base. As this field is rapidly evolving, the possibilities are limitless and often classified, such as the following:
    - Facial recognition systems
    - Artistic graphical rendering systems

    - Language processing
- **Agriculture**: As agriculture around the world has begun to embrace operations at ever-growing scales, technology has been needed to assist with the following operations:
    - Heavy equipment monitoring solutions
    - GPS-guided equipment navigation
    - Produce grading and sorting systems
    - Production tracking systems
- **Automotive**: Embedded Linux reigns supreme in this rapidly evolving transportation space. Countless embedded Linux systems can be found today, stealthily providing many critical services, such as the following:
    - Autonomous vehicle operating systems and components
    - Sensors and safety systems
    - GPS navigation
    - Electric vehicle charging infrastructure
- **Aviation**: Civilian aviation and transportation is a booming business for Linux-embedded systems solutions. These complex appliances can be found in the following areas (assuming you have the proper security clearance):
    - Baggage scanning systems
    - Passenger screening solutions
    - Baggage shipping and tracking systems
    - Radar and air traffic control systems
    - Traveler identification and verification systems
- **Building management and construction**: Building management controls have long been solved by Linux-based solutions providers. The following is but a mere example of some of the high-tech solutions:
    - Card key access systems
    - Environmental control systems
    - HVAC management systems
    - Fire detection and suppression systems
    - Alarm systems
- **Telecommunications**: In my opinion, this vertical has seen the most growth, which is second only to those related to military projects. **Virtualization** has empowered the proliferation of a whole new breed of **network functions virtualization** (**NFV**) solutions. Both physical and virtual appliances in this space dwarf deployments virtually anywhere else. These virtual appliances can be found in server rooms, telco closets, data centers, and cloud providers' marketplaces in order to make it easier to provide access to the following services:
    - **Radio access networks** (**RANs**)
    - Load balancers

    - Routing solutions
    - **Push-to-talk** (**PTT**) phone systems
    - Geo-locational-tracking solutions
    - Domain Name System (DNS) solutions
    - IP address management solutions
    - Firewall solutions
    - Email filtering solutions
    - Operations management systems that control fiber-based connectivity to the internet

- **Geology**: Mostly driven by energy companies and government agency needs, this vertical has seen exceptional growth in the past five years. Although they may not be mainstream, these appliances provide many valuable services to academia and the energy industry, such as the following:
    - Near real-time geological scanning and assessment solutions
    - Modeling systems
    - Seismic tracking solutions
- **Healthcare**: Linux systems are vastly deployed in almost every facet of healthcare, whose focus on security and patient safety is paramount. Hospital efficiency has been greatly improved as well. Many of the new appliance solutions can be found in virtually any healthcare practice, providing some of the following life-saving services:
    - Patient safety solutions
    - Electronic records solutions
    - Patient monitoring solutions
    - Ventilators
    - Lab testing solutions
    - Imaging systems (PET scan, MRI scan, CAT scan, etc.)
- **Industrial and manufacturing**: Industrial automation has been prolific for over two decades, however, some key players in this space have revolutionized what used to be manual processes. Embedded Linux systems reduce costs and accelerate production rates by providing some of the following services:
    - Industrial manufacturing automation systems
    - Environmental and safety monitoring systems
- **Military**: So, in order to keep those guys in black suits from coming to visit me in an unfriendly manner, I'll generalize, especially in this vertical, to preserve my life (and add a little intrigue to the mix). Linux appliances can stealthily be found providing critical security and defense-related services, such as the following:
    - Facial recognition systems
    - Data gathering solutions
    - Components to countless vehicles and aircraft

    - Radar systems
    - Guidance systems
    - Navigation systems
- **Space**: Without Linux and open source technologies, this vertical would not be achieving what it is today. Embedded Linux systems are reaching for the heavens. A few examples are as follows:
    - Space exploration vehicles (example: Cassini)
    - Satellites
    - Rovers
    - Sensor systems
    - GPS systems
    - Ground-based management systems
    - Space telescopes
- **Weather**: Government agencies, aviation, highway safety, oceanic shipping, and other transportation agencies all rely upon data provided by Linux-embedded systems solutions for continuity of travel and public safety. These appliances, which can be found globally, provide the following services:
    - Weather modeling and prediction solutions
    - Ground and aerial radar systems

In summary, we've highlighted just some of the critical and interesting services embedded Linux systems provide today that enrich our lives, protect us silently, and help us travel to places that were once beyond our reach.

Why is securing Linux-embedded systems so important?

Why is securing Linux-embedded solutions systems (appliances) so important? Well, this answer truly has multiple aspects depending on who you ask.

The most obvious answer would simply be to prevent liability and loss of credibility in the marketplace. Lately, it seems that there's an ever-growing list of companies whose offerings have become compromised and, as a result, their customers are grievously impacted.

The impact of such a breach can be the loss of personal information for the users (insert gasp here!) or even worse, unauthorized access to systems or data without the knowledge of those running the systems. Each year, these breaches result in billions of dollars in correlated losses and open these companies to legal action by their user base. The image impact on one's brand can last far longer in people's minds than you'd think.

It's highly documented that data leaks, breaches, and other security lapses are often done by internal staff and contractors. This does not imply malfeasance. Often, it can be a lapse of judgment, forgetting to remove sensitive data from development environments, or accidentally taking home sensitive data on a device that itself gets lost, stolen, or compromised.

Then, there are the situations caused by intentional misuse of a system. Sadly, these events, in my opinion, can be the easiest to prevent. We will definitely dive deep into this matter in later chapters. Misuse comes in many forms. It starts with unauthorized access to a system or exceeding one's intended limits. Then, there are those events where someone intentionally uses an application or system in a manner that it was not intended for. The worst-case scenario is when software on a system is modified, installed, or disabled so that an individual can perform tasks or monitor others. These are the situations where the wheels fall off the wagon.

Examples where embedded Linux systems had a security breach

In 2019, hackers compromised Ring security cameras and gained access to their video and audio streams, wreaking havoc with those devices' owners' lives. Several incidents were reported in several states. Software updates and better password requirements keep those bad actors out now.

Another example of a breach of security is when, in 2017, it was found that St. Jude's cardiac implants could be hacked. When I heard of this, I was amazed and scared for those patients. Thankfully, St. Jude has patched the vulnerability and no patients were actually harmed.

A research team at IBM was able to compromise the firmware update mechanism of a Jeep in mid-2015. They demonstrated how a bad actor could actually take control of the vehicle by speeding it up, steering the wheel, or applying the brakes. It's kind of frightening to think that a bad actor could wreck your family drive. These gaps have quickly been closed.

Hopefully, these few public examples can help reinforce just how important it is to design an embedded Linux appliance with security in mind.

Detecting these unauthorized changes or breaches can be rather difficult, especially if you no longer have access to your product once the customer takes delivery and begins consuming its services. Preventing them altogether is so much easier.

You may have heard the old saying, *an ounce of prevention is worth a ton of cure*. If you apply even some or all of the lessons this book seeks to impart upon you, it's like having ten pounds of prevention and *all* the cure. We'll dive deep into hardware security, operating systems security, secure connectivity, and how to know what threats you must prepare for.

Not every method or technology in this book may apply to your solution or even be feasible, but the more you apply these methodologies, the more likely you will create a rock-solid product. That said, may the only headlines your company creates be those of success and accolades.

Summary

In this first chapter, you will have gained a solid understanding of how an embedded Linux system is defined. We've shown you some wide-ranging examples of where embedded Linux systems are leveraged today. Most importantly, we've reviewed some critical reasons why having these devices delivered to their customers as securely as possible is mandatory for those building such solutions. In the next chapter, we'll dive deep into why security starts with a great design.

2
Security Starts at the Design Table

Security is more than a process. Security cannot be an afterthought. Security is a mindset. When you are considering offering a new or upgraded product to the market, security starts at the design table. The choices you make here will define outcomes you may not have even pondered in the past.

In the previous chapter, we did a brief review about what embedded Linux systems (AKA appliances) are and often how they solve real-world problems. The next few chapters will focus on the design and build phase. Each chapter will build upon the concepts of the previous, and as you'll soon readily see, it is all considerations to be aware of.

While in the design phase, a product team has the highest chance of success in mitigating future risks. Through proper scoping, planning, and execution, your team can achieve great success in creating a highly productive appliance solution. Each of the factors that we are about to review, if accounted for at this stage, will become a factor toward success, not a risk factor.

Security auditors, systems architects, and product managers, all by virtue of their expected job descriptions, are risk mitigators. That's why we are here doing this right now. Thank you for taking this journey with me.

Measure twice and cut once is a saying many of us are quite familiar with. Here in the design phase, I'd say that's radically insufficient. I say it's *Measure twice, change rulers, compare the new ruler to the old one, measure two more times with the new ruler, and then consider the cut carefully*. If still in doubt, you should get a third ruler (and start again!). I would also recommend a peer review before the initial cut. A peer review is something I cannot recommend highly enough. A second set of eyes (or several more) can often find and point out something you may have overlooked.

Truly, I am being overly dramatic. This is for many good reasons. Failing to account for key aspects of the solution can result in a weak or inadequate product. In the worst case, you're stuck with a smoldering pile of garbage and also putting your career in jeopardy. In this chapter, we are definitely going to dive deep into many factors that get overlooked in the design phase of products. Not all of these risks are purely technical. Yet, each of them alone can create problems. Some are social, some are political, and most are related to design factors that must be vetted.

The process flow that this chapter will cover can be illustrated through the following diagram:

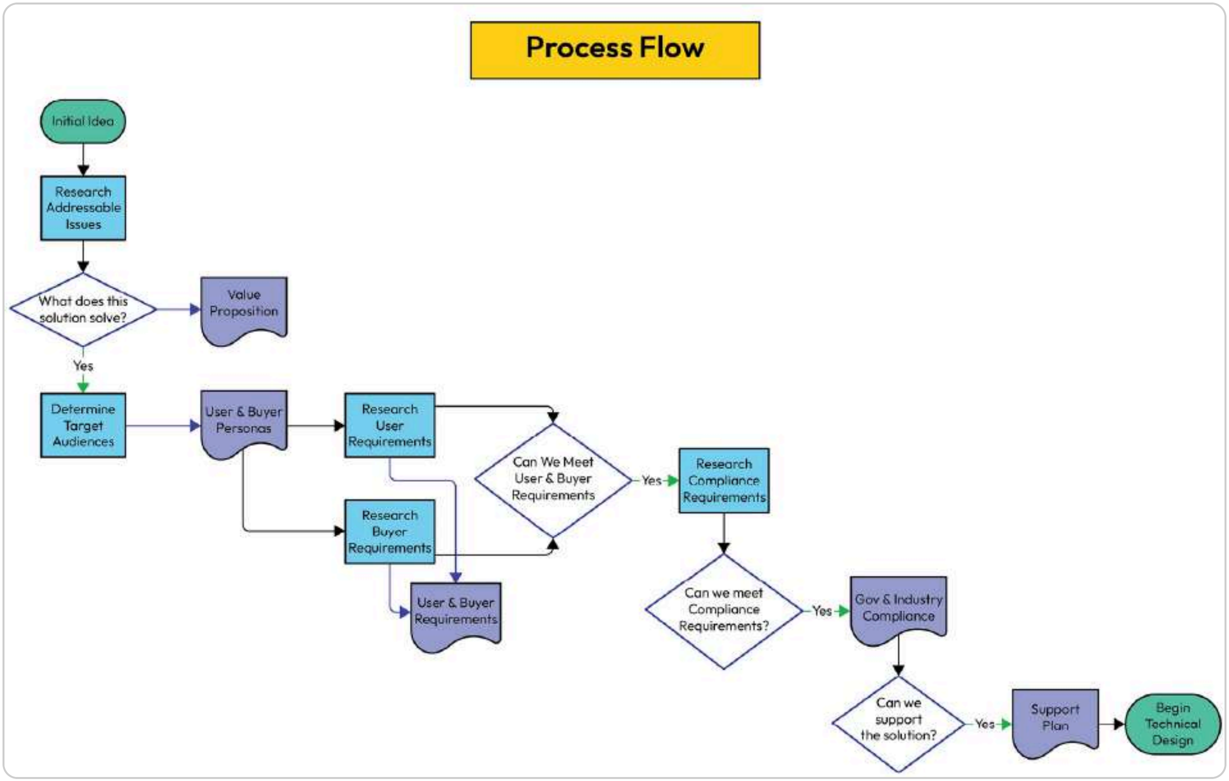

Process Flow
Initial Idea
Research Addressable Issues
What does this solution solve?
Value Proposition
Yes
Determine Target Audiences
User & Buyer Personas
Research User Requirements
Research Buyer Requirements
User & Buyer Requirements
Can We Meet User & Buyer Requirements
Yes
Research Compliance Requirements
Can we meet Compliance Requirements?
Yes
Gov & Industry Compliance
Can we support the solution?
Yes
Support Plan
Begin Technical Design

Figure 2.1 – Process flow of this chapter

In this chapter, we will cover the following topics:

- What are the business needs that the solution caters to?
- Who is my target buyer and my target user?
- Will any specific government compliance standards drive the decision tree?
- How will we support this appliance solution?
- Other product-impacting needs and concerns

Let's get started.

What are the business needs that the solution caters to?

You've got an idea for a new product. That's awesome. Before you spend time pitching the proposal to management, it's absolutely a necessity to spend some time where you need to roll up your sleeves and set some clear design goals for the product.

Let's walk through this together. You need to ask yourself some questions and be able to answer them in vast detail. The age of building a technological solution and throwing it on the market just because it's shiny is long over. Your prospective customers have tight budgets and massive time constraints. Adding value to the customer must be the single greatest attribute of any proposed solution.

First, what services exactly will your solution provide to its future end-users? This answer must be a clear and exacting response to this question, *What business need or problem is your appliance solution designed to address in order to add value to the end-customers?* To answer this question, one must know a few key points. What exactly is the problem? What are the workarounds (if any)?

So, what do we mean by *adding value*? Your solution must place a checkmark in several boxes. Next, ask yourself, *How complex this solution is to implement and does the average customer have the in-house ability to design/implement a home-grown solution themselves?* The answer must be *My solution adds value to people's lives or provides a valuable service that otherwise may be hard to obtain.*

If your solution could have easily been designed, built, or implemented by others, then the future chance of selling your idea could be bleak. Solutions that solve complex and difficult problems are those that add immediate value to their customers.

Research, research, research! I say to all those who are evaluating or considering creating a new product. *Why?* one may ask. My response is crucial yet simple. If someone else has already solved this problem and created a product around it, you are starting in a bad position. The incumbent has the obvious advantage and most likely control of the market.

What if there are multiple solutions for this problem already in the market? This is where I jokingly ask you to question your sanity. Your company will have an uphill battle and fight for every possible customer. Here's where I hope your solution crushes the competition and this journey aids you in your efforts.

Now that you are empowered by knowing the exact use cases your product is designed to address, we can move on to our next set of design criteria to account for, and that's determining who your target **personas** are.

Who is my target buyer and my target user?

Why should I care about these personas? or *I thought this book was about security!* you might be thinking silently to yourself. Excellent! I've got you thinking. This book is absolutely about security and mitigating risks. This is actually crucial to your future product and company's success, and we'll address this in two parts. The answers can be as complex as the personas that your company's sales teams, product managers, and presales engineers will be interacting with in the future.

There is more than an ounce of applied psychology necessary for gathering detailed knowledge of your targeted buyers' and users' requirements. This insight will greatly aid in determining which features you must plan for to achieve the technological win and eventually the sale of your embedded Linux systems appliances. Both have vastly different drivers and motivators that explicitly impact what they demand from a solution. Preemptively getting access directly or indirectly to these people within your target industry in order to gather intelligence will be a strategic activity for your teams. Focus groups and surveys can also be an invaluable tool in the requirements information collection effort as well. Use every tool and method at your disposal. Business can be like war, so don't play fairly. Take every advantage.

Before we move on to the details about the various personas' importance, please indulge me a moment for one more warning. Failure to address the needs and wants of your target audiences will ultimately result in poor adoption rates for the solution. I'd be remiss if I didn't add in the fact that politics within your customers' organizations can be a hidden factor as well. Gather as much intelligence as you can before you speak to these personas. Preparation and having answers to questions they will or may ask is crucial. The political and hidden motives of the customer, although not directly involved with security, can easily shift to a security feature discussion or bashing session where your competition is thrown on center stage with you (metaphorically, of course).

The target buyer

The target buyer can be the hardest persona to account for. They may not be anyone who will be actually involved in the deployment, testing, or usage of your product. These customer personas might be a department head, an executive, someone in the finance department, or a procurement specialist. So, let's dive into the differences in driving factors for some of these examples.

> ***IMPORTANT NOTE***
>
> *The target buyer may be a combination of these examples (that is, an executive and a department head or someone in the finance department and a procurement specialist).*

It is also a fair statement, depending on the scale and cost of your solution, who and how many different personas might be involved in the decision process to acquire your solution.

An executive

Generally, they have the ultimate power and say. Their drivers are more focused on achieving a competitive edge in the marketplace, creating profits, and maintaining enough compliance in order to avoid tarnishing the corporate brand and staying out of jail.

Getting an executive to sponsor an initiative based on your project can also mean the difference between success, a reduced adoption rate, or, even worse, a failed deployment altogether. The weight they carry in their respective organizations often can move obstacles for your team, but remember this

– if you fail to plan for their requirements in your solution, the very mountains they move might be ones moved against your initiative.

A department head

A department head can become your easiest ally or your hardest obstacle, as their realm of control and experience is deeply in tune with the needs of their organization. They may have already had experience (or general knowledge) of your solution or a competitor's solution in the past when they were in a different role. Their security requirements can be both strategic and tactical based on their unique domain knowledge.

The department head may also be in favor of a home-grown solution, or your solution may be in the running to replace one they had previously created (which is why they got promoted to department head in the first place).

Commonly, a department head persona will also most likely lean on one or more of their staff for their input and consideration. That level of collaboration makes them the most *in the know* of the target buyers. Because of this, they often come to the table with their lists of *must-have* security features along with secondary lists of *nice-to-haves*.

A procurement specialist

Procurement specialists are masters of driving down costs. They are compensated wholly upon that notion. These people are the bane of any sales team's existence. They truly do not care about the value of a solution or the enablement it provides. Oftentimes, these individuals come from a legal or financial background, which makes sense as their purpose is to negotiate purchasing contracts.

In seeking out solutions on behalf of their company, they are often given a shopping list of features, but often, they are not educated exactly on what the solutions do nor how they work. This persona might require additional enablement to understand why your solution is the best. If you can get to them during your design phase, they can help drive the required features and help the product team set an initial price point for your solution. In my experience and my humble opinion, these folks are the toughest customers to deal with in any sales/presales cycle.

Identifying and planning for the requirements of your key target buying audience is a challenge, but knowing their drivers gives you an advantage to design against. Now, let's move on to our next subsection and review the target user base.

The target user

The target users are your key influencers. They are the people who will be, hopefully, using your solution on a regular basis. They are the most informed on what features make their lives easier. They will have the greatest level of opinion on your interface and the operation of the appliance. They may already have an older version of your solution or, even worse, a competitor's solution already in place. This group can make or break a deal. As they are the ultimate consumers of the solution, their influence is key in the procurement of products and services for their organizations.

It is not fair, in many cases, to assume this group understands the technology that you are building into the solution. Your solution's very purpose is to make their jobs easier and to grant them additional capabilities that they may not have on their own. Appliance solutions' sole purpose is to provide advanced functionality to those who do not have technical abilities or advanced levels of support within the organization for that function.

This group is also your best ally in the adoption of your solution. They can give you the greatest intelligence as to what is really happening in the field. Their input can drive functionality and security requirements.

We reviewed the most critical personas whose needs and wants in your solution must be accounted for within your design. This is not all-inclusive, but it should give a massive advantage in the elimination of certain guesswork regarding specific security features being required, which security features are a *nice-to-have*, and, finally, which security features may be considered optional initially. Take this all with a pinch of salt as the landscape changes, personnel change, and this list of design requirements you are creating is also subject to change.

In our next set of key factors to consider, we'll review the impact of compliance on your product's design.

Will any specific government compliance standards drive the decision tree?

The mandate for compliance with one or more government standards has the probability of limiting the functionality of your solution while adding a level of difficulty in its design and preparations for the market. As this book may be available globally, my years of experience are more directly focused on the North American marketplace. It can also be argued that the largest marketplace for embedded systems is North America itself; this fact alone creates the opportunity for manufacturers worldwide to create solutions to sell there.

Knowing your target buyers, users, and their locales will obviously determine which compliance standards you'll need to account for. It is definitely vital to understand that many countries have similar and different restrictions and applicable security laws in place.

Many countries leverage standards originating in the US as the foundation. In the US, we also take notice of what other countries mandate. The global security community is as diverse as they are technical. Much as with the open source community, information sharing and peer reviewing exist in their core spirit. Many of these resources we will review in considerable depth and breadth in future chapters.

Let's dive through some example industry verticals and which compliance standards can be applied directly to any computer system in their midst. These examples will highlight initiatives at a federal/country level, but please remember that there can also be state or local legislature driving compliance standards as well. Before we dive in, I want to leave you with yet another disclaimer: worldwide there is a myriad of laws, rules, and regulations that regulate all aspects of businesses; however, with the following list of markets and their primary concerns, we will focus on some of the more important initiatives that will directly impact security and your design specifications. Take a deep breath. Let's go!

Healthcare systems (and data privacy)

First, let's talk about healthcare. The healthcare industry's customer base in North America alone exceeds 383,000,000 people. Protecting their systems, privacy, and their patients' data is mandated by laws. Some of them are explained ahead.

HIPAA

Let's start with the US federal government standard known as the **Health Insurance Portability and Accountability Act** (**HIPAA**). Since its inception in 1996, HIPAA has driven vast investments and scrutiny in the industry across all of the US. It's a shining example of how data privacy needs to be handled.

Follow this link for more details on the law and its requirements for healthcare systems: https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996.

Other critical standards for healthcare systems that will impact your embedded Linux systems are much more intrusive and may require you to submit your appliance for testing and certification.

IEC 60601

This standard applies to electrical safety for medical equipment and electromagnetic emissions (such as radio signals, radiation, and Wi-Fi) and defines testing for basic safety.

This one standard has many names globally; here are some examples of how other countries follow the standard but call it by a different name:

- Canada calls it *CAN/CSA C22.2 Number 60601-1*
- The **European Union** (**EU**) calls it *EN 60601-1*
- Japan calls it *JIS T0601-1*
- Australia and New Zealand call it *AS/NZ 3200.1.0*

This is not an exhaustive list. This is why I recommended researching what standards apply to you. Sadly, I know I will be repeating that from time to time throughout this book.

Follow this link for more details on the law and its requirements for healthcare systems: https://www.iso.org/standard/65529.html.

CE mark/certification

This is an EU requirement to prove that medical equipment has been assessed and meets strict safety, health, and environmental standards for healthcare equipment. It also provides proof of **Restriction of Hazardous Substances** (globally known as **RoHS**).

Follow this link for more details on the law and its requirements for healthcare systems: https://single-market-economy.ec.europa.eu/single-market/ce-marking_en.

Healthcare can be tricky, and as we move onto our next vertical, you should notice the scrutiny intensifies even more. Let's move on to financial services systems.

Financial services systems

All over the world, the financial services industry is massively regulated in order to protect the average consumer. This industry covers a vast range of services from general banking and lending to credit cards, real estate loans, business finance, the stock market, bonds, annuities, and other investment vehicles.

As a result, several government agencies have been created. They are tasked with monitoring and regulating various aspects of the financial markets. So, let's take a look at some of the more important entities (in the US). There are vastly more globally:

- The **Federal Deposit Insurance Corporation** (**FDIC**) is an agency created by the US Congress that is tasked with maintaining sustainability and public trust in the US's financial systems (basically the entire banking industry).
- The **Securities and Exchange Commission** (**SEC**) maintains wide responsibilities overseeing the securities and stock market industry.

- The **Federal Reserve Board** was established to control the operations of the 12 established Federal Reserve banks and oversight of their operations.
- The **Payment Card Industry Security Standards Council** (**PCI SSC**) was established to oversee and regulate the credit card industry.

In the US, these entities, along with law enforcement agencies, the **Internal Revenue Service** (**IRS**), and the US Treasury Department, ensure rules and regulations to protect the public are enforced strictly. This situation is similar all over the world.

Regardless of what country you reside or work in, this is where technology comes into play. The systems in this industry fall under the governance of many laws and standards for security. Let's take a look at a few of the big ones globally (not just the US) that can drive more scrutiny in your products' security footprint:

- The EU's **General Data Protection Regulation** (**EU-GDPR**) applies to any institution that holds the personal data of any EU resident. Follow this link for more details on the law and its requirements for financial industry systems: https://gdpr-info.eu/.
- **UK-GDPR** (the British version of EU-GDPR) applies to any institution that holds the personal data of UK citizens. Follow this link for more details on the law and its requirements for the industry: https://www.gov.uk/data-protection.
- The **Sarbanes-Oxley Act** (**SOX**) established critical security standards for financial systems within the US financial services industry. Follow these links for more details on the law and the requirements for the industry:
    - https://www.sec.gov/divisions/corpfin/faqs/soxact2002.htm
    - https://sarbanes-oxley-act.com/
- The **Payment Card Industry Data Security Standard** (**PCI DSS**) was created to create a uniform policy of requirements to protect systems and consumer data. Follow this link for more details on the law and its requirements for the industry: https://www.pcisecuritystandards.org/.
- The **Bank Secrecy Act** (**BSA**) regulates electronic transfers in the banking industry in an effort to prevent the act of money laundering. Follow these links for more details on the law and its requirements for the industry:
    - https://bsaefiling.fincen.treas.gov/main.html
    - https://www.occ.treas.gov/topics/supervision-and-examination/bsa/index-bsa.html
- The **Gramm-Leach-Bliley Act** (**GLBA**) enforces that companies that provide financial services disclose to their customers comprehensive information regarding their information-sharing practices in order to protect their data. They provide a framework for data privacy.

  Follow this link for more details about the law and its requirements for the industry: https://www.ftc.gov/business-guidance/privacy-security/gramm-leach-bliley-act
- **Payment Services Directive 2** (**PSD2**) is an EU regulation managing payment services that might be considered quite similar to the US's PCI DSS regulations. Follow these links for more details on the law and its requirements for the industry:
    - https://www.gov.uk/government/publications/the-revised-payment-services-directive-psd2-rpc-opinion
    - https://ec.europa.eu/commission/presscorner/detail/en/qanda_23_3544

- The **Federal Financial Institutions Examination Council** (**FFIEC**) is tasked with ensuring banking systems remain secure through detailed cybersecurity standards and review. Follow this link for more details about the law and its requirements for the industry: https://www.ffiec.gov/.

OK! Wow – that was a bit intense. Are you ready for a quiz?! I'm just kidding. That was a lot of regulation and government red tape to cut through. Let's move on to the next industry.

Retail and online marketplace systems

Retail and online marketplace systems (otherwise known as e-commerce systems) fall under a lot of the same regulations as financial services to mandate systems standards. To me, this makes a lot of sense. It's logical. Both verticals deal with personal data and the handling of financial transactions. Here are some familiar initiatives and some you may not be familiar with. All of these (and more not listed) could be huge factors in retail and e-commerce.

Retail has similar scrutiny to the financial services industry as laws such as PCI DSS, EU-GDPR, and UK-GDPR also apply to this sector. However, as this industry also provides a service to the public, its responsibilities extend beyond simple data protection as seen with these next laws:

- **Children's Online Privacy Protection Act** (**COPPA**) regulates what data can be collected and used from children under the age of 13 years old along with what things can be marketed to them. Follow this link for more details on the law and its requirements for the industry: https://www.ftc.gov/legal-library/browse/rules/childrens-online-privacy-protection-rule-coppa.
- **Americans with Disabilities Act** (**ADA**) (https://www.ada.gov/) enforces online commerce (that is, websites are compatible with views that enable people with disabilities to view them without discrimination).

So, there you have it. A brief summary of the agencies trying to protect your shopping experiences and your digital privacy has been laid out for your review. Let's move on to the whale in the room, and that's the governments themselves.

Government and military systems

As a military veteran and former US government contractor, I view this specific set of compliance standards as near and dear to my heart. OK – I'll admit there's been pain and scar tissue, but living up to then exceeding the standards was always in my DNA. That said, you'll need to pay close attention here.

US government agencies

The following is a listing of some of the more prolific US agencies that have a significantly defined level of oversight in the cybersecurity space and often share their standards internationally. Albeit not an all-inclusive listing, these government agencies are the ones to keep an eye on, and we'll also be talking about many of their publicly available resources in future chapters:

- **Cybersecurity and Infrastructure Security Agency (CISA)** (https://www.cisa.gov/): CISA is the US's premier cybersecurity agency and leads other government agencies and industry partners in ensuring the digital safety of the country, its citizens, and their digital assets.
- **National Institute of Standards and Technology (NIST)** (https://www.nist.gov/): Although NIST is actually part of the Department of Commerce, NIST's involvement with cybersecurity is most well known in its certification of systems to security standards.
- **National Security Agency (NSA)** (https://www.nsa.gov/Cybersecurity/): The NSA leads US government agencies in cryptology and intelligence gathering, It provides various products and services to government agencies and has been greatly involved in making **open source software** (**OSS**) and Linux more secure for everyone.
- **Department of Homeland Security (DHS)** (https://www.dhs.gov/topics/cybersecurity): DHS is yet another vastly broad-reaching agency in the US. Its role in cybersecurity usually involves investigations of cybersecurity crimes and related activities.

Now that we have taken a look at some key government agencies with responsibilities and units focused on cybersecurity, let's take a look at some **non-governmental organizations** (**NGOs**) and foreign government agencies that also have weight on security and standards.

Non-US government cybersecurity agencies

As this text will hopefully be read in the US and abroad, I wanted to include these agencies in a separate listing as they too are ones to follow and take notice of their work. They also produce and often enforce cybersecurity within their own domains. Let's look at a few key examples:

- **European Network Information Security Agency** (**ENISA**) (https://www.enisa.europa.eu/): ENISA is focused on setting security standards that apply across all of Europe. It not only creates policies but also processes and has mechanisms for certifications. It is the top entity to follow in the EU in terms of cybersecurity standards.
- **Organization for Security and Co-operation in Europe** (**OSCE**) (https://www.osce.org/): The OSCE is a non-profit organization in which each European country has a voice on economics, counter-terrorism, law enforcement strategies, and (of course) cybersecurity.
- **Cyber and Information Security Division** (**C&IS**) (https://www.mha.gov.in/en/divisionofmha/cyber-and-information-security-cis-division): C&IS is an Indian government division that deals with all matters of cybersecurity, cybercrime, and national security policies, and also with technology security standards and recommendations.
- **Center for Internet Security** (**CIS**) (https://www.cisecurity.org/): The CIS is a community-based, not-for-profit organization that funds itself through the sale of security best practices materials and related services.
- **National Center of Incident Readiness and Strategy for Cybersecurity** (**NISC**) (https://www.nisc.go.jp/eng/index.html): The NISC is Japan's cybersecurity watchdog. It is responsible for creating and enforcing standards for commercial and government systems.

Now that we've reviewed some international and foreign government agencies in the cybersecurity space along with their responsibilities, let's take a more introspective look at the impact and criticality definitions of data in the next section.

Impact Levels

The US government has a well-defined definition of implied levels of criticality to determine a specific level of systems security and scrutiny to be applied. They classify these levels as **Impact Levels**. A great graphical depiction of the implications of these levels can be found on the US **Department of Defense** (**DoD**) website – https://media.defense.gov/2020/May/18/2002302035/-1/-1/1/NAVY_TELEWORK_CAPABILITIES_V14.PDF.

Here's a quick summary of the IL determinations by the US DoD:

- **IL-2**: Impact Level 2 refers to public or non-critical mission information. Products with **Federal Risk and Authorization Management Program** (**FEDRAMP**) authorization achieve this designation automatically.
- **IL-4**: Impact Level 4 refers to **Controlled Unclassified Information** (**CUI**), non-mission critical information, not related to National Security Systems.
- **IL-5**: Impact Level 5 refers to systems with a much higher sensitivity of CUI, actual mission-critical information, or actual National Security Systems.
- **IL-6**: Impact Level 6 refers to information systems processing Classified *SECRET* data and other National Security Systems.
- **IL-7**: Impact Level 7 refers to information systems processing Classified *TOP SECRET* data and critical National Security Systems.

The US government takes data classification levels quite seriously, as we have highlighted in the last section. They do not stop there. The US government has a wealth of directives and standards that we'll be addressing in our next section.

US government standards and certifications

There are many federally enforced standards and certifications that are required by government or military institutions. Let's take a look at the key ones to be aware of:

- **Federal Information Security Modernization Act** (**FISMA**): Compliance with the standards drawn out by FISMA enforces annual detailed systems security audits in companies (https://security.cms.gov/learn/federal-information-security-modernization-act-fisma).
- **FEDRAMP**: FEDRAMP certification for products and web services enables them to be sold to and utilized by government agencies assuming the products also meet the proper Impact Level requirements (https://www.fedramp.gov/).
- **Secure Technical Implementation Guides** (**STIGs**): These publicly shared guides for the hardening of operating systems, network components, and applications are the baseline for obtaining and maintaining other government and military certifications. They also serve non-governmental entities with heightened standards that they can apply to their own systems regardless of which industry vertical they belong to. I have personally used these countless times previously in my career (especially when I was a government contractor working on sensitive systems) (https://public.cyber.mil/stigs/downloads/).
- **Common Criteria** (*ISO/IEC 15408*): This is a shining example of multinational collaboration where multiple governments have standardized and enforced adherence to a standard baseline of cybersecurity principles. Many countries participate in this global standard, and several offer certifications to the standards (https://www.commoncriteriaportal.org/index.cfm).
- **Federal Information Processing Standards** (**FIPS**) (*FIPS 140-2/140-3*): Both FIPS standards and their certification enforce detailed adherence to cryptography via comprehensive testing and certification. *FIPS 140-3* is the newer and current standard. This extremely comprehensive process is not only costly to manufacturers who take their products through it but also time-consuming

to the order of multiple years in some cases. NIST is the certifying agency. I have extensive personal experience in this process and have assisted several of my current company's partners through it. Use the following links for more details:

  - https://csrc.nist.gov/pubs/fips/140-3/final
  - https://csrc.nist.gov/pubs/fips/140-2/upd1/final

- **The US government standards profile (USGv6/USGv6-r1)** (https://www.nist.gov/programs-projects/usgv6-program): This program was created and also enforced by NIST to regulate all systems that use **Internet Protocol version 6** (**IPv6**) in their products.

In summary, we have seen a respectable number of governments asserting influence in the usage of computer systems. If you plan to market your solution to a government entity, it's clear that your product better bring its security A game as there are a plethora of standards that could apply to your embedded Linux system appliance.

We reviewed some critical examples of how compliance needs must be addressed in your product's design. In later chapters, we will dig deep into how to apply very specific standards, and we'll have some hands-on exercises for you to practice. Compliance adherence is not one of those *nice-to-haves*, as failure to comply may have the detrimental impact of criminal charges or civil liability; that puts your company at risk.

Now that we've seen how some standards and compliance initiatives will drive the security footprint of your solution, let's take a look at how your organization will consider how to support the solution once it's sold to your end-customers.

How will we support this appliance solution?

Another crucial set of design factors that can also impact your business model for your appliance solution is focused on how you plan to deliver support and services to your customer audience.

Support and services planning will directly impact the security of your appliance. Remote access, **virtual private network** (**VPN**), user accounts, patching methodologies, end-user self-support capabilities, and more are a few of the options that must be accounted for on the design table. Now, let's take a look at some delivery and support models that will impact access controls and the security footprint of your solution.

Managed service

A **managed service** offering allows your company to completely control many aspects of your solutions usage, update cycle, and access control. Regardless of the business model around this, several key factors will need to be addressed architecturally in your solution, such as the following:

- Level of customer (end-user access)

- Initial network configuration
- Scale and redundancies
- Remote access or on-premise access
- Will the solution be air-gapped?
- Auditing requirements

A managed services model is not for everyone. Let's move on to our next model, which is the online support model.

Online support

By online support, I mean web-based delivery of updates to the appliance and information for the customer. It often may include a method of creating a help request via a web page or via email. There may also be a situation where the user can get help/support online but the appliance itself is offline or air-gapped. With the online support model in place, let's take a look at key factors that may impact its success:

- How do you account for the initial appliance setup and configuration?
- Does the appliance have internet access?
- What method(s) will the appliance use to obtain and process updates?
- How will you deliver automatic updates vs. user-initiated updates?

Providing online support creates easy access to resources for the end-customer and a simple delivery method for providing information and updates. However, not all business models (or usage models) will find this acceptable. In this situation, we move on to our next model for those customers: the offline support model.

Offline support

In this case, the appliance receives nothing directly from the manufacturing company. The appliance solution itself is presumed to be in a restricted, air-gapped, or non-networked environment. Let's look at some key considerations that will need to be planned for:

- How do you account for the initial appliance setup and configuration?
- What method(s) will the user use to obtain and process updates for the solution?

No support/self-support

No support or self-support is the least common model on the market. It forces the end-user to do everything themselves, while the manufacturer of the solution has no control over the product once it ships. This is the set-and-forget model. This may be appropriate for some very low-cost items such as sensor appliances but ultimately makes your solution *disposable* in the eyes of the customer. Now, as this oversimplified limited support model has some cost advantages to your company, there are still a few factors to plan for in order to keep customers happy:

- Initial appliance setup and configuration
- Break/fix options for the end-customer

Replacements

No hardware platform is perfect. Sometimes, things just break. For some solutions, updating or troubleshooting them in the field (that is, remotely) is just too difficult or unfeasible. The solution for this is rudimentarily simple. Your company would swap out a new system with the customer in order to replace an inoperable unit or to provide the customer with a more updated or upgraded unit. Now, let's look at what you must consider and plan for:

- Is customer data backup and transfer between the old and new systems feasible?
- How do you account for the initial appliance setup and configuration?

As this is a common pain point for any product team, we must still plan for the inevitable. Now, let's look at a few other items you may not have planned for.

Other product-impacting needs and concerns

In this section, we'll perform continued deep introspection into factors that may have been overlooked or forgotten. In doing so, we'll be planning for the security and longevity of our product and ultimately taking strides toward ensuring its success.

Hardware life cycle

If you are a product manager, this section is really geared for you. Planning your alignment in concurrence with your hardware vendor's support life cycle is crucial in ensuring you have access to all the units and replacement parts for the planned life cycle of your appliance.

It is more than helpful to know where in a life cycle your vendor's current hardware release resides. Most mainstream hardware vendors will maintain a specific platform for 3 to 5 years. During this period, some components or minor motherboard changes are common. Replacement parts may be available for a couple of years beyond this timeframe but at a cost premium.

Assuming for a moment that your solution is using an off-the-shelf platform by a mainstream vendor, let's review a few key considerations:

- How much of a budget will we dedicate to life cycle support?
- Will minor changes in my vendor's platform be an issue for the appliance?
- What is my upgrade and replacement plan for future versions of my appliance?
- What third-party hardware components do I use in my appliance?
- How many spare systems and spare parts do I plan to keep on hand for immediate replacements?

Alright… Why does this tie into security? My answer is simple. If you fail to secure your hardware supply chain, you induce massive amounts of risk in the future in supporting your product once it's been adopted by end-customers. What does this actually translate into? It becomes an issue if you can no longer support the existing hardware initially deployed and may create a complex situation where your product has multiple platforms that become a support nightmare and a logistics nightmare. Or, even worse, by changing platforms, you introduce an unknown vulnerability or software incompatibility. And let's be honest, we're here because we want to reduce risk, right?!

Oh no! I just opened a can of worms. So, let's break it down into the simplest of terms. Securing your supply chain and having spares on hand will save you from the risk of deploying multiple platforms and your product becoming even harder to support. The whole purpose of my guidance here is to help you reduce risk and reduce costs.

Let's walk this together. *How much of a budget will we dedicate to life cycle support?* is the single most important question a product manager can ask of a product that is in active sales as this ties directly into the **profit and loss** (**P&L**) accounting that most PMs track daily. Life cycle support is a complex calculation.

One must plan for many different types of expenditures:

- Staffing costs
- Support methodologies (web, email, phone, and so on)
- Replacement costs
- Spare parts costs
- Additional costs that may be incurred by misalignment of hardware or software (or both) with the actual planned solution's life cycle

Now that we've reviewed how the life cycle of your hardware platform can impact the security of your pipeline and product delivery, let's look at the obvious adjoining issue tied to the hardware: the operating system life cycle.

Linux distribution life cycle

Most Linux distributions, regardless of community or enterprise, have their own predetermined life expectancy. This is intentional. Having a solidly defined life cycle empowers those supporting the distribution to plan their updates, feature releases, and version transitions.

This knowledge also enables you and your team to best plan for the needs of your products. Key milestone dates related to your operating system will drive design, deployment/release, and ultimately, the life cycle of your products.

The life cycle state will also be directly related to product cost structures. Paid, supported, enterprise-grade distributions incur costs on a per-unit basis. These can take the form of an annual subscription, perpetual license, and additional support costs. Some vendors provide minimalist support for distributions that have exceeded their normal life cycle. This extended life cycle support comes at a premium and usually only relates to a small subset of the operating system. What is also provided in this case is generally only the backporting of security fixes deemed critical or important. Lesser **common vulnerabilities and exposures** (**CVEs**) and bug fixes are not included, nor are new features or new hardware compatibility supported.

There are countless Linux distributions available today. Most claim to be in a niche and targeted to specific use cases. Others are leaders in the enterprise and government spaces. Some are easy to use (that is, they target beginners to Linux), and others require deeper knowledge and skill sets that exceed what is common in the workforce today. With there being so many, I thought it best to trim down the list to some of the most common and vocal players in the embedded Linux systems ecosystem.

I offer my disclaimer here yet again: this list is not all-inclusive, but these are what most appliances are built upon today in 2024. If I have omitted a distribution that you feel should be on this list, please accept my apologies. Detailing all the possible Linux distributions available today in and of itself could be its own book. These listed distributions are presented to you in no particular order and with no biases. In this section, we'll review lifespan, payment model, and support cycles. In upcoming chapters, we will go far deeper into several of these operating systems as to how they may or may not best serve your product.

How I'll format and display the data regarding the sampling of distributions you may come across being used as the foundation of embedded Linux systems appliances will go as follows:

- Distribution name

- URL for the distribution's main page
- Summary of the release schedule
- Primary life cycle duration
- Licensing/subscription information
- Support options information
- Miscellaneous important note (optional)

So, let's dive into some of these great examples, based on the aforementioned format:

- **Rocky Linux™** (https://rockylinux.org/)

  New major version releases every 3 years

  Update sub-releases every 5 months

  10-year life cycle

  Free to use

  No native support options

> ***IMPORTANT NOTE***
>
> *Moving from each major release is a fresh install – they offer no path, for example, to go from version 8.x to 9.x.*
> *This distribution was created by the founders of CentOS in response to Red Hat ending their downstream support of the operating system, hence transitioning CentOS to a development release set in between Fedora and* ***Red Hat Enterprise Linux*** *(****RHEL****) rather than a downstream free-to-use somewhat similar operating system to RHEL. Rocky is named in honor of one of the original founders of CentOS who has sadly passed away.*

- **AlmaLinux™** (https://almalinux.org/)

  New full version releases every 3 years

  Update sub-releases every 6 months

  10-year life cycle

  Free to use

  Paid support options are offered by third parties listed on the Alma Linux website

- **CentOS™** (https://centos.org/)

  Officially replaced by a non-production grade developers' playground named CentOS Stream

  The last of the updates stopped being published in June 2024

  Free to use

Third-party paid support options still exist, and several may be able to extend for years past June 2024

- **SUSE Linux Enterprise Server™ (SLES)** (https://www.suse.com/)

  10-year life cycle with an additional 3 years of paid long-term systems support (https://www.suse.com/lifecycle/#product-suse-linux-enterprise-server)

  Subscription model

  Support options included with the subscription

> ***IMPORTANT NOTE***
>
> *Free unsupported variants of SLES are available (openSUSE): https://www.opensuse.org/*

- **Wind River Linux™** (https://www.windriver.com/products/linux)

  Only one release annually as intended to align with Yocto releases

  10-year life cycle with paid support available beyond 10 years

  Paid license model only (nothing free)

  Support options are included with the licensing options: https://www.windriver.com/products/linux/support-maintenance

- **Red Hat Enterprise Linux™ (RHEL)** (https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux)

  New full version releases every 3 years

  Update sub-releases every 6 months

  10-year life cycle with additional paid support for up to 4 years of **Extended Lifecycle Support** (**ELS**)

  Paid subscription model

  Multiple support levels are determined by subscription version (that is, Premium versus Standard)

> ***IMPORTANT NOTE***
>
> *Free development-only subscriptions for individuals are also available: http://developers.redhat.com. Corporate development subscriptions have multiple options for paid and no-cost subscriptions that may be with or without support.*

- **Oracle Enterprise Linux™** (https://www.oracle.com/linux/)

  New full version releases every 3 years

Update sub-releases every 6 months

10-year life cycle with additional paid support for up to 3 years of Oracle Linux Extended Support (https://blogs.oracle.com/scoter/post/oracle-linux-and-unbreakable-enterprise-kernel-uek-releases)

Paid subscription model for support – free to use and distribute (https://www.oracle.com/linux/technologies/oracle-linux-downloads.html)

- **Ubuntu™ Server** (https://ubuntu.com/)

  Major versions – 10 years (5 years Standard support + 5 years of Pro support)

  Supplemental releases – 6 months

  Long Term Support (**LTS**) releases – 2 years (https://ubuntu.com/about/release-cycle)

  Free to use and distribute – paid support options are available

Now that we've looked at the most likely operating systems and their supported life cycles, let's move on to other supply chain issues as a whole.

Supply chain issues

A supply chain issue; how does that tie into security? you may ask. Well, that is a good question to ask. It's even more important to have multiple plans to mitigate it. This is vastly more critical if you are manufacturing a hardware-based appliance solution.

It is common, if not critical, to align your planned life cycle with your hardware vendors. What if your solution requires additional third-party components? What other sources do you have for spare parts? Is there a plan to maintain some parts on hand at repair facilities? And let's not forget to ensure that support staff have every possible combination of platform hardware and third-party components in their lab in order to troubleshoot customer issues and provide quality support.

Another instance where this adds a level of intricacy is when changes in hardware components induce changes in required drivers, kernel modules, libraries, and so on. This means each combination must be fully vetted, scanned, and validated to the same level of security measures. Keeping the number of combinations to a minimum will ultimately assist your teams in being able to deliver at scale. Otherwise, it becomes death by a thousand cuts; every customer has a slight variation from the next, or even worse, not every system for an individual customer is even the same. There you have it. A security *AND* support nightmare on your hands. With each new release or patching schedule, every combination must be accounted for. This can also impact how you provide updates to your customers and further impact the customer experience.

Summary

Wow! We've covered many different yet all-important criteria that will go into your design and product planning strategies. I highly recommend creating a detailed checklist based on what your needs and desires are for your product's future. Ensuring the stability, supportability, and longevity of your solution will make it easier to design, create, and sell a better, more secure appliance. Ultimately, you'll just be giving the end-customer more of what they expect from your solution.

In our next chapter, we'll review how to apply that design criteria to your hardware selection and procurement process.

3
Applying Design Requirements Criteria – Hardware Selection

In this chapter, we'll be performing a deep introspective into the design criteria that will determine what hardware you select. The simple decision of selecting a platform alone could drive thousands of other dependencies you may not have even thought of previously. This chapter and the next (which is this chapter's companion) will become the cornerstone of all future efforts when you're building your appliance.

Additionally, It's my understanding that if you're reading this book (which I deeply appreciate you choosing by the way!), you're somewhat vested in its core subject matter. As the author, I'm assuming that not only are you interested in **embedded Linux systems** but that you are probably deeply involved in such a project right now. Knowing that, I assume that you also have a good baseline of general technical knowledge. I trust that the level of the information presented here is informative enough to add value to your project and not too general enough to be an affront to your already acquired skills. This chapter and the next are meant to help you connect all the metaphorical dots and illuminate a greater picture of what your solution can become.

> ***IMPORTANT NOTE***
>
> *If you're planning to build a* ***virtual appliance****, this chapter may not be as relevant for you and your team as we will be focusing on hardware. This chapter is mostly geared toward the physical. We will only briefly review virtual appliances. In terms of virtual appliances (or cloud offerings), you'll need to consider what platform(s) you need to create images for. In recent years, virtual appliances have become more and more prevalent, especially in terms of the app stores on any of the* ***hyperscalers*** *(Azure, AWS, or Google). Many still build virtual appliances for OpenStack, KVM, Hyper-V, and (mostly) VMware ESXi.*

Whether you choose to make your products physical, virtual, or both, this chapter will help you and your design team sort out all the criteria to make an appropriate hardware platform selection. Additionally, we'll review how best practices can impact these selections in terms of performance and scalability. These best practices tie in directly to your security footprint but they'll also assist you in building a better product.

In this chapter, we will cover the following topics:

- What are the targeted performance requirements?
- Are there any environmental limitations?
- **Common off-the-shelf** (**COTS**) versus custom-built hardware

- What mainstream hardware platforms are available?
- Hardware and use case criteria will dictate your operating system selection

Let's get started.

What are the targeted performance requirements?

At this point, it's time for us to get technical. Hopefully, the developers of your solution have at least a general idea of what the end-state requirements of the appliance will be. Not having this crucial information at your disposal may make your prototyping exercises rather expensive as you may have to test multiple iterations of hardware platforms.

Regardless of the size and scale of your solution, or whether it's physical or virtual, you must ensure that extensive performance testing is conducted to ensure that the security measures that are implemented within it still allow the solution to perform its tasks as advertised. We'll get into how to integrate these security measures in later chapters.

Another critical thing to consider is the future state of your solution. Ensure that the specifications you set forth for the applications, security measures, and scale of user data have some level of a safety buffer. It's impossible to plan for the performance impact of future security patches or changes in your applications over time. Giving your solution a little extra CPU, memory, and storage can save your support team a trunk full of grief in the future.

For now, let's learn what we need if we decide so that we can create our solution virtually.

Virtual appliances

If you plan to deliver your offering as a virtual appliance, this exercise is going to be far less stressful. You won't be testing hardware! You don't need to worry about procuring a platform that meets your life cycle needs. It's liberating. Deciding on creating a virtual appliance (or a public cloud offering) puts the resource requirements on the customer.

Your customer can then decide exactly how much vCPU, RAM, storage, and networking bandwidth they wish to assign to the virtual appliance. They provide the platform. The added benefit of this is that you know that the operating system that you need to choose must support Intel/AMD as that is the general standard in the enterprise. Your company, of course, can make recommendations for those baselines. In terms of public cloud providers, you can set the baseline VM sizing as part of the marketplace offering.

At this point, the key decision becomes which virtualization platforms you're planning to make your solution available for – VMware, OpenStack, KVM, Hyper-V, Google, Azure, or AWS. Initially, you may start with one platform and then decide to add platforms as it is feasible for your teams.

I would be remiss in not mentioning that a good product manager does their research into which is the best selection for platforming. Depending on said research, you may find division in your target user base and their preferences, their procurement cycles, and, most importantly, their preferences for cloud and/or on-premises virtualization platforms.

Although many solutions cannot be delivered as a virtual appliance, if it is feasible for your product, I'd recommend giving that methodology strong consideration. There's a wealth of flexibility it can offer, something we will cover later in this book.

Now that we've covered the performance of physical appliances, let's consider hardware design.

T-shirt sizing

T-shirt sizing is a method of scaling and delivering a solution based on a desired capacity. It's meant to standardize and simplify the delivery of your solution to your end-customer at the scales most commonly sought after.

Allow me to present an example to help describe the T-shirt sizing methodology. For discussion's sake, let's say that a company wants to deliver a standalone storage solution. This company plans to offer a small, medium, and large solution. It becomes quite clear why they call it T-shirt sizing.

They decide that the small version of the solution will have the following hardware attributes:

- Single multi-core CPU (with low core counts)
- 32 GB of RAM
- Dual-gigabit Ethernet interfaces
- 4x 1 TB SATA SSD drives

Moving on to their next level, they have decided that the medium version of their storage solution has the following hardware attributes:

- Single multi-core CPU (with high core counts)
- 64 GB of RAM
- Four-gigabit Ethernet interfaces
- 8x 1 TB SATA SSD drives

Closing the loop on their sizing, they have decided that the large version of their storage appliance has the following hardware attributes:

- Dual multi-core CPUs (with high core counts)
- 128 GB of RAM
- Four-gigabit Ethernet interfaces
- Dual 10G Ethernet interfaces
- 16x 1 TB SATA SSD drives

These examples have been simplified to illustrate a point. The fictitious company selected three levels, each one being a larger scale than the previous. In the physical appliance world, this sizing methodology will also imply that to create a T-shirt-sized offering, much more work will be needed to ensure each release of the solution delivers the expected performance and security levels consistently. It is not uncommon that the choice to create such stepped-sized offerings may force the usage of different hardware vendors and platforms altogether.

T-shirt sizing your solutions invokes additional work for your teams, but I argue that it is a great methodology to consider. Doing such allows you to deliver your solution to a wider range of audiences with different scales of need. Now that we've reviewed the impact scaling has on design, let's look at other factors in the hardware realm.

CPU/VCPU

The CPU is the single greatest determining factor of everything else in your solution. There is a broad range in terms of performance, power consumption, and price. Your solutions' form factor, environmental thresholds, and performance needs will ultimately drive the decision.

Before we get into all the greatness each CPU family brings to the table, I want you to know about the dark side of this equation. That's right – I'm starting with the bad news and hopefully ending on a positive note in these next few sections. As this is a security-focused book, we must dive deep into the limitations, vulnerabilities, exploitation of said weaknesses, and other general malfeasance that becomes possible when choosing the wrong platform.

Later in this book, we'll dive into all the resources your teams will have available to inform you of new threats, vulnerabilities, fixes, and mitigations. For now, let's just go to the swampy waters of what to avoid today. The following list is based on entries in the **Common Vulnerabilities and Exposures** (**CVE**) database.

It's worth noting that the nasty ones get named, but not friendly human names such as tropical storms, hurricanes, and typhoons. They get names that instill the doom and gloom in those that mention them. Often, the team that discovers the vulnerability will give the vulnerability a logo as well.

As there are currently about 240,000 known vulnerabilities of all types in the CVE database, I will only review some of those that grabbed headlines worldwide. This list will get you started, but don't stop here – continue your research before making a selection. The following list highlights some of the most impactful CVEs that should influence your CPU and platform due diligence. We will also look at their impacts and mitigation techniques:

- **Meltdown**: This family of vulnerabilities, which is contained in multiple CVEs, is centered upon the ability to exploit a race condition in the CPU and gain access to privileged information. It has 5 CVEs – CVE-2023-46836, CVE-2022-42331, CVE-2018-7112, CVE-2018-19965, and CVE-2017-5754:
    - **Impacts**: Intel x86 microprocessors, AMD microprocessors, IBM Power processors, and some ARM microprocessors.
    - **Mitigations**: Most operating systems vendors have created patches to mitigate the vulnerability, but there are serious performance impacts on the systems once these fixes are applied. Newer processors built after 2019 should have hardware and firmware mitigations built in but still, there are new transient execution CPU vulnerabilities as recent as 2023.

- **Spectre**: This family of vulnerabilities, which is contained in many CVEs, is very similar to Meltdown in terms of how CPU conditions are exploited to attain privileged information access by unauthorized users. It has 31 CVEs, so I'll highlight the most important two – CVE-2017-5753 & CVE-2017-5715:
    - **Impacts**: Intel x86 microprocessors, as well as AMD and ARM microprocessors.
    - **Mitigations**: Most operating systems vendors have created patches to mitigate this vulnerability, but there are serious performance impacts on these systems once these fixes are applied. Newer processors built after 2019 should have hardware and firmware mitigations built in but still, there are new transient execution CPU vulnerabilities as recent as 2023.

- **Downfall**: Also known as **Gather Data Sampling** (**GDS**), Downfall is another breed of transient execution CPU vulnerability that attacks the speculative execution functions in the CPU called **Advanced Vector Extensions** (**AVX**). Through this weakness, it is possible to run an exploit to access data contained within the vector registers (CVE-2022-40982):
    - **Impacts**: Intel X86-64 consumer microprocessors of product generations 6 through 11, Intel Xeon X86-64 processors of generations 1 through 4.
    - **Mitigations**: It is expected that version 6.5 or greater of the Linux kernel will have code to mitigate this. However, even in 2024, almost every consumer computer today is vulnerable to it.

- **Foreshadow**: This vulnerability is also known as **L1 Terminal Fault** (**L1TF**). It targets virtual machines, hypervisors, and then ultimately the operating systems involved by attacking memory management (CVEs: CVE-2018-3615 & CVE-2018-3620):
    - **Impacts**: Virtually all families of Intel and AMD microprocessors.
    - **Mitigations**: Processor replacement appears to be the best fix to resolve the problem. This issue has been resolved within the Intel Xeon processor family, starting with the Cascade Lake release. All newer chipsets should be immune to the exploits.

- **Inception**: This vulnerability, also known as **Return Address Security** (**RAS**), is a new type of side-channel attack that targets speculative execution (CVE-2023-20569):
    - **Impacts**: Virtually all recent AMD microprocessors.

- **Mitigations**: AMD has released BIOS and firmware patch bundles to mitigate this exploit. Additionally, operating systems vendors may have released workaround patches. As of February 28, 2024, AMD has updated its published whitepaper with new information on how to mitigate this attack on future AMD processors (https://www.amd.com/content/dam/amd/en/documents/corporate/cr/speculative-return-stack-overflow-whitepaper.pdf).

This list was my version of the top five exploitable vulnerabilities to be actively aware of. Each is generally tied to multiple CVEs and not all have been mitigated to the fullest extent. New variations or similar attacks continue to plague all CPU platforms as threats and AI usage to find new vulnerabilities evolve.

Fueled with that baseline information, you can see how hardware selection impacts your security footprint. As we move on to the next section, remember that all of these factors will resurface when we discuss operating system selection in the next chapter. For now, we'll look into various memory considerations.

Memory

Memory type is deeply tied to the CPU and its associated memory controllers. These are all unchangeable parts of the system's main board or motherboard. Because of the inferred dependencies, this is one of the easier selection points. However, from a security and reliability perspective, there are some differences to consider when making design choices. Each of these factors has a cost and other implications.

In my experience, the only questionable option that you may be forced to consider is utilizing **Error-Correcting Code RAM** (**ECC RAM**) or non-ECC RAM.

What is ECC RAM? It is fault-tolerant memory that's primarily used in high-end servers and workstations. Although it does provide greater resiliency, there can be a mild performance degradation in using this. If you need resilience, chances are that a ~2% hit to speed doesn't matter.

Due to its higher cost than standard RAM modules, ECC RAM is uncommonly found in IoT devices or small-scale embedded Linux systems.

Next, we'll consider a more important factor that can be impacted by your security footprint.

Disk input/output (I/O)

To best understand how this can impact your design, you must know the definition of the term **input/output operations per second** (**IOPS**). Since the point of this book is security, not teaching technical terms, I thought it appropriate to leverage a standard, publicly accepted definition here.

IOPS is defined as "*an input/output performance measurement that's used to characterize computer storage devices such as hard disk drives (HDDs), solid-state drives (SSDs), and storage area networks (SANs). Like benchmarks, IOPS numbers published by storage device manufacturers do not directly relate to real-world application performance.*"

> ***TIP***
>
> *For vastly more on this subject, visit the Wikipedia page on this: https://en.wikipedia.org/wiki/IOPS. Wikipedia is an invaluable public resource that you should consider donating to if it helps you. It sure has helped me over the years.*

Secondly, you must understand the impact encryption has on system performance in terms of overhead and degraded IOPS.

The third and final factor to be considered is what type of storage your solution will leverage. This still matters regardless of whether you are creating a virtual appliance or a physical one. The painful difference between the two delivery methods is that you can control what storage types are built into a physical appliance, whereas with a virtual appliance, you have no control and can only provide recommendations. If the end-user can control what storage class is used, your solution may not function as advertised or – even worse – not be able to function at all if the storage is too slow to meet the applications' needs. So, let's look at what storage choices there are.

In the following storage classification list, I will be focusing on the base unit of storage itself and not going into different mass storage options, such as arrays or network-attached storage. That could be a whole book in itself. So, let's look at your options for individual storage drives:

- **Hard disk drive** (**HDD**) is one of the oldest storage devices for computer systems. Over the decades, the size of the spinning magnetic platters has greatly shrunk in size but gained in terms of storage capacity. Early HDDs had their capacity in megabytes, though today, they can contain as much as several terabytes. Other factors that will limit the overall performance of this storage platform are the bus that connects the drive and the amount of IOPS:
    - **Bus**: SATA
    - **IOPS**: Between 50-80 on average

> ***TIP***
>
> *The bus and IOPS are provided for all noted storage platforms.*

- **Solid state drive** (**SSD**) is a leap forward in storage technology from HDD disks. SSDs have no moving parts and are created in very small form factors. They are generally delivered in the form factor of a 2.5 in disk or card form:
    - **Bus**: SATA or PCIe
    - **IOPS**: Depending on the platform, 4,000 to tens of thousands (or more)

- **Non-Volatile Memory Express** (**NVME**) is the latest in consumer storage platforms. NVME drives are delivered as small cards. Sometimes, multiple NVME drive cards can be mounted onto a larger PCIe card for storage density:
    - **Bus**: PCIe
    - **IOPS**: Depending on the platform, 10,000 up to millions

Now that we've taken a deep dive into some of the most common storage types, let's look at how adding encryption impacts their performance.

Encryption at the disk, volume, or filesystem level can easily be broken down into two delivery methods. These are hardware-based encryption and software-based encryption. Each has its positives and negatives.

Hardware-based solutions are normally delivered via PCIe RAID cards. They can not only allow you to create resilient volumes but also offload the overhead of providing encryption from the main CPU or operating system. The only drawbacks are having proper slots to fit the cards into your solution, and more so the costs of said cards.

Linux Unified Key Setup (**LUKS**) is the standard software-based encryption method for Linux distributions. Widely used across the world in solutions of all scales, LUKS does add a bit of overhead cost in terms of RAM and CPU performance. On scale, it requires external key management, but on an individual system, it will force the user to enter a passphrase before the filesystem can be accessed. This can be cumbersome for booting and updating processes as it requires manual intervention. Other than those factors, it costs nothing to implement.

Encrypting data at rest is considered a cornerstone of systems security. On the other hand, encrypting data in transit is often taken care of at the application or networking layer.

Networking

In this section, we'll review a few options you can implement to add security to your networking profile while reducing the load on your solution's CPU. With each of these options, we'll provide you with some links to further educate you on their technology. Each of these technologies, which will be presented in brevity here, can greatly assist your team in designing a more secure solution.

What I don't want to do here is waste your time with general information on firewalling, VLANs, whitelisting address segments, or other basic network security. As this book focuses on designing, building, and supporting secure embedded Linux systems appliances, I intend to highlight hardware that can assist you. Let's take a look at a couple of interesting technologies that you should consider putting into your solutions.

TCP offload engine

For several years now, some higher-end **network interface cards** (**NICs**) have had additional built-in functionality to offload the entire TCP/IP networking stack from the main CPU of the system. This comes in the form of a solution known as a **TCP offload engine**. These NICs are supported by virtually all operating systems now.

Leveraging one or more of these cards can improve the performance of your solution but realistically, you must still have to account for encryption and security in your applications and systems configuration as well. Moving the networking stack to the NIC allows for other types of application security to run unimpeded on the main CPU.

You can learn more about this technology at https://en.wikipedia.org/wiki/TCP_offload_engine.

TLS accelerator cards

Transport Layer Security (**TLS**) accelerators (formerly also known as **SSL accelerators**) are cards that offload processor-intensive work related to encrypting and decrypting network traffic. The performance boost and network security capability increase by leveraging one of these solutions is beyond substantial. Some high-end appliance manufacturers embed this hardware directly on their mainboards, though they can be added easily via a PCI slot (if available) on the system.

You can learn more about this technology at https://en.wikipedia.org/wiki/TLS_acceleration.

GPU

OK – I know that I may ruffle some feathers here with my take on this subject. In recent years, GPUs have been used more for offloading workloads than processing video output in many appliance solutions worldwide. It can be debated and argued that most modern GPUs are more competent at workload processing than the majority of the mainstream CPU market.

Regardless of your opinion on their usage, offloading the processing of computational power to GPUs is a mainstream function in many Linux systems. This is most commonly associated with efforts in **artificial intelligence** (**AI**) and **machine learning** (**ML**) applications.

This is a design concern based on what may be supported in your chosen operating system (which we plan to dive deep into within the next chapter). Adding a GPU to your system is no small decision. Most of these cards are quite significant in terms of size and they require a notable amount of additional power (provided via direct cables from the **power supply unit** (**PSU**) of the system itself). In most circumstances, this additional power powers the card's cooling fans as they generate heat like an oven!

Implementing a solution that requires a significant GPU or multiple GPUs will require a larger PSU and more significant cooling to be built within the appliance. I felt that it was important to mention this as we will soon discuss the impact environmental concerns have on your solution.

Next, we'll consider other custom hardware that you may require in your appliance.

Custom hardware and peripherals

This is a section that may be near and dear to your heart since you're building a custom solution. There are vast security implications and factors to consider, depending on what your solution will be connected to or contain internally. Building an appliance solution virtually implies some level of customization at not just the software level but the hardware leveraged as well. This is best described by the fact that you are solving an issue or several issues that the end-customer could not account for easily.

Physical security concerns aside, let's take a journey down the path of what security concerns may exist with the usage of custom hardware and/or external peripheral hardware devices. These can be internal or connected to your solution externally. They may require your solution to have third-party drivers and software installed that you have little control over or input in their life cycle.

For externally connected devices, the concern not only resides in software that is out of your control but also in the physical medium to which your solution is connected to these devices. Perhaps your solution is built to control or maintain external industrial machines. The very manner in which your solution connects with them and the protocols in play is the greatest security concern.

If the custom hardware or peripheral needs third-party support and software, your team must ensure that the life cycle and support cycle will align with what you project for your solution. If you're building a solution that's anticipated to last for approximately 5-10 years but the third-party software needed to maintain the functionality may not be supported after only a few years, there is a massive gap. Failure to plan for this is a security problem. If there's a newly discovered vulnerability but your vendor is no longer providing support or updates, your solution is ultimately compromised.

The implications for your product might be substantial here, so I recommend having done a bit of due diligence in confirming the supportability of anything third party that you build into your solution. *The devil is in the details*.

Now, let's move on to another crucial factor – encompassing the implications of how and where your solution may be deployed by your end-customers. This is the environment in which it will reside.

Are there any environmental limitations?

In this section, we will cover a group of often overlapping concerns that will impact the design and physical security of your product. Not all hardware is capable of residing in challenging environments. Let's face the facts here and acknowledge that most electronics need a pretty stable and safe environment in which to operate efficiently. Exposure to the elements, limited cooling, power availability, lack of internet, and physical security all are at play here (and more). These factors may greatly influence whether or not your team can leverage existing off-the-shelf hardware platforms or whether you must design and build a custom platform. There are many vendors worldwide that can assist you regardless, but it's up to you and your team to make that call. Let's look at the factors you must consider.

Power

Power consumption, which is directly correlated to cooling requirements, is yet another crucial calculation and requirement for your solution. Power consumption is rate-limited by the power supply in the hardware platform. These can vary vastly, depending on the scale of capacity and the components in your solution.

Another power consideration has to do with where and how it will be deployed. Let's review some examples.

The average CPU doesn't consume that much power. However, if you have built a solution that has incorporated one or more GPUs, then plan for additional power and aggressive amounts of cooling. This may entail additional components, such as additional fans, liquid cooling systems, or additional heat sinks.

Systems that run too hot don't live long productive lives. They are much more likely to crash randomly and create situations where your solution is compromised or create negative customer experience issues. Heat is a key environmental factor. We will revisit environmental considerations later in this chapter and also discuss how some vendors counter them.

Let's move on to another set of impacting factors.

Offline/air-gapped

Albeit very common for embedded Linux systems appliances, being cut off from the internet is not a bad thing. Being shielded from exposure to external threats can be seen as a positive attribute for a system to have. Sadly, such protection also comes at a cost. Simplistic methods to provide device updates directly from web-based resources become untenable.

These systems require additional coding for their interfaces and alternate methods for processing software updates. This can be as simple as creating a DNF repository behind the firewall for the devices to update or be as complex as having an interface that leverages the ingestion of an update via an encrypted tarball, ISO image, or binary file.

It's paramount for you to understand your target market so that you can plan for this requirement. For some industries, such as the government, this is considered a baseline standard to adhere to. However, there are exceptions, hence why we covered impact levels earlier in this book. Some agencies may allow systems that are not critical nor contain any sensitive data to get their updates from the internet. It happens, but this cannot be assumed to be the standard. Often, the standards are much less forgiving.

What if your appliance is a component in a vehicle? Let's just assume this vehicle itself has no internet access and that its updates are only provided when the vehicles' owners take them to the dealership for service. Sound futuristic? Not exactly. This situation is a reality today. In terms of some of the most advanced vehicles, maybe they can self-update when they are parked outside your home via your Wi-Fi connection. Some even have satellite connections. Can you see where this leads? But mostly, such updates are strictly controlled by the manufacturers. I honestly believe they'd prefer to be able to charge you money via the dealership for such services.

Many appliances are deployed in areas that simply have no luxury of getting connected to a cellular network, Wi-Fi, or LAN. So, simply knowing your target audience and planning to serve them best is key.

We'll talk more about this subject in upcoming chapters and dive deeper into this via practical exercises in future chapters (in which you'll also have access to the resources via this book's GitHub repository).

I could go on about offline systems. I bet you're thinking, *Please don't!*. So, for now, let's move on to the next critical factor we need to plan for.

Climate control

Where exactly will your solution be deployed? This is yet another key bit of information. Why? Brace for impact, because it truly matters. If your target audience works in a desert environment outdoors, for example, then standard off-the-shelf hardware will not suffice. You'll need hardware that can survive in that environment and thrive. Such an example poses other system design concerns. Will this system be in sunlight and be exposed to the elements?

Hopefully, you can see where I am going with this. But if you can't yet, let's take a look at another example situation. Let's say that your appliance is targeting the agricultural industry and that it is mounted on very large tractors that service massive farmlands. Is your solution going to be able to withstand the vast differences in temperatures, humidity, and weather in that situation?

Now, let's consider the ultimate extreme example that has existed for decades. What if your solution were to be a component of something traveling in outer space, such as a satellite or a deep space probe? Is your solution capable of enduring radiation, vast heating and freezing situations, and most of all the prospect of a piece of space dust or debris traveling at 100,000 km per hour?

Environmental conditions can easily compromise your solution if they are not planned for. However, there's also another level of security you need to plan for if a machine is not in a protected location: the appliance's physical security. Few platforms have tamper-proofing built in, so this also needs to be considered if your solution could be in a place outside or exposed to the public. Perhaps you need to ensure the solution has a locking enclosure or other protections.

Protect your solutions from their target environment. Protect your solutions from unauthorized access. Protect your solutions from the end-users themselves. Yes, I said it – protect your appliances from the end-users themselves. The overall environment in which your solutions could be deployed determines our next design factor. Taking all, you've gathered thus far, let's move on to choosing to use off-the-shelf hardware or having something custom-built.

COTS versus custom-built hardware

Yes, I saved this *little* section for last in this group as all the previous ones led to this. This is where I truly hope that you've been paying close attention to the treasure trove of circumstances that can impact your overall final design for your solution.

As part of my intense research for this book, I've reached out to a few of the most likely hardware vendors that impact the embedded Linux systems market here in North America and globally. I was shocked to see how several of them no longer offer fully customized solutions rather than custom bezels and covers, hence why I felt this review needed to be here.

Some smaller or foreign shops will build you a system to specification but know that you will pay a premium for such services. I humbly acknowledge that based on your solutions' needs, this may be your only recourse.

Some of these global vendors, such as Dell, HPE, and Lenovo, are more than happy to create a rebranded version of the hardware they already build. Few, however, at least in the USA, will build a custom system from the motherboard up. Those that will create an entire custom build from the motherboard onwards will not be inexpensive unless the order of scale in the number of systems requested is astronomical.

As customization is at such a premium cost, leveraging an existing form factor/build from a major vendor does have its advantages. Firstly, you know the system that will be delivered to you is supported, has a level of quality, and is easily repeated. The bigger factor in leveraging a vendor like this is to be able to get a system that meets your environmental and form factor needs.

In this section, we'll take a look at a couple of global companies that build good systems that can be leveraged as your platform for your embedded Linux system. My day job has empowered me with excellent access to these two companies (and several others) but I offer the disclaimer that there are more out there across the globe. Both have many excellent products and services that can assist your team in its journey to create the ultimate secure embedded Linux system.

Dell

So, who is Dell and what do they do in the embedded Linux systems space? As one of the largest OEM manufacturers of technology solutions in the world, Dell has advanced capabilities and services that can be tailored to assist their embedded partners in logistics, building, and providing key solutions' life cycle services. In other words, they can help you build, design, and then actually do all the heavy lifting for your team. This means that once you've settled on a design specification, Dell's services can build, ship to customers, and even provide end-customer updates and support. You can learn more at www.dell.com or on YouTube at https://www.youtube.com/channel/UC01FW5V9UVohbPtqKSmXX-w.

Dell makes a wide range of server and workstation products. Another thing that Dell is known for is being one of the world's first major OEM vendors to ship Linux pre-installed on their systems.

OnLogic™

So, who is OnLogic™ and what do they do in the embedded Linux systems space? As mentioned on the official website (https://www.OnLogic.com/company/press/cl250/), *"OnLogic™ is a global industrial computer manufacturer who designs highly configurable, solution-focused computers engineered for reliability at the IoT edge. Their systems operate in the world's harshest environments, empowering customers to solve their most complex computing challenges, no matter their industry. Founded in 2003, the company has offices in the US, the Netherlands, Taiwan, and Malaysia. OnLogic™ has helped more than 70,000 customers worldwide advance their ideas with computers that are designed* *to last, built to order, and delivered in days.* Learn more at www.OnLogic™.com or on YouTube at www.youtube.com/OnLogic™."

OnLogic makes a vast range of customizable systems for those creating solutions, ranging from very small ARM™-based systems, to a range of rugged PCs, and closing the loop with ruggedized modular small industrialized servers. Any of these systems are geared toward a wide range of environmental conditions. These resilient systems empower operations in climates and situations that would destroy lesser systems.

Let's consider an example of an OnLogic system being used in an industrial situation. The following figure shows an ARM-based OnLogic Factor 201™ system mounted in an electrical cabinet and being used as a **programmable logic controller** (**PLC**) that controls functions for machine controls within a factory environment:

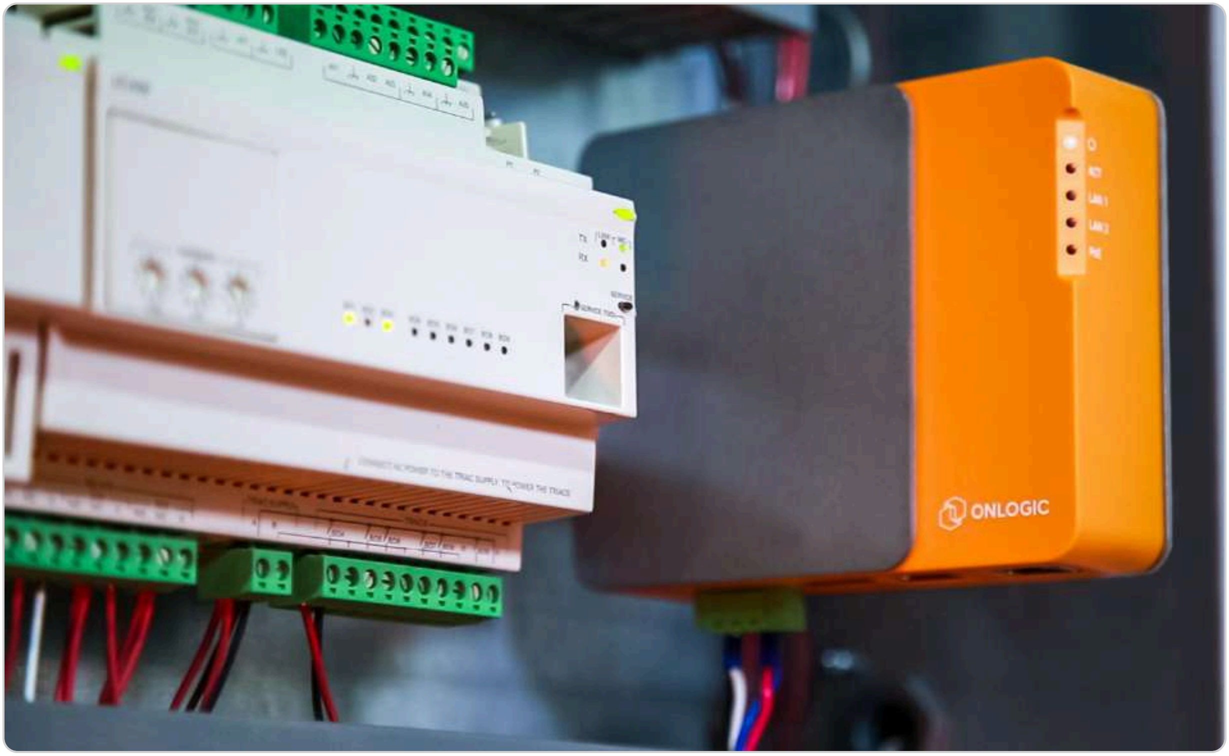


Figure 3.1 – OnLogic Factor 201

For more information on the comprehensive line of OnLogic Factor 201 system solutions, please visit the OnLogic Factor 201 system website, a link to which was provided previously.

Now that we've reviewed a couple of examples of COTS systems, if you don't believe they meet your needs, then it's time to consider custom builds from the ground up. This is a herculean undertaking and not something I have any deep recommendations for.

What mainstream CPU/hardware platforms are available?

Now, let's take a quick tour of the most commonplace platforms (and some not-so-common) that can be found being used by companies building embedded Linux systems. Each of these platforms has a unique performance profile and costs. The following platforms have been used by partners that I have worked with in the past decade. Some of these are vastly more popular than others. Let's dive in.

Xeon™

The Intel Xeon™ processor family has been the flagship of high-performance server and workstation solutions since its introduction in 1998. These processors are commonly used in AI, HPC, database and analytics, and critical application workloads.

For more information on choosing the right Xeon processor for your solution, visit the Intel® website at https://www.intel.com/content/www/us/en/products/docs/processors/xeon/server-processor-overview.html.

Core™

The Intel® Core™ family of processors covers a wide range of devices, from IoT devices to laptops to the highest-end gaming PCs (i3, i5, i7, and i9). This family of processors can effortlessly handle everything from edge computing to business applications to high-end gaming.

For more information on how to choose the right Core processor for your solution, visit the Intel website at https://www.intel.com/content/www/us/en/products/details/processors/core.html.

Atom™

The Intel® Atom™ family of processors delivers a low-power consuming x64 platform that's geared toward appliances and IoT devices. You might even think that these processors were tailor-made for the diverse environmental needs of most networking devices, edge appliances, and definitely for embedded Linux systems solutions. Don't let their lightweight size and reduced power consumption fool you – these processors pack a punch in terms of processing capability. To learn more about Atom processors, visit the Intel website at
https://www.intel.com/content/www/us/en/products/details/processors/atom.html.

Ryzen™

The AMD® Ryzen™ family of processors boasts onboard AI capabilities and Radeon GPU features on-chip. This empowers these chipsets to not just address the gaming PC market but readily provide the ability to process AI workloads within small form factors. AMD is also committed to maintaining the socket technology used by this family of processors for years to come. This is a pretty cool feature if you wish to future-proof your solution or be able to simply change the CPU itself for T-shirt sizing options. To learn more about the Ryzen™ family of processors, visit the AMD website at https://www.amd.com/en/products/processors/business-systems/ryzen-ai.html.

Advanced RISC Machine (ARM™)

ARM® isn't just a platform – it's a licensed architecture and because of this, many manufacturers build custom variations of this architecture into their systems. In recent years, NVIDIA tried to buy the company that owns the rights and patent of the architecture. This didn't happen as planned, but NVIDIA is now known for bringing the most advanced ARM-based solutions coupled with their industry-leading GPU technologies, and these are trending in the ever-growing AI/ML battlefield.

The number of use cases where ARM is used as the platform is as vast as its ability to scale in terms of performance. This ranges from cellular phones to IoT devices and transcends to large-scale data center clusters of ARM with multiple onboard GPU-powered juggernaut servers. All of these require their manufacturers to pay license fees to the ARM corporation. If you're interested in learning more about how this all works, I recommend taking a look at their website: https://www.arm.com/.

With all these different companies making their own version of the architecture, there have been hurdles in the Linux community in trying to support the platform as a whole. Many of these issues will be addressed in depth in the next chapter. So, if you're looking for a hint of what's to be discussed, I'll offer a quick summary: UEFI BIOS compatibility, driver issues, and common tooling are the major hurdles in supporting this platform from a Linux perspective.

Regardless of the perceived issues, some distributions of Linux thrive in this space, while others struggle to keep up and can only offer support for limited implementations of the ARM platform architecture. This platform and the next to be discussed are truly areas to keep your eyes on closely. Let's move on.

RISC-V®

RISC-V® is a new international open standard architecture that provides a framework for anyone to build a solution upon. Licensing is similar to the Open Source Software models. I find this to be game-changing. There is very little out there for support on this emerging platform at the time of writing, but in the next year or so, I suspect that RISC-V may give ARM a significant amount of competition, if not overtake it in the marketplace. Recently, I started playing with this platform in my home lab. Many Linux distributions are rushing to add support for this game-changing platform. You should follow this one closely. Very, very close.

For more information on this open architecture, visit the RISC-V website at https://riscv.org/.

Power® and IBM Z®

These two IBM® platforms are generally known for their massive scale and resilience. IBM Z® is the mainframe platform that has been in production for longer than I can imagine and it's still going strong in the Linux world. IBM® Power® systems haven't been around as long but they too provide a certain niche environment that is still present in some industry verticals.

These systems are the pillars of resilience and uptime. They are the cornerstone of taking damage and still going onward. That said, they are generally not the platform of choice for an embedded Linux systems appliance. They are large and require a very specific set of skills that are no longer prevalent in the IT industry.

This is where my many years of experience come into play. Over the past decade, I've seen a couple of examples where these platforms are delivered as appliance solutions. Yes, I said it. These behemoths are delivered as appliances to their end-customers. These rare solutions generally focus on provisioning advanced storage and disaster recovery solutions for their end-customers, who need such advanced services but find them too complex in today's X86-driven world.

Designing, building, and supporting one of these solutions nowadays is considered a herculean effort. The skills required to master these platforms are becoming more and more rare as time progresses. For more information on the Power platform, visit the IBM website at https://www.ibm.com/power. Additionally, for more information regarding the Z series platform, please visit the IBM website at https://www.ibm.com/z. These systems are greatly resilient but require some deep, deep investments.

With that, we've come to the end of this chapter. So, let's summarize what we've learned.

Summary

In this chapter, we covered a myriad of selection criteria. All of the data that we have reviewed in this chapter will directly relate to what operating system we select in the next chapter. It's codependent. The hardware you choose for your solution will limit your choices of what Linux distribution can support your hardware and requirements properly.

There is no *Easy Button* here for you to push and get answers. The platform you decide upon will be deeply tied to the options you have available for possible Linux distribution choices. This may not be a bad thing as there are many distributions of Linux to choose from. Any assistance in driving the selection may be useful. In this chapter, you learned how many of the complex hardware factors can impact the supportability of your system, limit the security profile, or simply become so complex that you may need to hire more specialized personnel to work with the platform.

The greatest advice I can offer here is that you do your research and be diligent in not limiting yourself in this selection process. Having more options is a good thing. I wish the best of luck to you in this endeavor.

In the next chapter, we will take everything we've reviewed here and apply that knowledge to various selection criteria so that you have the most adequate Linux distribution for your company's solution.

4
Applying Design Requirements Criteria – the Operating System

In the last chapter, we dove deep into many factors to consider in your hardware platform selection criteria. In earlier chapters, we reviewed other factors to keep in mind as well. Everything has been building up throughout each chapter. Everything ties into the security of a solution – some directly, and others more indirectly.

In this chapter, we're hopefully going to educate you on all the factors that your team needs to consider when choosing the right Linux operating system for your appliance. Upon completion of this chapter, you'll be ready to finalize your design criteria and actually start building your initial prototypes.

Since the late 1990s, I must have tried countless distributions of Linux. In the earliest days, new distributions popped up all over the place. Back before high-speed internet, we used to have to buy monthly Linux magazines that contained CD-ROMs with new distributions. I must have bought hundreds of those magazines over the years. Sadly, only a handful of Linux distributions have survived the test of time and popularity. Over all these years, you could definitely say that I have been and continue to be a true Linux enthusiast.

In this chapter, we will cover the following:

- Matching an operating system to your base hardware platform
- Driver support, vendor support, and stability
- Enterprise versus community distributions of Linux
- The life cycle of the operating systems versus your solution
- Hard costs versus soft costs

Let's get started.

Matching an operating system to your base hardware platform

The calculated selection of the Linux distribution to best fit your hardware platform can be compared to the pairing of a fine wine with your gourmet chef-prepared meal. My hope for the outcome here is that your team can take into consideration all relevant factors, along with the findings of your own research, and finally, be able to come to a general design consensus.

This selection process can be quite tedious if you plan to leverage Intel or AMD `x86_64`-type processors, as virtually all present Linux distributions' primary platform is such. It's been this way from the beginning when Linux itself arose as the x86 inexpensive alternative to large expensive RISC systems that run very expensive and proprietary Unix distributions. In 2024, there are at least a hundred distributions of Linux geared toward the `x86_64` platform. Most are obscure forks of mainstream distributions. I'd say it's pretty fair to assume that there's a distribution out there for anyone's desires.

Where the selection gets narrowed down for you is when you decide to build upon a non-x86_64 platform. Power, System z, Arm, or RISC-V as your choice may certainly cut down what you can choose from to fewer than a handful of distributions. Some may see this as limiting, while others might embrace not having to research or even trial implementations of many distributions in their labs.

For example, here in my own home lab, I maintain over 20 systems, all paid for by me (not my employer). Although they are expensive to maintain, it is key for me to be able to address the needs of multi-platform and operating system testing. This also affords me the freedom to build what I want, when I want, and utilize whatever tooling I deem appropriate for the task. I build, test, tear down, and repeat over and over again, changing operating systems or the hardware platform itself as needed.

Obtaining a strong understanding of the factors considered when pairing your chosen hardware to a secure operating system can be compared to how a chef in a fine restaurant pairs an excellent wine with their culinary delights. The next few sections in this chapter will do just that. We are heading to our architectural kitchen now, so let's move on and get cooking.

IBM Power

The IBM® Power® series servers are quite scalable and have quality hardware. These systems operate quite similarly to a mainframe in their ability to carve out sections of physical hardware and dedicate them to an individual operating system. In the Power series, this functionality is referred to as an **LPAR** (**Logical Partition**).

This platform is becoming more and more rare, as lightweight systems are plentiful in the embedded Linux systems space.

This platform is regarded as a niche for those who utilize it today. Be aware that your operating system choices are greatly limited to paid-for distributions. Red Hat® Enterprise Linux®, SUSE Linux Enterprise Server™, and Ubuntu® appear to be the selections available today.

Let's move on to the next platform.

IBM System z

The IBM Z® series (AKA System z) mainframe is obviously the most expensive possible hardware platform for any possible embedded Linux system appliance in existence and also, in my opinion, the ultimate example of the rarest platform for such appliances in existence today.

Additionally, this platform is the largest in physical size and, in my professional opinion, quite possibly the hardest to find staffing resources for. If you don't absolutely demand what benefits the mainframe provides (like massive amounts of hardware-based redundancies), I will definitely state this is not the platform for you. That said, I also doubt that, to this day, there is a hardware platform more fault-tolerant and resilient than the Z series mainframes.

However, in the context of today, I must use the great analogy of pets versus cattle, which is basically the service model of cloud-based technologies – using vastly inexpensive expendable machines clustered together (i.e., the cattle) versus a single expensive monolith. As there are countless references to this on the internet, I will leave you to make your own opinion on this methodology.

Ultimately, I must concede that the Z machines are the epitome of resiliency and reliability. Sadly, their size and cost alone take them out of 99.99% of the embedded Linux systems market itself. The Z machines are literally the most expensive pets.

Options here are limited. Much like the Power platform, with the Z Series, you are limited to only paid-for distributions such as IBM LinuxOne, zLinux, and Red Hat Enterprise Linux. These seem to be the limited offerings on the market today for those unorthodox embedded Linux platforms.

Let's move on to another platform.

RISC-V

The latest shiny object in the IoT space is RISC-V® systems. I recently just picked up my first of these **bleeding-edge systems**. Currently, when choosing this platform, your options are not just limited but, additionally, harder to get on the system itself. Remember that this platform is rapidly evolving and growing daily. That said, I personally have high hopes for this emerging platform.

As the test unit that I've been playing with does not have a UEFI BIOS, installing any operating system on this device is somewhat tedious.

Here's a picture of the sample RISC-V IoT board I was able to secure from Amazon.

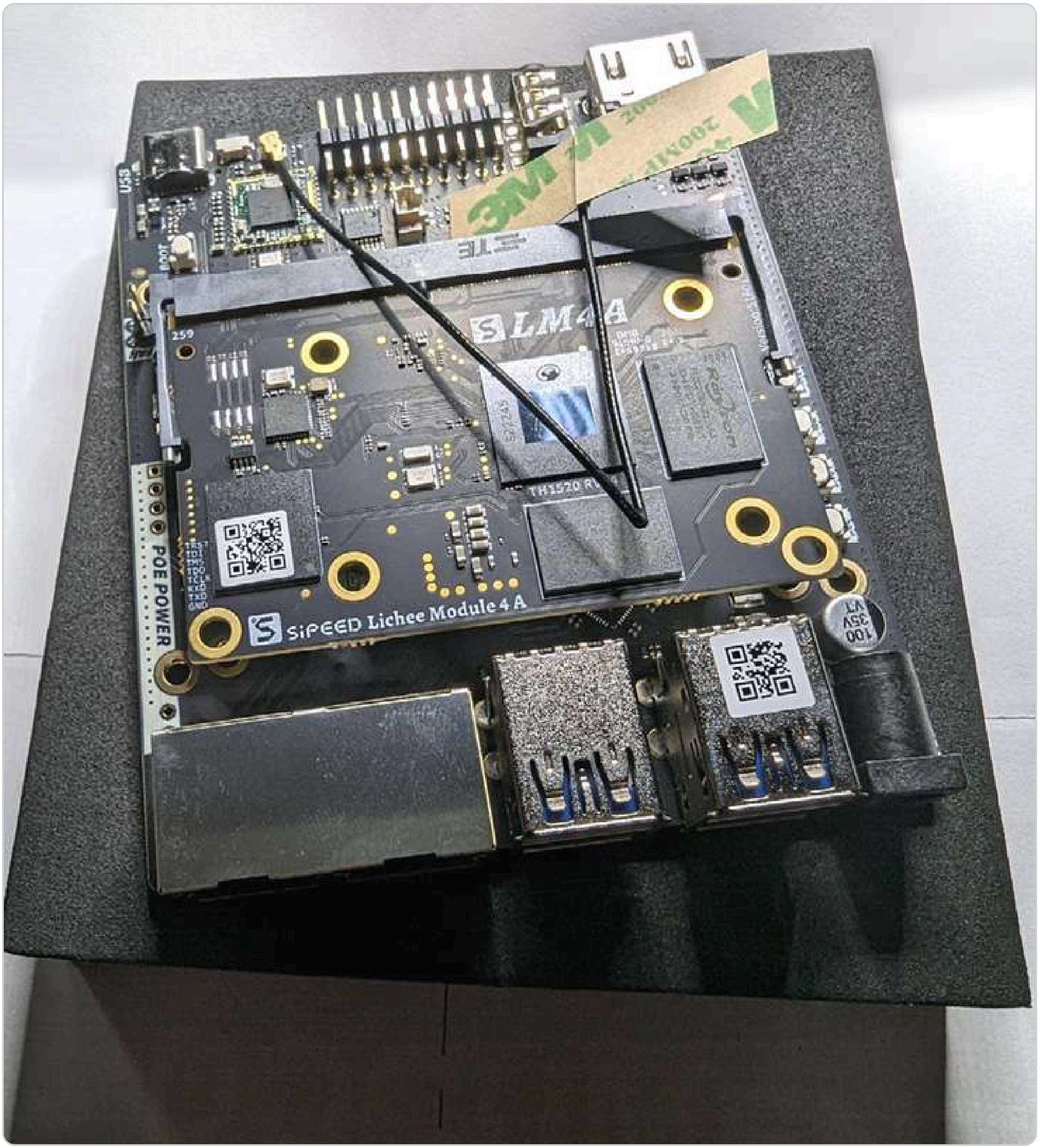


Figure 4.1 – Initial generic RISC-V boards available on Amazon

I have just recently started playing with this new hardware, so at the time of writing, support for this emerging platform is quite limited. This is where community distributions really shine, as they are at the bleeding edge of hardware support. Thus far, what I have found available is some compatibility with Debian, Ubuntu, and Fedora®.

Ubuntu seems to be the leading distribution here, with Fedora closing in fast. That said, I'm looking forward to a future where paid enterprise-grade operating systems place this new platform under their support umbrella.

We'll definitely be talking more about this platform in the future. For now, let's move on to what might just be the most important platform in the entire realm of embedded Linux systems. Let's move on to the **Advanced RISC Machine** (**ARM**) architecture.

ARM

Over the past few years, the ARM® architecture has gone deep and wide in the ecosystem of embedded Linux systems. As we covered in the previous chapter, ARM is a licensed architecture from the company, Arm, or a blueprint if you will. Hardware manufacturers pick and choose what chips they're going to put onto their custom boards. Additionally, there's also a treasure trove of pre-built, extremely inexpensive ARM systems, such as Raspberry Pi-type boards.

In this ecosystem, Linux distributions that are the closest to the bleeding edge of what's new in the Linux world tend to have the greatest adoption. In other words, the community distributions (where most of the latest developer work is tested) are likely to have the deepest compatibility with the newest boards. Distributions such as Raspberry Pi OS, Ubuntu, and Fedora are key players in this space. Other open source projects such as Yocto (https://www.nvidia.com/en-us/data-center/) also provide Linux users with great build and deployment tools.

This gap between community and enterprise is closing fast. With the explosion of IoT devices being created worldwide, the enterprise market has been forced to join in or be left behind. Wind River Linux appears to be a leader in this, especially in terms of industrial ARM systems. The Ubuntu Pro server, SUSE Linux Enterprise Server, and Red Hat Enterprise Linux are not that far behind in their efforts to gain their own lion's share of the ARM ecosystem.

The edge and IoT market is not the only market that the enterprise distributions want to capture. As NVIDIA is now bringing the ARM platform to the data center with their new super-powered AI processing juggernaut systems, which combine large-core-count ARM processors with their GPU chips, NVIDIA Grace Hopper systems will change the computer and operating system landscape globally.

ARM support in all form factors is improving fast. I credit NVIDIA for bringing them to the data center and driving greater platform awareness. ARM is definitely a great platform to consider. Let's move on.

Driver support, vendor support, and stability

As we go through all the hardware platforms that are available in the Linux ecosystem, I find it crucial to remind you and your team to ensure that whichever Linux distribution you consider also meets your products' needs (and your customers' expectations).

What do I mean by that? Basically, I am recommending that you and your team evaluate more than one operating system before settling on a winner. Ensure that your hardware performs at the expected rate with the candidate operating system. Record metrics. Record package versions. Stress-test your prototypes. This process guarantees that when you compare and contrast the candidates, your team can best decide on what truly should be the best selection.

Not all distributions are based on the same kernel versions, which directly impacts which drivers may be included (or, in some cases, deprecated from the distribution). Additionally, depending on your planned hardware, you may also require third-party drivers not included with your distribution. Will these drivers be compatible with your distribution's kernel and libraries?

Moreover, each distribution is curated with a different set of core packages and an extended set of optional packages. Will your distribution support the correct package/version combinations required by your appliance's applications stack? Will you need to seek third-party libraries or applications to complete your solution?

Why am I making you ask all these questions of yourself and the team? Allow me to explain. Sadly, it's pretty simplistic. Most community distributions do not fully test combinations of packages along with their base kernel. Some community distributions are basically a kitchen sink of what their developers assume you may want to try out, but in actuality, they're hoping you can assist them in vetting stability and bug reporting.

This is again where I make my argument for the adoption of an enterprise Linux operating system of your choice. Enterprise distributions pride themselves on stability, security, support, and binary compatibility because that's what business demands from them. So, let's move on to the next section where I dive deeper into this thought process.

Enterprise versus community distributions of Linux

In this section, we will discuss community versus enterprise Linux distributions, or, as many of us in the Linux industry refer to them, *free versus fee*. What distribution you decide upon is driven by many of the factors we have reviewed in previous chapters. Compliance, support, longevity, and your own general needs will come into play here.

This point of the selection criteria not only impacts the bottom line of building the appliance but can also better position your product as more secure, as the enterprise distributions bring security features, documentation, training opportunities, automation, tools, and best practice acumen to support your team.

Paid distributions tend to carry significant longevity, as it's expected by the customer base. The average supported enterprise Linux lifespan is 10 years. These distributions also tend to offer an extended timeframe beyond that (for additional fees). This is important to ensure that your team can get access to critical security fixes, even when the community-driven operating systems have ceased supporting their distribution.

A great reason why you should consider an enterprise distribution over a free community one is software supply chain security. This is a topic that we will touch upon many times in this book. In March 2024, the highest level CVE was discovered in distributions that are the closest to the upstream developers. *CVE-2024-3094* (https://www.cve.org/CVERecord?id=CVE-2024-3094) is caused by the supply chain of a software package called **xz** having had its GitHub repo compromised, where malicious code was injected into the package. This malicious code then interferes with OpenSSH by creating a backdoor into the system.

This vulnerability directly compromised hundreds of thousands of systems that run community-based Linux operating systems; however, those customers of enterprise Linux distributions, such as Suse or Red Hat, were unaffected, as their distributions' supply chains were considered heavily secured and well-tested.

As we're talking about community versus enterprise, I thought it prudent to show you a contrast listing of examples for both. There are actually hundreds of small community distributions out there, but the majority of those most likely to appear in an embedded Linux system appliance are listed here.

Here are some examples of good, free-to-use community distributions that could be used in an embedded appliance:

- Fedora
- OpenSuse
- AlmaLinux™
- Rocky Linux™
- Ubuntu
- Debian
- Arch Linux™
- Slackware

- Raspberry Pi® OS

Here are some examples of paid enterprise distributions that offer greater security, support options, and possibly extended life cycle options:

- Red Hat Enterprise Linux
- Ubuntu Pro Server
- Suse Enterprise Linux Server
- Oracle Enterprise Linux
- Wind River Linux™

Now that we've reviewed a major swath of distributions by listing the more common free/community distributions and contrasted them with the vastly more stable and secure enterprise offerings, I trust that you can see the true attributes in their comparisons. Let's move on to see how they stand the test of time.

Lifecycle of operating systems versus your solution

Ensuring that the support window of time for the physical hardware that your solution will be built upon aligns with the support window and availability of the Linux distribution you have chosen is crucial.

Changing an operating system on an appliance in the field is an effort in pain and suffering. It's generally a safe assumption that your support team will not have direct access to any system sold that's in use by a customer in the field. Unless your offering is supported as a remotely managed service for the end-customer, it's common for embedded systems to have no communication with the mothership (i.e., its manufacturer), except for those that are allowed to acquire their updates from the vendor automatically online.

Changing an operating system is like performing a fresh installation. Generally, we try to keep this out of the end-customers' view, as if they are involved and have access during the process, it can be easily compromised and grant the end-customer access levels that were not intended for them. This puts your solution at risk of having its configuration altered or broken, or levels of access granted to those who are not authorized. Yes, it's a security problem!

So, choose your operating system and hardware wisely. Plan for extensive buffers of time in the support windows of hardware and Linux distribution. If this cannot be aligned completely, ensure that you have a plan to manage upgrades in the future that keep a solution secure, while preventing data loss. Keep the end-customer as far away from this process as possible (wherever feasible). I think you get my point. Let's move on.

Hard costs versus soft costs

This discussion might pose a tricky situation for your company, yet it's a discussion that must take place. This will definitely impact design decisions and future support structures. Ultimately, what you charge for your solution must be reasonably above what it costs to design, build, and support it. I must strongly advise you and your team to work out all cost models extensively in the design phase of your project. Feature creep, training costs, logistics costs, technical debt, delays, research and marketing budgets, and other factors are all deeply in play here. Simply, this ties into security in that if not all costs are planned for beforehand, it's more than probable you will operate on a fixed budget, and something may not get the attention it fully deserves, thereby creating a situation where maybe something gets overlooked and, in itself, becomes a security vulnerability.

Cost models drive everything in the modern world. Profit matters. Finding the balance between minimizing costs and providing the best product possible is a constant challenge that product managers face. It's up to you to determine what the proper balance is for your product. I beg you to not cut corners on factors that impact your product's security. Having a true understanding of your labor costs, hardware costs, software costs, and what your research has determined to be the truest sweet spot for your product's price to the customer will have a great impact on your decision-making. Let's look at some of those factors in the next sections.

Hardware costs

These costs may be the easiest to account for yet the largest expenditure for your project. These go beyond the platform you have chosen as well. You may need to build out a whole new toolchain in order to build and support your appliances. I encourage you and your team to ensure that you plan for a decent reserve of systems, keeping them on hand for testing and troubleshooting.

As mentioned in the previous chapter, custom hardware has additional costs. These are not just monetary but can also apply to how much time is spent. Custom hardware takes longer to design and prototype; moreover, it often requires substantially more testing. If you plan on utilizing an enterprise Linux distribution, the company that provides it may require you to certify your hardware for compatibility with their platform. Failure to account for that may also result in the hardware solution not being supportable by the operating system's vendor. This can create problems down the road for your company, either perceived or actual.

I understand that the drive to cut costs in any organization is paramount, but I feel compelled to encourage you to account for and acquire sufficient resources to maintain a group of systems for future developments, and to troubleshoot any possible future end-customer issues. Testing directly impacts the quality and, yes, the security of your product. Don't skimp on testing.

Let's move on to other costs.

Software costs

Just like with planning for hardware costs, planning for your team's software needs is no less important. This goes beyond simple operating system costs for the product itself, as it extends to every aspect of designing, building, and supporting your appliance solution. All software must be accounted for – operating systems, applications, source code control systems, automation systems, security scanning systems, backup and recovery systems, and so on.

Partnering with your software vendors can greatly assist in reducing costs. Additionally, such partnering may be required in order to legally utilize their product as part of your own. These partner programs generally also have a cost of membership, whether it is a program fee, minimum purchase requirement, or requirement to maintain a subscription to software or services.

Soft costs

Soft costs are the creeping death of any project, as they can be difficult to account for or forecast. While hard costs refer to materials, hardware, and software, soft costs are generally related to your staff. More specifically, this is impacted by what skill sets they currently have versus what they need to design, build, and then support your desired appliance solution.

These needs may not be apparent when your design team prototypes the appliance. It's quite possible for there to be knowledge gaps between those designing/building the embedded Linux systems and the teams tasked with providing the customer support services. These gaps must be filled with training and knowledge transfer.

Not addressing these knowledge gaps can cause many problems. What if your design team is not familiar with the targeted platform, or even worse, its chosen operating system? This leaves the end design open to possibly countless vulnerabilities that, in this case, may not have been thought about. Additionally, failure to address any gaps in the skills required to put forward the best product will ultimately induce technical debt that in itself is a security risk.

It is best to fill these gaps with training or, in a worst-case scenario, with new hires or consultants who can ramp up the rest of the product and support teams. Regardless of where the knowledge gaps reside (e.g., hardware, configuration, operating system, or general usage), it becomes a security challenge if not addressed.

Okay, I'm done beating this subject to death. I trust it was not too painful. Granted, I do acknowledge that I can sometimes appear overly dramatic when I want to ram a point home. But let's not underestimate the true value of just how many skills each team member brings to the design table and beyond it. Not bringing your *A-game* has a negative impact. Let's move onward.

Summary

All we have reviewed thus far has been leading up to this moment. In the past four chapters, we have covered a wealth of factors that impact design criteria, wrapping all that up in this chapter with the Linux operating system that is to be the foundation of your applications within your solutions.

The lessons that you have learned from this chapter are complex. There are many great Linux distributions and companies out there. Each of their distributions or, in some cases, their custom spins, which are variants of their distributions, target specific hardware platforms or end-user use cases. These use cases can range from custom graphical interfaces to industrial control management, from artificial intelligence-driven analytics to advanced security policy enforcement.

We have covered a lot in this chapter, so let's summarize some of the points you need to remember. Your hardware selection will totally empower or limit you in terms of the Linux distributions you choose. Within the vast galaxy of Linux distributions, hopefully, you now have a better understanding of the differences between free community distributions versus the stable and secure, paid-for enterprise offerings. You should also understand how the life cycle of your hardware and software selections can empower or limit your solution. Finally, your team should also understand the overall cost model relative to the decisions you have made. Altogether, this forms the basis of your architectural plan for your product.

Now, it's time to put what you have learned to the test. We'll move on to our next chapter, which will contain practical exercises for setting up your build chain, tooling, and implementing some of the technologies we mentioned earlier, along with some things that may be new to you.

Part 2: Design Components

In this part, we'll dive deeply into security-related activities for your appliance while getting hands-on with advanced Linux configurations and security technologies in your lab.

This part has the following chapters:

- *Chapter 5, Basic Needs in My Build Chain*
- *Chapter 6, Disk Encryption*
- *Chapter 7, The Trusted Platform Module*
- *Chapter 8, Boot, BIOS, and Firmware Security*
- *Chapter 9, Image-Based Deployments*
- *Chapter 10, Childproofing the Solution: Protection from the End-User and Their Environment*

5
Basic Needs in My Build Chain

Security begins at the design table. True. Yet it is enhanced and comes alive in the build chain. It is here where we will implement and validate our security policies, test, scan our prototypes, and leverage all the informational sources at our disposal to ensure that our product is as secure and robust as it possibly can be, long before it sees its first customer.

Here is truly where all the magic happens (or sadly, it doesn't if you choose to ignore it). In the previous chapters, we have addressed countless design factors that have led up to your conceptual initial design criteria. From here and forward, we will be working hands-on to ensure those embedded systems' best practices are adhered to along with the application of security measures.

Here's a non-exhaustive sample of what may be in your company's build environment. These tools should be accessible only to those working directly on this product or supporting production releases. From secure repositories to scanning tools and everything else in between, these critical tools will greatly contribute to the success of your product's lifecycle.

The many tools you will need are vast and include (but are not limited to) the following:

- Local repositories
- Source code control systems
- Project management systems
- Customer support systems
- Build automation systems
- Compliance and security scanning systems
- Update infrastructure for deployed appliances
- Most importantly, a comprehensive set of test systems

Here's a perspective of systems and tools needed for a successful build chain infrastructure:

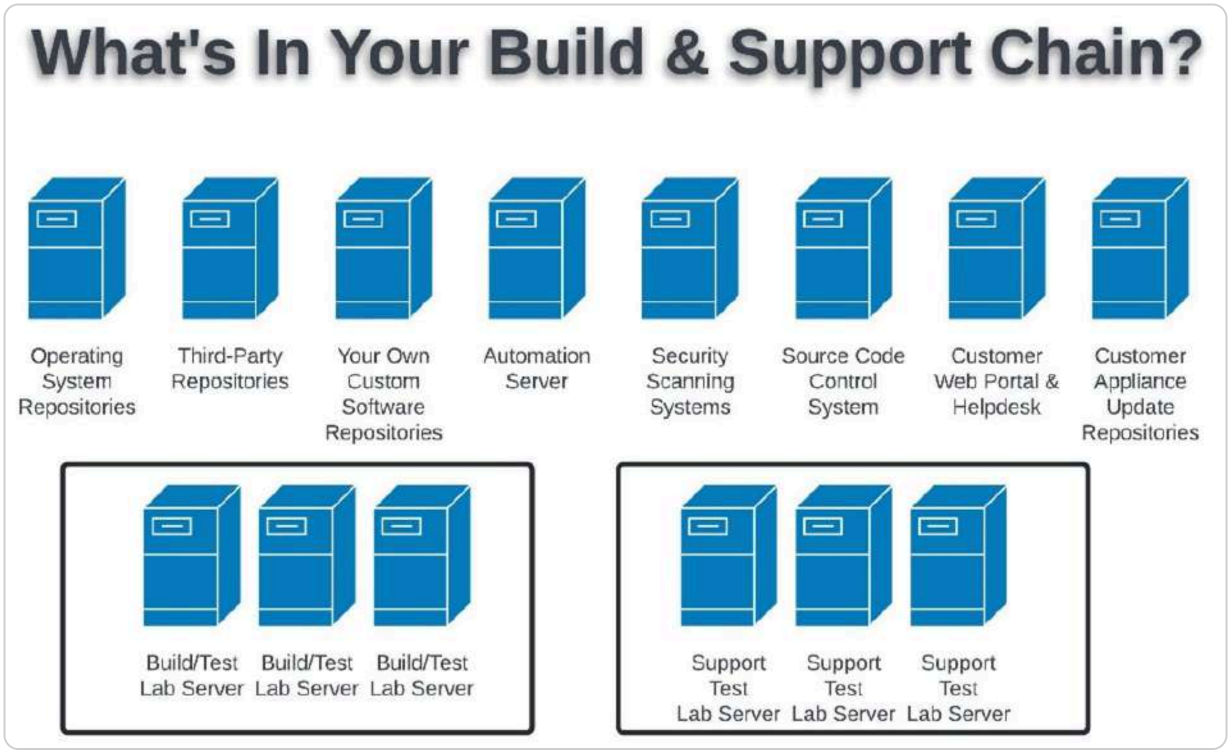

What's In Your Build & Support Chain?
Operating System Repositories
Third-Party Repositories
Your Own Custom Software Repositories
Automation Server
Security Scanning Systems
Source Code Control System
Customer Web Portal & Helpdesk
Customer Appliance Update Repositories
Build/Test Lab Server
Build/Test Lab Server
Build/Test Lab Server
Support Test Lab Server
Support Test Lab Server
Support Test Lab Server

Figure 5.1 – Example systems in a secure build and support chain

Grab your favorite beverage. Clear your calendar. Grab some USB thumb drives and let's go to the lab. I invite you to join me on this journey, roll up your sleeves, and get your keyboard thumping. Are you ready to start practicing how to secure your software supply chain?

In the following sections, I will address key concepts and then walk you through some detailed exercises:

- Software supply chain control
- Automation and tool integration – a brief overview
- Security scanning, testing, and remediation
- Manifest and configuration tracking
- Update control mechanisms

Let's get started.

Technical requirements

If you'd like to follow along with the exercises within this chapter, you will need at least two machines (physical or virtual) running the same distribution of Linux. In the exercises themselves, I will call out which distribution I have used for the exercise and any other pertinent configuration information. For the purposes of this book, it is assumed that you and your team have a substantial level of experience with whichever solution(s) for source code control/management.

This book itself has its own GitHub page and repository. Many of the exercises and example configuration files can be found there:

https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook/tree/main/Chapter05/exercises

Software supply chain control

As a solution provider, it is your responsibility to know, track, and maintain records of each component that goes into your solution. If your solution falls under any sort of government or industry compliance regulations, this requirement may have dire consequences if not maintained.

This is a situation where again, I recommend leveraging an enterprise Linux distribution. Their software sources are secure, from source code to compilation to packaging and, ultimately, its delivery to you. They maintain great records. They have to! These vendors can provide your team with what we call a **Software Bill of Materials** (**SBOM**). This is a complete listing of the components and their versions. Additionally, it is an attestation that they use secure, tested, and validated software.

There are several commercial solutions available on the market. Companies like Aqua, Synopsys®, and Red Hat® (just to name a few) create some excellent comprehensive solutions to securing the software pipeline. There are lots of viable options for your team to consider in this aspect, so let's move onto the next section, Source code control.

Source code control

Intellectual property, that very special code that makes your solution oh so special, is meant to be protected. Sometimes, at virtually all costs.

There are many excellent commercial and open source solutions for this. Oftentimes, these solutions are integrated with other solutions that may provide services for bug tracking, agile project management, software packaging solutions, etc. Regardless of which source code control system your organization has chosen, I recommend ensuring that all feasible steps to minimize access and secure the platform are taken. Protect your company's intellectual property. Let's move on to other parts of a good software supply chain.

Automation and tool integration – a brief overview

No DevOps shop can be complete without a substantial level of automation throughout the operation. Just like with source code control, there are many excellent commercial and open source solutions for this. Red Hat Ansible®, SaltStack®, Puppet Enterprise®, Chef®, Ansible AWX®, and Puppet Bolt® are the most commonly used.

Other products often found in a build chain may also include a significant level of automation. One such example is the platform offered by a company called CloudBees. Their Jenkins product has been at the leading edge of DevOps shops globally. This is just one example in a market of many solutions.

Automation is an area that I can rant about for hours on end. That is definitely not what I intend for this brief section. The message I want to impart to you here is that automation, when done correctly, can ease the burden of many cumbersome and menial tasks that can be error-prone when humans are doing them repetitively. Risk mitigation and efficiency, pure and simple.

Whichever tool (or tools) your team selects for the project, I recommend their usage and efficiency be reviewed in the same manner you would review any other agile process at the end of a sprint. There is no wrong selection here, except in the case of not leveraging some sort of automation at all in your build, test, and support chain. Don't just find something. Find what works best for your budget and your team's skills, and can be leveraged easily within the timeframes you are allotted for the project.

We will definitely be covering much more about tools and integration in later chapters, so let's move on.

Security scanning, testing, and remediation

By using **Free Open Source Software** (**FOSS**), your team can achieve many, if not all, aspects of your product's lifecycle. Yes, I am definitely an open source advocate and have been since the late 1990s. There's a good reason, especially in the Linux world. It's the portability of skills.

Virtually all Linux distributions share about 90% of the same available commands and utilities. What they don't share, that other 10%, is what makes those distributions unique, scalable, or more secure than the others. For discussion's sake, I want to focus on that 90% – the common stuff.

When at a command prompt in Linux (regardless of distribution), there's a baseline of commands we all come to know and rely upon. Granted there are variations out there, but the most common tools are what I am focusing on here.

The first command line tool I want to bring up is **Nmap**. Nmap has a GUI counterpart called **Zenmap** (or in older distributions, **nmap-fe**). It is commonly found in virtually all distributions of Linux and on all platforms. Why? Because it is great at finding out which ports are open and even doing something called **OS fingerprinting** (i.e., figuring out what operating system the host is using).

So, let's take a quick look at Nmap and Zenmap in action in a quick hands-on exercise.

Exercise – executing a network port scan

In this exercise, we will be running detailed network port scans on some test machines. We'll be using two similar tools. The first tool, called Nmap, is an open source command-line port scanning tool. The second tool, called Zenmap, is also an open source tool; however, it has a graphical user interface.

> ***REQUIREMENTS FOR THE EXERCISE***
>
> *For this exercise, you'll need access to a Linux host with its graphical desktop enabled along with a defined target host (to scan). In the accompanying screenshots, I've used Fedora® as the Linux distribution but virtually any distribution is acceptable here. Nmap should be available from your distribution's own repositories, but we'll need to download Zenmap from Flathub™. On Fedora, this repository is preconfigured for you if you enable third-party repositories in the gnome-software application.*

First, you need to log in to the host and open a terminal session. We can usually assume that it's not installed by default, hence we will install it before continuing. If you've already installed the tools, you may skip this step.

Then, we'll install the Flathub `flatpak` repository (if it's not already installed):

```
$ flatpak remote-add --if-not-exists flathub \
https://dl.flathub.org/repo/flathub.flatpakrepo
[mstonge@bm03 ~]$
```

Next, we'll confirm that the Flathub repository is available at the command line.

> ***TIP***
>
> *Please note that even if this is configured within gnome-software, it may not be available on the command line unless you physically install/enable it yourself.*

```
$ flatpak remotes
Name    Options
fedora  system,oci
flathub system
```

Now we'll install the `zenmap flatpak`.

```
$ sudo flatpak install zenmap
Looking for matches…
Found ref 'app/org.nmap.Zenmap/x86_64/stable' in remote 'flathub' (system).
Use this ref? [Y/n]: y
Required runtime for org.nmap.Zenmap/x86_64/stable (runtime/org.gnome.Platform/x86_64/45)
found in remote flathub
Do you want to install it? [Y/n]: y
org.nmap.Zenmap permissions:
    ipc    network    x11    file access [1]
    [1] home
        ID                                  Branch          Op
Remote           Download
 1. [✓] org.gnome.Platform.Locale           45              i
flathub           18.1 kB / 369.6 MB
 2. [✓] org.gnome.Platform                  45              i
flathub          242.7 MB / 378.4 MB
 3. [✓] org.nmap.Zenmap                     stable          i
flathub            7.0 MB / 8.6 MB
Installation complete.
```

Next, we'll install Nmap out of the operating system's own repository.

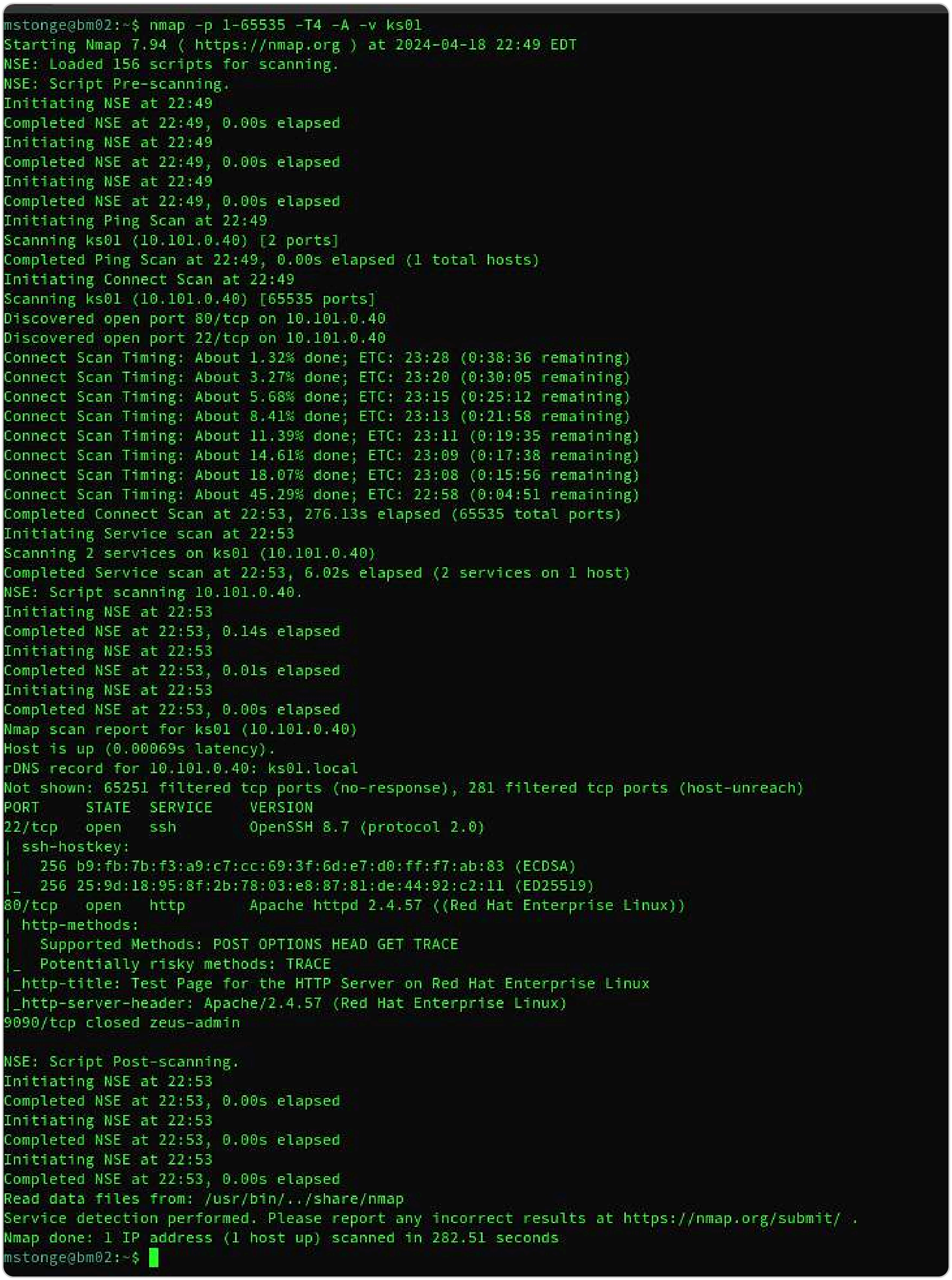
```
$ sudo dnf install -y nmap
```

We will get the following output:

```
Last metadata expiration check: 0:00:27 ago on Tue 30 Apr 2024
09:08:18 PM EDT.
Dependencies resolved.
=========================================================================
 Package                Architecture            Version
          Repository                 Size
=========================================================================
Installing:
 nmap                   x86_64                  3:7.95-1.fc40
            updates                  5.8 M
Transaction Summary
=========================================================================
Install  1 Package
Total download size: 5.8 M
Installed size: 25 M
Downloading Packages:
nmap-7.95-1.fc40.x86_64.rpm
            299 kB/s | 5.8 MB     00:19
-------------------------------------------------------------------------
--------------------------------------------
Total
            297 kB/s | 5.8 MB     00:20
Fedora 40 - x86_64 - Updates
            1.6 MB/s | 1.6 kB     00:00
Importing GPG key 0xA15B79CC:
 Userid     : "Fedora (40) <fedora-40-primary@fedoraproject.org>"
 Fingerprint: 115D F9AE F857 853E E844 5D0A 0727 707E A15B 79CC
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-40-x86_64
Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
                                         1/1
  Installing       : nmap-3:7.95-1.fc40.x86_64
                                         1/1
  Running scriptlet: nmap-3:7.95-1.fc40.x86_64
                                         1/1
Installed:
  nmap-3:7.95-1.fc40.x86_64
Complete!
```

Here's an example of me running a scan in my network with the Nmap utility.

It's actually me running a detailed port scan against a host called `ks01`. What we will see is the open ports and the services running.



```
mstonge@bm02:~$ nmap -p 1-65535 -T4 -A -v ks01
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-18 22:49 EDT
NSE: Loaded 156 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 22:49
Completed NSE at 22:49, 0.00s elapsed
Initiating NSE at 22:49
Completed NSE at 22:49, 0.00s elapsed
Initiating NSE at 22:49
Completed NSE at 22:49, 0.00s elapsed
Initiating Ping Scan at 22:49
Scanning ks01 (10.101.0.40) [2 ports]
Completed Ping Scan at 22:49, 0.00s elapsed (1 total hosts)
Initiating Connect Scan at 22:49
Scanning ks01 (10.101.0.40) [65535 ports]
Discovered open port 80/tcp on 10.101.0.40
Discovered open port 22/tcp on 10.101.0.40
Connect Scan Timing: About 1.32% done; ETC: 23:28 (0:38:36 remaining)
Connect Scan Timing: About 3.27% done; ETC: 23:20 (0:30:05 remaining)
Connect Scan Timing: About 5.68% done; ETC: 23:15 (0:25:12 remaining)
Connect Scan Timing: About 8.41% done; ETC: 23:13 (0:21:58 remaining)
Connect Scan Timing: About 11.39% done; ETC: 23:11 (0:19:35 remaining)
Connect Scan Timing: About 14.61% done; ETC: 23:09 (0:17:38 remaining)
Connect Scan Timing: About 18.07% done; ETC: 23:08 (0:15:56 remaining)
Connect Scan Timing: About 45.29% done; ETC: 22:58 (0:04:51 remaining)
Completed Connect Scan at 22:53, 276.13s elapsed (65535 total ports)
Initiating Service scan at 22:53
Scanning 2 services on ks01 (10.101.0.40)
Completed Service scan at 22:53, 6.02s elapsed (2 services on 1 host)
NSE: Script scanning 10.101.0.40.
Initiating NSE at 22:53
Completed NSE at 22:53, 0.14s elapsed
Initiating NSE at 22:53
Completed NSE at 22:53, 0.01s elapsed
Initiating NSE at 22:53
Completed NSE at 22:53, 0.00s elapsed
Nmap scan report for ks01 (10.101.0.40)
Host is up (0.00069s latency).
rDNS record for 10.101.0.40: ks01.local
Not shown: 65251 filtered tcp ports (no-response), 281 filtered tcp ports (host-unreach)
PORT     STATE  SERVICE    VERSION
22/tcp   open   ssh        OpenSSH 8.7 (protocol 2.0)
| ssh-hostkey:
|   256 b9:fb:7b:f3:a9:c7:cc:69:3f:6d:e7:d0:ff:f7:ab:83 (ECDSA)
|_  256 25:9d:18:95:8f:2b:78:03:e8:87:81:de:44:92:c2:11 (ED25519)
80/tcp   open   http       Apache httpd 2.4.57 ((Red Hat Enterprise Linux))
| http-methods:
|   Supported Methods: POST OPTIONS HEAD GET TRACE
|_  Potentially risky methods: TRACE
|_http-title: Test Page for the HTTP Server on Red Hat Enterprise Linux
|_http-server-header: Apache/2.4.57 (Red Hat Enterprise Linux)
9090/tcp closed zeus-admin

NSE: Script Post-scanning.
Initiating NSE at 22:53
Completed NSE at 22:53, 0.00s elapsed
Initiating NSE at 22:53
Completed NSE at 22:53, 0.00s elapsed
Initiating NSE at 22:53
Completed NSE at 22:53, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 282.51 seconds
mstonge@bm02:~$
```

Figure 5.2 – Nmap portscan and results

In that example scan, we can see that SSH and HTTP are both accessible to external hosts.

First, let's see what our options really are (*hint – there are MANY!!*).

```
$ nmap --help
```

We will get the following output:

```
Nmap 7.95 ( https://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target specification}
TARGET SPECIFICATION:
  Can pass hostnames, IP addresses, networks, etc.
  Ex: scanme.nmap.org, microsoft.com/24, 192.168.0.1; 10.0.0-255.1-254
  -iL <inputfilename>: Input from list of hosts/networks
  -iR <num hosts>: Choose random targets
  --exclude <host1[,host2][,host3],...>: Exclude hosts/networks
  --excludefile <exclude_file>: Exclude list from file
HOST DISCOVERY:
  -sL: List Scan - simply list targets to scan
  -sn: Ping Scan - disable port scan
  -Pn: Treat all hosts as online -- skip host discovery
  -PS/PA/PU/PY[portlist]: TCP SYN, TCP ACK, UDP or SCTP discovery to given ports
  -PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
  -PO[protocol list]: IP Protocol Ping
  -n/-R: Never do DNS resolution/Always resolve [default: sometimes]
  --dns-servers <serv1[,serv2],...>: Specify custom DNS servers
  --system-dns: Use OS's DNS resolver
  --traceroute: Trace hop path to each host
SCAN TECHNIQUES:
  -sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans
  -sU: UDP Scan
  -sN/sF/sX: TCP Null, FIN, and Xmas scans
  --scanflags <flags>: Customize TCP scan flags
  -sI <zombie host[:probeport]>: Idle scan
  -sY/sZ: SCTP INIT/COOKIE-ECHO scans
  -sO: IP protocol scan
  -b <FTP relay host>: FTP bounce scan
PORT SPECIFICATION AND SCAN ORDER:
  -p <port ranges>: Only scan specified ports
    Ex: -p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080,S:9
  --exclude-ports <port ranges>: Exclude the specified ports from scanning
  -F: Fast mode - Scan fewer ports than the default scan
  -r: Scan ports sequentially - don't randomize
  --top-ports <number>: Scan <number> most common ports
  --port-ratio <ratio>: Scan ports more common than <ratio>
SERVICE/VERSION DETECTION:
  -sV: Probe open ports to determine service/version info
  --version-intensity <level>: Set from 0 (light) to 9 (try all probes)
  --version-light: Limit to most likely probes (intensity 2)
  --version-all: Try every single probe (intensity 9)
  --version-trace: Show detailed version scan activity (for debugging)
SCRIPT SCAN:
 -sC: equivalent to --script=default
  --script=<Lua scripts>: <Lua scripts> is a comma separated list of
           directories, script-files or script-categories
  --script-args=<n1=v1,[n2=v2,...]>: provide arguments to scripts
  --script-args-file=filename: provide NSE script args in a file
  --script-trace: Show all data sent and received
  --script-updatedb: Update the script database.
  --script-help=<Lua scripts>: Show help about scripts.
           <Lua scripts> is a comma-separated list of script-files or
           script-categories.
OS DETECTION:
  -O: Enable OS detection
  --osscan-limit: Limit OS detection to promising targets
  --osscan-guess: Guess OS more aggressively
TIMING AND PERFORMANCE:
  Options which take <time> are in seconds, or append 'ms' (milliseconds),
  's' (seconds), 'm' (minutes), or 'h' (hours) to the value (e.g. 30m).
  -T<0-5>: Set timing template (higher is faster)
  --min-hostgroup/max-hostgroup <size>: Parallel host scan group sizes
  --min-parallelism/max-parallelism <numprobes>: Probe parallelization
  --min-rtt-timeout/max-rtt-timeout/initial-rtt-timeout <time>: Specifies
      probe round trip time.
  --max-retries <tries>: Caps number of port scan probe retransmissions.
  --host-timeout <time>: Give up on target after this long
  --scan-delay/--max-scan-delay <time>: Adjust delay between probes
  --min-rate <number>: Send packets no slower than <number> per second
  --max-rate <number>: Send packets no faster than <number> per second
FIREWALL/IDS EVASION AND SPOOFING:
  -f; --mtu <val>: fragment packets (optionally w/given MTU)
  -D <decoy1,decoy2[,ME],...>: Cloak a scan with decoys
  -S <IP_Address>: Spoof source address
  -e <iface>: Use specified interface
  -g/--source-port <portnum>: Use given port number
  --proxies <url1,[url2],...>: Relay connections through HTTP/SOCKS4 proxies
  --data <hex string>: Append a custom payload to sent packets
  --data-string <string>: Append a custom ASCII string to sent packets
  --data-length <num>: Append random data to sent packets
  --ip-options <options>: Send packets with specified ip options
  --ttl <val>: Set IP time-to-live field
  --spoof-mac <mac address/prefix/vendor name>: Spoof your MAC address
  --badsum: Send packets with a bogus TCP/UDP/SCTP checksum
OUTPUT:
  -oN/-oX/-oS/-oG <file>: Output scan in normal, XML, s|<rIpt kIddi3,
     and Grepable format, respectively, to the given filename.
  -oA <basename>: Output in the three major formats at once
  -v: Increase verbosity level (use -vv or more for greater effect)
  -d: Increase debugging level (use -dd or more for greater effect)
  --reason: Display the reason a port is in a particular state
  --open: Only show open (or possibly open) ports
  --packet-trace: Show all packets sent and received
  --iflist: Print host interfaces and routes (for debugging)
  --append-output: Append to rather than clobber specified output files
  --resume <filename>: Resume an aborted scan
  --noninteractive: Disable runtime interactions via keyboard
  --stylesheet <path/URL>: XSL stylesheet to transform XML output to HTML
  --webxml: Reference stylesheet from Nmap.Org for more portable XML
  --no-stylesheet: Prevent associating of XSL stylesheet w/XML output
MISC:
  -6: Enable IPv6 scanning
  -A: Enable OS detection, version detection, script scanning, and traceroute
  --datadir <dirname>: Specify custom Nmap data file location
  --send-eth/--send-ip: Send using raw ethernet frames or IP packets
  --privileged: Assume that the user is fully privileged
  --unprivileged: Assume the user lacks raw socket privileges
  -V: Print version number
  -h: Print this help summary page.
EXAMPLES:
  nmap -v -A scanme.nmap.org
  nmap -v -sn 192.168.0.0/16 10.0.0.0/8
  nmap -v -iR 10000 -Pn -p 80
SEE THE MAN PAGE (https://nmap.org/book/man.html) FOR MORE OPTIONS AND EXAMPLES
```

Now let's run a simple scan of a target host. Please substitute the name of your target host where I have defined my target (`ks01`); otherwise, this will not work for you. This first example is the *fast* port scan.

```
$ nmap -F ks01
Starting Nmap 7.95 ( https://nmap.org ) at 2024-04-30 21:41 EDT
Nmap scan report for ks01 (10.101.0.40)
Host is up (3.4s latency).
rDNS record for 10.101.0.40: ks01.local
Not shown: 68 filtered tcp ports (no-response), 30 filtered tcp ports (host-unreach)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
Nmap done: 1 IP address (1 host up) scanned in 28.40 seconds
$
```

Now let's do a more complex scan of our target host. Again, please substitute the name of your target host where I have listed my target (`ks01`). In this scan, we'll be looking for detailed versioning information of the services running (when detected).

```
$ nmap -sV ks01
Starting Nmap 7.95 ( https://nmap.org ) at 2024-04-30 21:26 EDT
Nmap scan report for ks01 (10.101.0.40)
Host is up (0.021s latency).
rDNS record for 10.101.0.40: ks01.local
Not shown: 980 filtered tcp ports (no-response), 17 filtered tcp ports (host-unreach)
PORT     STATE SERVICE  VERSION
22/tcp   open  ssh      OpenSSH 8.7 (protocol 2.0)
80/tcp   open  http     Apache httpd 2.4.57 ((Red Hat Enterprise Linux))
9090/tcp open  ssl/http Cockpit web service 282 or later
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 95.55 seconds
```

Next, I will demonstrate that this Nmap tool can not just scan individual hosts but entire subnets at once. This operation can take significantly longer and the total time to execute can vary depending on how many targets are in the network.

> ***WARNING***
>
> *Don't try this at work without prior coordination with those teams who run the network and systems – you don't want to set off any alarms (or get yourself into trouble).*

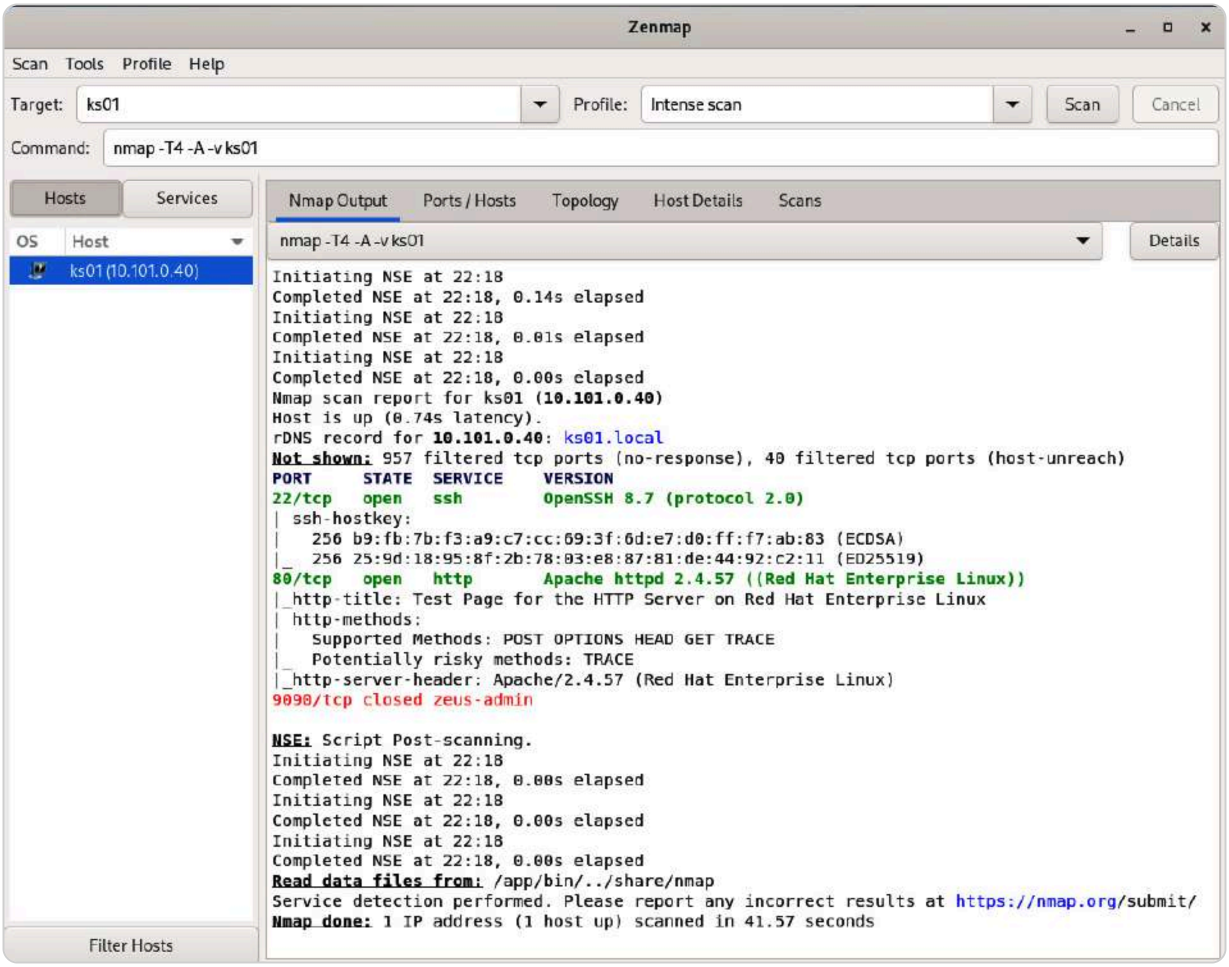
```
$ nmap 10.82.0.0/25
Starting Nmap 7.95 ( https://nmap.org ) at 2024-04-30 21:18 EDT
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
RTTVAR has grown to over 2.3 seconds, decreasing to 2.0
Nmap scan report for _gateway (10.82.0.1)
Host is up (0.0013s latency).
Not shown: 996 closed tcp ports (conn-refused)
```

```
PORT     STATE SERVICE
53/tcp   open  domain
80/tcp   open  http
443/tcp  open  https
1900/tcp open  upnp
Nmap scan report for 10.82.0.41
Host is up (0.93s latency).
Not shown: 856 filtered tcp ports (no-response), 141 filtered tcp ports (host-unreach)
PORT     STATE  SERVICE
22/tcp   open   ssh
80/tcp   open   http
9090/tcp closed zeus-admin
Nmap scan report for 10.82.0.42
Host is up (3.7s latency).
Not shown: 847 filtered tcp ports (no-response), 151 filtered tcp ports (host-unreach)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
Nmap scan report for bm03.local (10.82.0.52)
Host is up (0.00060s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT     STATE SERVICE
22/tcp   open  ssh
9090/tcp open  zeus-admin
Nmap done: 128 IP addresses (4 hosts up) scanned in 151.69 seconds
```

Nmap is capable of running many types of scans and can also be somewhat intrusive if you are not careful. I recommend examining the documentation and perhaps grabbing one of the many cheat sheets on Nmap that are freely available online.

Now that we've done a brief introduction to port scanning at the command line, let's move on to using the slightly more feature-rich GUI tool: Zenmap.

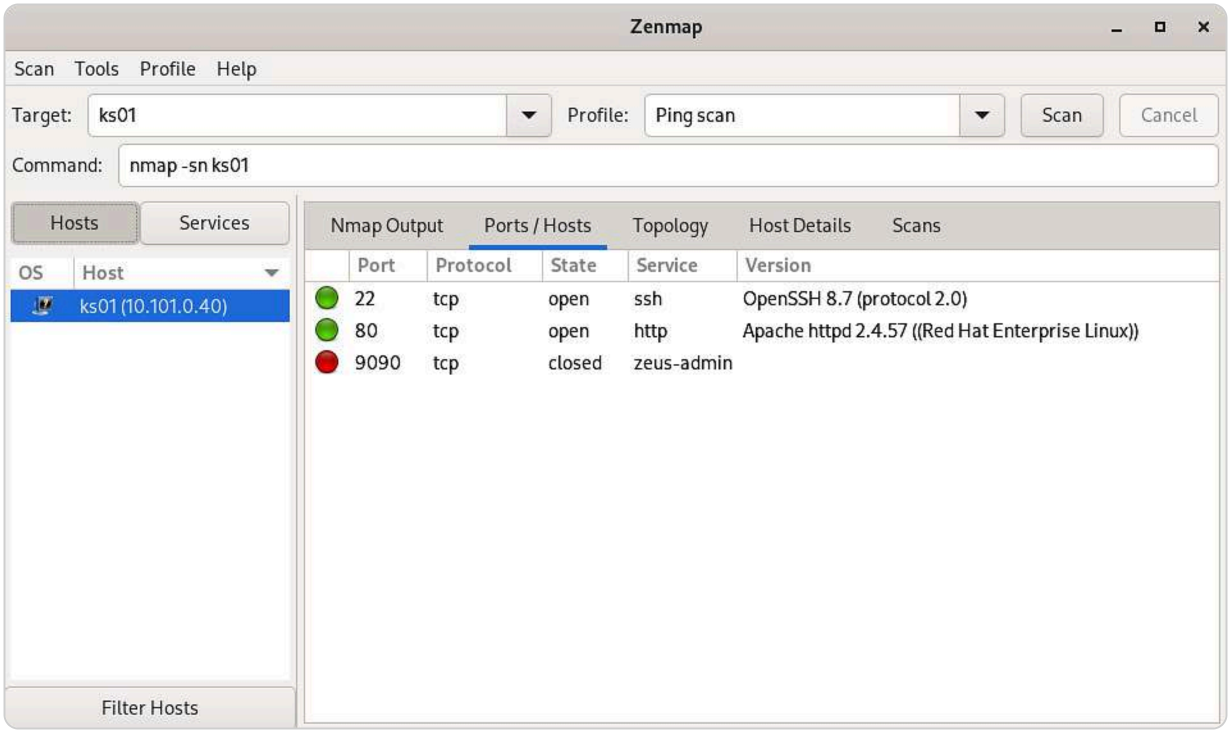
Here's an example Zenmap scan that I ran, also targeting the same host as in the previous exercise and demonstration. Please note that the output is formatted much better than the command-line tool Nmap and there are options made available to the user for expanding upon details.



```
Initiating NSE at 22:18
Completed NSE at 22:18, 0.14s elapsed
Initiating NSE at 22:18
Completed NSE at 22:18, 0.01s elapsed
Initiating NSE at 22:18
Completed NSE at 22:18, 0.00s elapsed
Nmap scan report for ks01 (10.101.0.40)
Host is up (0.74s latency).
rDNS record for 10.101.0.40: ks01.local
Not shown: 957 filtered tcp ports (no-response), 40 filtered tcp ports (host-unreach)
PORT     STATE  SERVICE    VERSION
22/tcp   open   ssh        OpenSSH 8.7 (protocol 2.0)
| ssh-hostkey:
|   256 b9:fb:7b:f3:a9:c7:cc:69:3f:6d:e7:d0:ff:f7:ab:83 (ECDSA)
|_  256 25:9d:18:95:8f:2b:78:03:e8:87:81:de:44:92:c2:11 (ED25519)
80/tcp   open   http       Apache httpd 2.4.57 ((Red Hat Enterprise Linux))
|_http-title: Test Page for the HTTP Server on Red Hat Enterprise Linux
| http-methods:
|   Supported Methods: POST OPTIONS HEAD GET TRACE
|_  Potentially risky methods: TRACE
|_http-server-header: Apache/2.4.57 (Red Hat Enterprise Linux)
9090/tcp closed zeus-admin

NSE: Script Post-scanning.
Initiating NSE at 22:18
Completed NSE at 22:18, 0.00s elapsed
Initiating NSE at 22:18
Completed NSE at 22:18, 0.00s elapsed
Initiating NSE at 22:18
Completed NSE at 22:18, 0.00s elapsed
Read data files from: /app/bin/../share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 41.57 seconds
```

Figure 5.3 – Zenmap scan

Here's another example scan that I have run. This time, once I have the results, I have clicked on the **Ports / Hosts** tab in the interface.



	Port	Protocol	State	Service	Version
🟢	22	tcp	open	ssh	OpenSSH 8.7 (protocol 2.0)
🟢	80	tcp	open	http	Apache httpd 2.4.57 ((Red Hat Enterprise Linux))
🔴	9090	tcp	closed	zeus-admin	

Figure 5.4 – Another Zenmap scan; focus on the Ports / Hosts tab

I cannot overemphasize the importance of ensuring that you scan release candidates for vulnerabilities, to validate open ports/services. We will touch on different types of scans in later chapters.

Now that we have reviewed open source scanning, let's move on to how one can track changes in the appliances.

Manifest and configuration tracking

The creation of a software manifest is a crucial step in knowing how to maintain the security of your systems. This ultimately is a detailed list of software packages and their respective versions. It is the sum of all packages in your operating system and your application stack, along with any additional third-party packages or combined dependencies.

Knowing what goes into each and every release and where those packages come from, and curating a secure anthology of said packages will empower you to control the lifecycle of your solution.

I am not saying this will be easy. It can be, but as solutions become more and more complex, this truly becomes a labored effort – especially if not everything your solutions have consumed is provided in the same format.

Let me go into detail here. What if your operating system provider gives you their packages as RPMs but you have some dependencies that are downloaded as **tarballs**, or as flatpaks? And then what if we also run some containerized microservices on top? Tracking everything can get complex really fast. In fact, it almost becomes impossible to track at all. This is why change tracking and the SBOM matter. So, let's take a look at how we can track changes in our own appliances.

Exercise – tracking changes in your product

In this exercise, we'll dive deep into how to correlate what's in an existing build versus what changes (aka deltas) are present in the next round of updates for your appliance. Knowing which packages are new (those deltas again) will blatantly highlight what your team will need to provide to existing users as an update bundle either in a custom repository or via other methods.

> ### *REQUIREMENTS FOR THE EXERCISE*
>
> *For this hands-on exercise, you will only need access to a Linux machine (physical or virtual). Root (or* **`sudo`***) access is mandatory. I am doing this example on my Fedora 40 box, but you could potentially follow along with me on any RHEL-like distribution.*

First, we'll create a script, using your favorite editor. Create a file called **`my-inventory-creator.sh`**. Ensure that you set the file permissions so that it's executable.

```
#!/usr/bin/bash
#############################
#
# my-inventory-creator.sh
#
#############################
MYDATE=$(date +"%Y%m%d%H%M")
MYFILE1="my-software-sources_$MYDATE.txt"
MYFILE2="my-software-details_$MYDATE.txt"
# Notify user of the output filenames for this run
echo "Checking for RPMs and Flatpak sources"
echo "and what's installed..."
echo
echo "Your output files for this check are: "
echo "$MYFILE1"
echo "$MYFILE2"
# Determine software sources and output to file
dnf repolist | sort > "$MYFILE1"
flatpak remotes | sort >> "$MYFILE1"
# Determine the software & versions installed and output to file
rpm -qa | sort > "$MYFILE2"
flatpak list | sort >> "$MYFILE2"
echo "Completed."
```

Now let's run the script and take note of the output file names (which include date stamps). These will be unique for you.

```
$ bash ./my-inventory-creator.sh
```

We will get the following output:

```
Checking for RPMs and Flatpak sources
and what's installed...
Your output files for this check are:
my-software-sources_202405011944.txt
my-software-details_202405011944.txt
Completed.
```

Now let's review those output files. These will also be unique for your build/system. We will start with the software sources file.

```
$ cat my-software-sources_202405011944.txt
fedora-cisco-openh264          Fedora 40 openh264 (From Cisco) - x86_64
fedora                         Fedora 40 - x86_64
google-chrome                  google-chrome
repo id                        repo name
updates                        Fedora 40 - x86_64 - Updates
fedora  system,oci
flathub system
```

Now let's see the software details file (this will be long). The output of this command is huge so I will warn you upfront – I had to truncate it for space reasons.

```
$  cat my-software-details_202405011944.txt
aajohan-comfortaa-fonts-3.105-0.3.20210729git2a87ac6.fc40.noarch
aardvark-dns-1.10.0-1.fc40.x86_64
abattis-cantarell-fonts-0.301-12.fc40.noarch
abattis-cantarell-vf-fonts-0.301-12.fc40.noarch
abrt-2.17.5-1.fc40.x86_64
abrt-addon-ccpp-2.17.5-1.fc40.x86_64
abrt-addon-kerneloops-2.17.5-1.fc40.x86_64
abrt-addon-pstoreoops-2.17.5-1.fc40.x86_64
abrt-addon-vmcore-2.17.5-1.fc40.x86_64
abrt-addon-xorg-2.17.5-1.fc40.x86_64
abrt-cli-2.17.5-1.fc40.x86_64
(((output truncated)))
Red Hat, Inc.   io.podman_desktop.PodmanDesktop 1.9.1   stable  flathub system
Zenmap  org.nmap.Zenmap 7.94    stable  flathub system
$
```

Now we have a complete manifest of software and sources for our build. Let's do a quick search for a specific package. For my example, I am using `httpd`, but you can practice searching for whichever package/version you prefer.

```
$ grep httpd my-software-details_202405011944.txt
fedora-logos-httpd-38.1.0-5.fc40.noarch
httpd-2.4.59-2.fc40.x86_64
httpd-core-2.4.59-2.fc40.x86_64
httpd-filesystem-2.4.59-2.fc40.noarch
httpd-tools-2.4.59-2.fc40.x86_64
[mstonge@bm03 ~]$
```

Now let's install a package that will drag in some dependencies along with it. I'm going to install a package called `kdiff3`, which belongs to the vast family of "diff" tools that find differences in files or from command-line outputs. Again, here, the output from this command is pages and pages long, hence I have truncated it again.

```
$ sudo dnf install -y kdiff3
Last metadata expiration check: 0:57:55 ago on Wed 01 May 2024 07:06:13 PM EDT.
Dependencies resolved.
==========================================================================================
===============
 Package                         Architecture
Version                               Repository     Size
==========================================================================================
===============
Installing:
 kdiff3                          x86_64       1.10.7-
3.fc40                         fedora        1.9 M
Installing dependencies:
 SDL_image                       x86_64       1.2.12-
37.fc40                        fedora        45 k
 aspell                          x86_64       12:0.60.8-
14.fc40                     fedora        713 k
 breeze-icon-theme               noarch       6.1.0-
1.fc40                          updates       9.7 M
 daala-libs                      x86_64       0-
27.20200724git694d4ce.fc40          fedora        211 k
 dbusmenu-qt5                    x86_64       0.9.3-
0.34.20160218.fc40              fedora         79 k
 docbook-dtds                    noarch       1.0-
85.fc40                          fedora        335 k
 docbook-style-xsl               noarch       1.79.2-
22.fc40                        fedora        1.5 M
 hspell                          x86_64       1.4-
21.fc40                          fedora        684 k
 kde-settings                    noarch       40.0-
1.fc40                           fedora         40 k
 kdsoap                          x86_64       2.2.0-
4.fc40                          fedora        133 k
 kf5-kactivities                 x86_64       5.115.0-
1.fc40                        fedora        137 k
 kf5-kactivities-stats           x86_64       5.115.0-
1.fc40                        fedora        111 k
((( output truncated )))
Installed:
  SDL_image-1.2.12-37.fc40.x86_64                        aspell-12:0.60.8-14.fc40.x86_64
  aspell-en-50:2020.12.07-10.fc40.x86_64                 breeze-icon-theme-6.1.0-
1.fc40.noarch
  daala-libs-0-27.20200724git694d4ce.fc40.x86_64         dbusmenu-qt5-0.9.3-
0.34.20160218.fc40.x86_64
  docbook-dtds-1.0-85.fc40.noarch                        docbook-style-xsl-1.79.2-
22.fc40.noarch
  hspell-1.4-21.fc40.x86_64                              kde-settings-40.0-1.fc40.noarch
  kdiff3-1.10.7-3.fc40.x86_64                            kdsoap-2.2.0-4.fc40.x86_64
  kf5-kactivities-5.115.0-1.fc40.x86_64                  kf5-kactivities-stats-5.115.0-
1.fc40.x86_64
  kf5-karchive-5.115.0-1.fc40.x86_64                     kf5-kauth-5.115.0-1.fc40.x86_64
  kf5-kbookmarks-5.115.0-1.fc40.x86_64                   kf5-kcodecs-5.115.0-
1.fc40.x86_64
(((output truncated)))
  speech-dispatcher-libs-0.11.5-5.fc40.x86_64            speexdsp-1.2.1-6.fc40.x86_64
  vlc-libs-1:3.0.20-12.fc40.x86_64                       vlc-plugin-pipewire-3-
2.fc40.x86_64
  vlc-plugins-base-1:3.0.20-12.fc40.x86_64               voikko-fi-2.5-6.fc40.noarch
Complete!
```

In the last step, we installed a single package (but it brought along 113 dependencies). So, let's now rerun our inventory script and do some introspection on what really changed.

> ***TIP***
>
> *Take note of the new files that were created in this run.*

```
$ bash ./my-inventory-creator.sh
Checking for RPMs and Flatpak sources
and what's installed...
```

Your output files for this check are:

```
my-software-sources_202405012008.txt
my-software-details_202405012008.txt
Completed.
$
```

Now let's generate a list of differences between the two file sets.

```
$ diff my-software-sources_202405011944.txt my-software-sources_202405012008.txt
```

> ***IMPORTANT NOTE***
>
> *There was no change in the sources we drew our files from but there will be significant changes in the software details lists. Let's do a side-by-side comparison. Please note the output will be big.*

```
$ diff -y my-software-details_202405011944.txt my-software-details_202405012008.txt  |
less
aajohan-comfortaa-fonts-3.105-0.3.20210729git2a87ac6.fc40.noa   aajohan-comfortaa-fonts-
3.105-0.3.20210729git2a87ac6.fc40.noa
aardvark-dns-1.10.0-1.fc40.x86_64                               aardvark-dns-1.10.0-
1.fc40.x86_64
abattis-cantarell-fonts-0.301-12.fc40.noarch                    abattis-cantarell-fonts-
0.301-12.fc40.noarch
abattis-cantarell-vf-fonts-0.301-12.fc40.noarch                 abattis-cantarell-vf-
fonts-0.301-12.fc40.noarch
abrt-2.17.5-1.fc40.x86_64                                       abrt-2.17.5-1.fc40.x86_64
abrt-addon-ccpp-2.17.5-1.fc40.x86_64                            abrt-addon-ccpp-2.17.5-
1.fc40.x86_64
((( output truncated )))
brcmfmac-firmware-20240410-1.fc40.noarch                        brcmfmac-firmware-
20240410-1.fc40.noarch
                                                              > breeze-icon-theme-6.1.0-
1.fc40.noarch
brlapi-0.8.5-13.fc40.x86_64                                     brlapi-0.8.5-
13.fc40.x86_64
((( output truncated )))
```

You'll now be able to see what was added (or changed) with this method. In the above example, there are 114 changes (actually, additions). I only truncated the output to save space and not bore you with the details.

This is where I prefer to leverage some GUI-based tools like the one I just installed. Let's take a quick look at `kdiff3`.

We'll select the same files from the previous steps.

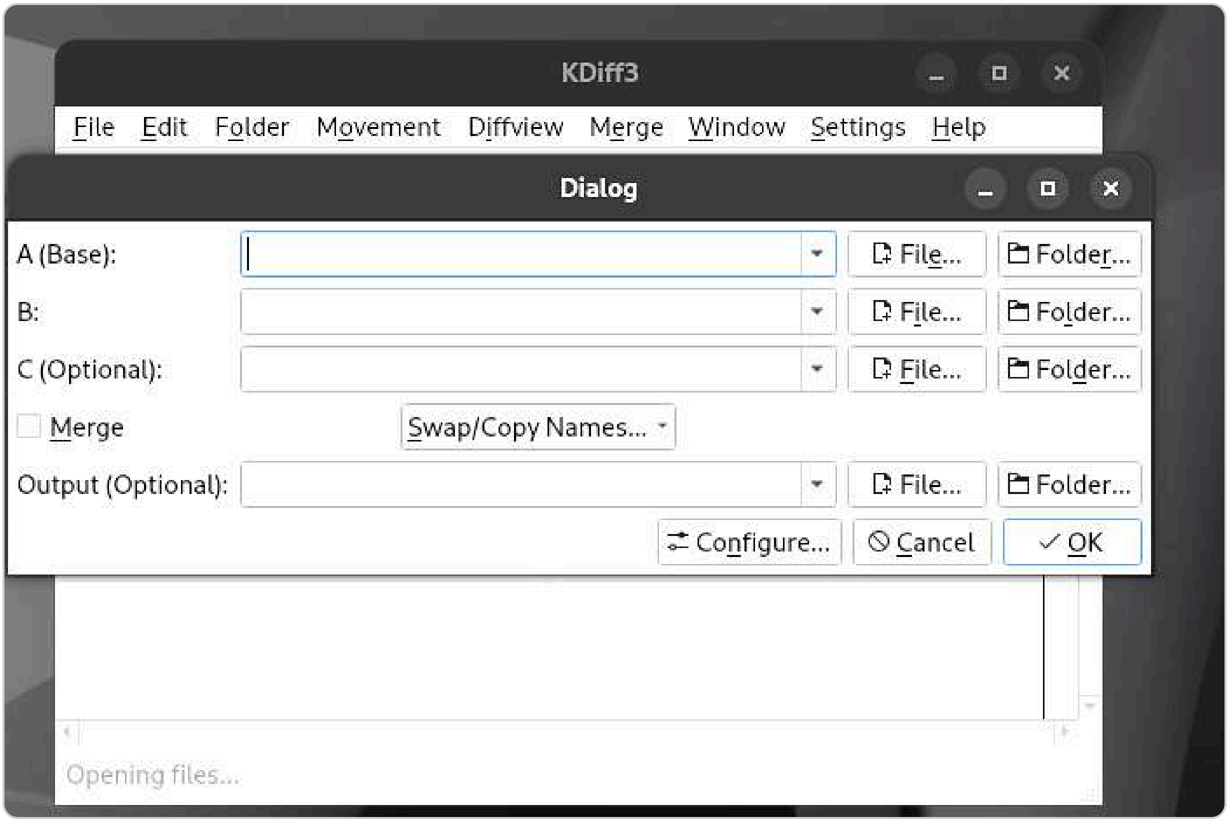
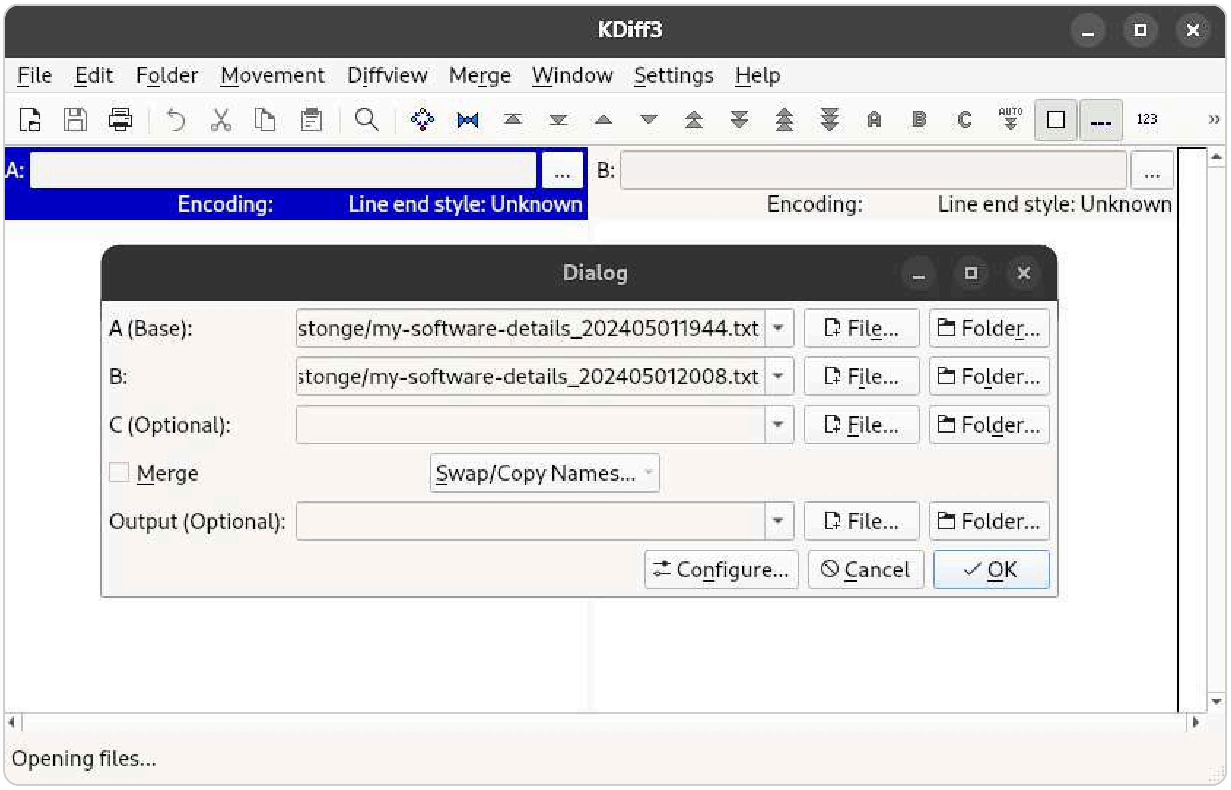


Figure 5.5 – Starting the kdiff3 comparison operation

Next, select the files in the interface. **A (Base)** is the older software details file. Finally, select the file for **B** as the latest one.

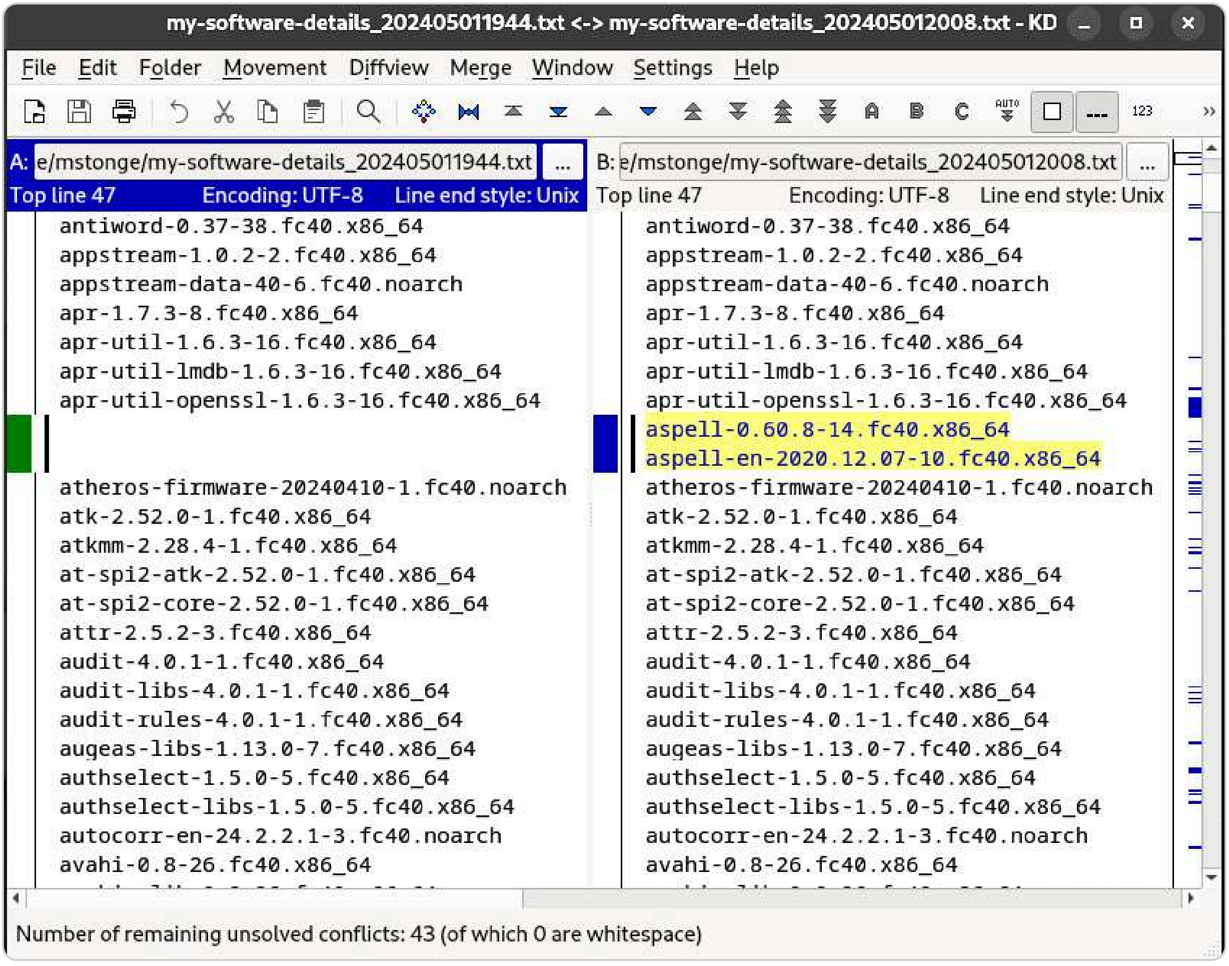


A (Base): stonge/my-software-details_202405011944.txt
B: stonge/my-software-details_202405012008.txt
C (Optional):
Merge
Output (Optional):

Figure 5.6 – Selecting the files

Now that we've defined all the files to compare, we click **OK**.

The output of this will highlight the differences graphically in the two lists.


my-software-details_202405011944.txt <-> my-software-details_202405012008.txt - KD
File Edit Folder Movement Diffview Merge Window Settings Help
A: e/mstonge/my-software-details_202405011944.txt
B: e/mstonge/my-software-details_202405012008.txt
Top line 47 Encoding: UTF-8 Line end style: Unix

```
antiword-0.37-38.fc40.x86_64                      antiword-0.37-38.fc40.x86_64
appstream-1.0.2-2.fc40.x86_64                     appstream-1.0.2-2.fc40.x86_64
appstream-data-40-6.fc40.noarch                   appstream-data-40-6.fc40.noarch
apr-1.7.3-8.fc40.x86_64                           apr-1.7.3-8.fc40.x86_64
apr-util-1.6.3-16.fc40.x86_64                     apr-util-1.6.3-16.fc40.x86_64
apr-util-lmdb-1.6.3-16.fc40.x86_64                apr-util-lmdb-1.6.3-16.fc40.x86_64
apr-util-openssl-1.6.3-16.fc40.x86_64             apr-util-openssl-1.6.3-16.fc40.x86_64
                                                  aspell-0.60.8-14.fc40.x86_64
                                                  aspell-en-2020.12.07-10.fc40.x86_64
atheros-firmware-20240410-1.fc40.noarch           atheros-firmware-20240410-1.fc40.noarch
atk-2.52.0-1.fc40.x86_64                          atk-2.52.0-1.fc40.x86_64
atkmm-2.28.4-1.fc40.x86_64                        atkmm-2.28.4-1.fc40.x86_64
at-spi2-atk-2.52.0-1.fc40.x86_64                  at-spi2-atk-2.52.0-1.fc40.x86_64
at-spi2-core-2.52.0-1.fc40.x86_64                 at-spi2-core-2.52.0-1.fc40.x86_64
attr-2.5.2-3.fc40.x86_64                          attr-2.5.2-3.fc40.x86_64
audit-4.0.1-1.fc40.x86_64                         audit-4.0.1-1.fc40.x86_64
audit-libs-4.0.1-1.fc40.x86_64                    audit-libs-4.0.1-1.fc40.x86_64
audit-rules-4.0.1-1.fc40.x86_64                   audit-rules-4.0.1-1.fc40.x86_64
augeas-libs-1.13.0-7.fc40.x86_64                  augeas-libs-1.13.0-7.fc40.x86_64
authselect-1.5.0-5.fc40.x86_64                    authselect-1.5.0-5.fc40.x86_64
authselect-libs-1.5.0-5.fc40.x86_64               authselect-libs-1.5.0-5.fc40.x86_64
autocorr-en-24.2.2.1-3.fc40.noarch                autocorr-en-24.2.2.1-3.fc40.noarch
avahi-0.8-26.fc40.x86_64                          avahi-0.8-26.fc40.x86_64
```

Number of remaining unsolved conflicts: 43 (of which 0 are whitespace)

Figure 5.7 – Running the comparison

The differences are crucial knowledge for you and your team. With this knowledge, you know which files you'll need to curate proper update releases for your customers (assuming you run this type of comparison as part of your release candidate review process). This is the perfect segue to our next section on how to provide update control mechanisms within your product.

Update control mechanisms

One of my favorite discussions over the years with my embedded partners has been how to create a proper update methodology along with the processes and the infrastructure to support it. Ultimately, we already know – based on the customers' operating environment, use cases, and compliance footprint – whether these mechanisms will be simple and online, or offline and complex.

Your choices here will directly impact the User Interface (UI) of your product. Packaging your software and configurations will simplify lifecycle management and support costs while improving end-user experiences.

For this next set of exercises, we will assume that your team has chosen to package up your special sauce along with using RPM packages and custom repositories in both your build and support chains. This is the easiest route but not always the most feasible.

We will cover other update control mechanisms later (and online in the book's GitHub repository). They can be (but are not limited to) encrypted tarballs, ISO images, thumb drives, FTP bundles, and many others. Each of those will require additional work in your build chain and vastly more attention in the support chain, not to mention more support staff who are better trained.

Now that we have reviewed how to track and document changes in your appliance, let's move on to another key facet of maintaining control of your SBOM – this time, via custom packaging.

Exercise – building custom software packages

I hope you are ready for some typing. We're going to start building here. As there are a million things that you can do within your own custom packages, I will only cover something that's near and dear to some embedded systems best practices, and that's creating your own release definition file.

The following link that I am sharing is somewhat the *go-to* reference for all things RPM-based. I recommend spending some serious time there:

https://rpm-packaging-guide.github.io/

REQUIREMENTS FOR THE EXERCISE

You'll need access to a Linux server. Root or **`sudo`** *access is mandatory. A little bit about my demo environment – I am running Fedora 40 Workstation and I have created a user called* **`mattbuild`** *(you can create your own build environment user before starting this exercise) and I've added it to the* **`sudoers`** *file. Now let's set up your RPM package building environment.*

First, we'll install the packaging tools (along with their numerous dependencies).

```
$ sudo dnf install -y fedora-packager fedora-review \
rpm-build rpm-devel rpmlint make python \
coreutils diffutils rpmdevtools
  ((( OUTPUT Truncated )))
Complete!
```

Next, we'll modify the user account we are using to build the packages to become a member of the *mock* system group.

```
$ sudo usermod -a -G mock mattbuild
```

Finally, we'll confirm the changes to the user account. You can either leverage the `newgrp` command or log out and log back in to reset your environment. Once you've done either of those, validate that the group changes have taken effect.

```
$ newgrp
$ id
uid=1001(mattbuild) gid=1001(mattbuild) groups=1001(mattbuild),135(mock)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[mattbuild@fedora  ~]$
```

Now let's create a proper rpm build environment for that user. Running the `rpmdev-setuptree` command will build a folder structure that is required for the build process.

```
$ rpmdev-setuptree
$ tree rpmbuild/
rpmbuild/
├── BUILD
├── RPMS
├── SOURCES
├── SPECS
└── SRPMS
6 directories, 0 files
```

Now let's create a `.spec` file for our project. I use `vi` or `vim`, but feel free to use your favorite. I'll place a copy of this file in the GitHub repository in case you do not want to type this one out.

```
$ vi myapprel.spec
```

Now, we'll edit the contents of the file. Be sure to save your file when done.

> ***IMPORTANT NOTE***
>
> *Only use spaces and not tabs anywhere in this file.*

```
(code - myapprel.spec)
Name:          myapprel
Version:       1.0
Release:       1
Summary:       Example custom app release file installer
License:       MIT License
%description
This will install a custom release text for your appliance
%prep
# we have no source code - so let's skip this
%build
cat > myappliance-release << EOF
Welcome to my secure embedded linux system
version 1.0
EOF
%install
mkdir -p %{buildroot}/etc
install -m 644 myappliance-release %{buildroot}/etc/myappliance-release
%files
/etc/
/etc/myappliance-release
%changelog
# nothing to report on the first attempt
```

Now let's try our first rpm build. The output for this will be huge (so I'm not going to bother listing it all).

```
$ rpmbuild -ba myapprel.spec
warning: source_date_epoch_from_changelog set but %changelog is missing
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.w4pHxJ
+ umask 022
+ cd /home/mattbuild/rpmbuild/BUILD
+ RPM_EC=0
++ jobs -p
+ exit 0
Executing(%build): /bin/sh -e /var/tmp/rpm-tmp.q2Jeer
+ umask 022
+ cd /home/mattbuild/rpmbuild/BUILD
+ CFLAGS='-O2 -flto=auto -ffat-lto-objects -fexceptions -g -grecord-gcc-switches -pipe -
Wall -Werror=format-security -Wp,-U_FORTIFY_SOURCE,-D_FORTIFY_SOURCE=3 -Wp,-
D_GLIBCXX_ASSERTIONS -specs=/usr/lib/rpm/redhat/redhat-hardened-cc1 -fstack-protector-
strong -specs=/usr/lib/rpm/redhat/redhat-annobin-cc1  -m64   -mtune=generic -
fasynchronous-unwind-tables -fstack-clash-protection -fcf-protection -fno-omit-frame-
pointer -mno-omit-leaf-frame-pointer '
+ export CFLAGS
(((output truncated)))
```

Now let's do a test install.

```
$ cp ./rpmbuild/RPMS/x86_64/myapprel-1.0-1.x86_64.rpm ~/
[mattbuild@fedora  ~]$    total 12
-rw-r--r--. 1 mattbuild mattbuild 6861 Apr 20 16:37 myapprel-1.0-1.x86_64.rpm
-rw-r--r--. 1 mattbuild mattbuild  572 Apr 20 16:35 myapprel.spec
drwxr-xr-x. 8 mattbuild mattbuild   89 Apr 20 16:05 rpmbuild
[mattbuild@fedora  ~]$    sudo dnf install -y ./myapprel-1.0-1.x86_64.rpm
[sudo] password for mattbuild:
Last metadata expiration check: 0:08:09 ago on Sat 20 Apr 2024 04:31:26 PM EDT.
Dependencies resolved.
=======================================================================
 Package                     Architecture            Version
           Repository                       Size
=======================================================================
Installing:
 myapprel                    x86_64                  1.0-1
           @commandline                   6.7 k
Transaction Summary
=======================================================================
Install  1 Package
Total size: 6.7 k
Installed size: 55
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
                                              1/1
  Installing       : myapprel-1.0-1.x86_64
                                              1/1
  Verifying        : myapprel-1.0-1.x86_64
                                              1/1
Installed:
  myapprel-1.0-1.x86_64
```

```
Complete!
$
```

Alright! We have our first custom package and our first custom file installer (and it works!)

Let's show it off.

```
$ cat /etc/myappliance-release
Welcome to my secure embedded linux system
version 1.0
```

Great job! Now you are on your way to securing your software supply chain by creating your own packages. I truly hope you had at least a little fun in this exercise.

Exercise – signing your custom RPM package

Now it's time for extra credit. This is not an introductory course. This is a master-level course. Yes, it's great we can now create a custom RPM package but we are focusing on securing your software supply chain in this chapter. Time to step things up. It's time to add your own GPG signature to your custom package!

First, we're going to leave the build account and switch over to root access.

```
$ su -
#
```

We're now going to ensure you have the right software installed to sign the RPMs that you build for your appliance. Install the `rpm-sign` package.

```
# dnf install rpm-sign
Last metadata expiration check: 1:37:57 ago on Sat 20 Apr 2024 04:31:26 PM EDT.
Dependencies resolved.
================================================================================
========================================
 Package                    Architecture            Version
       Repository                  Size
================================================================================
=======================================
Installing:
 rpm-sign                   x86_64                  4.19.1.1-1.fc39
      updates                     22 k
Transaction Summary
================================================================================
=======================================
Install  1 Package
Total download size: 22 k
Installed size: 22 k
Is this ok [y/N]: n
Operation aborted.
[root@bm02 ~]# dnf install -y rpm-sign
Last metadata expiration check: 1:38:20 ago on Sat 20 Apr 2024 04:31:26 PM EDT.
Dependencies resolved.
=======================================================================
 Package                    Architecture            Version
```

```
       Repository                 Size
========================================================================
Installing:
 rpm-sign                   x86_64                  4.19.1.1-1.fc39
      updates                    22 k
Transaction Summary
========================================================================
Install  1 Package
Total download size: 22 k
Installed size: 22 k
Downloading Packages:
rpm-sign-4.19.1.1-1.fc39.x86_64.rpm

       49 kB/s |  22 kB     00:00
--------------------------------------------------------------------
---------------
--------------------------------------
Total
                     28 kB/s |  22 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
                                                                  1/1
  Installing       : rpm-sign-4.19.1.1-1.fc39.x86_64
                                               1/1
  Running scriptlet: rpm-sign-4.19.1.1-1.fc39.x86_64
                                         1/1
  Verifying        : rpm-sign-4.19.1.1-1.fc39.x86_64
                                              1/1
Installed:
  rpm-sign-4.19.1.1-1.fc39.x86_64
Complete!
```

In this next set of steps, you're going to generate your own super-cool secure `gpg` key. When you do this for real in production, you must ensure that you are using official names, email addresses, and so on for your `gpg` key.

For this example, we'll use the name of John Doe with an email address of `john_doe@gmail.com` and the not-so-secure passphrase of `Embedded`. This next command is interactive and some of the dialogs actually present what I comically call screen vomit in order to keep the users' attention to the dialogs. Others may simply call this a loosely managed **Text User Interface** (**TUI**).

IMPORTANT NOTE

Use ***`gpg --full-generate-key`*** *for a full-featured key generation dialog.*

```
# gpg --gen-key
gpg (GnuPG) 2.4.4; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
GnuPG needs to construct a user ID to identify your key.
Real name: John Doe
Email address: john_doe@gmail.com
```

```
You selected this USER-ID:
    "John Doe <john_doe@gmail.com>"
Change (N)ame, (E)mail, or (O)kay/(Q)uit?
```

Next, the following dialog will appear; you'll enter the passphrase `Embedded` and select `<OK>` to continue.

```
Please enter the passphrase to protect your new key
 Passphrase: ________________________________________
       <OK>                              <Cancel>
```

You will most likely get scolded by the system for not using a super-secure passphrase, but this is training, so select `<Take this one anyway>` to continue. One just has to love these interactive text interfaces. They really do prevent you from having to learn a command line that's a mile long.

```
Warning: You have entered an insecure passphrase.
A passphrase should contain at least 1 digit or special character.
 <Take this one anyway>            <Enter new passphrase>
```

We're almost there in getting the key created. We must confirm the passphrase `Embedded` to continue.

```
 Please re-enter this passphrase
   Passphrase: ________________________________________
       <OK>                              <Cancel>
```

And let's watch it do its work. Your output should appear much like this (but not exactly as each key generation is unique).

```
We need to generate a lot of random bytes. It is a good idea to perform some other action
(type on the keyboard, move the mouse, utilize the disks) during the prime generation;
this gives the random number generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform some other action
(type on the keyboard, move the mouse, utilize the disks) during the prime generation;
this gives the random number generator a better chance to gain enough entropy.
gpg: directory '/root/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/root/.gnupg/openpgp-
revocs.d/417CAC67C4F673DEDC13C95CA01133131459C4AD.rev'
public and secret key created and signed.
pub   ed25519 2024-04-20 [SC] [expires: 2027-04-20]
      417CAC67C4F673DEDC13C95CA01133131459C4AD
uid                 Matt St. Onge <matt_st_onge@yahoo.com>
sub   cv25519 2024-04-20 [E] [expires: 2027-04-20]
[root@fedora  ~]#
```

So now let's take a look at all the keys on our system so far.

```
# gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2027-04-20
[keyboxd]
---------
pub   ed25519 2024-04-20 [SC] [expires: 2027-04-20]
      417CAC67C4F673DEDC13C95CA01133131459C4AD
uid           [ultimate] John Doe <john_doe@gmail.com>
sub   cv25519 2024-04-20 [E] [expires: 2027-04-20]
```

Your key is now imported. Let's query the `rpm` database to see if it's actually there. A simple query may not be sufficient, so we might need to add in some field parsing.

```
# rpm -q gpg-pubkey
gpg-pubkey-18b8e74c-62f2920f
gpg-pubkey-7fac5991-45f06f46
gpg-pubkey-d38b4796-570c8cd3
gpg-pubkey-1459c4ad-6624378b
#    rpm -q gpg-pubkey --qf '%{name}-%{version}-%{release} --> %{summary}\n'
gpg-pubkey-18b8e74c-62f2920f --> Fedora (39) <fedora-39-primary@fedoraproject.org> public
key
gpg-pubkey-7fac5991-45f06f46 --> Google, Inc. Linux Package Signing Key <linux-packages-
keymaster@google.com> public key
gpg-pubkey-d38b4796-570c8cd3 --> Google Inc. (Linux Packages Signing Authority) <linux-
packages-keymaster@google.com> public key
gpg-pubkey-1459c4ad-6624378b --> John Doe <john_doe@gmail.com> public key
```

Okay. Now, let's set up the signing environment for root and then verify that it is all ready to go. As root, use your favorite editor to create the `.rpmmacros` file in the root home directory.

```
# vi .rpmmacros
```

Here's what you'll need to place in the `.rpmmacros` file. Don't forget to save the file.

```
 %_signature gpg
 %_gpg_path /root/.gnupg
 %_gpg_name Matt St. Onge
 %_gpgbin /usr/bin/gpg2
```

As many have heard me say (probably far too often), *Trust but verify*. So verify!

```
# rpm --showrc | grep __gpg_sign_cmd -A 5
-13: __gpg_sign_cmd     %{shescape:%{__gpg}}
        gpg --no-verbose --no-armor --no-secmem-warning
        %{?_gpg_digest_algo:--digest-algo=%{_gpg_digest_algo}}
        %{?_gpg_sign_cmd_extra_args}
        %{?_gpg_name:-u %{shescape:%{_gpg_name}}}
        -sbo %{shescape:%{?__signature_filename}}
#
```

The magic moment that you've been waiting for. Let's sign the RPM that you just built with your shiny new `GPG` key! Remember you'll be asked for our super-secret passphrase – `Embedded`. Enter it and then select `<OK>`.

```
# cp ~mattbuild/myapprel-1.0-1,x86_65.rpm ~/
# rpm --addsign myapprel-1.0-1.x86_64.rpm
 Please enter the passphrase to unlock the OpenPGP secret key: x
  "Matt St. Onge <matt_st_onge@yahoo.com>"
   255-bit EDDSA key, ID A01133131459C4AD,
 created 2024-04-20.
  Passphrase: _________________________________________ x
      <OK>                                   <Cancel>
[root@fedora  ~]#
```

You've created your first custom file that aligns with embedded best practices by installing a release file in your appliance. Since security is paramount in the software supply chain, you've also ensured the sanctity of your package by adding your own `GPG` key signature. Pat yourself on the back.

Show off that awesome new skill set by testing the package again.

```
# rpm -K ./myapprel-1.0-1.x86_64.rpm
./myapprel-1.0-1.x86_64.rpm: digests signatures OK
#
```

Here we are. Journey complete. Let's move on to the next steps in securing our software supply chain.

Exercise – creating a custom DNF repository

There are actually two schools of thought on this. One creates separate repositories for the applications stack and the operating system, and the other simply puts it all in a monolithic custom repository. I prefer the latter. I say just put it all together as tracking gets simpler. I also believe in the creation of numerous custom RPMs in the said repository that shall assist your team in the maintenance of your solution. I can literally feel my own eyes rolling when I type that. Am I repeating myself? Probably.

What else should go in this custom repository? First, I recommend an rpm that defines your release, much like we created just a little while ago. Second, I recommend the creation of a documentation rpm to be separate from your application itself. Finally, and most importantly, I strongly recommend the creation of an rpm that installs your repository information directly into the appliance. Defining the repository in its own rpm is key. It aids in the support and lifecycle of the solution and if you need to make changes to the repository (or the GPG keys), it gives you the simplest way of updating it for the end-customer.

These custom repositories are not just for the appliances' updates. They can be greatly leveraged internally as part of your secure build chain. So, with all that said, how do we build a custom repository?

REQUIREMENTS FOR THE EXERCISE

*For this hands-on exercise, you will only need access to a Linux machine (physical or virtual). Root (or **`sudo`**) access is mandatory. I am doing this example on my Fedora 40 box, but you could potentially follow along with me on any RHEL-like distribution.*

First, we'll ensure that `httpd` and the `createrepo` utility are installed. Then, we'll also make sure that the web server is actively running.

```
# dnf install httpd createrepo -y
….output truncated
Complete!
# systemctl enable –now httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service →
```

```
/usr/lib/systemd/system/httpd.service.
# systemctl status httpd
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: active (running) since Mon 2024-04-22 21:39:01 EDT; 31s ago
       Docs: man:httpd.service(8)
   Main PID: 2712 (httpd)
     Status: "Total requests: 0; Idle/Busy workers 100/0;Requests/sec: 0; Bytes
served/sec:   0 B/sec"
      Tasks: 177 (limit: 38325)
     Memory: 19.7M
        CPU: 93ms
     CGroup: /system.slice/httpd.service
             ├─2712 /usr/sbin/httpd -DFOREGROUND
             ├─2713 /usr/sbin/httpd -DFOREGROUND
             ├─2715 /usr/sbin/httpd -DFOREGROUND
             ├─2716 /usr/sbin/httpd -DFOREGROUND
             └─2717 /usr/sbin/httpd -DFOREGROUND
Apr 22 21:39:01 bm02.local systemd[1]: Starting httpd.service - The Apache HTTP Server...
Apr 22 21:39:01 bm02.local (httpd)[2712]: httpd.service: Referenced but unset environment
variable evaluates to an>
Apr 22 21:39:01 bm02.local httpd[2712]: Server configured, listening on: port 80
Apr 22 21:39:01 bm02.local systemd[1]: Started httpd.service - The Apache HTTP Server.
lines 1-22/22 (END)
```

Next, we'll create a folder under the html directory for our repository. Then we'll copy the rpm we created in the previous exercise to the directory.

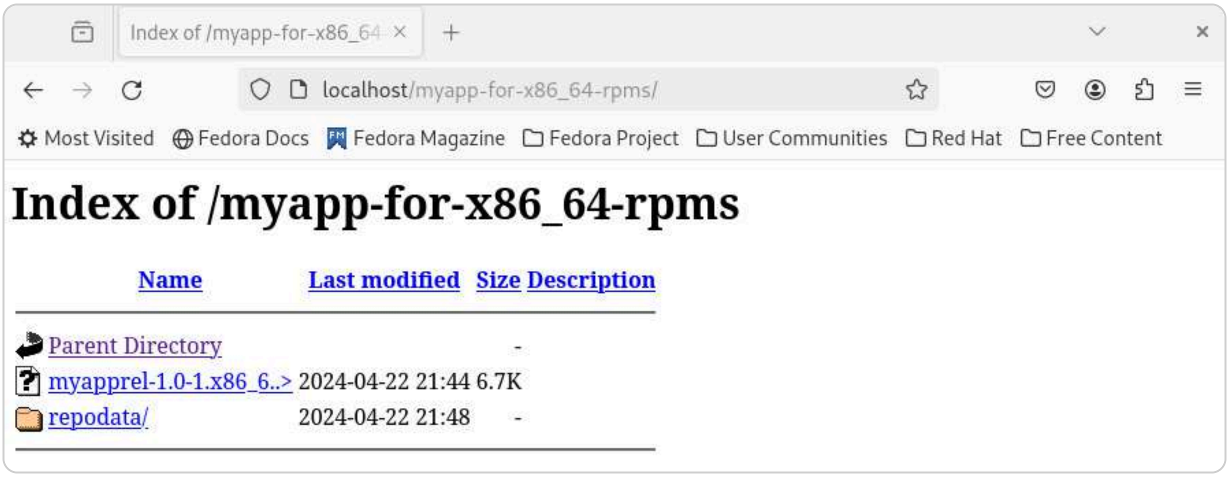
```
# cd /var/www/html
# mkdir myapp-for-x86_64-rpms
# cd myapp-for-x86_64-rpms
# cp ~/myapprel-1.0-1.x86_64.rpm .
# ls -l
total 8
-rw-r--r--. 1 root root 6861 Apr 22 21:44 myapprel-1.0-1.x86_64.rpm
[root@bm02 myapp-for-x86_64-rpms]#
```

Next, we'll initialize the repository.

```
# createrepo /var/www/html/myapp-for-x86_64-rpms/
Directory walk started
Directory walk done - 1 packages
Temporary output repo path: /var/www/html/myapp-for-x86_64-rpms/.repodata/
Pool started (with 5 workers)
Pool finished
#
```

Test to ensure that we can access the repository via a web browser locally.

Here's an example of what you'll see by testing on a local browser. It will show you an accessible set of contents in your newly created custom repository.


Index of /myapp-for-x86_64-rpms

Name	Last modified	Size	Description
Parent Directory		-	
myapprel-1.0-1.x86_6..>	2024-04-22 21:44	6.7K	
repodata/	2024-04-22 21:48	-	

Figure 5.8 – Localhost view of repository in a web browser

Other steps for this internal repository are to enable `http` on the firewall (so our new repo can be accessed by machines in your lab internally) and, if in use, set the `selinux` file context for the repositories directory and files to `httpd_syscontent_t`. As this is not a basic sysadmin book, I trust you and your team already know how to do those simple tasks.

This is generally an OK setup for a lab. This setup is beyond unacceptable for anything externally facing (i.e., internet-facing). For that use case, there are many books and resources available on how to secure your web server; however, I will make some baseline recommendations that should go without saying.

For a public-facing web server, your system should be using `https` with a signed certificate. The server should be in your company's external DNS so the customers can find it. Methods to control access, perhaps at an individual customer or user level, should be applied. Protect your system as best you can.

Now that we've created a custom package and hosted it via a custom repository, all that remains is how to configure the appliances to consume the new custom content. This next exercise will walk you through just that.

Exercise – configuring your solution to use your custom repository

Let's now configure our appliance to access our custom repository. This is a key step in ensuring that only your tested and vetted content is provided to your end-users in the future.

> ***REQUIREMENTS FOR THE EXERCISE***

For this hands-on exercise, you will need access to the same Linux machine where you previously created a sample repository along with another machine to configure for accessing the said repository. Root (or **`sudo`***) access is mandatory. I am doing this example on my Fedora 40 box, but you could potentially follow along with me on any RHEL-like distribution.*

These files are stored under **`/etc/yum.repos.d/`** and have the file extension of **`.repo`**. They should be owned by root, readable by all, but not writable by anyone else other than root. This will keep them secure.

So before you create your own example repository definition file, let's take a look at a detailed breakdown of one I created for the custom repository in the earlier exercise. Here is an example file called **`mycustomstuff.repo`**. I will place a copy of this file in the GitHub repository in case you do not wish to type this one out.

```
# ID definition of the repository
[my-custom-stuff]
# NAME of the repository
name=my-custom-stuff
# the base URL - update this for your systems information
baseurl=http://bm02.local/myapp-for-x86_64-rpms/
# Repository enabled =1 ... disabled =0
enabled=1
# setting up a gpg key for the repo is a great idea if public facing
# repo gpg check enabled =1 disabled =0
repo_gpgcheck=0
# definition for the gpg key for the repo itself (if enabled)
# example formatting:
# gpgkey=file:///(path to file)
# (or)
# gpgkey=(URL to gpgkey)
#
```

Now that you have an example, create your own **`mycustomstuff.repo`** file and place it on the Linux machine that is not hosting the repo. The file should be owned by root but be readable by all groups and all users.

Once you have completed that, test the functionality by installing the RPM package **`myapprel`** on the machine where you've set up the repository access.

```
$ sudo dnf install -y myapprel
Last metadata expiration check: 0:21:52 ago on Wed 01 May 2024 10:45:50 PM EDT.
Dependencies resolved.
=======================================================================
===================================
 Package                Architecture          Version
Repository                      Size
======================================================================
====================================
Installing:
 myapprel               x86_64                1.0-1
my-custom-stuff                 6.7 k
Transaction Summary
=====================================================================
=====================================
Install  1 Package
Total download size: 6.7 k
Installed size: 55
Downloading Packages:
myapprel-1.0-1.x86_64.rpm
         1.9 MB/s | 6.7 kB     00:00
-----------------------------------------------------------------
-----------------------------------------
Total
        670 kB/s | 6.7 kB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :
                                   1/1
  Installing       : myapprel-1.0-1.x86_64
                                   1/1
  Verifying        : myapprel-1.0-1.x86_64
                                   1/1
Installed:
  myapprel-1.0-1.x86_64
Complete!
```

Now that we've successfully attached our system to the repository and installed our custom package, we can celebrate our success. These exercises are crucial to how your team will be able to support your appliances in the future.

> **_OTHER CONCERNS TO ACCOUNT FOR_**
>
> *This should go without saying, but I will say it anyway. Have actual people test your prototypes and release candidates. Only this will let your team know if you have truly achieved success. I am not saying that you should not test with automation – actually, quite the opposite. Do both. Do all the automated tests, the penetration tests, and the scans first. Once satisfied with those results, move on to human trials. You must also test your processes and support infrastructure. Test how new releases will impact the support chain and how each new release will be consumed by your customers. Failed or broken updates can cause outages, service blockages, unusable appliances, and, worst of all, very disgruntled and dissatisfied customers.*

Summary

So, let's recap. We truly have covered a lot of material in this chapter. You should feel that you've achieved some level of success. You have come a long way in a short time. Initially, you established your appliance's bill of materials (SBOM), and then you defined your own custom packages, releases, and custom repositories. All of this culminated in putting them into action by granting your appliance access to your custom repository and installing new packages. You have secured your own software supply chain! I hope you now feel that you're empowered to track the lifecycle of your future offering, so let's move on to the next chapter where we will dive deep into the usage of encryption and protecting the data within your solution.

6
Disk Encryption

The **Linux Unified Key Setup** (**LUKS**) standard for encrypting block devices within Linux was created way back in 2004. No wonder I feel like I've been using it forever! Twenty years is a long time for a tool to get stable and feature-rich. It's also been around long enough to be universally loved and appreciated by engineers around the world. It's virtually a mainstay of every Linux distribution's installer options when configuring storage for your Linux systems. But each of those installers only lets you take LUKS so far. You will be forced to manually enter keys every time you boot or reboot. As this book is meant to be by no means introductory, let's assume you have some great baseline Linux skills and continue our journey down the mineshaft of complexities and advanced skill sets.

This chapter's goal is to open your mind to more ways to secure the storage of your system. Here, I plan to expand your insights into how that can be configured alternatively.

Grab some caffeinated beverages and a snack. This will be a deeper dive into the employment of LUKS for your future products.

In this chapter, we will cover the following topics:

- Introduction to LUKS
- Implementing LUKS on an appliance with automated keys
- Is recovery possible?

Let's get started.

Technical requirements

For the exercise in this chapter, you will need a physical or virtual machine that you can (re)install Linux onto to complete these tasks. Administrative (root) access is implied. You will be required to create some custom partitions/filesystems. You will need a fresh installation of a Linux system with the regular filesystems you may have regularly created. Also create a 500 MiB XFS LUKS encrypted partition, labeled `data3`, with the mount point set as `/data3`. Use `CreatePass` as the initial key passphrase. Root or sudo access is mandatory. I cannot highlight enough how important it is that this prerequisite is done as specified – not doing this will impact your ability to easily complete this chapter's exercises.

Introduction to LUKS

With LUKS in play, any Linux filesystem can be encrypted. There are some caveats that you should be aware of ahead of time.

Encrypting your data at rest (that is, everything stored on disk, SSD, or NVME) is not just a nice-to-have option, it's almost assumed to be present depending on whom your target clientele may be. For discussion's sake, let's imply that the expected customer for your solution is a government entity. Most government customers (regardless of the country we are referring to) are mandated to have an exceptional level of security within whatever may be deployed within their walls. Their standards are significantly higher, as are their risks. It's safe to say that disk encryption is assumed to be present. We shall cover how to implement some of these more stringent government security standards later in *Chapter 13*.

Crucial to the encryption process is an open source utility called `cryptsetup`. This relies on functionality provided by the `dm-crypt` Linux kernel module. These tools, along with your own distribution-specific tooling for managing storage, are installed generally by default in every Linux distribution. Once a volume is encrypted with LUKS, the encryption can only be fully removed when the volume is offline and not mounted.

LUKS, most simply stated, leverages **Advanced Encryption Standard** (**AES**) cryptographic algorithms and the device-mapper facility through the `dm-crypt` module to cipher/decipher data on disks. LUKS requires a passphrase to be entered each time a volume is mounted and accessed. Its inner workings, which are thankfully obscured from us end-users, are rather complex. Automating this process is even more complex.

For systems using the **Unified Extensible Firmware Interface** (**UEFI**), the first stage boot loader partition (i.e., the EFI system partition) can never be encrypted. I suppose someday soon that may change but, for now, we do not have that option. Then comes the second stage of the boot process, your system's boot loader, in the event you dare to encrypt the `/boot` partition. GRUB and GRUB2 both support such encryption. There are other bootloaders that may not do this, so check before considering this option.

Most commonly, the root partition along with any application partitions should be encrypted to protect the system from tampering. Swap and user home directory partitions should also be considered.

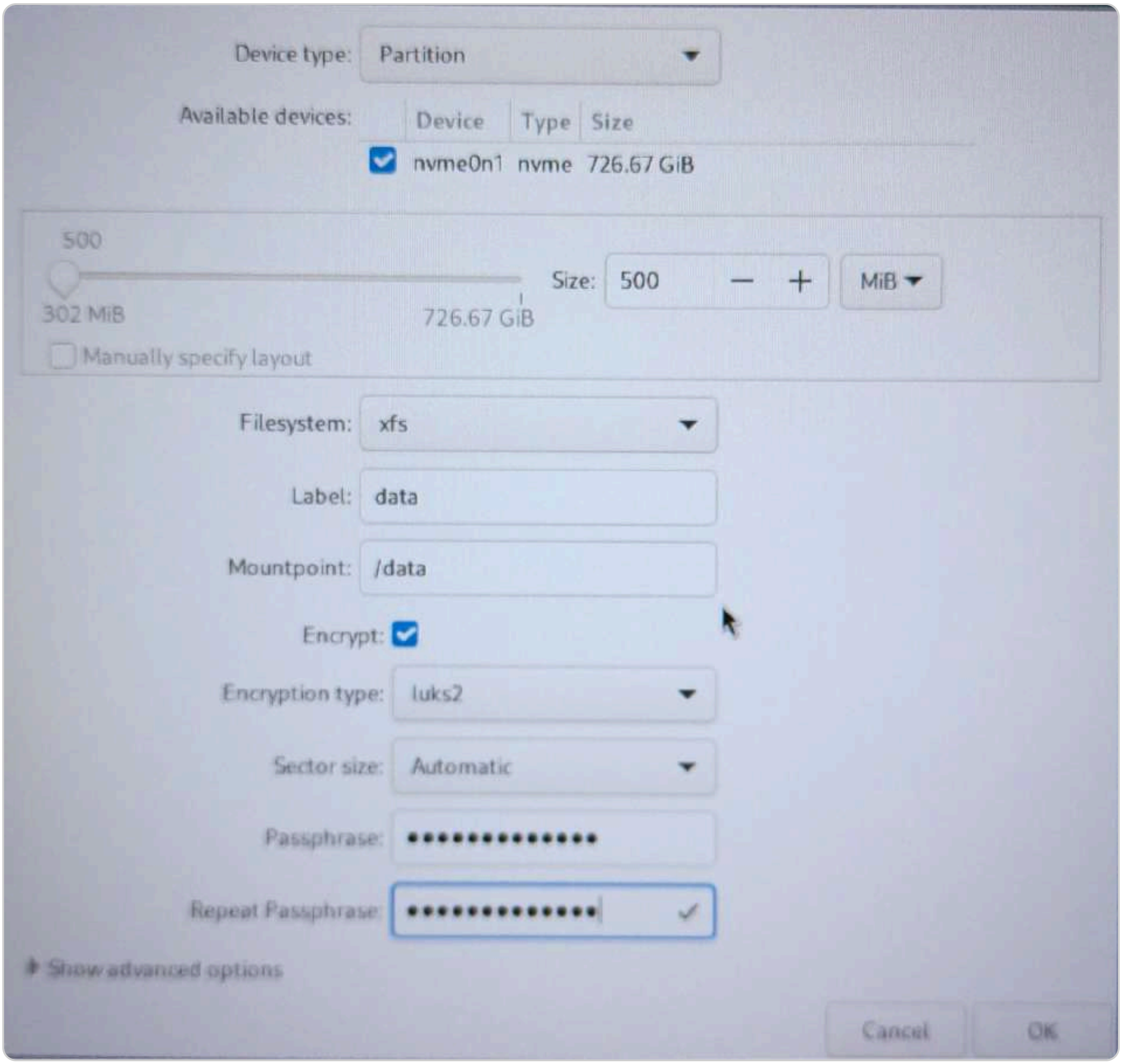
The current version of LUKS (LUKS2) supports up to 32 encryption keys per encrypted volume, whereas LUKS1 only supports up to 8 keys. This will be a crucial golden nugget to remember when we discuss recovery later on in this chapter.

Basic implementation review

With the excellent graphical installers available in today's Linux distributions, you have probably seen exactly how they try to assist you with the configuration of LUKS encryption as part of the build process. We won't be covering LUKS basics in any depth, but if you need a reference, please check the project's repository at https://gitlab.com/cryptsetup/cryptsetup/blobmC5#WdW07?dhVJ4aster/README.md. However, I do want to highlight specifically that configuring LUKS via the base installer will force the manual entry of the key passphrase every time the system is booted or rebooted.

As this is not a beginner's guide to installing Linux, I'd like to make an important point. The installer can only minimally configure LUKS encryption. This configuration may be suitable for people building things in their lab, but it is not acceptable when one takes on the customers' perspective in the situation where they're paying prime money for a well-crafted secure solution.

Here's an example of configuring LUKS via the Linux distribution's installer:


Device type: Partition

Available devices:

	Device	Type	Size
✓	nvme0n1	nvme	726.67 GiB

500

302 MiB 726.67 GiB

Size: 500 MiB

Manually specify layout

Filesystem: xfs

Label: data

Mountpoint: /data

Encrypt: ✓

Encryption type: luks2

Sector size: Automatic

Passphrase: ••••••••••••••

Repeat Passphrase: ••••••••••••••

Figure 6.1 – Configuring encryption via the installer

Creating encrypted filesystems through the installer is definitely a positive jumpstart to securing your appliance build, but if you stop there, it’s only a matter of time before the customer complaints over usability and security (as they’ll have to give out the passphrase to all the admins at minimum) will force you to make changes.

As we continue to build upon each piece of security and usability criteria in this book, let’s move on to the next section, which will definitely improve the customers’ end-user experience and, without a shadow of a doubt, will improve the security posture of the appliance itself. Automating the encryption/decryption process and securing this information from the end-user becomes a key initiative. The fewer people with access to the ability to decrypt the data, the better the solution will be perceived. Let's now explore exactly how such automation looks and let's get our hands dirty.

Implementing LUKS on an appliance with automated keys

A key point to keep at the forefront of your thought process in the implementation of any security factor in an appliance solution is your end-user experience. This book was created to help you and your team create a secure but usable embedded Linux system.

Depending on what your solution is and how it is utilized by the end-users, it is paramount to prevent the need for those end-users to have to enter keys every time a system is turned on. Firstly, that appliance may not even have a console or a keyboard attached. Secondly, forcing the end-user to manually enter such a key will ultimately result in them writing the key down on a note somewhere taped to the machine or elsewhere in clear view in the workplace. Making the end-user enter the key passphrases should be avoided if at all feasible.

Here's an example screenshot of how you may be prompted for a passphrase before the boot sequence can continue:

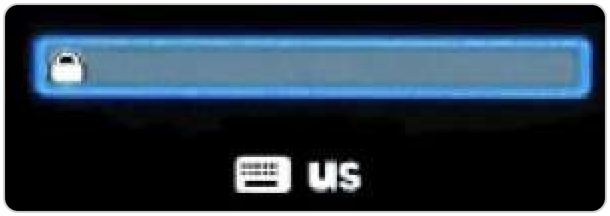


Figure 6.2 – Manual encryption key passphrase entry at boot

Now that you can see how having your end-user be forced to enter a passphrase is not a great idea, we'll take a journey in the next section to see how this process can be securely automated. Let's move on.

Exercise – implementing LUKS with stored keys and leveraging the crypttab file

Before we start pounding the keyboard, please allow me to introduce an ally that you may have known that you have in this battle. The `/etc/crypttab` file is used by all distributions of Linux to store information about encrypted block devices for them to be automatically unlocked at boot time. In my opinion, this is one of the true hidden gems in Linux.

For this exercise, we'll be using the machine mentioned in this chapter's *Technical requirements* section and we will be automating the unlocking of that `/data3` filesystem. Let's begin with the following steps:

1. Log in as root.
2. Check device availability. Identify the UUID and the device name of the partition you have created for this exercise. Your output will most likely have differences:

```
$ sudo lsblk
NAME           MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda            8:0    0 476.9G  0 disk
├─sda1         8:1    0     2M  0 part
├─sda2         8:2    0     3G  0 part  /boot
├─sda3         8:3    0    55G  0 part  /
├─sda4         8:4    0    64G  0 part  [SWAP]
├─sda5         8:5    0    40G  0 part  /var
├─sda6         8:6    0    40G  0 part  /home
└─sda78:7    0   500M  0 part
  └─luks-8e1fb810-b471-491a-adcf-32048a0eb534 253:0    0   484M  0 crypt /data3
zram0          252:0    0     8G  0 disk  [SWAP]
```

3. Now, let's determine the UUID for our specific volume. Yours may be a different device so please pay attention:

```
$ sudo blkid /dev/sda7
/dev/sda7: UUID="8e1fb810-b471-491a-adcf-32048a0eb534" TYPE="crypto_LUKS"
PARTUUID="b5906739-06d2-44d2-8770-17f2ffd75212"
```

IMPORTANT NOTE

For me, my device name was `/dev/sda7` *– yours will be different for a variety of reasons (disk type, the partitioning scheme of your system, etc.) Wherever I have typed* `/dev/sda7` *or my unique UUID for the volume, you must replace it with your own information.*

4. Generate a random passkey as root and save it under `/etc` as `luks-keyfile`:

```
# dd if=/dev/random of=/etc/luks-keyfile \
bs=1024 count=4
4+0 records in
4+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.000156324 s, 26.2 MB/s
```

5. Set the permissions properly for the new `luks-keyfile` file as root:

```
# chmod 0400 /etc/luks-keyfile
```

6. Ensure that SELinux contexts are applied to the new `luks-keyfile` file:

```
# restorecon -vvRF /etc/luks-keyfile
Relabeled /etc/luks-keyfile from unconfined_u:object_r:etc_t:s0 to
system_u:object_r:etc_t:s0
```

7. As root, add the new `luks-keyfile` file as another way of decrypting the drive we just created. You will be prompted for the original passphrase that you used in the prep for this exercise – `CreatePass`:

```
# cryptsetup luksAddKey /dev/sda7 /etc/luks-keyfile
Enter any existing passphrase:
```

8. Let's look at the **/etc/crypttab** file created by the Linux installer. We'll need to see whether the installer created a definition for the manual decryption of our volume (in my case, it did):

```
# cat /etc/crypttab
luks-8e1fb810-b471-491a-adcf-32048a0eb534 UUID=8e1fb810-b471-491a-adcf-32048a0eb534
none discard
```

9. Now, we'll unmount the volume and ensure LUKS hasn't kept the volume open. First, unmount the **/data3** volume and then run the command to ensure the LUKS has also closed the volume:

```
# umount /data3
# cryptsetup -v luksClose luks-8e1fb810-b471-491a-adcf-32048a0eb534
Command successful.
```

IMPORTANT NOTE

*For this exercise, we will also name the volume **data3**. As my **/data3** partition was created during the installation (for speed purposes), there was already an entry with the same UUID as my **/data3** volume so I created a new entry with the label of **data3** as the new first entry and commented out the original entry. This is a very important step. You may later choose to delete the older entry, but in testing, I recommend simply commenting it out. Additionally, I must remind you that your own UUIDs will be different than mine.*

10. Edit the **/etc/crypttab** file using your favorite editor:

```
$ sudo vi /etc/crypttab
```

/etc/crypttab

```
data3   UUID=8e1fb810-b471-491a-adcf-32048a0eb534 /etc/luks-keyfile luks
###luks-8e1fb810-b471-491a-adcf-32048a0eb534 UUID=8e1fb810-b471-491a-adcf-32048a0eb534
none discard
```

11. Let's manually force LUKS to use our new keyfile and test open the encryption and then mount the volume:

```
# cryptsetup -v luksOpen /dev/sda7 data3 \
--key-file=/etc/luks-keyfile
No usable token is available.
Key slot 2 unlocked.
Command successful.
```

12. Let's now mount the device:

```
# mount /data3
```

13. Verify that **/data3** is mounted. Depending on how you built your test machine, your output may vary. Please pay attention:

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        55G  7.2G   48G  13% /
devtmpfs        4.0M     0  4.0M   0% /dev
tmpfs            16G     0   16G   0% /dev/shm
tmpfs           6.3G  1.9M  6.3G   1% /run
tmpfs            16G   76K   16G   1% /tmp
/dev/sda2       3.0G  406M  2.6G  14% /boot
/dev/sda5        40G  2.6G   38G   7% /var
/dev/sda6        40G  875M   40G   3% /home
```

```
tmpfs           3.2G  224K  3.2G   1% /run/user/1000
/dev/dm-0       444M  135K  415M   1% /data3
```

14. Finally, we need to do a reboot of the system and re-verify all filesystems mounted automatically.

We've come a long way with LUKS thus far through some serious advanced automation. Your customers will truly appreciate your attention to detail and the ease of use your solution provides (along with the encryption). We are not done yet. Let's dive into the million-dollar question in our next section.

Is recovery possible?

Wow. Is recovery possible? That's the million-dollar support question. Please allow me to paint a picture and offer you what may just be the only solution. The truest answer is both *yes* and *no*. It's mostly *No!* So, let's explore what I mean and why I call it the million-dollar support question.

Please humor me for just a moment. Let's envision that the worst possible situation has arrived – due to a perfect storm of either a broken TPM module or filesystem corruption or, even worse, a malicious act by a rogue employee or hacker, the keystore for your automated encryption passkeys is lost on one of your customers' appliances.

Your support team is flustered. Not only is that customer offline but there will probably be some significant data loss involved. "*Oh man*," you think to yourself, "*Why didn't I call in sick today?*"

How can your support team help this customer? Will your company lose credibility or the end-customers' trust? How could you have prevented this?

Okay, here's the bad news. Brace for impact. If you only have one passkey and it's lost or compromised or whatever, you and that customer are definitely going to have a very, very bad day. There is no recovery from the loss of the sole key/passphrase store. Period. This type of encryption has ramifications. It's built into the design. No key or passphrase means zero access, and that is the point!

The implications of not being able to help your customer recover can be costly, but it may not be as easy to predict just how costly it could be. If you are unable to assist a customer recover from a passkey/encryption issue, they've lost data, and they've lost time. We don't know and can't possibly calculate whether this impacts their customers too. You're probably going to lose them as a customer. You will lose credibility in the market. Hopefully, they don't sue your company for losses. As I said, it's costly.

Now this is where I say, "*Where there's a will there's a way!*" In the fullest disclosure, nothing can prevent unforeseen failures entirely, but some additional engineering and planning will give your support staff the virtual lifeboat to survive the shipwreck if it ever happens. Please indulge me for a little more time.

We have already covered that the current version of LUKS (i.e., LUKS2) supports up to 32 slots for keys or passphrases. There's a good reason for this. You need more than one. In my jaded opinion, you truly need several. Don't be lazy. Set multiple. Document them for your product and support staff. They can be a lifeboat for your customer in the event of a catastrophe.

How you apply that knowledge is up to you. What I recommend is at minimum having three options, and I repeat, this is the bare minimum. They are as follows:

- The passphrase for LUKS used in the installer
- The key you create for the automated decryption of filesystems
- An emergency passphrase for all encrypted filesystems that is known only to your support team

That said, depending on the scale of some of your customers' implementations of your solution, you might consider adding a standard recovery passphrase as the fourth option. This would empower your support team to work more closely and freely with the customer to deal with any encryption issues.

At this point, your own internal documentation and processes for building and support must be spot on. Yet in all of this, who actually has access to those specific keys/passphrases must be controlled and regulated. "*Why?*" one may ask. Let me be blunt. The worst-case scenario would be a disgruntled employee with access to this critical information and sharing it – *anywhere* – such as on the Internet. The level of compromise for all your customers would be devastating.

You might be thinking to yourself, "*Geez, this whole book is chock full of doom and gloom!*" Well, yeah – it's a security book. Fear is a motivator. Move on. But all kidding aside, this is a very serious subject and a design decision you and your team must make.

Summary

Now, let's review what we've covered in this chapter. We have gone way beyond the base configuration of LUKS from common Linux installers. We have reviewed advanced ways of automating LUKS to improve security and the end-user experience; we have also covered more advanced ways of configuring the encryption keys and passphrases; and finally, and just as importantly, we have covered preventative measures with multiple keys and passphrases to virtually eliminate the probability of data loss during a critical support issue. I hope you have enjoyed this deep dive into LUKS as much as I have.

The benefits of leveraging this kind of encryption automation are extensive. In doing this, you will immediately create a better end-user experience as they will not be forced to memorize or type the passphrase every boot cycle. This implemented encryption protects your appliance and your end-customers' data or intellectual properties. Having this strong encryption gives the end-customer greater *peace of mind* in acquiring and using your solution. At the end of the day, it truly is all about solving your customers' problems and creating repeat customers who, in turn, tout your solution to their friends and colleagues.

Using LUKS should not add complexity for your customers and end-users. This is where implementing the automation of the passkeys is crucial. Ultimately, positive user experience and security can both be achieved in this instance. I hope that this deep dive into LUKS has been inspirational for you and given you new ideas on securing your filesystems.

Let's now move on to our next chapter where we'll drill deep into BIOS and boot security.

7
The Trusted Platform Module

Commonly just called **TPM**, short for **Trusted Platform Module**, this security-focused microcontroller chipset uses advanced cryptography to store critical or sensitive information. This could be in the form of credentials, passwords, biometrics, encryption keys, or other very sensitive data.

The exercises in this chapter will challenge your perceptions of what can or should be automated in an appliance. In this chapter, we will build upon methodologies to leverage this tool to further automate your encrypted passphrase authentication within your solutions, as initially introduced in *Chapter 6*. TPM, when leveraged properly, can become your enabler to a higher level of security and positive end-user experience. That said, TPM is not without its drawbacks and eccentric warts. It is notoriously not user-friendly and, sometimes, some of its registers can be unreliable. Whether or not this is a hardware or software issue, I know not. Regardless, I will demonstrate how to manipulate it for success.

Whether you choose to leverage TPM or not is not a simple decision to make. Weigh the pros and cons. It can truly take your appliance to the next level if you let it. This chapter will have the following headings:

- What is TPM?
- Configuring TPM by example

Let's get started.

What is TPM?

Beyond being one of hundreds of chips on your motherboard that most people could never identify, TPM is rapidly becoming a mission-essential tool for most operating systems. As of this book's writing, the average Linux system does not even require you to have TPM activated. Other operating systems, such as Microsoft Windows 11, actually require it for the operating system to be installed.

TPM provides mechanisms to securely store a variety of information securely. These objects can be (but are not limited to) license keys, user credentials, encryption keys, or other types of data that provide for the consumption of said data without user interactions.

There are different types of TPM implementation – firmware, discrete, and integrated TPM. TPM can also be done via software (aka virtual TPM) but it can possibly be worked around since it has no more protection than any other software programs running on top of your operating system. Avoid software TPM unless it's a last resort.

Firmware TPMs take advantage of the system CPU's trusted execution functions. This, by default, creates what is known as a **trust anchor** for that system. These can only be cleared by being physically present at the console and cleared out via setting in the **UEFI BIOS**. These are the most commonly found – in my opinion, the most reliable as well.

Discrete TPMs are similar to firmware-based TPMs except that they are controlled outside of the UEFI BIOS through other APIs and software. Discrete TPMs leverage functionality within your CPU itself rather than the firmware. If you replace your CPU or change your operating system, you will lose all the TPM data previously stored. Otherwise, these are virtually just as good as those in one's firmware.

Finally, there are the **integrated TPMs**; these are chipsets that perform many functions but also have the TPM functionality built in. This type of TPM functionality is the least common in the field.

I've been asked *"Is using one over the others leaving oneself vulnerable?"*. The short and correct answer is that not using any TPM solution is what may leave one's solution vulnerable. All TPM implementations work. Which you choose is solely based on your own criteria and what is available for your chosen platform.

Currently, the gold standard is TPM version 2.0. TPM 2.0 is feature-rich and can store multiple keys and values. It is found virtually on all the latest servers, PCs, and laptops. Its algorithms and built-in cryptography are impressive.

Older systems may have the earlier implementation of TPM 1.2. This version has limited capacity and lesser-strength cryptography, yet it is leaps and bounds better than not having anything.

Most government requirements standards globally set TPM 1.2 as a minimum requirement for any system that touches sensitive data. Of course, TPM 2.0 is requisite on the most sensitive platforms.

Let's now move on to a brief history of TPM.

The history of TPM

I'll summarize a quick historical overview for you, and I'll intentionally keep it brief and spare you any drudgery. TPM was the result of brainstorming by a technology industry think tank called *Trusted Computing Group* back in 2009. This concept was turned into a global standard by the **International Organization for Standardization** (**ISO**) in conjunction with the **International Electrotechnical Commission** (**IEC**). Hence, TPM was born as *ISO/IEC 11889:2009*.

TPM 1.2 became a global standard back in 2011 and reigned for many years. It was only replaced by TPM 2.0 in 2019 as the *ISO/IEC 11889:2015* publication. Sadly, it is now considered insufficient and obsolete by most security organizations. I still say using a TPM 1.2 module is better than using nothing. So clearly, you can make your own decisions.

Here's a graphical description of how TPM works (credit, Wikipedia, https://upload.wikimedia.org/wikipedia/commons/thumb/b/be/TPM.svg/2880px-TPM.svg.png):

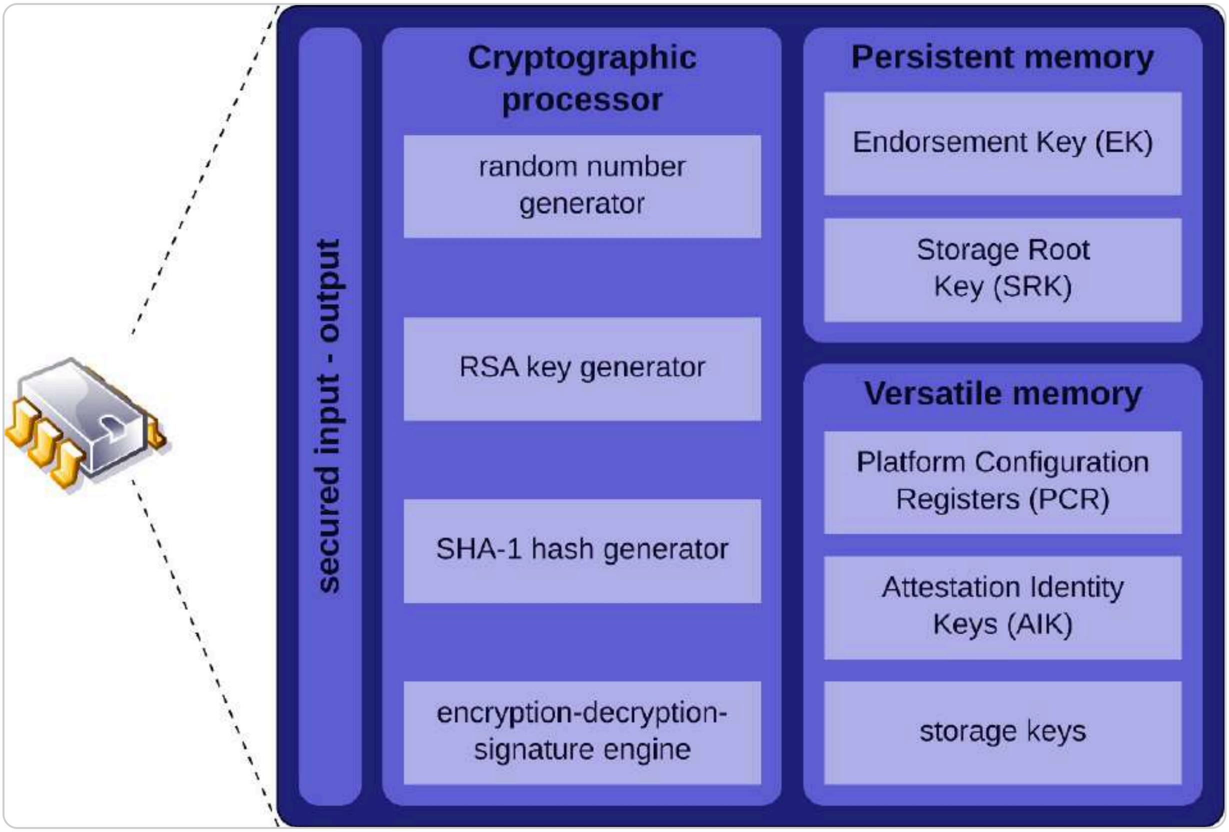


Figure 7.1 – TPM description

TPM 2.0 is still, as of this book's publishing, the global standard. The exercises in this chapter will solely focus on TPM version 2.0. So, let's move on to those exercises now.

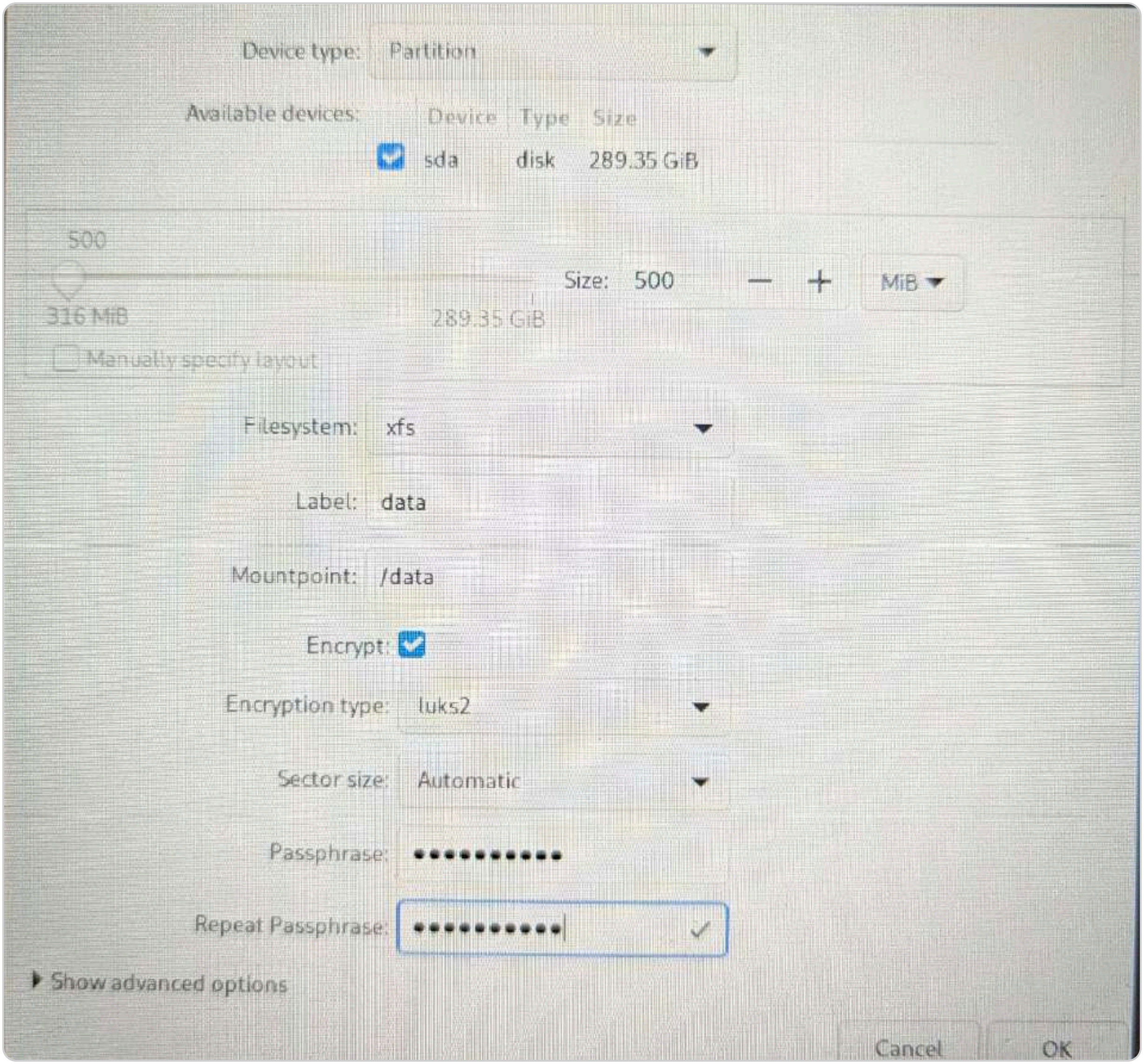
Configuring TPM by example

In this exercise, we'll implement the automation of a storage volume's decryption by storing an encryption passphrase within the TPM securely. Doing such in a real-world appliance makes your solution more secure and prevents having to share the encryption passphrase with the public, hence, by nature, making your solution vastly more secure.

REQUIREMENTS FOR THE EXERCISE

For this lab, I installed Fedora40 (Workstation Edition) onto bare-metal hardware. During this installation, I created a 500 MiB encrypted XFS filesystem mounted onto **`/data`**. *I used* **`TPMmaster2024`** *as the encryption key.*

Here's what that setup looked like during the creation process:



Device type: Partition

Available devices:

Device	Type	Size
✓ sda	disk	289.35 GiB

Filesystem: xfs

Label: data

Mountpoint: /data

Encrypt: ✓

Encryption type: luks2

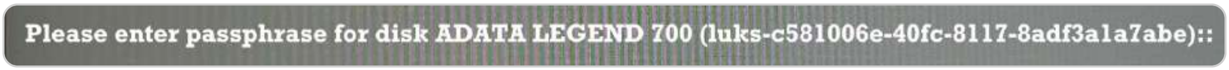
Sector size: Automatic

Passphrase: ●●●●●●●●●●●

Repeat Passphrase: ●●●●●●●●●●●

Figure 7.2 – Encrypted filesystem setup during Fedora installation

Upon completion of the installation, when the system boots, we are prompted to enter the **LUKS key** in order to boot and mount that filesystem:



```
Please enter passphrase for disk ADATA LEGEND 700 (luks-c581006e-40fc-8117-8adf3a1a7abe)::
```

Figure 7.3 – Entering the LUKS key manually at boot

For this exercise, I recommend you add a really small LUKS encrypted partition to an existing test machine without doing a complete reinstall. It's up to you. In my example, I've used a non-system partition/filesystem, but in production, you would be encrypting almost everything except for `/boot` and the EFI partition.

Exercise – enabling TPM 2 in conjunction with LUKS encryption

First, let's ensure that your system actually has the correct hardware. We'll browse the logs to see whether a TPM 2.0 module was found at the last boot cycle. If this doesn't return good results, you may not have the requisite hardware for this exercise:

```
$ sudo dmesg | grep TPM
```

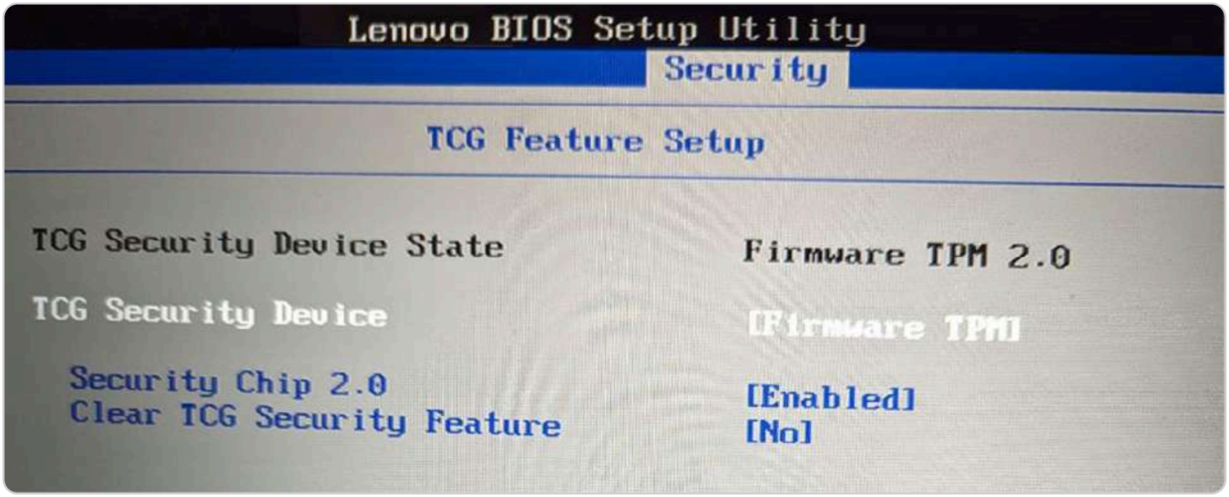
The output for this command can be somewhat lengthy, so I will not be displaying all of the possible output here for this instance. The key thing to observe in the output you get from running the command is `TPM2`:

```
[    0.011277] ACPI: TPM2 0x000000009AECAF08 000034 (v04 LENOVO TC-M1U   00001450 
AMI  00000000)
[    0.011320] ACPI: Reserving TPM2 table memory at [mem 0x9aecaf08-0x9aecaf3b]
[    1.105253] tpm_tis MSFT0101:00: 2.0 TPM (device-id 0x1B, rev-id 16)
[    1.726489] systemd[1]: systemd 255.6-1.fc40 running in system mode (+PAM +AUDIT 
+SELINUX -APPARMOR +IMA +SMACK +SECCOMP -GCRYPT +GNUTLS +OPENSSL +ACL +BLKID +CURL 
+ELFUTILS +FIDO2 +IDN2 -IDN -IPTC +KMOD +LIBCRYPTSETUP +LIBFDISK +PCRE2 +PWQUALITY 
+P11KIT +QRENCODE +TPM2 +BZIP2 +LZ4 +XZ +ZLIB +ZSTD +BPF_FRAMEWORK +XKBCOMMON +UTMP 
+SYSVINIT default-hierarchy=unified)
[    4.604247] systemd[1]: systemd 255.6-1.fc40 running in system mode (+PAM +AUDIT 
+SELINUX -APPARMOR +IMA +SMACK +SECCOMP -GCRYPT +GNUTLS +OPENSSL +ACL +BLKID +CURL 
+ELFUTILS +FIDO2 +IDN2 -IDN -IPTC +KMOD +LIBCRYPTSETUP +LIBFDISK +PCRE2 +PWQUALITY 
+P11KIT +QRENCODE +TPM2 +BZIP2 +LZ4 +XZ +ZLIB +ZSTD +BPF_FRAMEWORK +XKBCOMMON +UTMP 
+SYSVINIT default-hierarchy=unified)
[    5.477528] systemd[1]: systemd-pcrextend.socket - TPM2 PCR Extension (Varlink) was 
skipped because of an unmet condition check (ConditionSecurity=measured-uki).
[    5.507674] systemd[1]: systemd-pcrmachine.service - TPM2 PCR Machine ID Measurement 
was skipped because of an unmet condition check (ConditionSecurity=measured-uki).
[    5.508733] systemd[1]: systemd-tpm2-setup-early.service - TPM2 SRK Setup (Early) was 
skipped because of an unmet condition check (ConditionSecurity=measured-uki).
```

IMPORTANT NOTE

If you didn't get any **`TPM2`** *output, it's probably safe to say that your lab hardware is insufficient for the exercises in this chapter.*

Now that we've established you have a TPM version 2.0 module, let's ensure it is set up properly in the UEFI BIOS. Reboot your machine, interrupt the boot process so that you can enter the UEFI BIOS setup, and then go to your TPM configuration:



```
Lenovo BIOS Setup Utility
Security

TCG Feature Setup

TCG Security Device State            Firmware TPM 2.0
TCG Security Device                  [Firmware TPM]
  Security Chip 2.0                  [Enabled]
  Clear TCG Security Feature         [No]
```

Figure 7.4 – UEFI BIOS – setting up TPM

You may have more than one option for TPM; if so, choose **Firmware TPM** over **Discrete TPM**. Set the chipset to **Enabled**, and to ensure that there's nothing legacy left inside the chip, clear it before proceeding. Do not forget to save and exit your UEFI BIOS (and reboot).

Let's install the packages we'll need to automate the decryption process leveraging our TPM:

```
$ sudo dnf install -y clevis clevis-luks clevis-dracut clevis-systemd clevis-pin-tpm2
```

The output for this is rather lengthy, so I have truncated what is displayed. What is important is that your package installations are completed successfully:

```
Installed:
  clevis-20-2.fc40.x86_64               clevis-dracut-20-2.fc40.x86_64    clevis-luks-20-
2.fc40.x86_64
  clevis-pin-tpm2-0.5.3-5.fc40.x86_64   clevis-systemd-20-2.fc40.x86_64   jose-13-
1.fc40.x86_64
  libjose-13-1.fc40.x86_64              libluksmeta-9-22.fc40.x86_64      luksmeta-9-
22.fc40.x86_64
Complete!
```

We're not done yet. There are more packages that we must ensure are installed properly:

```
$ sudo dnf install -y tpm2-tss tpm2-tools tpm2-abrmd tpm2-pkcs11
```

The output for this command is rather verbose, so I have truncated it to only show what I recommend that you check for on your execution (i.e., the package installations completed successfully – please note that some packages may already have been installed previously depending on how you configured your lab machine):

```
Installed:
  tpm2-abrmd-3.0.0-5.fc40.x86_64                tpm2-abrmd-selinux-2.3.1-10.fc40.noarch
  tpm2-pkcs11-1.9.0-5.fc40.x86_64
Complete!
```

Next, we'll need to determine the exact device name for our encrypted device:

```
$ lsblk
```

This command is rather important in its lengthy output as it will guide you as to how your disks are carved out and the sizes of filesystems:

> ***IMPORTANT NOTE***
>
> *Your system's output might vary from the results I have – make note of how your disks are carved out. This will be important in this exercise.*

```
NAME                                          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda                                             8:0    0 476.9G  0 disk
├─sda1                                          8:1    0   200M  0 part  /boot/efi
├─sda2                                          8:2    0     3G  0 part  /boot
├─sda3                                          8:3    0    50G  0 part  /
├─sda4                                          8:4    0    64G  0 part  [SWAP]
├─sda5                                          8:5    0    50G  0 part  /var
├─sda6                                          8:6    0    50G  0 part  /usr
├─sda7                                          8:7    0    50G  0 part  /home
└─sda8                                          8:8    0   500M  0 part
  └─luks-463aba53-7189-4920-a128-b4db2a314848 253:0    0   484M  0 crypt /data
zram0                                         252:0    0     8G  0 disk  [SWAP]
```

So, for example, my block device is `/dev/sda8` (please note that yours will be different).

We'll take that information and feed it into the next command, which will bind our key into TPM:

```
$ sudo clevis luks bind -d /dev/sda8 tpm2
'{"hash":"sha256","key":"rsa","pcr_bank":"sha256","pcr_ids":"6,7"}'
```

Please note that this command will ask you to authenticate to use elevated permissions. Then, it will ask you to confirm the existing LUKS passphrase you wish to use to automatically decrypt the volume:

```
[sudo] password for mstonge:
Enter existing LUKS password:
```

Now, let's tell `systemd` to always use the TPM2 module first when trying to access that drive:

```
$ sudo systemctl enable clevis-luks-askpass.path --now
$ sudo systemd-cryptenroll --tpm2-device=auto --tpm2-pcrs=6+7 /dev/sda8
```

This command will ask you to confirm the existing LUKS passphrase:

```
 Please enter current passphrase for disk /dev/sda8: •••••••••••••
New TPM2 token enrolled as key slot 2.
```

All right, we're not done yet. We'll use `dracut` to regenerate the proper settings for booting.

```
$ sudo dracut -fv --regenerate-all
```

Again, the output for this will be exhaustingly extensive, but I do ask that you pay close attention to your own output results. Here, I will truncate my results and only highlight the end state:

```
dracut[I]: *** Stripping files ***
dracut[I]: *** Stripping files done ***
dracut[I]: *** Creating image file '/boot/initramfs-6.8.5-301.fc40.x86_64.img' ***
dracut[I]: Using auto-determined compression method 'pigz'
dracut[I]: *** Creating initramfs image file '/boot/initramfs-6.8.5-301.fc40.x86_64.img'
done ***
```

Now we've configured the new booting parameters and kernel image, it's time to reboot your system. If all goes well, you will *not* be prompted to enter your LUKS password (it'll be done for you automatically thanks to TPM):

```
$ sudo systemctl reboot
```

> ***IMPORTANT NOTE***
>
> *TPM2 can be rather problematic. Depending on your system, it's common to have to rebind LUKS passphrases with* `clevis` *after some patching cycles.*

TPM is known to be troublesome, if not uncooperative, to Linux. Many Linux developers are working hard to iron out the issues. I suspect that, in the near future, as these issues are resolved, we'll see some distributions forcing TPM2 usage in the same manner that Windows does today.

I trust you enjoyed this platform exercise module. (See what I did there? Dad jokes – sorry, hahaha!)

Summary

In this chapter, we covered TPM, its versions, and its history. I kept it brief in order to not bore you to death. The point of this book is to help you understand how to make things more secure rather than being an anthology of technological evolution.

You got a rare glimpse into how this understated TPM technology can aid you in automating your appliance's security. Although TPM is an imperfect solution, it has its merits and its risks. More importantly, you should consider implementing TPM to automate your encrypted filesystems with your product.

In our next chapter, we'll go even deeper into disk encryption functionality. Let's move on.

Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

https://packt.link/embeddedsystems

8
Boot, BIOS, and Firmware Security

So far, we have covered TPM and LUKS disk encryption to protect the data. This chapter will push you to reflect deeper into the basest of hardware functions and your boot system itself. I'm going to warn you upfront. The settings we will review here are truly a double-edged sword, and by that, I mean simply that they can protect the system but also, they can make supporting the same system at a customer site painfully difficult.

Understanding how to best lock down and protect your BIOS (short for Basic Input Output System), its firmware, and your operating systems' ability to boot securely will ensure that your customers are rewarded with a more resilient product and your team has reduced your company's exposure to risks.

Locking down your BIOS and boot options, albeit a great idea, does have support implications that I feel compelled to highlight. We'll cover some of these in greater detail.

In this chapter, we will cover the following topics:

- Deep dive into various booting system components
- Understanding boot-level security using examples
- Possible threats in firmware

Deep dive into various booting system components

Ever wonder what happens when you turn on a Linux system? Let's take a quick look at what really goes on.

First, we'll dive into your system's BIOS. Virtually all modern x86_64 platforms leverage a Unified Extensible Firmware Interface (UEFI) BIOS, which is basically a firmware interface that serves as a gateway from your operating system to the hardware it controls. Older platforms have vendor-specific or legacy BIOS types that follow limited standards from one system to another. For ease of understanding the differences between legacy BIOS and the new standard UEFI BIOS, I've created a table here to contrast their features or functionalities.

Concern	**Legacy BIOS**	**UEFI BIOS**
Standardization	Each vendor has its own feature/functionality	Industry-standard functionality regardless of vendor

Concern	Legacy BIOS	UEFI BIOS
Digital signatures	None	Secure Boot
Partition table support	MBR partition table only	GUID partition table support added
Platform	Limited	x86, X86_64, ARM, ARM64, PowerPC, and others
Modularity	None	Modular – vendors can add additional support (i.e., networking, storage, etc.)

Table 8.1 – Contrast of the two BIOS types

It all starts with the pushing of the Power On button. A modern marvel of chain reactions begins. The system's firmware BIOS/UEFI kicks into action. It initiates a POST (short for Power-On Self-Test). Once it's completed checking itself, it searches for and initiates a Master Boot Record (MBR).

The MBR kicks off the bootloader, which, in our example, is GRUB2. GRUB2 then loads the defined kernel executable. The Initramfs then kicks into gear and takes care of tasks such as decryption, software RAID, and filesystem loaders.

Once all of that orchestration has concluded and the root filesystem is loaded, systemd gets started. Systemd loads all the other filesystems and starts up all enabled network functions and application services. And there you have it… a functional Linux system.

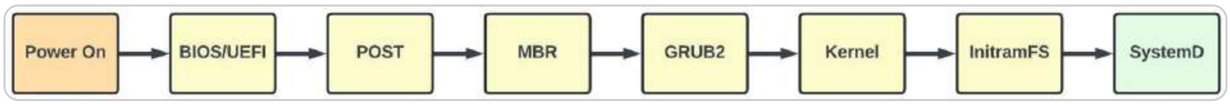


Figure 8.1 – Booting up

None of this goodness happens without your firmware UEFI BIOS. Locking down what can be accessed and how boot devices can be used is crucial. Without such precautions, it will be impossible to protect the system from massive changes like having an alternative boot source becoming available, which would enable anyone with direct access to the hardware not only to boot from a non-approved device but possibly introduce malicious code into your system or, even worse, erase your entire system's drives.

Let's move on to some security examples in the next section.

Understanding boot-level security using examples

In this section, we'll look at how to access the UEFI configuration and then we'll walk through some of the key security settings that you need to be aware of. We'll also review some of the pros and cons of such configurations.

Depending upon the distribution (and version of said distribution) of Linux you choose to deploy, the tooling available to you and the complexities around using such are as varied as the weather. Again, I will reiterate that choosing an enterprise-supported version of Linux will ultimately provide a better outcome.

Additionally, the newer your hardware platform is, the more likely you are to have the latest security enhancements in your firmware, CPU, and, of course, your BIOS. Let's now move on to how one can access those security settings.

Accessing the UEFI configuration

Let me begin by addressing the key step in securing the UEFI configuration. This requires that you have physical access to the machine's console.

How does one access the UEFI BIOS interface on their machine? There are a few paths that all lead to the same destination.

First, one can interrupt the boot sequence; usually, this is done by pressing a function key before GRUB kicks in on its part of the boot process. The specific key to be pressed is often different between vendors and models of systems. This step is a true interruption of the POST process and is the easiest way of getting to the BIOS, in my opinion.

Second, one can use GRUB2's menu. Often, there is an entry that can allow you to enter the UEFI configuration, but this may not be present on all systems or distributions. Some platforms may also make editing UEFI firmware settings a manual transaction separate from the bootloader.

Third, from within the OS… Well, not exactly. The OS can be rebooted into the firmware setup without prompting you. Here's a quick but exact example of how to tell your system to directly reboot and enter the UEFI configuration utility. Like all administrative functions, this one must be run as root:

```
 # systemctl reboot --firmware-setup
```

The above command has no output, per se, but will immediately reboot into the UEFI configuration screen.

Well, since we're talking about boot processes, let's mention the password that can be present (or set) here. The UEFI admin password setting, in my professional opinion, is a double-edged sword. For the exact same reasons, I feel the same way about also setting a bootloader (i.e., GRUB2) password. This is quite similar in effect to the issues I have highlighted in leveraging things like LUKS without TPM automation, as mentioned in *Chapters 6* and *7*.

Assume you elect to set a password for either of these settings. Let's review the implications for your end-users. Granted, for the UEFI admin password, it's a 1-in-100,000 odds situation where the end-user would have a legitimate excuse for accessing it. It may never be an issue. Sadly, this is not the case in setting a bootloader (i.e., GRUB2) password. Every boot/reboot would force the user to manually enter the password on the console for the appliance to boot. For appliances with no keyboard perpetually attached, this is impossible besides impractical. For appliances with a keyboard, the result of this is that a lazy end-user tapes a note with the password to boot the system directly on it, and that eliminates any benefit of having the password to begin with.

So, you might be asking yourself why this is an issue. My response is human nature and bad behaviors. It's generally safe to assume that all people most often will take the path of least resistance. This will obviously be the case in something that is required at every boot cycle. The end-customer would absolutely require access to this password.

In the case of the UEFI admin password, let's assume that it's a support issue that forces your company to give that information to a customer. In both cases, the security such passwords are implied to carry is devalued by an end-user not only having them, but the probability that they will be shared outside your customers' organizations is the greatest threat to be mentioned.

I urge caution in leveraging either unless each individual appliance has unique passwords assigned and your support team has a database of them. Even then, once shared with the end-customer, they are to be considered compromised, negating the protections they were designed to provide.

What is Secure Boot?

Long answer short, **Secure Boot** is an excellent way of establishing with your users that your operating system is running on a kernel and binaries that your company personally has tested and signed with a key.

There are some key variables at play, as shown in *Table 8.2*:

Credential/Variable	**Purpose**	**Creating Entity**

Platform Key (PK)	Root Key	Platform Manufacturer
Key Exchange Key (KEK)	List of certificate owners	OS Partner or OEM
Authorized Database (db)	List of allowed driver or app	OS Partner or CA
Exclusion Database (dbx)	List of revoked signers	Signing Authority
SetupMode	1 = Setup Mode, 0 = Secure Boot Enabled	
SecureBoot	1 = Secure Boot is enforced, 0 = Secure Boot is disabled	

Table 8.2 – Key variables and their roles in Secure Boot

Before we start getting our hands dirty with exercises, let's look at the differences in the boot process without and then with Secure Boot enabled. Here's a graphical representation of the boot process without Secure Boot in place.

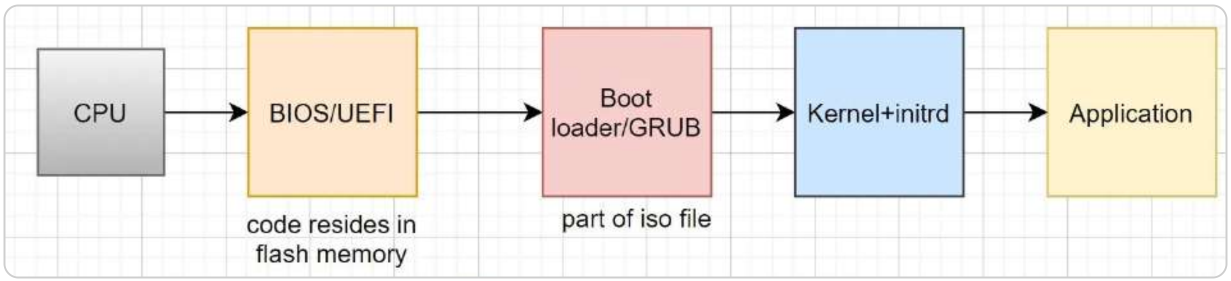


Figure 8.2 – Non-secure boot chain

As you can see from the preceding graphic, anything could be loaded without any checks or balances. Generally, that's a bad idea when building a product. So let's work toward guaranteeing to our customer base that what they are running is verified, true, and secure. Here's a graphic that describes the Secure Boot process.

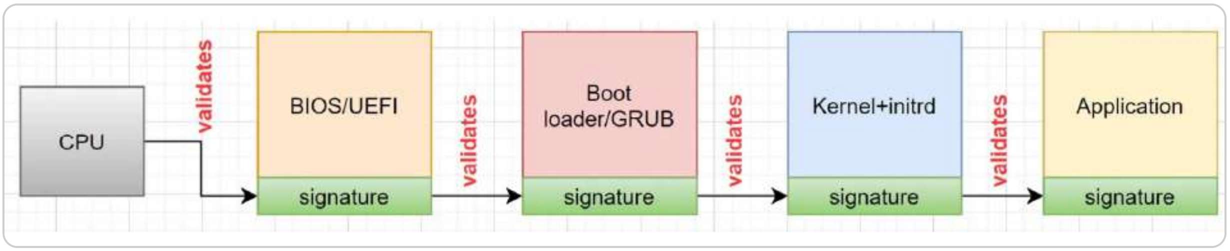


Figure 8.3 – Leveraging Secure Boot and keys

Configuring Secure Boot in Linux just got harder for most people trying to leverage their own keys. As of the writing of this book, the primary tool for achieving this functionality was a package called **efitools**. This extensive toolbox provided for the backup and recovery of keys, directly setting keys in the UEFI firmware, and much more. Sadly, as happens in open source from time to time, a critical tool is abandoned without its replacement being ready for primetime.

The planned replacement, from what best information I have seen, is a new tool called **sbctl**. There is limited information regarding this tool as of now. I am hopeful that it may find its way into Fedora 42 and other distributions very soon. Here's a link to the project's GitHub – https://github.com/Foxboron/sbctl – where you will find some interesting information on this exciting project.

So, this does create a temporary quandary for those of us wanting to use our own keys for Secure Boot in our appliances. There are a few options. This is where you must make your own judgement call until the new tooling gets itself into the mainstream Linux distributions. It's difficult for me to advise in any direction here:

- **Option 1**: You could rely solely on your Linux distribution's keys themselves for the enablement of Secure Boot. This may or may not work depending on whether or not your solution requires custom kernel modules, which may or may not contain the correct signatures or even one at all.
- **Option 2**: Forgo the usage of Secure Boot temporarily until you can run it on your own terms and with your own keys in the driver's seat. There's no timeline for when you may be able to veer away from this course yet unless you consider exploring one of the other options. As a close friend pointed out to me, this also lacks a disclaimer that notifies you that you may be exposed to malware risks.
- **Option 3**: Albeit the immediate path to using your own keys and truly using Secure Boot with only your own keys, you could download an existing *efitools* package for your distribution or compile it yourself from source code.
- **Option 4**: If you're feeling daring, you could attempt to use an early-release version of *sbctl* by getting it from the project's GitHub and building it yourself.

You are not without choices on this Secure Boot issue, but that is just one issue you'll need to consider. Let's move on to other issues that can impact your firmware.

Possible threats in firmware

Malicious code infecting firmware, such as a BIOS rootkit, seems to be the newest attack vector on a global scale. It is also a difficult-to-detect issue for security teams. This problem is industry-wide and, by that statement, it impacts virtually all hardware vendors creating solutions for the x86_64 platform family. One high-profile example of such evil comes through the appropriately named LogoFAIL attack.

LogoFAIL exploits a feature within all manufacturers' UEFI BIOS, which enables them to create a custom splash screen at boot, hence displaying their company logo. It has found a way of injecting malicious code into the process, which enables the execution of code without the users' knowledge with severe security implications.

In the past couple of years, two dozen high-severity CVEs have been created for vulnerabilities that impact millions of systems globally – network hardware, storage systems, servers, industrial controllers, edge devices, and laptops. Virtually all global manufacturers have been forced to make emergency updates to battle these possible attack vectors. As of today, when I'm writing this page (September 2024), when I search in NIST's National Vulnerability Database, the query returns over 4,500 results for firmware CVEs alone. To me, that quantity is grievously staggering to think about. The only good news I can offer in this situation is that the vast majority of those are already reported as resolved.

Some of these vulnerabilities stem from hardcoded credentials for support access being compromised, others from their very own bytecode having massive security gaps. The worst cases involve situations where actual malicious code has been installed without the end-user even knowing. Regardless of what the inception of the vulnerability is, its impact can be devastating. These are ticking time bombs, as bad actors can compromise a system and simply wait before leveraging the exploit.

For a product manager, these CVEs must be noted, tracked, and fixed. Your key takeaway from these firmware horror stories is twofold. First, ensure that the firmware on your appliances is at the version recommended by your hardware vendor. Second, ensure your firmware supply chain itself is secure. Accept updates only from credible sources or the vendors themselves. Do not assume that the firmware your systems arrived with should be the version that your team will ship. Again, trust but verify. Double check. Triple check.

Virus scanners cannot help with detection. They look at dissected files, not firmware. They have their place, but they are only a part of an overall assessment of security, not totality.

A new generation of tools has been created to help administrators detect, report, and update vulnerable firmware and BIOS modules. Open source has greatly assisted in this effort. Hardware vendors can openly share their updates with the community via tools such as a firmware update manager (aka the Linux Vendor Firmware Service). Vendors can choose to either share information regarding the versioning of specific firmware or to also provide updates to the ecosystem. With this new service in Linux, administrators can be easily made aware of available updates and/or automatically install them on their appliances. This is also a great way of sending updates to an appliance in the field.

Summary

I hope you have taken this chapter to heart. We have covered a topic here that is often referred to as a dark art in security circles due to the complexities it creates. By that, I mean securing firmware and the boot process along with the digital signatures of your stack. I trust that this chapter has instilled a higher degree of understanding of how to configure boot-level securities. Finally, we have also touched upon the unseen and often untalked-about vulnerabilities that can exist in firmware. Through services like those now provided by Linux or via the NIST database, you can stay informed about threats as they are reported. Knowing is half the battle. In our next chapter, we'll explore a new way of deploying a Linux appliance through immutable images.

9
Image-Based Deployments

In this chapter, we are going to truly diverge from the existing norms of package-based installs (and updates), which have been the mainstay of Linux for over thirty years. Let's talk about **immutable image-based systems**. Immutable images have only been around for a few years. Just recently, they have captured the limelight. Some will argue (myself included) that immutable operating systems for appliances are the future of all embedded systems. The very thought of a system that is immutable and unchanging just gives us the impression of something even more secure than the rest. Or does it? If you too assume that it is more secure, you would be correct. More secure – yes – but not perfect...this chapter will give you a great overview of a topic that in my professional opinion deserves its own book due to its complexities and rapidly evolving future. In this chapter, we will review and also go through some focused exercises geared towards providing you with a greater understanding of the levels of security provided by immutable operating systems. We will take a look at some of the tools available and how to leverage them into building, deploying, and, later on, supporting your appliances once your customers have them in hand.

By the end of this chapter, you might just convince yourself that this new method of deploying Linux was tailor-made for us in the embedded Linux systems community. Conversely, you may also decide after digging deeper into the technology that perhaps it's not for you (yet).

In this chapter, we will have the following main headings:

- Introducing image-based Linux deployments
- bootc and bootable container images
- Special tooling and support infrastructure differences
- Limitations of image-based deployments
- Updating and rolling back changes
- Practical exercises – step-by-step walkthrough of how to deploy image-based systems

So, let's move on and dive deeper.

Technical requirements

To successfully navigate through the exercises in this chapter, you will need two bare metal or virtual machines that you can modify or reinstall the operating system upon. Root access, internet access, and DHCP IP addressing are mandatory. You will also need the ability to download ISO images and have a 16 GB (or greater) USB thumb drive. Finally, you will need a free Red Hat Developer account and access to your own **Quay** registry (also free). The requirements are greater for these exercises as the outcomes are more significant. I am hopeful you'll even have fun along the way, which in my opinion is also a requirement.

For these exercises. I am using CentOS Streams 9 as my build machine's operating system, and we'll be creating a CentOS Streams 9 bootable container image. Let's now move onward to the stars of the show, image-based Linux deployments.

Introducing image-based Linux deployments

This subject is something near and dear to my heart, as I am an embedded systems specialist and have dedicated the past decade to this endeavor. It's truly my opinion that this set of technologies was virtually tailor made for our embedded Linux systems appliances and the ecosystem of people who build and support them. It would be justifiable for you to question why I, along with many other leaders in the industry, see this as the future. I will give you the short answer here, but we will review it in depth as the chapter progresses. Simplified lifecycle management is the single greatest reason to adopt this new technology for your products.

In the upcoming sections, I will walk you through two of these methodologies, their features, their limitations, and their tooling, along with some exercises to make it all real for you.

rpm-ostree and atomic images

The first type of image-based installation method that we'll review is **rpm-ostree**. This type of deployment is sometimes also called *atomic* because of its immutability. Some vendors have their own marketing name for it. Red Hat® calls it RHEL for Edge™. Whatever they call it, I call it a game-changer. Many other distributions, in recent years, have added rpm-ostree deployment options. Maybe you'd like a few more examples? Fedora® Silverblue, NixOS, openSUSE® MicroOS, Nitrux, Vanilla OS, Talos Linux™, and BlendOS are some of the many purely atomic distributions.

Any use case for Linux you previously had can be addressed with greater security and stability with an image-based deployment. Where you are deploying is irrelevant; these systems run on bare metal, virtual machines, or cloud instances with ease.

The **OSTree** package is the star of the show here. It is used to compose, build, deploy, and update the image-based operating system. It's almost like a filesystem tree that is married to a version control system. OSTree has support for two persistent and writable directories whose contents are unscathed across upgrades: `/etc` and `/var`. Because of this image format, the old-school methodology of partitioning is essentially moot. …Or should I say immutable?

OSTree at its core replicates read-only trees via HTTP. It also has mechanisms for application layering and installation either in `/var` or `/home` (which more details about their actual filesystem layout will be covered next). This mechanism is like how rpm repositories are hosted; however, they are different in the manner of images as reference data (versioning, tags, etc.). For your solution, you will have to create the infrastructure to support your deployed appliances.

To give you a rough idea of how drastic a change in what might have been its own writable partition in the standard deployment model, here's a look at how they are handled in this new model. With `/var` being RW, there are several key directory/filesystem mappings that are different from what you are used to in Linux. Pay close attention.

Let's see how filesystems are mapped and their permissions:

- **`/home`** links to **`/var/home`** as RW
- **`/srv`** links to **`/var/srv`** as RW
- **`/root`** links to **`/var/roothome`** as RW
- **`/opt`** links to **`/var/opt`** as RW
- **`/mnt`** links to **`/var/mnt`** as RW
- **`/sysroot`** is RO
- **`/boot`** and **`/boot/efi`** are RW

NOTE

`rpm-ostree` *is the mechanism that enables* ***RPM packages*** *to be layered on top of an* ***`ostree`*** *image base. Installing an RPM package with* ***`rpm-ostree`*** *forces the creation of a new image that is merely an update of the base image.*

To oversimplify, I will try to relate this methodology as I see it: a series of commits that build upon each other that can be easily rolled back delivered as an immutable image-based appliance-like deployment. I'll say it again. I feel like sometimes this was tailor-made for embedded Linux systems and custom appliance solutions.

So, how can we take advantage of `rpm-ostree` and layer more awesomeness on top? Let's take a quick look at some of the ways as we already know that the base image itself is immutable. First, I give you this disclaimer: anything you layer on top might not be maintained as part of that image and could induce the risk of being forgotten as updates to that system progress.

That said, we can layer things on top of an rpm-ostree system (outside the image itself). This can be done through fully self-contained applications deployed via flatpacks or as a running container image deployed outside the image. In this use case, the base image is simply the immutable platform for delivering a dynamic application.

The other method is to install RPM packages on top of the image. I suppose someone has a use case where this makes sense, but I argue that it should just have been made part of the image, and the system should be updated with the latest image containing the additional RPM packages. Now that we understand rpm-ostree and its key features, here are examples of distributions that can be deployed *optionally* as rpm-ostree. Please note that this is a non-exhaustive list:

- Red Hat® Enterprise Linux®
- Fedora® (many variants; the most common is Silverblue)
- Vanilla OS (an Ubuntu®/Debian® variant)
- Debian® ostree
- openSUSE® (Slowroll)
- SUSE Linux Enterprise Server™
- CoreOS®
- Rocky Linux®
- AlmaLinux®
- Oracle Enterprise Linux™
- CentOS® Stream

So, please know that you have many options for testing the waters with rpm-ostree, for more details I invite you to check out the libostree documentation online: https://ostreedev.github.io/ostree/.

Let's move on to our next subject and that's bootable containers.

bootc and bootable container images

I have been involved deeply with product teams in the scoping, testing, and documentation of this new leading-edge technology since before I started writing this book. These efforts introduced me to some really amazing engineers who are adept at thinking outside the box.

What is bootc? A simplified way of looking at bootc would be to define it as a tool for layering a Linux kernel into a container image so the container image can boot itself without external operating systems hosting it.

If you are interested in contributing to this global effort to transform technology (or just learning more), here's the bootc GitHub repository: https://github.com/containers/bootc. There, you can find extensive documentation that is readily maintained.

Let's be clear, bootc is bleeding-edge technology. Red Hat® (the organization leading the charge in this new technology) considers this to have *Technology Preview* status and they will not consider it fully supportable until the tooling and support infrastructure are present. Basically, the technology will not be considered fully supportable, or as Red Hat calls it, **Generally Available** (**GA**), and production-workload-ready until the advent of RHEL 10 and RHEL 9.6, which are both released in May of 2025.

This doesn't mean you should be waiting around. I highly recommend doing an exhaustive level of testing and due diligence to see if this deployment method can add value to your product now and in the future.

What exactly is bootc? bootc is an open source project that leverages an **Open Container Initiative** (**OCI**) standard-based container image to create a full operating system image. It leverages the same layering techniques that existing container infrastructures use today for their creation and also for updating themselves. It leverages the functionality of other projects, such as rpm-ostree, to create and update those operating system images.

What makes bootc different from rpm-ostree? Both are image-based deployment methods. Both have levels of immutability. What sets them apart is how they are built and how they are updated. The single greatest defining point is bootc's unique ability to use a container as the basis for an immutable image.

Where the bootc methodology absolutely shines is that there's a vast ecosystem of developers and tools for building containers already. If you can build a container, you can build a whole system now. That whole build chain is now transformational, and for most companies, it will require little tooling to adapt to this new methodology if they're already using containerized apps today.

But hold on…in corporate America, and elsewhere globally, not every containerized application needs to be converted to an image-based appliance. The true transformations are, for us, the embedded Linux system community. With this technology, we can design, create, build, and automate the lifecycles of the products that drive our company's success.

Many engineers are currently working feverishly to deliver new tools to support this type of deployment method. By the time RHEL 9.5 ships, the community should have all new functionality available within what is considered the premier tool for building systems images and installers from the RHEL ecosystem and that tool is **Image Builder**. Image Builder has been around since RHEL 8.3 days (quite some time ago), but it has evolved and keeps getting better month by month.

For those of us building images in Fedora and CentOS Stream, we can leverage new functionality today at the risk of being on the bleeding edge without the support and obvious stability of the enterprise distributions. I highly recommend checking out the ongoing development of functionality for Image Builder in this GitHub repository: https://github.com/osbuild/bootc-image-builder.

Now that we've introduced rpm-ostree and bootc images, let's continue our journey and see how they differ.

Special tooling and support infrastructure differences

This is where we can easily set one methodology against the other. Both methods work eloquently; however, creating, maintaining, and supporting appliances built from rpm-ostree alone is more difficult, and their tooling is more intensively hands-on.

In the next few sections, I will primarily focus on open source tools that are the most relevant for either methodology. Let's start this investigation by looking at tooling that can help you deliver an rpm-ostree solution.

rpm-ostree open source tooling

Today, there's a vast set of tools available within the open source community for your teams to leverage, regardless of the operating system you have selected. That said, each distribution may have some slight alterations or functionality differences that tailor the tools to their specific distribution's base.

Some great examples of tools for building or maintaining functionality in an rpm-ostree image deployment that are available across distributions are:

- **Composer-cli:** A command-line tool for defining the complex contents of a Linux image and a way of rendering the image into multiple formats
- **Osbuild**: A command-line tool for building Linux images
- **Composer** (also known as **Image Builder** in some distributions): A graphical, web-based tool for defining, building, and updating a Linux image
- **Cockpit-composer**: A plugin for the web console on a Linux machine that enables a user to use Composer within the cockpit/web console web UI
- **Toolbox:** An interactive command-line tool that assists in the troubleshooting of an operating system

Let's move on to what tools we can leverage for other methods of image-based deployments.

bootc bootable container image tooling

This is where things get a little interesting, in my opinion. bootc image-based systems don't exactly require a plethora of complex tools to be built and supported. This concept we will dive into deeply in our hands-on exercises within this chapter.

Please allow me to elaborate. As we will see shortly, all you really need on your build machine is a text editor and access to a registry in order to build and update your appliances. We'll be leveraging some container tools as well during the build.

Here's an example list of the tools that you can leverage to create and support the lifecycle of a bootc image-based system:

- Editors (vi, emacs, nano, etc.): Simple text editing programs that are console-based or graphical, yet not as feature-rich as an IDE suite.
- Podman: A container management tool used for building, running, and managing containers on top of Linux
- IDE's (Podman-Desktop, Eclipse, Visual Studio): Various graphical comprehensive development tools
- Image Builder: A graphical, web-based tool for defining, building, and updating a Linux image.
- Cockpit-composer: A plugin for the web console on a Linux machine that enables a user to use Composer within the cockpit/web console web UI.
- Console.redhat.com: A comprehensive service hosted by Red Hat®, that can assist in the building, maintaining, and introspection of systems
- Registries: A registry is crucial for hosting container images and providing a central location to distribute updates, for example, Quay™, Amazon Elastic Container Registry™ (ECR), Harbor, Azure Container Registry (ACR), GitHub Container Registry™, Google Container Registry™ (GCR), JFrog Container Registry™.

Let's move onto a more serious topic, the limitations of images-based deployments today.

Limitations of image-based deployments

While image-based deployments are a paradigm shift in how we can create and maintain solutions, they do have idiosyncrasies that standard package-based deployments do not have. For image-based systems, updates happen at the speed of a reboot. Package-based systems' update operations can take hours and require multiple reboots, all while hoping no dependencies are broken and everything works in the end.

One significant perceived limitation is that filesystems are defined as non-writable. While many see the broader concept of immutability as a feature, if you are trying to install a third-party solution into your image and said solution expects to be installed in a very specific location (filesystem) that is readable and writable, but the actual location is read-only, issues are instantaneous.

This is a perceived limitation, not an actual one. With some creative usage of symbolic links, this most likely can easily be circumvented.

For those not involved in the ecosystem, these deployments can sometimes be miscategorized or perceived as inflexible, or as having limited ability to customize or tailor the solution easily, and often the necessary software may need to be layered onto them into non-standard filesystem locations so that those applications can function in read/write mode. Some of that is a myth, but not all of it. Let's go deeper.

rpm-ostree image limitations

Alright, rpm-ostree is a great solution. It really is. I have to give credit where it's due. I love it. That said, I have to address the perception that it can be inflexible. I propose the statement that in all actuality it's meant to be just that. It is a very secure, immutable image-based deployment. The very tooling itself helps to keep rpm-ostree a very prescriptive type of deployment methodology. rpm-ostree was basically designed to be a secure platform that can easily host containerized apps or virtual machines while retaining a very minimal footprint.

There are some limitations and frustrations that I must bring to light for you. First, kernel customizations are a bit of a challenge as the tooling does not help you in that endeavor (yet). I would say that if you have some customizations for anything in the end-state that you desire, your team should create custom RPM packages for them to be implemented. This also requires that you have a deep understanding of the exact hardware that you plan to leverage as your appliances' baseline. Managing operating-system-level user accounts can be a significant challenge because any changes, layered on such as passwords or group affiliations, that are added on after the deployment can easily be lost as they run the risk of being overwritten by the next image update deployment. In *Chapter 10*, we will cover this in more depth; we'll review the best practice of not having your end-users ever log into the operating system itself on your appliance. I can see how in a corporate environment, this can be seen as a massive issue, but in a secure appliance, I beg the question: is it an issue at all?

Additionally, the places where data is actually in a read/write capability does add additional work in the design and support process. Users' data, application data, and so on – these are seen as some of the complexities of this method of deployment. Planning, testing, and creative usage of symbolic links to locations that are writable will be key concepts that shall help you overcome most of these perceived issues.

I often argue that these facets mentioned are truly not limitations but simply design considerations because if you choose to deploy via rpm-ostree, you are embarking on a different and complex journey with many security rewards. Remember when I stated that security begins at the design table?

Let's move on to how other systems could be perceived as limited.

bootc bootable container image limitations

Here, I feel I must be a bit forgiving as the technology has not even reached version 1.0 status yet. bootc is still evolving, and its potential is astronomical. Additionally, I feel the need to reiterate that the advanced tools to support bootc are also still evolving as well. I might be biased, but I see a massively bright future for bootc images.

For an embedded appliance, having a stable and secure update methodology is a major design point. Today, bootc images are most easily supported by access to the online registry that created their existence. That's all good and well if your solution will have access to the internet once the customer has implemented it in their own environment. Today, for an embedded Linux system appliance based upon bootc, being *online* is best.

One of the perceived limitations I see today isn't exactly a limitation, but a serious design consideration that has to be addressed at the design table. Today, an offline bootc image-based deployment is possible. It's just a little bit more complex. If your product is to be deployed in an **air-gapped environment** or non-networked environment, one must build into the appliance's interface a way of ingesting its updates. This actually means a way of importing locally a new container image to use as the new baseline for its updates and rebuilding.

Again, as with rpm-ostree, I recommend obscuring the operating system from the end-user by only allowing them application-based access and keeping the operating system accounts limited. This will address the similar difficulty of maintaining operating systems' user accounts through each update/image rerendering. Yes, managing or controlling operating-system-level users is a concern in bootc images too. I suspect that there shall be additional tooling developed for this as bootc evolves.

In comparison with rpm-ostree, bootc exceeds flexibility and customization. Kernel modifications are easier to achieve. Why? Containers are incredibly flexible themselves and they're easier to maintain.

My professional opinion is that this is a technology worth keeping a very close eye on because it can transform and simplify how you build appliances. I won't say that it's without pain points, but I do have the opinion that it has a very bright future. Let's move on to how we can update or, in the worst cases, roll back.

Updating and rolling back changes

Package-based systems, regardless of the package system (RPM, DNF, APT, Zypper, etc.) can have issues and artifacts when either moving to a newer version of their operating system or during an attempt to roll back installations that produce unintended consequences. If you have ever experienced this, you know exactly what I mean and I offer my condolences. For those of you who have been blessed or simply lucky enough to not have suffered a failed package update process, here's the difference.

Package-based systems are forced to do their dependency checks in real time when they process their updates. What this process amounts to is some packages may be added or deprecated that you might not even be aware of during the process. Not only can it be clunky and obtuse, but it consumes significant time. Hopefully, you tested this process for each release *ad nauseum* before presenting updates to your customers. Sometimes, oftentimes, multiple reboots are required. I feel for anyone enduring this. This is downtime. Hopefully, it just works the first time, because if it doesn't, roll backs often require a recovery from a backup or a snapshot, Hahahahaha – if that even exists.

Image-based operating systems do not fall victim to those issues. Whether providing an incremental update or a completely new operating system, this delivery method allows for a rapid in-place upgrade while not impacting user or application data. How? These systems stage their new image and then simply reboot into it. Fast, clean, efficient. More importantly, since you and your team have tested these images, there's little worrying about whether or not this new image has issues.

So, let's assume the worst case: the image had some corruption in downloading. *No problem*. We can recover from the previous image with little to no effort. That's some amazing upside, right?! Let's move on to how these systems also can be upgraded in place without stress.

Upgrade of operating system version in place

This is yet another area where all image-based deployments shine: upgrades in place.

For atomic/rpm-ostree systems, all we need to do is create a new image based on a newer operating system and get that image staged in the appliance's update repository. The appliance can download/stage the new image and will be upgraded to the new version of the operating system upon reboot.

For bootc-based systems, all we need to do is base our container image upon a newer operating system base image, rebuild the container, and upload the container to the registry. When the system checks for updates, it will pull the new container and rerender itself as an upgraded system.

Like I've said, it's almost like these things are tailor-made for embedded Linux systems, right? Let's move on to our hands-on exercises now and build one for ourselves.

Practical exercises

This is the time when we get to roll up our sleeves and bash the keyboard. If you follow the exercises in the order presented, at the end you will have a bootc image-based deployment ecosystem set up for yourself.

Before we begin, I want to share with you some useful information. The tools for this type of infrastructure are evolving rapidly, and there is already more than one way to design, build, and deploy these images. I see this as an amazing opportunity for the Linux community. The method I am demonstrating here is simple and easy to either script or automate via something such as Ansible. Throughout the exercises, I will mention other ways or alternative tools that could enhance the developer experience and make things scale better.

As these exercises require precision in the configuration files that we will be creating together, I urge extreme caution should you choose to leverage a method of cutting and pasting to rapidly create the files. It is often too easy to copy over hidden ASCII characters (or formatting characters) that will render your working configuration files utterly useless and virtually impossible to debug because when you enter your text editor you may only see a blank space (if anything). I have added templated files that you can download and tailor yourself to your own lab's environment. They can be found in the book's GitHub repository (https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook/tree/main/Chapter09/exercises), and they will also enlighten you about other options that we have not covered.

So, grab a formattable 16 GB USB thumb drive along with your favorite beverage (maybe several), and let's get started.

Exercise 1 – preparing the environment

In this first exercise, we will install the necessary tools along with some optional tools to create our minimal bootable container image build chain. Let's do that using the following steps:

1. First, we will set up a build environment, configure our registry, and create a container that will become the basis for not just our application but our operating system as well:

```
$ sudo dnf install -y containers-common crun \
iptables netavark nftables slirp4netns \
composer-cli cockpit cockpit-composer \
skopeo buildah runc podman
```

2. Now we'll ensure that the web console has been enabled:

```
$ sudo systemctl enable cockpit.socket
```

3. And here, we'll start the socket for the web console:

```
$ sudo systemctl start cockpit.socket
```

4. Confirm the web console is active:

```
$ sudo systemctl status cockpit.socket
```

Your output should resemble something like this:

```
● cockpit.socket - Cockpit Web Service Socket
     Loaded: loaded (/usr/lib/systemd/system/cockpit.socket; enabled; preset: disabled)
     Active: active (listening) since Fri 2024-08-09 12:39:24 EDT; 1 month 13 days ago
   Triggers: ● cockpit.service
       Docs: man:cockpit-ws(8)
     Listen: [::]:9090 (Stream)
      Tasks: 0 (limit: 38320)
     Memory: 648.0K (peak: 2.5M)
        CPU: 26ms
     CGroup: /system.slice/cockpit.socket
Aug 09 12:39:24 bm03.local systemd[1]: Starting cockpit.socket - Cockpit Web Service Socket...
Aug 09 12:39:24 bm03.local systemd[1]: Listening on cockpit.socket - Cockpit Web Service Socket.
```

5. Next, we'll take a look at our free registry and make some configuration changes:

   I. In a browser window, log into https://quay.io/.

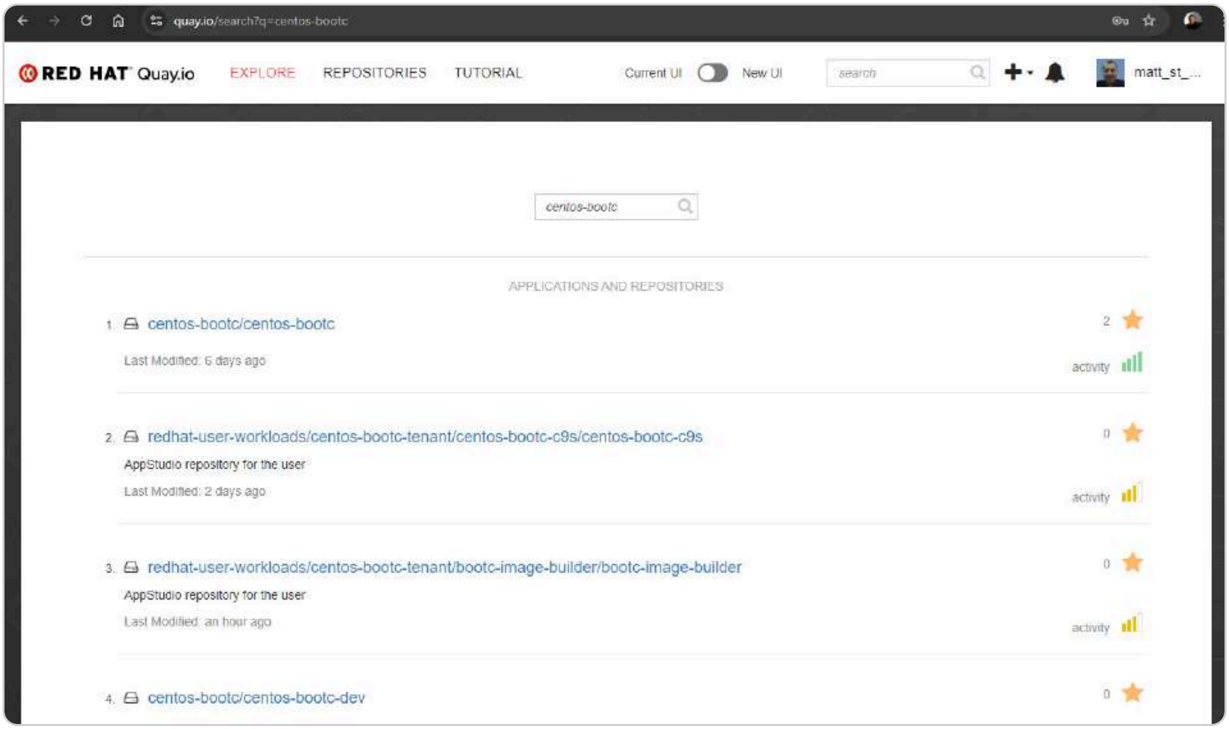
   II. Navigate to **Account** >> **Settings** >> **CLI Password**.

   III. Set a CLI password if you have not already done so (you may be asked to create an encrypted password – these are better). Make a note of this information; we will need it very soon.

   IV. Search for the `centos-bootc` repository.

   V. Click on the link for **centos-bootc/centos-bootc**.

   VI. Make note of the URL as we'll be using it soon. Consider bookmarking the page too.



1. centos-bootc/centos-bootc — Last Modified: 6 days ago — 2 — activity
2. redhat-user-workloads/centos-bootc-tenant/centos-bootc-c9s/centos-bootc-c9s — AppStudio repository for the user — Last Modified: 2 days ago — 0 — activity
3. redhat-user-workloads/centos-bootc-tenant/bootc-image-builder/bootc-image-builder — AppStudio repository for the user — Last Modified: an hour ago — 0 — activity
4. centos-bootc/centos-bootc-dev — 0

Figure 9.1 – Searching for CentOS Stream's bootc base image

6. Back in your terminal, configure your non-root user account to be able to search the registry:

```
$ cd ~
```

7. Create the requisite directories to store your container configuration so Podman will know where to seek for information:

```
$ mkdir -p ~/.config/containers
```

8. Let's change directories to the one we just created:

```
$ cd ~/.config/containers
```

9. Let's now create a **registries.conf** file to contain the defined contents:

```
$ vi registries.conf
```

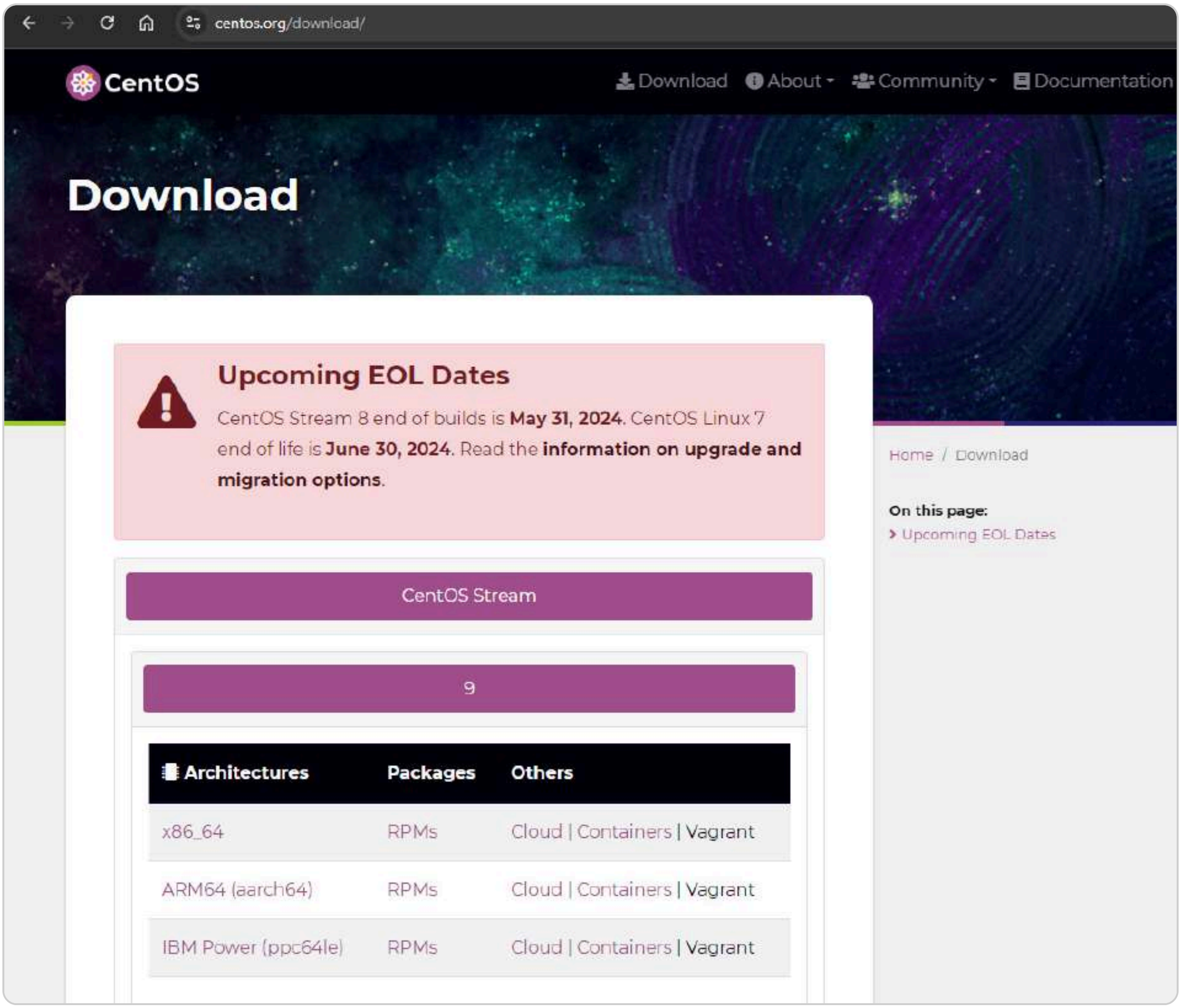
   Set the file's contents to match the following and save the file:

```
# we will use these registries only
[registries.search]
registries = ['registry.redhat.io','quay.io']
```

10. Now, we'll go back to our home directory:

```
$ cd ~
```

11. Let's download the CentOS Stream 9 DVD ISO image to be used later in the process when we create our custom installer. Use your web browser to go to https://centos.org/download/ and then click on the **x86_64** button.



centos.org/download/

CentOS

Download | About | Community | Documentation

Download

Upcoming EOL Dates

CentOS Stream 8 end of builds is **May 31, 2024**. CentOS Linux 7 end of life is **June 30, 2024**. Read the **information on upgrade and migration options**.

CentOS Stream

9

Architectures	Packages	Others
x86_64	RPMs	Cloud | Containers | Vagrant
ARM64 (aarch64)	RPMs	Cloud | Containers | Vagrant
IBM Power (ppc64le)	RPMs	Cloud | Containers | Vagrant

Home / Download

On this page:

› Upcoming EOL Dates

Figure 9.2 – CentOS Stream 9 download

12. Let's verify the file size and that it's fully downloaded:

```
$ ls -lh | grep *.iso
```

Your output should resemble something like this, but the files are owned by your user account:

```
-rw-r--r--. 1 mstonge mstonge  11G Sep 25 01:01 CentOS-Stream-9-latest-x86_64-
dvd1.iso
```

13. Back in your terminal, log in to `quay.io` via the command line. Use your own user account and the password you have previously set in order to access the `quay.io` registry:

```
$ podman login quay.io
```

If the login was successful, you will receive the following message:

```
Login Succeeded!
```

IMPORTANT NOTE

If you opted to generate the encrypted CLI password, the **`login`** *command should look a little different. Replace* **`[your username]`** *with your own username and* **`[encrypted pass]`** *with your long, encrypted password here:*

`$ podman login -u='[your username]' -p='[encrypted pass]' quay.io`

14. Let's pull down our base container image:

```
$ podman pull quay.io/centos-bootc/centos-bootc:stream9
```

The output for this one is astronomically long (again). Here, we've truncated the output to show you generally what you can expect to see:

```
((( output truncated)))
Copying blob 775d29f76a39 done   |
Copying blob 7eff373befa3 done   |
Copying blob 8c789e616763 done   |
Copying blob fd730fb4a24b done   |
Copying blob 54246c915569 done   |
Copying blob 232fb94490b0 done   |
Copying blob ad312c5c40cc done   |
Copying blob bd9ddc54bea9 done   |
Copying config a1163a9d15 done   |
Writing manifest to image destination
a1163a9d15d2f9a3f7f81748baf8fbcfc69690ed38030e770fe2006c090b0f83
```

15. Let's check our local container inventory and verify that we have the intended CentOS Stream 9 bootc image in our inventory:

```
$ podman images
```

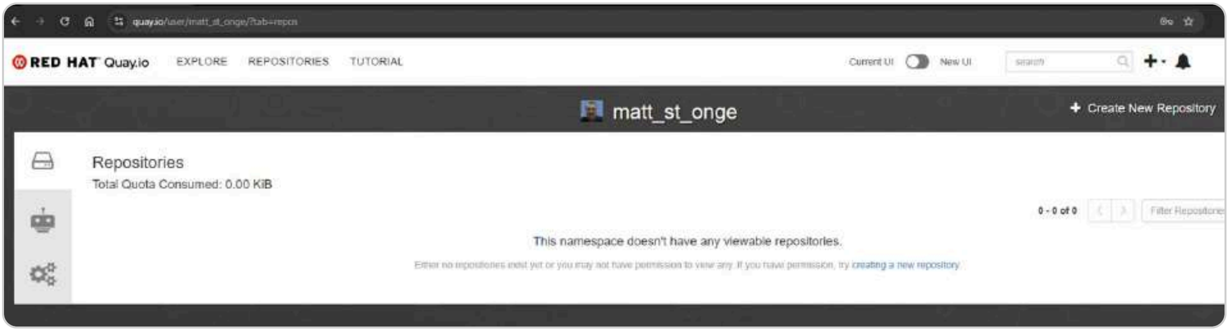
Your output should resemble this:

```
REPOSITORY                         TAG        IMAGE ID      CREATED        SIZE
quay.io/centos-bootc/centos-bootc  stream9    a1163a9d15d2  43 hours ago   1.52 GB
```

16. Now, in the Quay.io web interface, we will create our own public repository for our container image project. We'll start on the **Repositories** tab.

IMPORTANT NOTE

As a simple reminder, this is not exactly how one should ever do this in production. We'll be using a public configuration for lab purposes only. Private, secure registries are the only way I would recommend doing this deployment method while being connected to the internet.



Repositories
Total Quota Consumed: 0.00 KiB

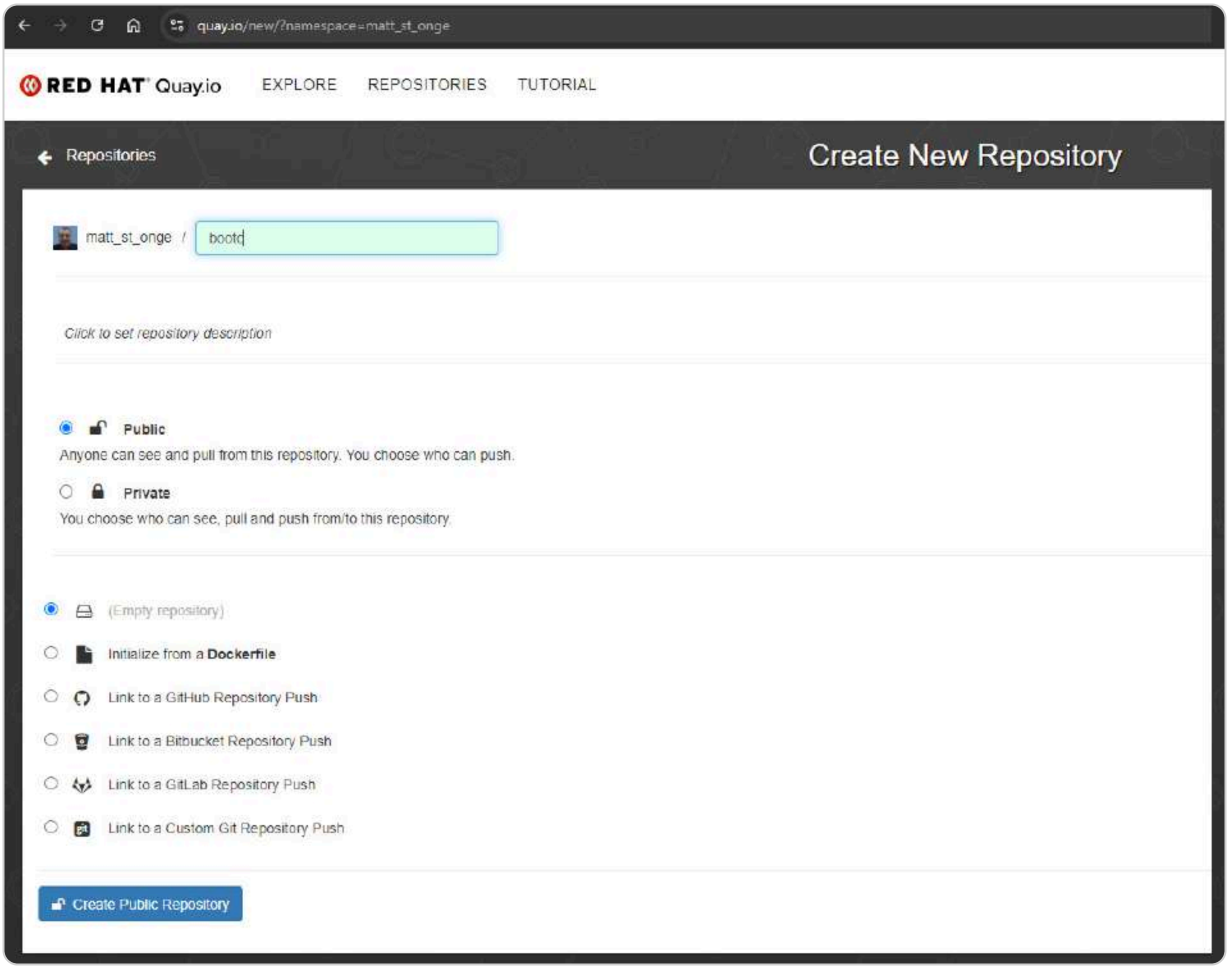
This namespace doesn't have any viewable repositories.

Either no repositories exist yet or you may not have permission to view any. If you have permission, try creating a new repository.

Figure 9.3 – Quay – Repositories main page

17. Next, we'll click on the **+ Create New Repository** button in the top-right-hand corner of the screen.

    Name your repository `bootc` and ensure that the **Public** radio button is selected along with the **(Empty repository)** radio button. Then, click the **Create Public Repository** button at the bottom.



matt_st_onge / bootc

Click to set repository description

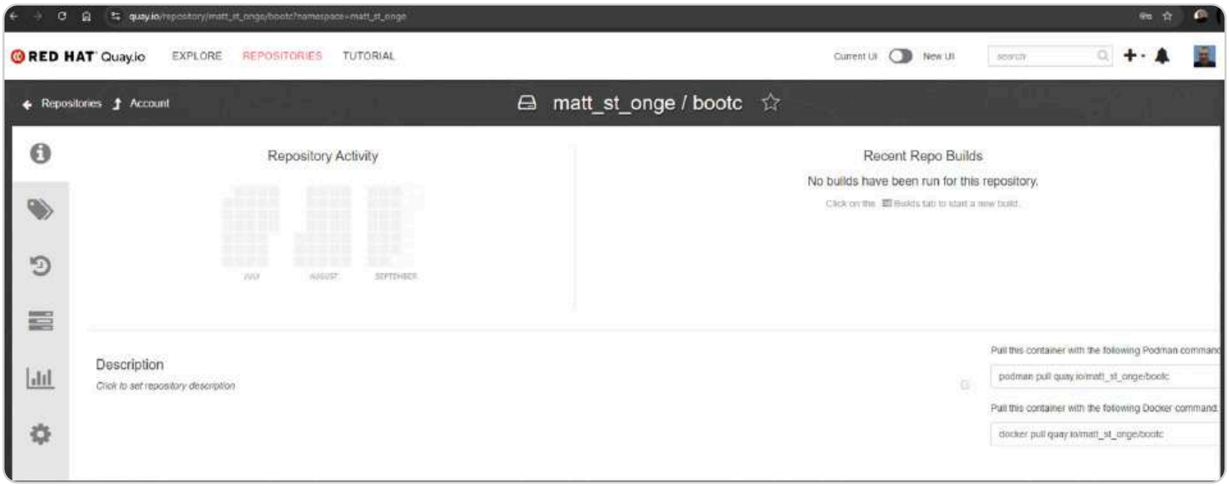
(● Public) Public
Anyone can see and pull from this repository. You choose who can push.

○ Private
You choose who can see, pull and push from/to this repository.

[✓ (Empty repository)]
○ Initialize from a **Dockerfile**
○ Link to a GitHub Repository Push
○ Link to a Bitbucket Repository Push
○ Link to a GitLab Repository Push
○ Link to a Custom Git Repository Push

Figure 9.4 – Creating a new repository in Quay

Make note of the URL for your repository; we'll be using it soon. I also recommend bookmarking it in your web browser.



Pull this container with the following Podman command

```
podman pull quay.io/matt_st_onge/bootc
```

Pull this container with the following Docker command:

```
docker pull quay.io/matt_st_onge/bootc
```

Figure 9.5 – Your custom Quay repository

Now, we have configured a baseline build environment on your system. By configuring how we leverage registries, installed container tools, and staged a container base image and a Linux installer ISO image, we have all we need to be successful. We additionally set up the web console on our build system, which will come in handy later. Let's move on to building our initial container, which will be the basis for our future immutable image.

Exercise 2 – creating a container file

In this exercise, we'll create a container file that we will build using the base image we downloaded in the previous exercise. The results of this exercise will give us a container that can run a simple **Linux Apache mySQL and PHP** (**LAMP**) stack. If you decide not to type this file, there will be a prebuilt one in the book's GitHub repository for reference.

Let's build our container:

1. Let's create the container file and name it **`mycontainerfile.cf`** :

   ```
   $ vi mycontainerfile.cf
   ```

   Set the contents of the file to look like the following:

   ```
   FROM quay.io/centos-bootc/centos-bootc:stream9
   #install the lamp components
   RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf clean all
   #start the services automatically on boot
   RUN systemctl enable httpd mariadb php-fpm
   #create an awe inspiring home page (all one command line)
   RUN echo '<h1 style="text-align:center;">Welcome to My Appliance</h1> <?php
   phpinfo(); ?>' >> /var/www/html/index.php
   ```

2. Let's build our container image:

IMPORTANT NOTE

Replace your own Quay.io username where ***`[my_account]`*** *appears in the command line.*

```
$ podman build -f mycontainerfile.cf -t quay.io/[my_account]/bootc/lamp-bootc:latest
```

The output for this one is really long too. We've truncated the output somewhat to save space while still showing you what you can expect to see:

```
STEP 1/4: FROM quay.io/centos-bootc/centos-bootc:stream9
STEP 2/4: RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf
clean all
--> Using cache a525b1bb126820c8522199f6d42b292210f06e4d178efbc148d97a92b94a64ed
--> a525b1bb1268
STEP 3/4: RUN systemctl enable httpd mariadb php-fpm
--> Using cache 9400a8bbc0287454ae0db9f42f9b49e518daf2b410fd3c3d0bb91a8b58e0a2a3
--> 9400a8bbc028
STEP 4/4: RUN echo '<h1 style="text-align:center;">Welcome to My Appliance</h1> <?php
phpinfo(); ?>' >> /var/www/html/index.php
--> Using cache 4bcb220e3de6429f9f83264e84f064f2101c715c78e1104e388d11f6007b560e
COMMIT quay.io/matt_st_onge/bootc/lamp-bootc:latest
--> 4bcb220e3de6
Successfully tagged quay.io/matt_st_onge/bootc/lamp-bootc:latest
4bcb220e3de6429f9f83264e84f064f2101c715c78e1104e388d11f6007b560e
```

Great! We now have our container image. Let's do a quick test to see how well it works:

IMPORTANT NOTE

Replace your own Quay.io username where ***`[my_account]`*** *appears in the command line.*

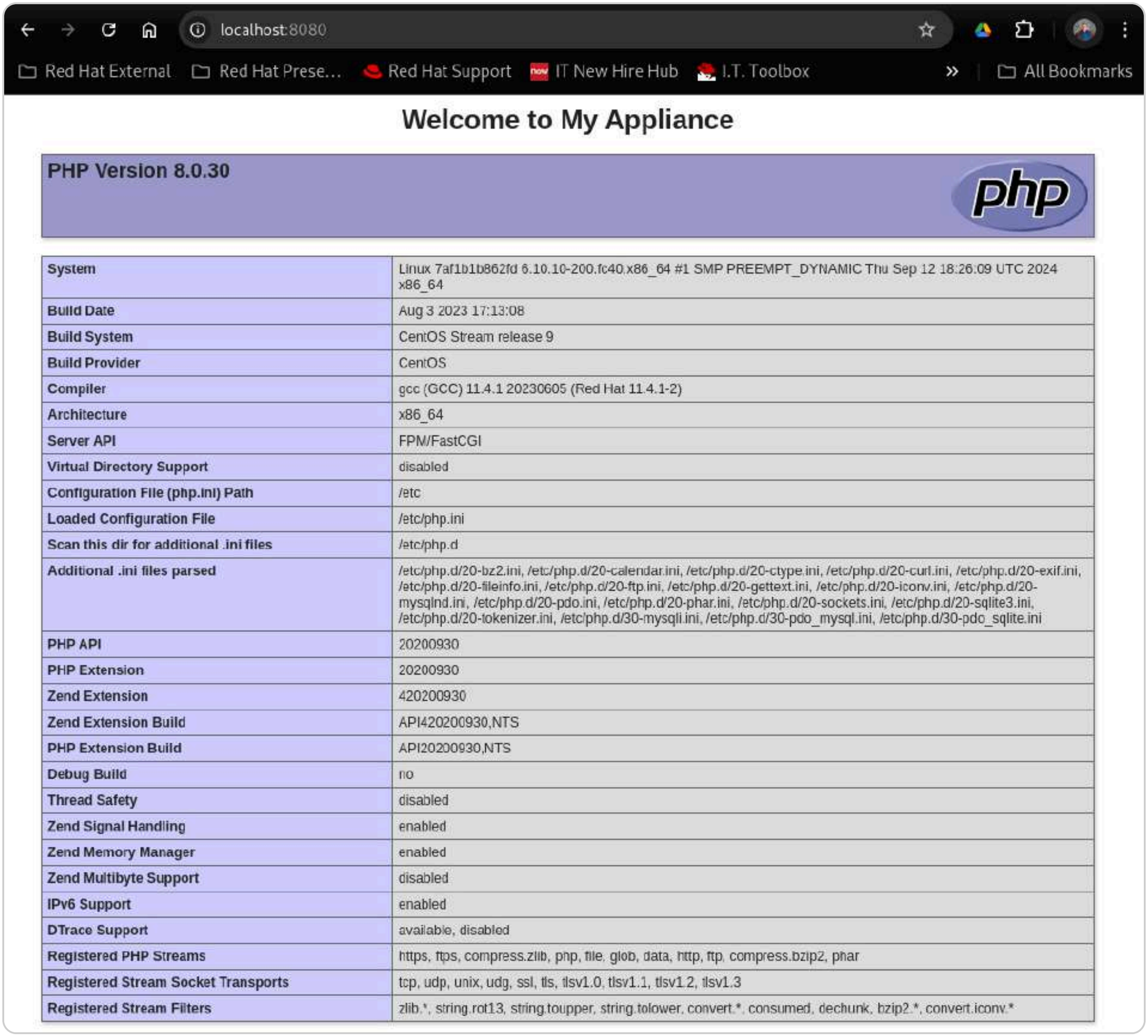
```
$ podman run -d --rm --name lamp -p 8080:80 \
quay.io/[my_account]/bootc/lamp-bootc:latest
```

Your output will resemble something like this:

```
7d9c474d9dd4e6ab32d910c72775cdb111adfed764f29887c110461ca67c54a6
```

3. With the container started, let's open a browser window and verify that you can view the served content: http://[your_ip_address]:8080.

   If the page doesn't load, double-check your firewall settings. If you are on the same system where you are running the container, your loopback address should also work.



Welcome to My Appliance

PHP Version 8.0.30

System	Linux 7af1b1b862fd 6.10.10-200.fc40.x86_64 #1 SMP PREEMPT_DYNAMIC Thu Sep 12 18:26:09 UTC 2024 x86_64
Build Date	Aug 3 2023 17:13:08
Build System	CentOS Stream release 9
Build Provider	CentOS
Compiler	gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
Architecture	x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

Figure 9.6 – Testing our container

4. We should now also be able to shell into the container while it's running:

   ```
   $ podman exec -it lamp /bin/bash
   ```

5. When given the prompt feel free to test some commands but remember to exit:

   ```
   bash-5.1# exit
   ```

   You will then be returned to your regular shell prompt on your system.

6. Stop the running container since we know that the image works:

   ```
   $ podman stop lamp
   ```

   The output from this command is rather terse, it will simply reply with the name of the container you asked it to stop. In our case, it will just reply `lamp`.

7. Our final step will be saving our functional container image to your own repository within Quay.io:

> **IMPORTANT NOTE**
>
> *Log back into Quay.io via the command line if you are not logged in already.*
> *Replace your own Quay.io username where* **`[my_account]`** *appears in the command line.*

```
$ podman push quay.io/[my_account]/bootc/lamp-bootc:latest
```

The output for this operation will be rather lengthy, so I'll only show you the last few lines:

```
(((output truncated)))
Copying blob 7685af3680f8 skipped: already exists
Copying blob 9046686a9227 skipped: already exists
Copying blob d1c1676ee4e9 skipped: already exists
Copying blob 7a1c4a9ce068 skipped: already exists
Copying blob 0811ec9b544a done   |
Copying blob abef090ec865 done   |
Copying blob 6394663daed5 done   |
Copying blob 2daf40f13a19 skipped: already exists
Copying blob 9dad063a624b skipped: already exists
Copying config 8a4585ebc8 done   |
Writing manifest to image destination
```

Excellent! You've created the basis of your future operating system via the base image and layered the applications stack, all by creating a working container image. In our next exercise, we'll create an installer so we can deploy it as a bootable image.

Exercise 3 – creating an installer

In this exercise, we will create a kickstart file and then take that kickstart along with a standard vendor-provided ISO install image and create our own custom ISO installer image for our amazing new system. This method of installation is great when you're working in your lab or data center. Alternative methods will be necessary if you are deploying in a cloud services provider. For a great reference on how to build kickstart files, you can check out this guide: https://docs.fedoraproject.org/en-US/fedora/f36/install-guide/appendixes/Kickstart_Syntax_Reference/.

Let's move on to the first step of this exercise:

1. In this first step, you will create a kickstart file (**`mykickstart.ks`**). Within this file, you will substitute your own account username, where I state **`[you]`**, and you'll also be setting basic configuration for the operating system's filesystem layout and root password. Save and exit the file when you are done. Should you choose not to type the file in its entirety, there's an example in the book's GitHub repository:

IMPORTANT NOTE

Replace your own Quay.io username where `[my_account]` appears in the file's contents.

```
$ vi mykickstart.ks
```

Ensure the contents of `mykickstart.ks` look similar (with your substitutions) to this:

```
# mykickstart.ks
# version 1
# anaconda installer type
text
# ensure that you connect your device to a Ethernet network with active DHCP
network --bootproto=dhcp --device=link --activate
# basic partitioning
clearpart --all --initlabel --disklabel=gpt
reqpart --add-boot
part / --grow --fstype xfs
# here's where we reference the container image
# notice this kickstart has no packages section
ostreecontainer --url quay.io/ [quay_username]/bootc/lamp-bootc:latest --no-signature-
verification
# additional settings for demonstration purposes
# in production use better settings
# the purpose of this exercise is not to tech you kickstart
# but to show how to leverage it in custom installers
firewall --disabled
services --enabled=sshd
# add your own user account to the system
user --name=mstonge --groups=wheel --plaintext --password=embedded
# set root password
rootpw --plaintext embedded
```

2. Now, we will install the `lorax` software package, which will enable us to create a custom installer ISO image:

   ```
   $ sudo dnf install -y lorax
   ```

   The output is rather lengthy; hence, I've only shown the end, which you should pay attention to... it should say `Complete`:

   ```
   ((( output truncated)))
   Complete!
   ```

3. Now, we'll utilize the `mkksiso` command, which is part of the `lorax` RPM package we just installed. This will create a custom installer for us:

IMPORTANT NOTE

You will need to substitute exact paths of your own here.
For example,

```
sudo mkksiso --ks /home/mstonge/mykickstart.ks \
/home/mstonge/CentOS-Stream-9-latest-x86_64-dvd1.iso \
/home/mstonge/mycustominstaller.iso
```

```
$ sudo mkksiso –ks [absolute path to mykickstart.ks] \
```

```
[absolute path to the CentOS Stream 9 ISO] \
[absolute path to the new ISO you want created]
```

The output for this step is significantly long. I have truncated it here to prevent my editor from murdering me. The part you must pay attention to is the last line, which confirms the operation has completed successfully and that it wrote your custom ISO image:

```
     ((( output truncated)))
xorriso : UPDATE : Writing:     830548s   74.1%   fifo   0%  buf  50%  137.6xD
xorriso : UPDATE : Writing:     932628s   83.2%   fifo   0%  buf  50%  150.7xD
xorriso : UPDATE : Writing:    1007616s   89.9%   fifo  29%  buf  50%  110.7xD
xorriso : UPDATE : Writing:    1097728s   97.9%   fifo  10%  buf  50%  133.0xD
ISO image produced: 1120832 sectors
Written to medium : 1121008 sectors at LBA 48
Writing to '/home/mstonge/mycustominstaller.iso' completed successfully.
```

4. Now that we have our own custom installer ISO image, let's create boot media. For this step, you will need to use Fedora Media Writer. It should already be on your system; if not, download it first. If you need assistance downloading or installing the tool (which works on all major platforms), check out this reference link: https://docs.fedoraproject.org/en-US/fedora/latest/preparing-boot-media/#_fedora_media_writer.

IMPORTANT NOTE

This step may be optional if you are working with virtual machines – you might be able just to boot from the ISO file itself within the hypervisor.

Let's look at how Fedora Media Writer can simplify the creation of boot media.

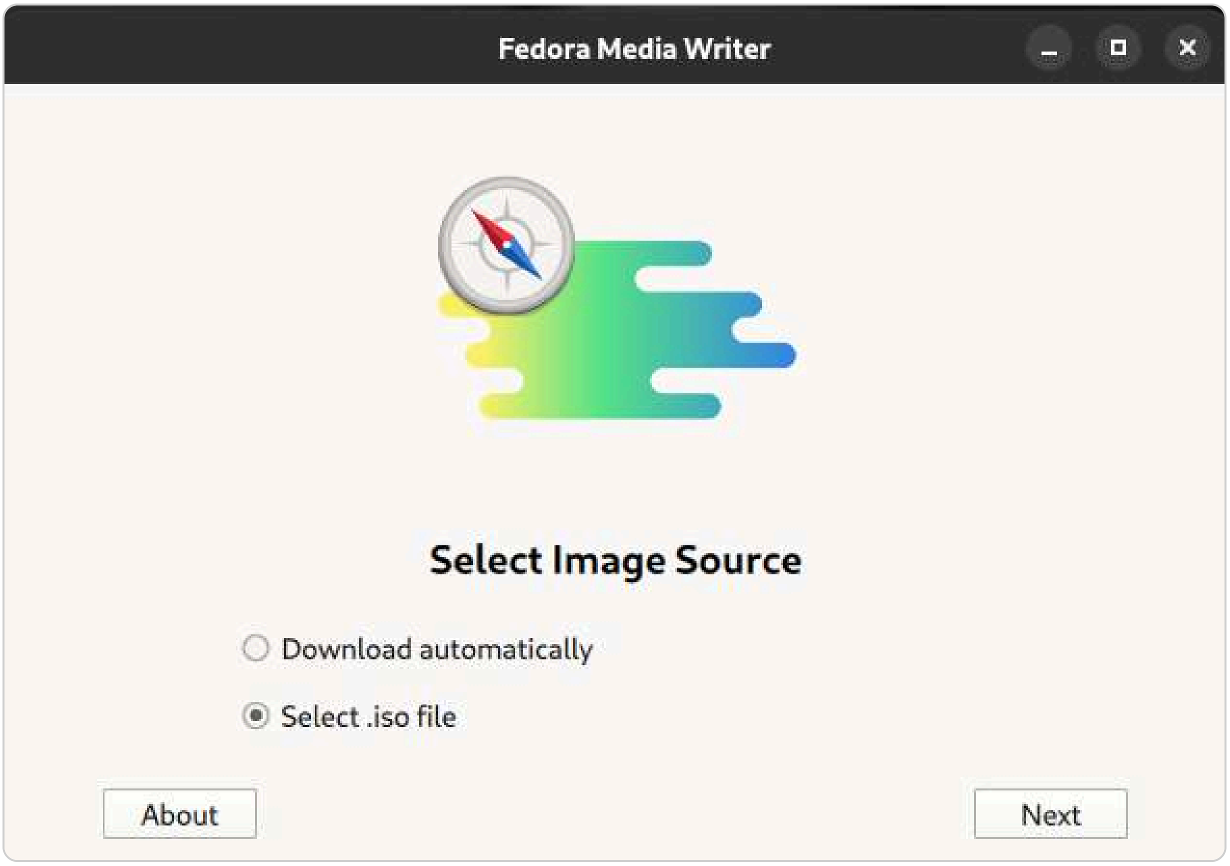
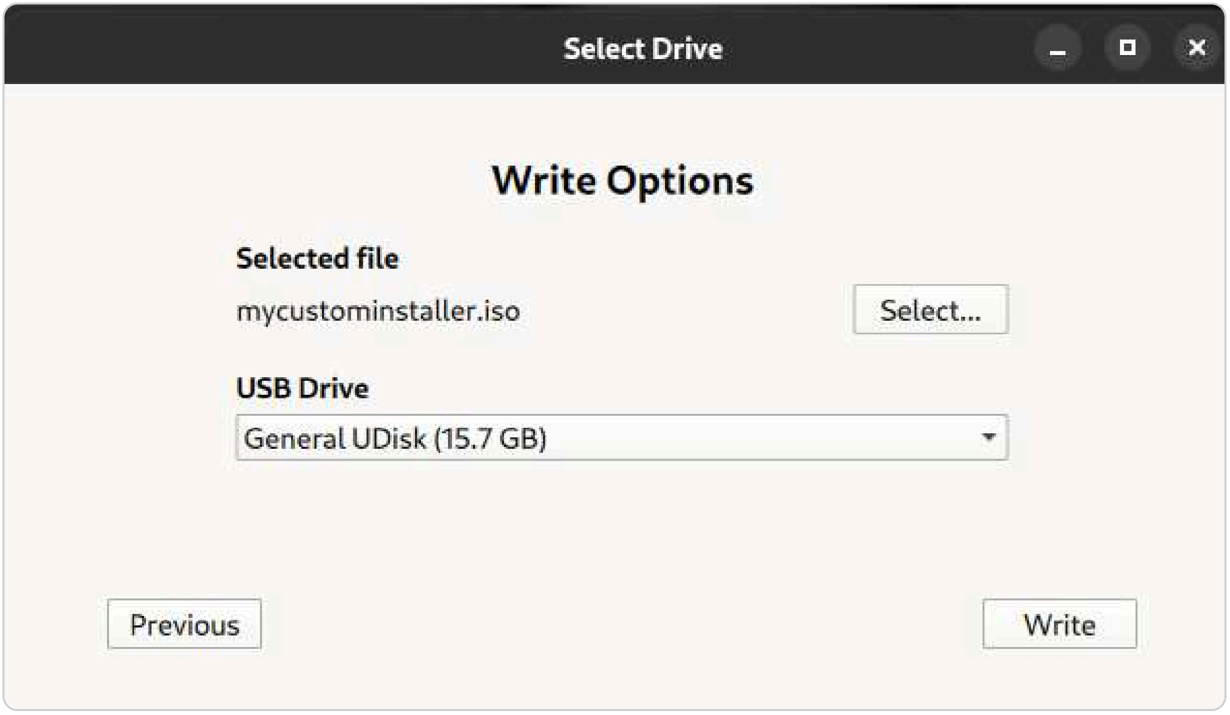


Figure 9.7 – Fedora Media Writer

5. Here, you select the ISO image and the USB thumb drive that you want to commit the bootable image to.

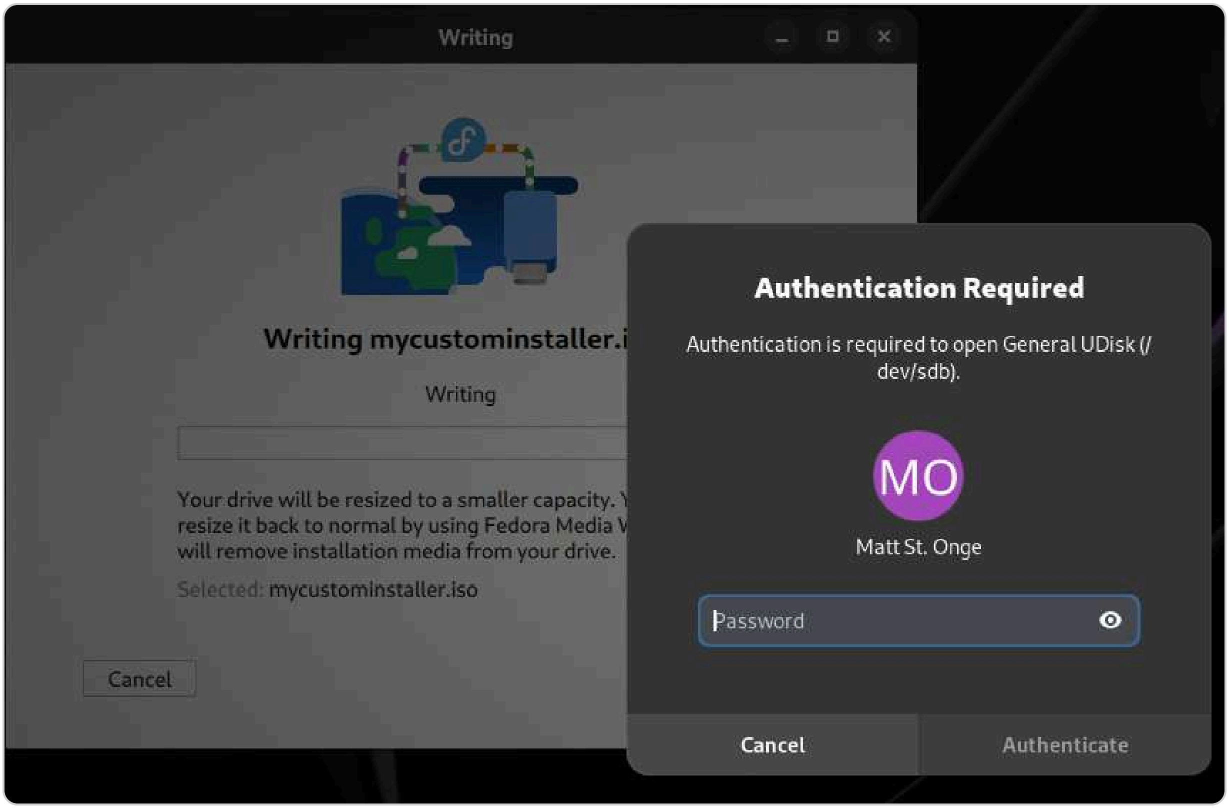


Selected file: mycustominstaller.iso

USB Drive: General UDisk (15.7 GB)

Figure 9.8 – Choosing ISO images

As the Fedora Media Writer requires elevated access, you'll be prompted for authentication to achieve `sudo` status.



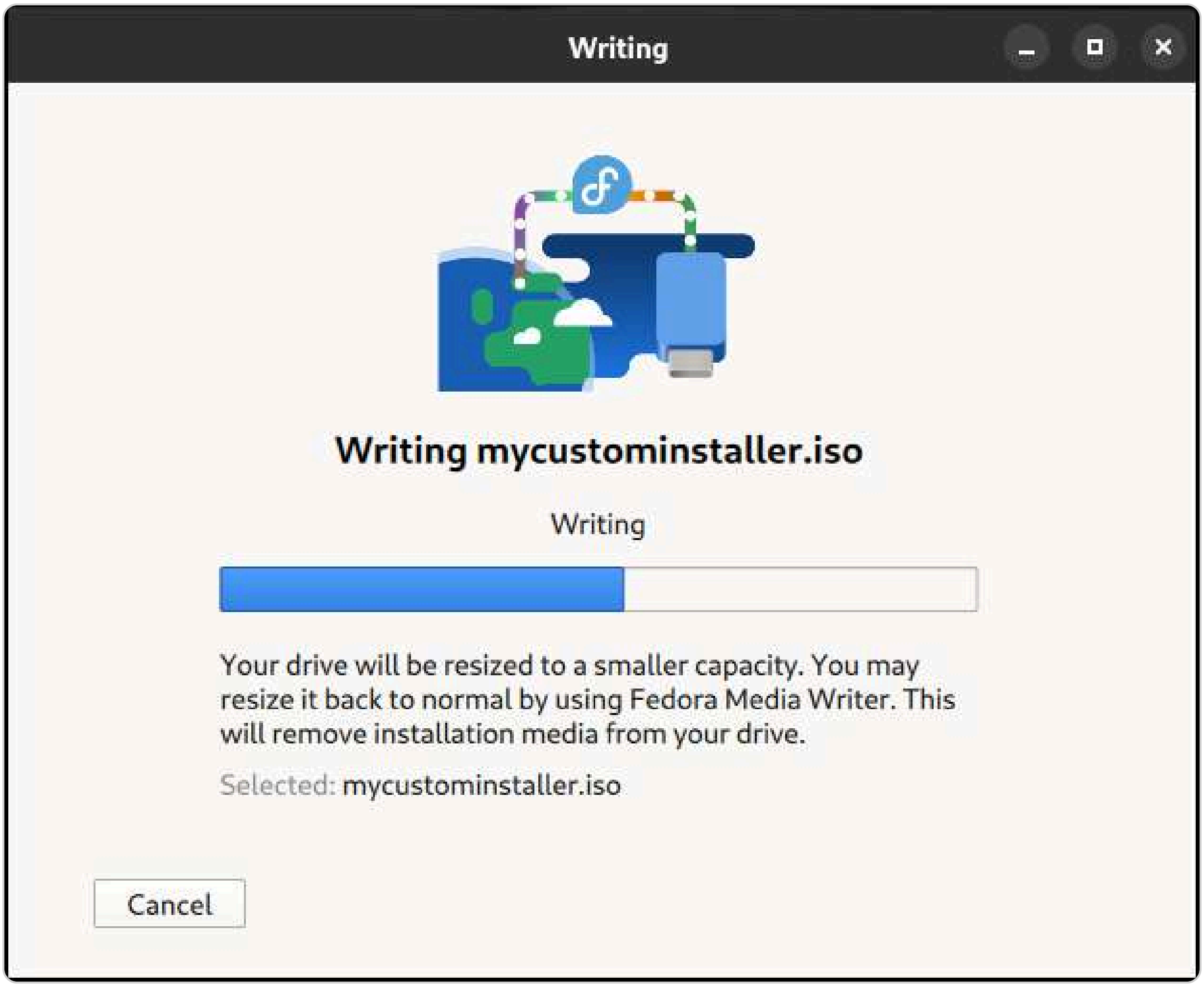
Your drive will be resized to a smaller capacity. Y
resize it back to normal by using Fedora Media V
will remove installation media from your drive.

Selected: mycustominstaller.iso

Authentication is required to open General UDisk (/dev/sdb).

Figure 9.9 – Elevated permissions – authentication

It will definitely take a few minutes to render the ISO image to the physical media. Have patience, grab a beverage, and enjoy the break.



Writing

Writing mycustominstaller.iso

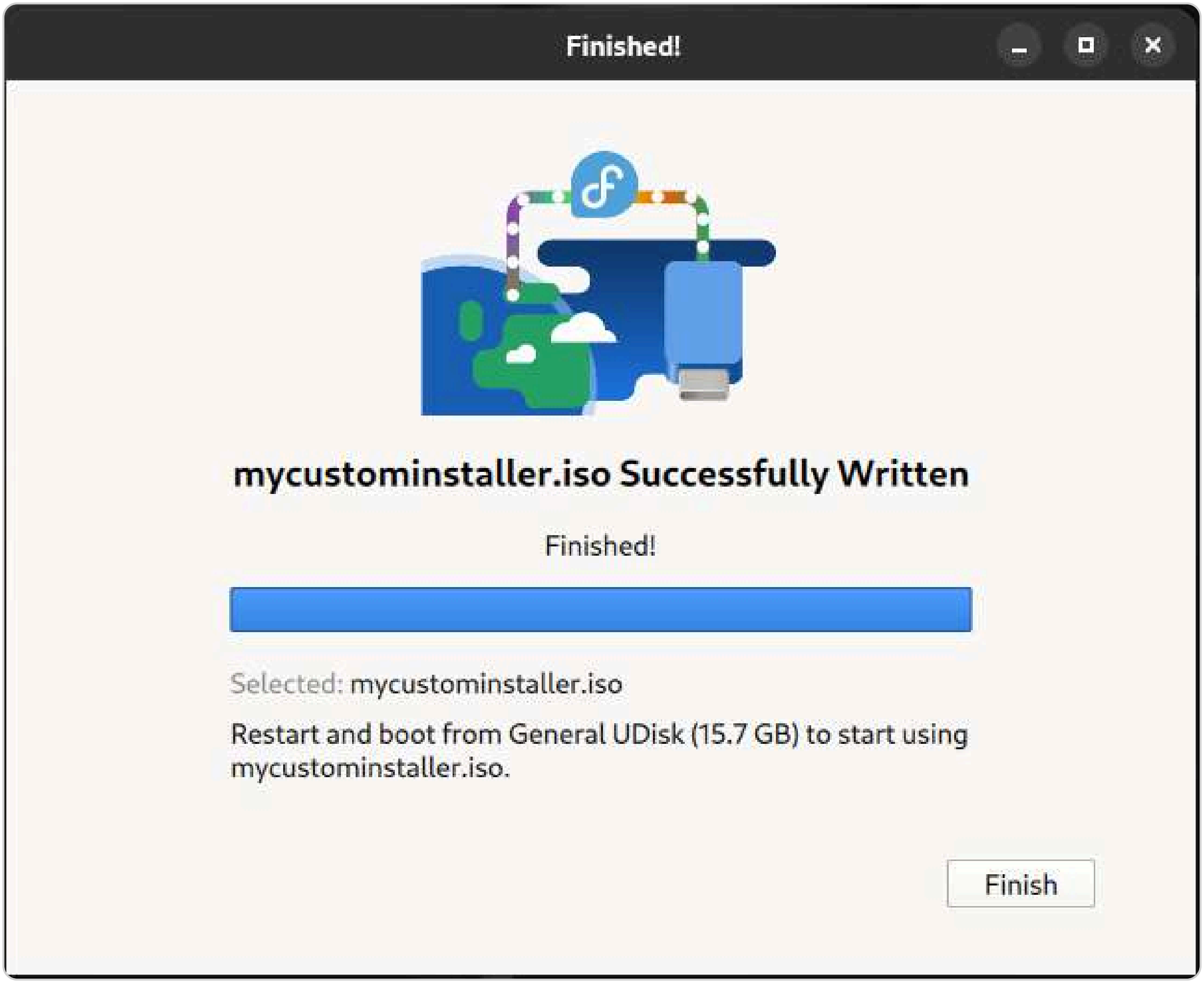
Writing

Your drive will be resized to a smaller capacity. You may resize it back to normal by using Fedora Media Writer. This will remove installation media from your drive.

Selected: mycustominstaller.iso

Figure 9.10 - ISO build in progress

Once the ISO build is completed, you'll be greeted by this screen.



mycustominstaller.iso Successfully Written

Finished!

Selected: mycustominstaller.iso

Restart and boot from General UDisk (15.7 GB) to start using mycustominstaller.iso.

Figured 9.11 – ISO image created

You can now remove the thumb drive from the USB port. We're just moments away from installation. You've successfully created your own custom installer. Let's move on and put it to good use.

Exercise 4 – initial installation

In this exercise, we will install our first **Image Mode system** with our newly created custom installer. You will boot the test system from the newly created thumb drive (or from the ISO file we just created in the case of a virtual machine). You may need to interrupt your system's normal boot process to get it to boot from the USB thumb drive.

Sit back, relax, and watch the magic happen. Let's move on to deploying our first system leveraging the automated installer that we just created:

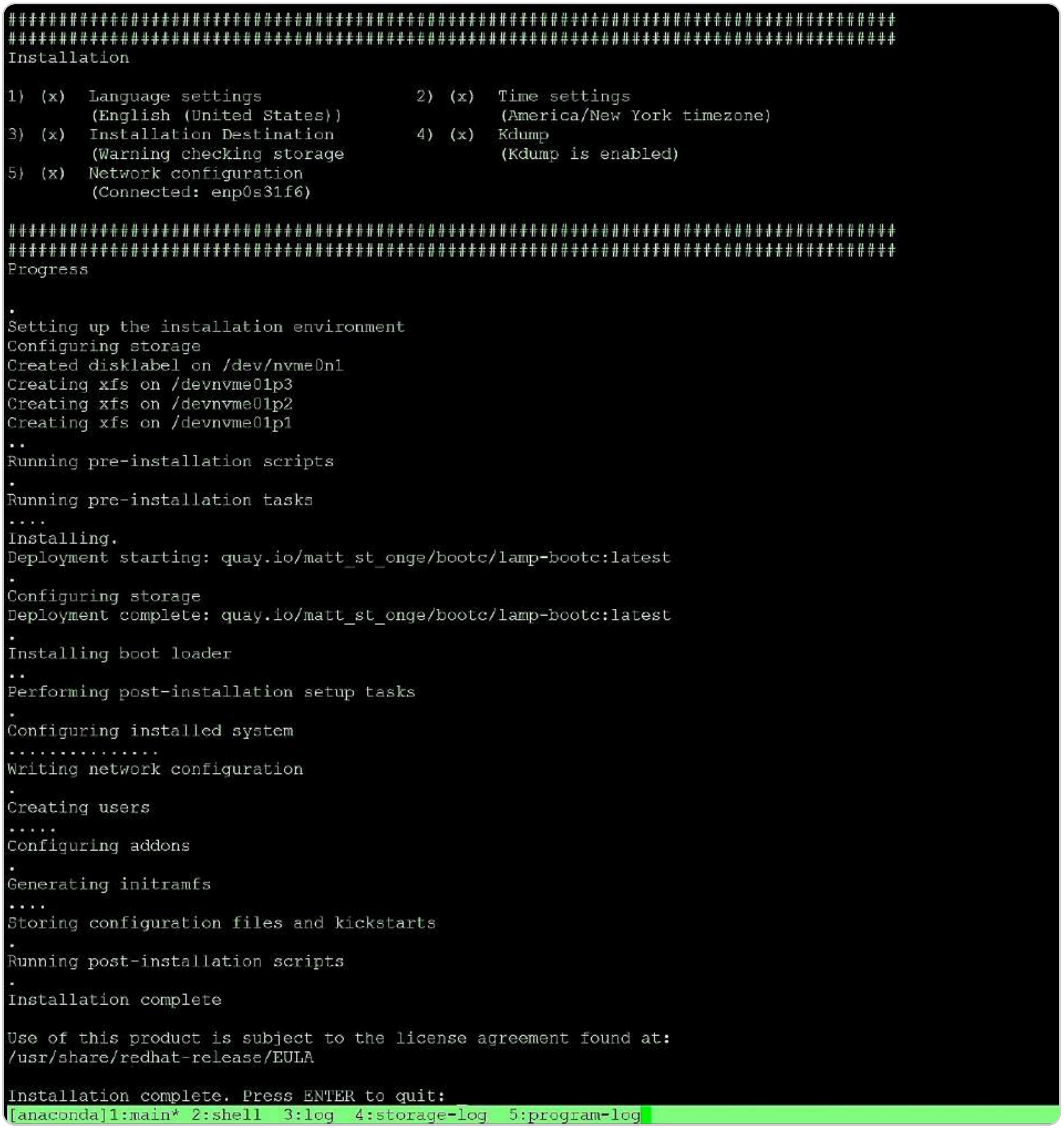
1. With your newly created boot media, use it to boot (or create a new virtual machine). If you are booting onto physical hardware, you need to be aware of some things before your installation:

I. Ensure all previous partitions are removed from the drive (especially the **UEFI** partition) before the installation process begins.

II. Ensure that, within your **UEFI BIOS**, any previous entries for **Secure Boot** are removed (**RESET**) and that **Secure Boot** is set to **DISABLED** before the installation.

III. Boot your system from the USB media (physical hardware) or directly from the ISO image (virtual machine). This is an automated install and it will notify you upon completion (or failure).

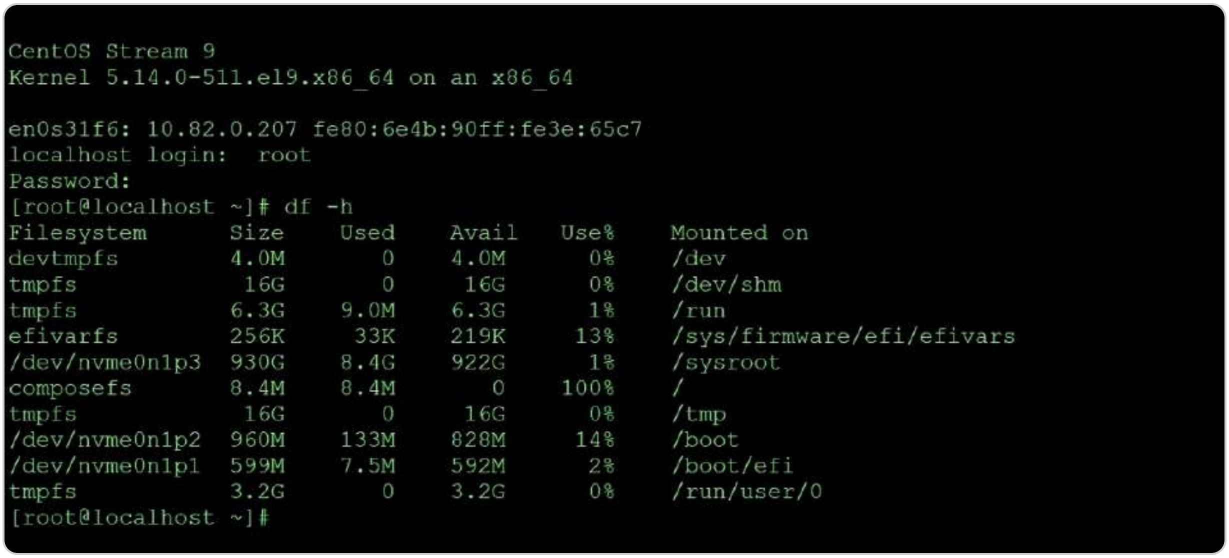
Here's what a successful text-based unattended installation looks like.



```
################################################################################
################################################################################
Installation

1) (x)  Language settings                  2) (x)  Time settings
        (English (United States))                  (America/New York timezone)
3) (x)  Installation Destination           4) (x)  Kdump
        (Warning checking storage                  (Kdump is enabled)
5) (x)  Network configuration
        (Connected: enp0s31f6)

################################################################################
################################################################################
Progress

.
Setting up the installation environment
Configuring storage
Created disklabel on /dev/nvme0n1
Creating xfs on /devnvme01p3
Creating xfs on /devnvme01p2
Creating xfs on /devnvme01p1
..
Running pre-installation scripts
.
Running pre-installation tasks
....
Installing.
Deployment starting: quay.io/matt_st_onge/bootc/lamp-bootc:latest
.
Configuring storage
Deployment complete: quay.io/matt_st_onge/bootc/lamp-bootc:latest
.
Installing boot loader
..
Performing post-installation setup tasks
.
Configuring installed system
...............
Writing network configuration
.
Creating users
.....
Configuring addons
.
Generating initramfs
....
Storing configuration files and kickstarts
.
Running post-installation scripts
.
Installation complete

Use of this product is subject to the license agreement found at:
/usr/share/redhat-release/EULA

Installation complete. Press ENTER to quit:
[anaconda]1:main* 2:shell  3:log  4:storage-log  5:program-log
```

Figure 9.12 – Installation success!

2. Once the installation is complete, test your login credentials at the console.



```
CentOS Stream 9
Kernel 5.14.0-511.el9.x86_64 on an x86_64

en0s31f6: 10.82.0.207 fe80:6e4b:90ff:fe3e:65c7
localhost login:  root
Password:
[root@localhost ~]# df -h
Filesystem       Size    Used    Avail   Use%    Mounted on
devtmpfs         4.0M       0    4.0M      0%    /dev
tmpfs             16G       0     16G      0%    /dev/shm
tmpfs            6.3G    9.0M    6.3G      1%    /run
efivarfs         256K     33K    219K     13%    /sys/firmware/efi/efivars
/dev/nvme0n1p3   930G    8.4G    922G      1%    /sysroot
composefs        8.4M    8.4M       0    100%    /
tmpfs             16G       0     16G      0%    /tmp
/dev/nvme0n1p2   960M    133M    828M     14%    /boot
/dev/nvme0n1p1   599M    7.5M    592M      2%    /boot/efi
tmpfs            3.2G       0    3.2G      0%    /run/user/0
[root@localhost ~]#
```

Figure 9.13 – First login to our new appliance

3. Next, let's determine the IP address of our new system:

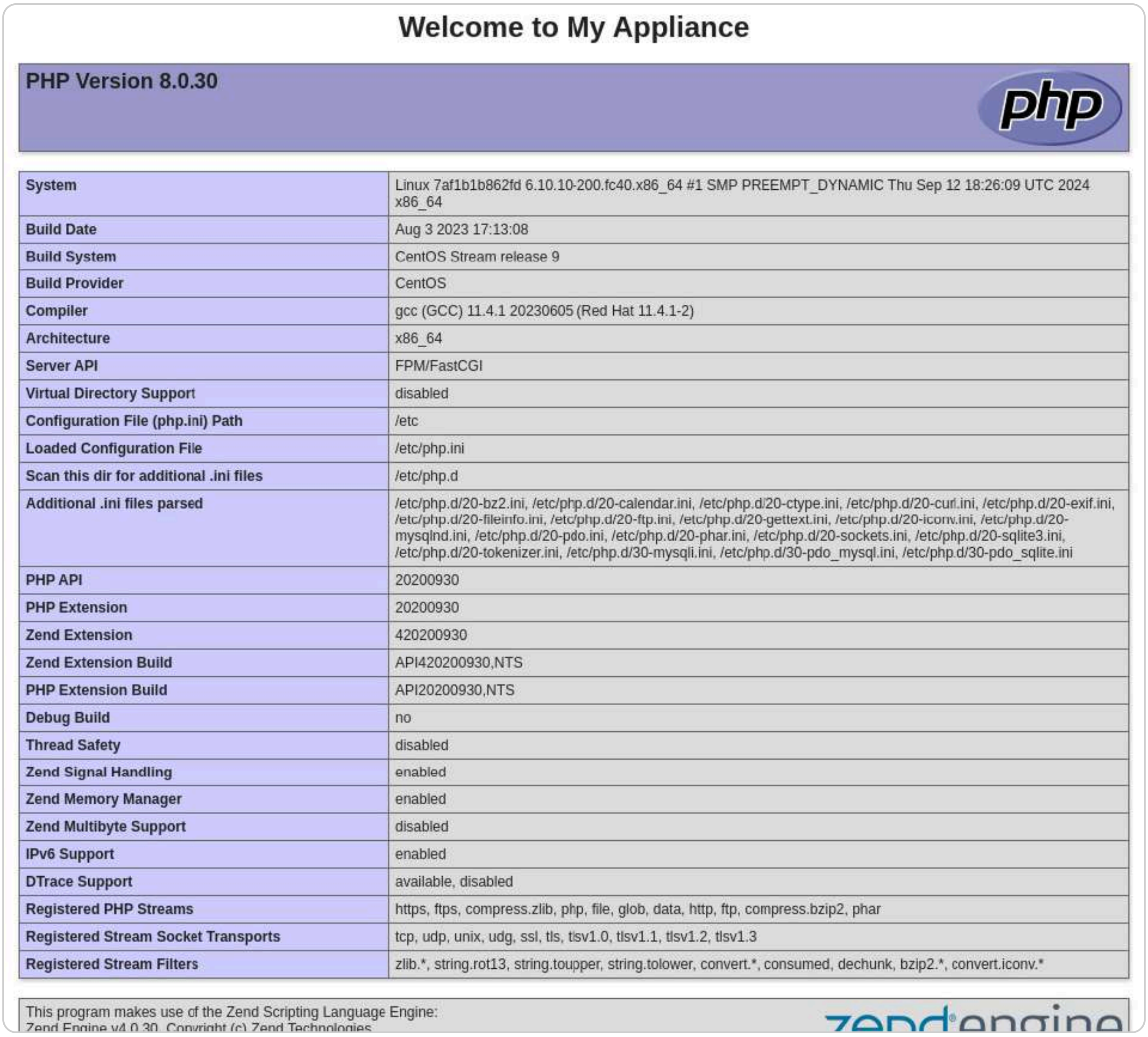
```
$ ip addr show
```

Your output should indicate that you have successfully obtained a DHCP address. Make a note of that IP address.

4. Now, let's open a web browser on another machine and test the LAMP stack.

Go to the IP address that you found in the previous step.

Your result should look something like this:



Welcome to My Appliance

PHP Version 8.0.30

System	Linux 7af1b1b862fd 6.10.10-200.fc40.x86_64 #1 SMP PREEMPT_DYNAMIC Thu Sep 12 18:26:09 UTC 2024 x86_64
Build Date	Aug 3 2023 17:13:08
Build System	CentOS Stream release 9
Build Provider	CentOS
Compiler	gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
Architecture	x86_64
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc
Loaded Configuration File	/etc/php.ini
Scan this dir for additional .ini files	/etc/php.d
Additional .ini files parsed	/etc/php.d/20-bz2.ini, /etc/php.d/20-calendar.ini, /etc/php.d/20-ctype.ini, /etc/php.d/20-curl.ini, /etc/php.d/20-exif.ini, /etc/php.d/20-fileinfo.ini, /etc/php.d/20-ftp.ini, /etc/php.d/20-gettext.ini, /etc/php.d/20-iconv.ini, /etc/php.d/20-mysqlnd.ini, /etc/php.d/20-pdo.ini, /etc/php.d/20-phar.ini, /etc/php.d/20-sockets.ini, /etc/php.d/20-sqlite3.ini, /etc/php.d/20-tokenizer.ini, /etc/php.d/30-mysqli.ini, /etc/php.d/30-pdo_mysql.ini, /etc/php.d/30-pdo_sqlite.ini
PHP API	20200930
PHP Extension	20200930
Zend Extension	420200930
Zend Extension Build	API420200930,NTS
PHP Extension Build	API20200930,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v4.0.30, Copyright (c) Zend Technologies

Figure 9.14 – Viewing your appliance's application

Welcome to the new world where, if you can create a container, you can build a whole system. Let's now move on to how we update these awesome beasts.

Exercise 5 – creating an updated container

In this exercise, we will make updates to our previously built container image, which will in turn provide updates to our Image Mode machine:

1. Your new appliance is defined by its container image. To create an update for your appliance, all we need to do is create a new container and then publish it to our registry. In this step, we will start by creating a new container file called **`mycontainerfile2.cf`**:

```
$ vi mycontainerfile2.cf
```

The contents of your file should look like this. Don't forget to save the file:

```
FROM quay.io/centos-bootc/centos-bootc:stream9
RUN dnf install -y httpd mariadb mariadb-server php-fpm php-mysqlnd && dnf clean all
RUN systemctl enable httpd mariadb php-fpm
# this next command is all one line although looks
# like two or more
RUN echo '<h1 style="Text-align:center;">Welcome to My Appliance</h1><?php
phpinfo(); ?>' >> /var/www/html/index.php
# new stuff
RUN dnf install -y cockpit
RUN systemctl enable cockpit.socket
```

2. Build the new version of your container image:

> ### IMPORTANT NOTE
>
> *Replace your own Quay.io username where* **`[my_account]`** *appears in the command line. You may also have to log in to Quay before running this command (see Exercise 1, step 13).*

```
$ podman build -f mycontainerfile2.cf -t \
quay.io/[my_account]/lamp-bootc:latest
```

The output for this one is significantly long. I have truncated the output in a few locations, so what we can see here is more of a short summary:

```
(((output truncated)))
Installed products updated.
Installed:
  PackageKit-1.2.6-1.el9.x86_64
  PackageKit-glib-1.2.6-1.el9.x86_64
  abattis-cantarell-fonts-0.301-4.el9.noarch
  adobe-source-code-pro-fonts-2.030.1.050-12.el9.1.noarch
((( output truncated more )))
  sscg-3.0.0-7.el9.x86_64
  tracer-common-1.1-2.el9.noarch
  udisks2-iscsi-2.9.4-11.el9.x86_64
  udisks2-lvm2-2.9.4-11.el9.x86_64
  webkit2gtk3-jsc-2.44.3-2.el9.x86_64
Complete!
--> 6ab95e317a3c
STEP 6/6: RUN systemctl enable cockpit.socket
Created symlink /etc/systemd/system/sockets.target.wants/cockpit.socket →
/usr/lib/systemd/system/cockpit.socket.
COMMIT quay.io/matt_st_onge/bootc/lamp-bootc:latest
--> fe247cf7e89d
Successfully tagged quay.io/matt_st_onge/bootc/lamp-bootc:latest
fe247cf7e89d97d5832d889718750d63cc5f2f24dcfd5ed4cce39dfafd150778
```

3. Now that you've rebuilt your container image, feel free to test it in the same way we did in a previous exercise, or don't (it's optional). We do, however, have to push this new image to our registry and set it as the latest version:

> ### IMPORTANT NOTE
>
> *Replace your own Quay.io username where* **`[my_account]`** *appears in the command line.*

```
$ podman push quay.io/[my_account]/bootc/lamp-bootc:latest
```

The output from this command has been truncated significantly due to its length. You can expect your output to resemble this:

```
     (((output truncated)))
Copying blob ad312c5c40cc skipped: already exists
Copying blob bd9ddc54bea9 skipped: already exists
Copying blob 386e8ecea514 done   |
Copying blob 2463de35bc3e skipped: already exists
Copying blob d4cfe3c3d422 skipped: already exists
Copying blob eedcea4f81f6 done   |
Copying blob 2bca4ceb08f4 skipped: already exists
Copying config fe247cf7e8 done   |
Writing manifest to image destination
```

Wow! This is all that you have to do if you want your system to pick up an update automatically. As we are impatient creatures, let's move on to the next exercise and force the update manually.

Exercise 6 – updating your system

In this exercise, we will leverage the latest updates to the container you have created to improve and update our bootable container (bootc) machine.

> ***IMPORTANT NOTE***
>
> *Your machine will check for updates automatically every few hours. The default time check period can be modified.*

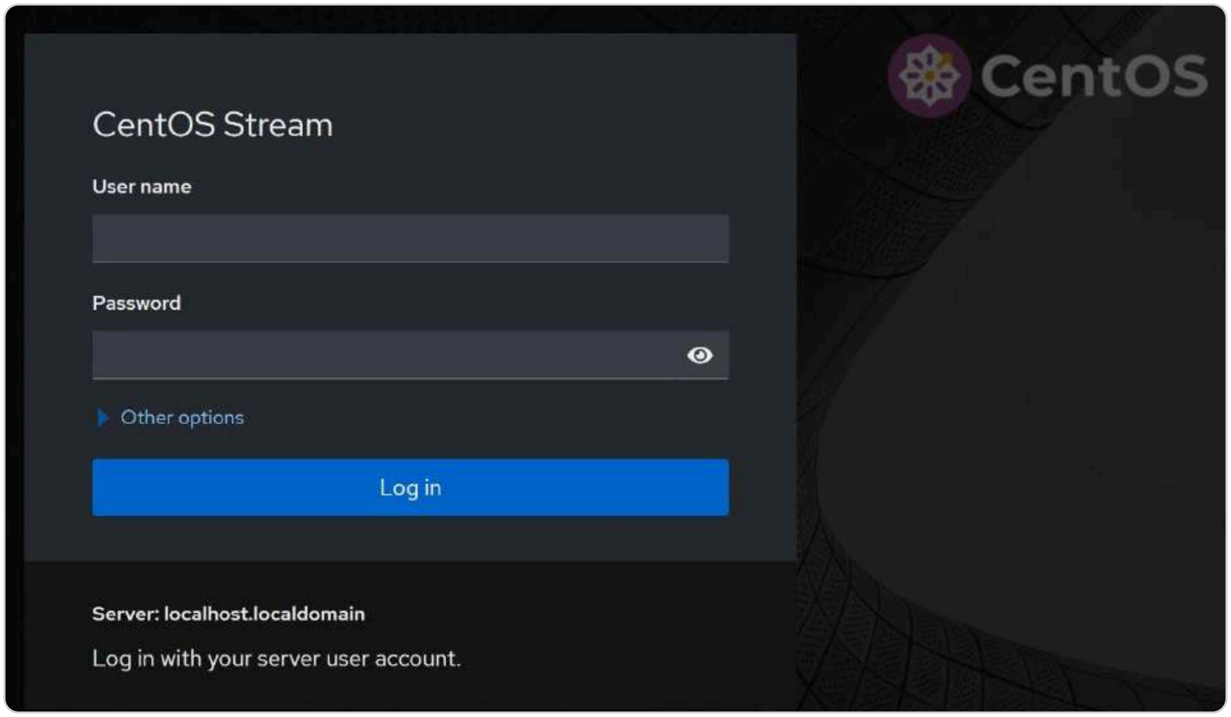
Log back into the console of your appliance machine. Run the following as root:

```
#  /usr/bin/bootc update --apply --quiet
```

Your machine will pull down its updates and reboot itself automatically.

Well done! You have not only created your first bootc machine, but you have established an update mechanism and successfully updated your new machine. Congratulations!

Success! You have updated your appliance by adding the web console to your image. Although you probably cannot log in as root, I hope you know that you can add additional users in the kickstart if you want to rebuild or you can add a user in your console now. Here, we only wanted to show just how easy it is to create an update. It works… gorgeously.



Server: localhost.localdomain

Log in with your server user account.

Figure 9.15 – Appliance is updated with new functionality

I hope that you have enjoyed walking through these exercises and that they have inspired thoughts as to how you could leverage this technology to build a better appliance. Additionally, I hope you continue to experiment and add to what we've covered in this chapter in your own lab.

Summary

Thanks for sticking with me. I never said this journey would be easy. rpm-ostree has been around now for several years, but its time in the limelight has been overshadowed by bootc and bootable container images, an upcoming technology that builds upon its positive facets and the management is much simpler.

In this chapter, we have done an overview of the option of deploying your Linux appliance as an immutable system. I truly believe this technology could fill an entire book itself, so I do use the term overview quite sparingly. As this does greatly enhance the security of the system, it clearly will add some additional complexity to your build and support processes. Now that you've been armed with the knowledge of the tools you may need, you will clearly have some homework to do to determine whether this methodology is right for your team or your product. If time permits, I highly recommend your team does additional research into the feasibility of leveraging this technology in your future solutions. I know that may be a heavy lift. So, let's now move on to the next chapter, where we will dive deep into the art of tamper-proofing.

10
Childproofing the Solution: Protection from the End-User and Their Environment

This chapter will give you a crash course on child-proofing your appliance while ensuring a positive customer experience. Yeah. Both must be done in parallel. Delivering a masterful UI is absolutely a challenge that will require a little extra effort in the solution's development. Just know in advance that your efforts here will pay back in adoption rates exponentially.

In this chapter, you will learn crucial information like why locking the end-user out of the operating system is crucial to security. Additionally, you will gain an understanding of some key security concerns at the hardware level. We shall also review where the hosted applications themselves are solely what the end-users really need to access. Finally, we'll wrap up the chapter by providing you with a deeper understanding of UI design implications that can enhance your appliance's chances for success.

This chapter has the following main headings:

- Introduction to childproofing (i.e., protecting the appliance from the end-user)
- Ensuring hardware-level protections
- Operating-system-level and application protections
- Building a UI to simplify configuration while providing a great User Experience (UX)

Introduction to child-proofing (i.e., protecting the appliance from the end-user)

How to perform **child-proofing** of an appliance prototype is definitely one of my favorite conversations to have with product teams. Please indulge me and allow me to elaborate. This process may be the single greatest factor in determining what meets the definition of an appliance. All efforts to keep the final solution secure and focused solely on its predetermined function are crucial. The end-users must be properly guided on the initial setup requirements and kept within your guardrails of what you choose to allow them to access beyond the application/function of the appliance.

There are many efforts to be considered here. How will my product's end-users actually access the solution? What can be done to minimize or prevent any end-user from obtaining unauthorized elevated or root access? What automation will need to be put in place and obscured via the appliance's UI?

A prime example of what I am describing here is preventing the end-user from leveraging an escape key sequence to *escape* from your application regardless of whether it is console-based or graphical. I call this *No Escape!* This is achieved through thoughtful modification of the general console settings or editing a keymap. Of all the possible guardrails we will cover in this chapter, I consider this a mandatory first step.

Let's proceed onward to securing hardware access.

Ensuring hardware-level protections

Okay, I know that we've discussed this before, in *Chapter 8* ... I feel that I must remind you that this is real. Ensuring that the end-user cannot change the configuration is key here. You have two main responsibilities in this realm. The first responsibility is the securing of the solution and the second responsibility is prevention of the end-user from controlling the operating system. By achieving both, you guarantee that your team controls the end-user experience.

If you create a user interface that simplifies how your appliance can operate securely, ultimately the appliance will be easy to use yet the end-user need not know how it works. I implore you to consider all these factors. Remove the end-user from all access to the operating system.

Create an experience where the end-user can trust that you have achieved, above all else, a secure but user-friendly platform. This is truly the field of dreams. This is what we all aspire to create. This should be our legacy.

Hardware security is something we have covered in quite some depth. Yet still, I feel that we must revisit this conversation, if just briefly. So, let's dive in. Forgive me in advance, but I will reiterate a few details in this chapter. This is meant to protect you and your solution. So, let's now move on to how to properly configure your BIOS settings.

Tamper-proofing with BIOS security

In this section, I want to reinforce what I have been building upon for many chapters up to this point. Choose your security settings carefully. Please understand the pros and cons of each configuration option. Not every setting will offer the best outcome. Some, if not most, have caveats as we have reviewed and we shall continue to dive deeper.

Choosing to lock down the BIOS admin password is a double-edged sword. Yes, you can completely ensure that the end-user cannot change anything in the BIOS… But… That also ensures that your support team is in the business of replacing a unit – not servicing it remotely. Feel free to revisit *Chapter 8* if you need more ideas on BIOS and boot security functions.

This is where your support plan is crucial. Know what it is and own it. Let's move on to the next hardware security option.

USB disablement

I'm on the fence about this security precaution and method of tamper-proofing and I'll elaborate why. Disabling the USB ports on a system, whether through a BIOS setting or via the operating system, can be an elaborate security measure or one that perpetually guarantees your support model is not one of troubleshooting but of direct replacement. Elimination of the ability to connect a device (like a USB thumb drive) adds another layer of security by preventing data transfer, unauthorized booting to a different OS, or the running of code on the device that your team has never authorized.

Choosing this level of security is a significant decision. Or… Maybe it's not. For some solutions, the decision to disable USB is a no-brainer. Let's use a couple of examples here to state a case for the disablement of any possible external USB ports on the chassis of the solution.

The first example is a hypothetical mobile ventilator used by hospitals and ambulance crews and with other installations across the healthcare industry. It has an LED display and all of its controls are built into a touch screen. Disabling the USB ports is a pure tamper-proofing effort. As most of these ventilators might never be attached to a network or be left waiting in hallways and closets for that all-too-crucial moment when their services are needed, tamper-proofing is critical. These machines are subject to government scrutiny and certification. So, updates are left to trained personnel anyway.

My second hypothetical example is a visual targeting system in a military vehicle. This appliance has its own display and custom hardwired controls. USB disablement is implied as this appliance will never be updated nor maintained in the field and will only receive service from trained engineers when the vehicle itself goes into the maintenance depot by those who work for the firm that built the solution. Field soldiers have no access, nor should they have access to the underlying system.

So that's plenty of reasons in the *pro* bucket. Here's a couple for the *con* bucket. I'll use the example of a medical imaging system. It could be an MRI or a CAT scanner – one of those hypotheticals. These are complex yet very well-maintained systems. They must be. They are not maintained by the hospitals that own (or lease) them. Regular maintenance is part of the support agreement when one acquires such an expensive toy. On average, these complex solutions are maintained on a monthly or quarterly schedule by field engineers who are sent by the manufacturer. These support engineers will absolutely need USB access to run diagnostics, patching operations, etc. The hospital staff is not remotely involved in this and these systems are kept in secure rooms with limited access, so tampering is far less of a concern.

My second example is a patient monitoring solution appliance for healthcare that requires a persistent connection with a keyboard, mouse, and, more importantly, a specified USB speaker that is used to alert the nurses at the nurses' station.

My final example against the disabling of USB ports applies to any physical appliance where perhaps the normal UI is delivered through a kiosk-like touch screen. If there is to be any support activity on said device, a physical keyboard and mouse may be needed to simply navigate the interface until help from support can remediate a situation. This is just an oversimplification of how I believe that, at a minimum, allowing for a keyboard and mouse to connect may always be necessary.

For this set of security precautions, I trust that you and your team will weigh the pros and cons. Let's move on to the next tamper-proofing option.

Case tamper-proofing

This is an option with limited value in the end game. Very few hardware manufacturers have enabled this level of detail. To determine if your end-user has actually opened a case without your permission or guidance is an effort in futility.

I say, who cares if they open up the system's case? If your team has properly locked down the operating system and the BIOS, there's little, if anything, a malicious actor can do to break the system. They surely will not be able to add new hardware without your help. Nor should they ever be allowed to do so.

At best, this will only be highlighted when the case has been *opened* by displaying a message at the system's boot time (assuming one can even see the console) ... For this type of lockdown, if your hardware even supports it, you (and your team) wouldn't know, and the end-user might have the ability to erase this altogether from any logs if, in fact, there is nothing preventing them from entering and altering the BIOS settings.

Oddly enough, this is where technology vendors could take a hint from the transportation industry – more specifically, trucking and shipping. Your enclosure will require something like a lock or anti-tamper clamp or numbered tag to prevent the end-user from opening the case (somewhat). Some examples are easily obtained via Amazon. Anything like this could greatly enhance and protect your product from being tampered with. I would recommend having something like the examples custom-made. You should consider two pieces of information to be placed on the tag itself. First, place a serial number that you track internally that is bound to the hardware that you sold that customer. Secondly, have a warning that breaking the seal invalidates any warranties (for your protection).

Examples from Amazon:

- Steel Security Cable Seal – Amazon: https://www.amazon.com/dp/B0CTJZL4QX/ref=sspa_dk_detail_1?pd_rd_i=B0CTJZL4QX&pd_rd_w=2YM0o&content-id=amzn1.sym.8c2f9165-8e93-42a1-8313-73d3809141a2&pf_rd_p=8c2f9165-8e93-42a1-8313-73d3809141a2&pf_rd_r=TAADTD9FHXQ62YCV3V3W&pd_rd_wg=IKKNk&pd_rd_r=cf146379-ab39-49c5-bb8a-048e05de0a7b&s=office-products&sp_csd=d2lkZ2V0TmFtZT1zcF9kZXRhaWw&th=1
- Locking tags for Containers, Cargo... – Amazon: https://www.amazon.com/Numbered-Security-Anti-Tamper-Self-Locking-Container/dp/B07QXWS9DV/ref=asc_df_B07QXWS9DV/?tag=hyprod-20&linkCode=df0&hvadid=693270340443&hvpos=&hvnetw=g&hvrand=8112534262958122670&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9001911&hvtargid=pla-757440487320&mcid=3443569f9f6738f8a4a870639b788988&th=1

As there are a few ways of tamper-proofing your case and protecting the hardware itself, I will humbly acknowledge that not all methods may be appropriate for your solution. Choose wisely. Let's now move forward to how you might look into locking down the operating system for your solution.

Operating-system-level and application protections

Right, here's where things might get a wee bit complicated. I know you may feel a bit of apprehension when I make such a statement. So, with that in mind, I offer this disclaimer: I do not know your application. How could I? I (and you) must trust that your team does.

In this section, I am going to challenge you and your team to build a comprehensive interface that isn't just for your application but user access, updates, etc. The more that you can obscure with your application/user interface the more secure your appliance will be. This, along with the next few sections, are dedicated to how you can make that happen. This challenge I present to you may ultimately determine the success or failure of your product. The more effort you put forth in providing a rich and enjoyable user experience, the more likely the end customers are to applaud your product and adopt it wholesale.

These actions are not just for security but can absolutely create a rich and enjoyable end-user experience. Let's dive into a few of these factors now.

Minimizing access to root

By definition, an appliance should be easy to use but not require the end-user to be vastly skilled in the intricacies of managing the application nor the operating system. These factors are implied. If the end-user's organization had the skills (or the desire) to maintain such solutions, they ultimately would have built them themselves.

Hence, my recommendations are brutally prescriptive in this thought process. I cringe as I write this, yet I feel that you should adopt this mentality as well. It will ultimately protect your support team from countless unnecessary calls. Tamper-proof your solution. Yes, I just said it. The more you prevent the end-user from directly accessing the operating system, the more secure your solution will be.

Failing to do so and an end-user acquiring root access could create a significant emotional event for your appliance. At this point, they can do anything they want with your appliance. Shall I list some of the horrors that could be created? Oh, yes. I shall.

If you have an end-user that has root access, they could possibly carry out any of the following acts with or without malicious intent:

- Add or remove software
- Alter the system's configuration
- Add or remove users from the system or applications
- Remove security precautions or configurations
- Alter or delete logs to cover their tracks (or delete logging altogether)
- Make a change in the system that renders your appliance unusable for the purpose it was procured for

I haven't even really mentioned the truly evil, malicious activities that could possibly occur. For those, I will allow your imagination to roam free. Enjoy! Perhaps now you understand why I implore you and your team to take every reasonable step to protect your solutions.

Regardless of how your solution is accessed or interacted with, having these settings obscured from all end-users and handled behind the scenes will guarantee the security and stability (hopefully the longevity too) of your solution.

SUDO and restricting console access

Depending on how exactly your solution is designed and implemented, this next recommendation may not be feasible. Well… at least as it might have been originally envisioned. Please open your mind. I'm trying to save you from future pain.

This is where I'm going to reiterate some best practices that I hope you already know. Minimize or totally eliminate end-user access directly to the operating system. An appliance *SHOULD* be a black box. The end-users should only have access to the services for which the solution was designed to create, along with any administrative tasks not already automated by the solution.

For the administrative tasks, I feel it is paramount to obscure and remove the end-user from directly accessing operating system too. Giving your user base root access will literally ensure that, at some point, some malicious (or accidental) end-users will break something intentionally, reverse engineer the solution, or – even worse – reconfigure the system to do malicious activities.

So, how can we achieve this? It's not difficult actually. Behind your graphical or web UI, have scripts that are run as a non-root user account that has restricted sudo capabilities. That's the easiest way to keep the user out of the OS or the shell.

But what if the end-user can log into the system and has terminal access... Limit what they can see. Limit what they can execute. Ensure that what they are allowed to execute is controlled by ACLs, SUID, SGID, and other mechanisms set within the files' permissions. Furthermore, I would present a limited interface (even if shell-based) to any user who has access to the system, with the only exception being the access you have created for your support personnel (if any).

In this, the end-user can be seen as an enemy – willing or accidental. Hence, every effort that you can create within your interfaces to prevent direct operating systems access should be seen as mandatory. Let's move on.

Non-interactive LUKS encryption

We have covered several ways of ensuring disk encryption (without end-user interaction) extensively in *Chapters 6* and *7*; however, I would feel remiss in not mentioning it again, as this is also seen as another way to child-proof one's solution.

In using automated disk encryption, you take the responsibility out of the end-user's hands, along with the access to alter it maliciously. Not automating this creates a poor end-user experience and leaves your solution vulnerable. As I know that you already understand my point, I'll say let's move on.

Keeping users in the application space

What does he mean by keeping the users in the application space? Good question. For that, I have a great recommendation. Please allow me to elaborate.

Another great way of ultimately preventing your application or your end-users from compromising the operating system is to keep all the end-users as application-level users, not operating-system-level users. In doing this, your applications' interface is all they can interact with – forever. By this, I mean the end-user only has a user account in your application stack, which is controlled by the application, not within Linux itself. This prevents the end-user from ever accessing a command-line shell.

Think of this in the same manner as application sandboxing... but we're doing it with the end-users... It's a great layer of separation that adds a protective layer of security to your solution.

Application auto-launch at boot

Ensuring that your interface and applications automatically start up at boot is assumed, but I recommend taking it a step further. Run them with **systemd** as a non-root user.

Here's an example of how to set up systemd and **linger** to run your applications as a defined user. This example assumes you already have your service defined via systemd. I used the example service name of `myapplication`. Enabling linger allows for the application or service to run appropriately under a non-root user's login without interactions.

Start and enable your application:

```
$ systemctl --user start myapplication.service
$ systemctl --user enable myapplication.service
```

To ensure the service runs even when the `$USER` is not logged in, we'll need to use `linger`. Remember that `$USER` is the user account you want to run your applications/services: as.

```
$ sudo loginctl enable-linger $USER
```

Now your application will run as the defined user at each startup sequence without any elevated or root access. By employing the customizations detailed in this section, we've taken more steps to ensure system security and stability. Now let's move on and look into how we can provide a great user experience whilst ensuring security.

Building a UI to simplify configuration while providing a great User Experience (UX)

At some point, your prospective customers have made the executive decision to seek a commercially ready solution to a problem that they have encountered. This fateful decision was whether to build a solution in-house versus buying one built by someone else. This truly is an important fact to keep in mind. At some point – hopefully soon – someone will choose to buy your future product!

Honor those who make that decision by giving them a positive and rewarding **User Experience** (**UX**). Delivering them a secure solution is simply implied. They have placed their trust in your company even more if your solution is a security-services-focused appliance. Trust, in this case, also equals responsibility. Your team is responsible for creating an appliance that doesn't need the customers' IT staff to be rocket scientists, or perhaps even an IT staff at all, to assist with the consumption of services from your solution. I challenge you to go beyond that and deliver to the end-customers an interface that goes above and beyond in the following areas:

- Exceeds all expectations in terms of intuitiveness (i.e., the users can navigate with ease)
- Provides the proper number of details to the end-user and doesn't over-burden them with administration concerns

- Accounts for all support and configuration needs that could arise so as the end-user should never need to get direct access to the operating system
- Provides the end-users with the ability to consume the appliance's functionality without extensive training

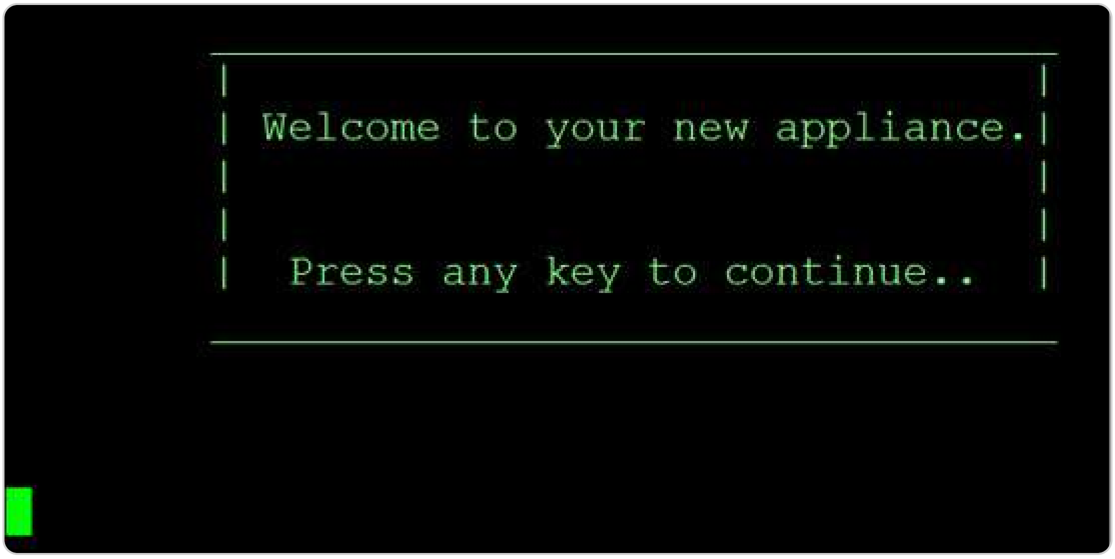
A secure and successful appliance experience can be highlighted by how much planning and automation goes into its interface. These decisions are made early in the design phase and are implemented here. Regardless of how the user interacts with the appliance, obscuring or totally removing the operating system from the user base is crucial. Let's explore some examples of both text and web UI's.

Initial config – text UI

For this set of example screens, I'll be showing you a bare-bones text UI for an initial configuration of an appliance. For argument's sake, let's assume the end-user has a keyboard connected and has console access.

For this type of initial configuration UI, you'll need to set, at minimum, a non-root user for them to log in as and, hopefully, upon the login as this user, they are given only the ability to run this simple setup utility.

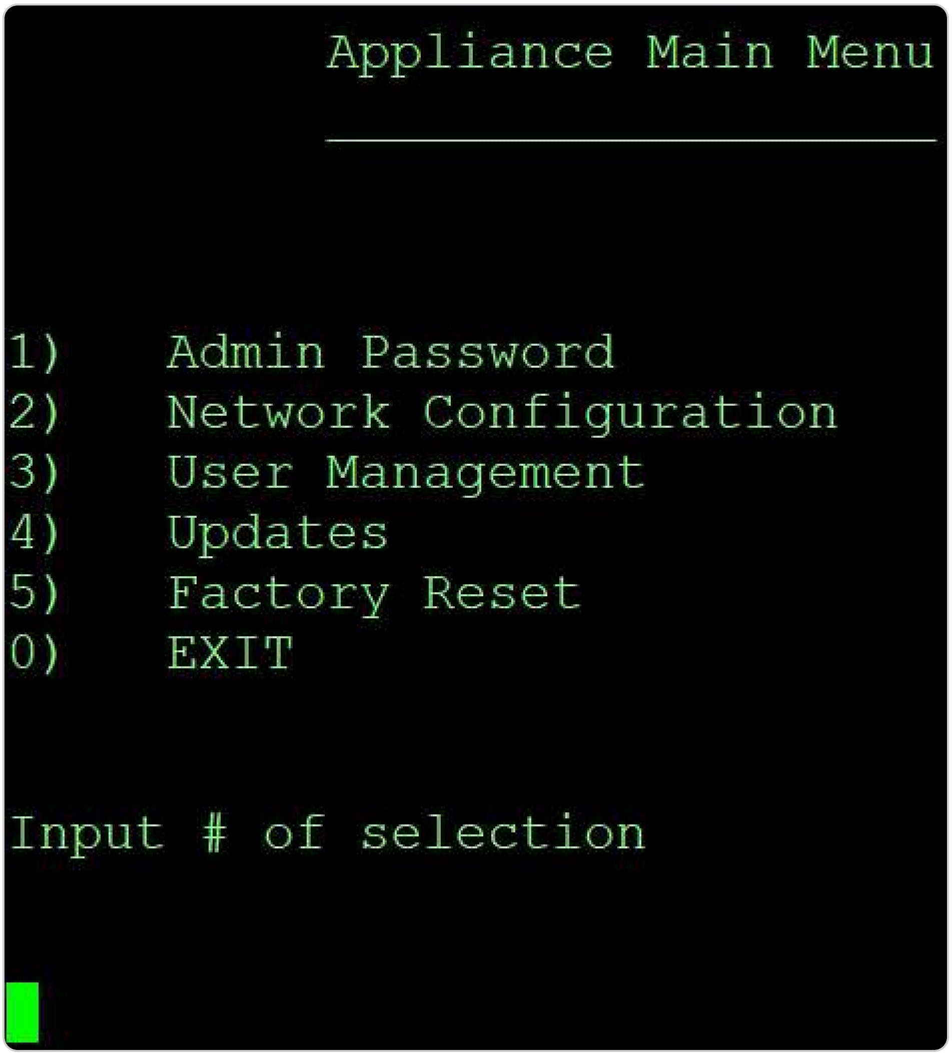
Our first screenshot here simulates a welcome screen for the end-user. You may also want to add in other friendly messages like *Thank you for choosing us…* or something similar.



```
 ______________________________
|                              |
| Welcome to your new appliance.|
|                              |
|                              |
|  Press any key to continue..  |
 ______________________________
```

Figure 10.1 – Text UI – welcome screen

Once past the welcome screen, the end-user would be presented with a main menu of configuration options.



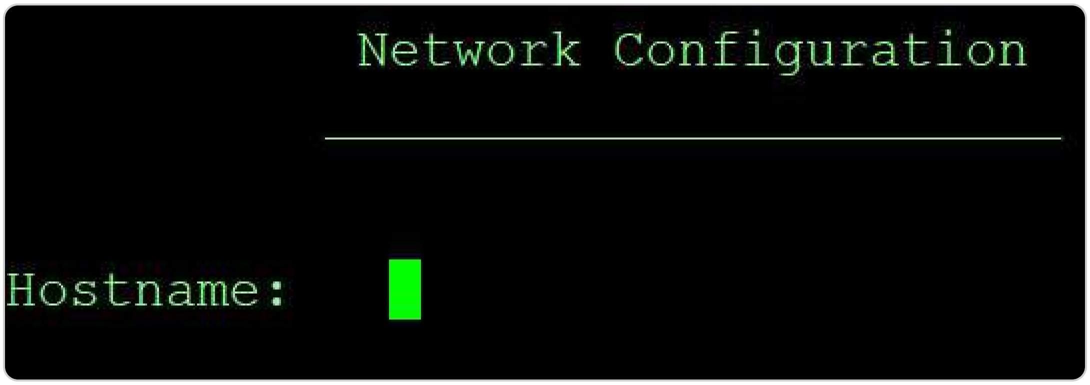
```
          Appliance Main Menu
          ___________________


1)   Admin Password
2)   Network Configuration
3)   User Management
4)   Updates
5)   Factory Reset
0)   EXIT


Input # of selection
```

Figure 10.2 – Text UI – main menu

Let's also assume this appliance serves some function on a network. So, let's help the end-user define the network settings. They will simply give the attribute values and your automation behind the scenes is what really makes the configuration changes, limiting what the end-user can access and do – no root access.

So, these next few screens will show the interactions where the end-user defines the network settings and is eventually asked to confirm them so they can be saved and applied.

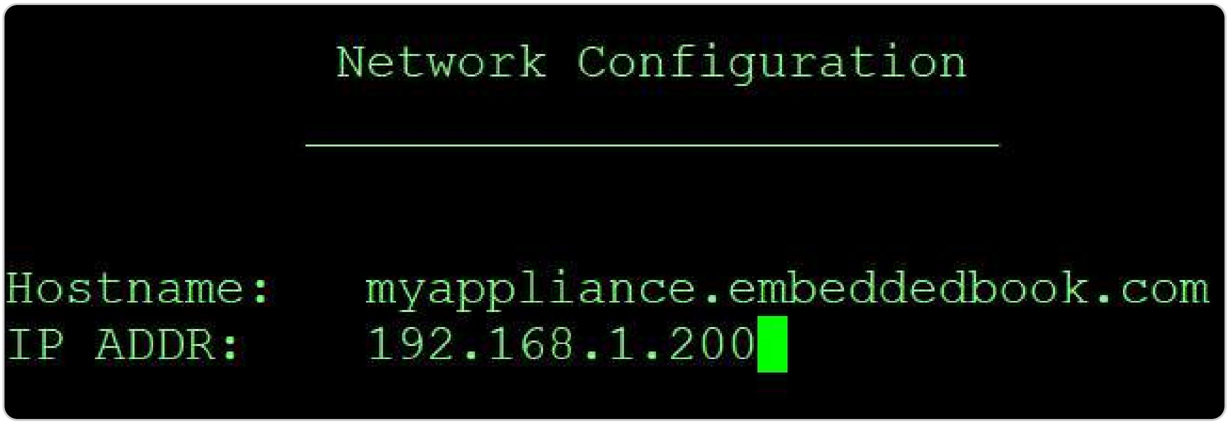
The first setting the user may be prompted for is the appliance's hostname.



```
           Network Configuration
         ______________________

Hostname:
```

Figure 10.3 – Text UI – network configuration – start

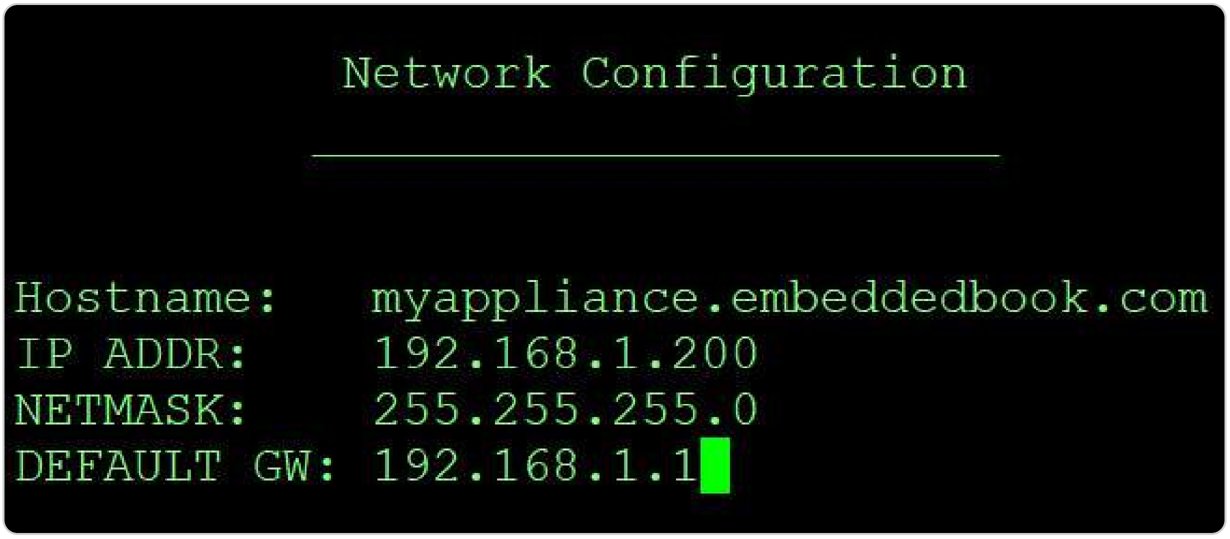
As we continue further into the network configuration, IP addressing begins:



```
           Network Configuration
         ______________________

Hostname:    myappliance.embeddedbook.com
IP ADDR:     192.168.1.200
```

Figure 10.4 – Text UI – network configuration – in progress

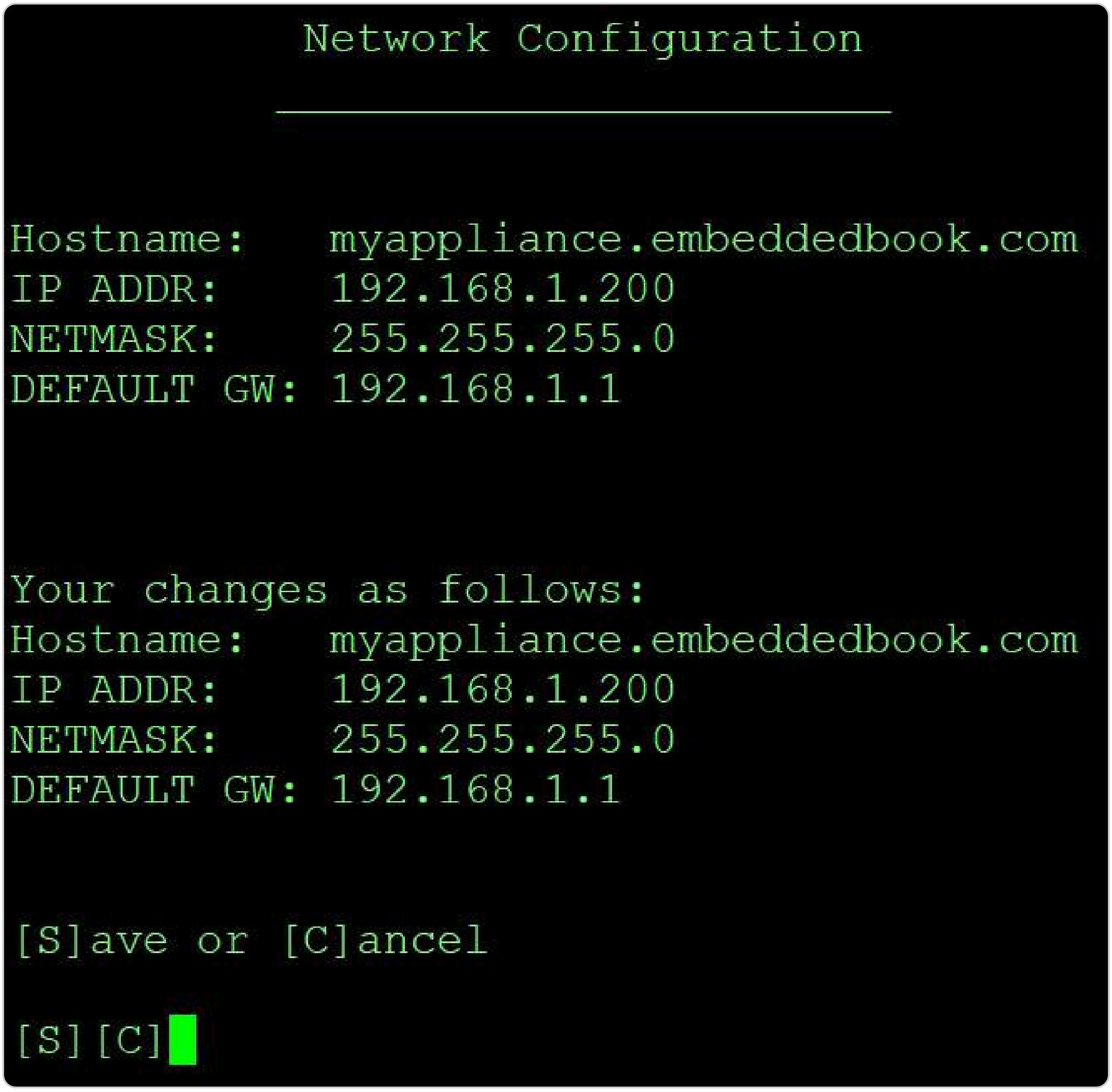
There are more network settings as we progress. Here, we are displaying a few more settings inward where netmask and DNS settings can be configured.



```
           Network Configuration
         ______________________

Hostname:    myappliance.embeddedbook.com
IP ADDR:     192.168.1.200
NETMASK:     255.255.255.0
DEFAULT GW:  192.168.1.1
```

Figure 10.5 – Text UI – network configuration – continued

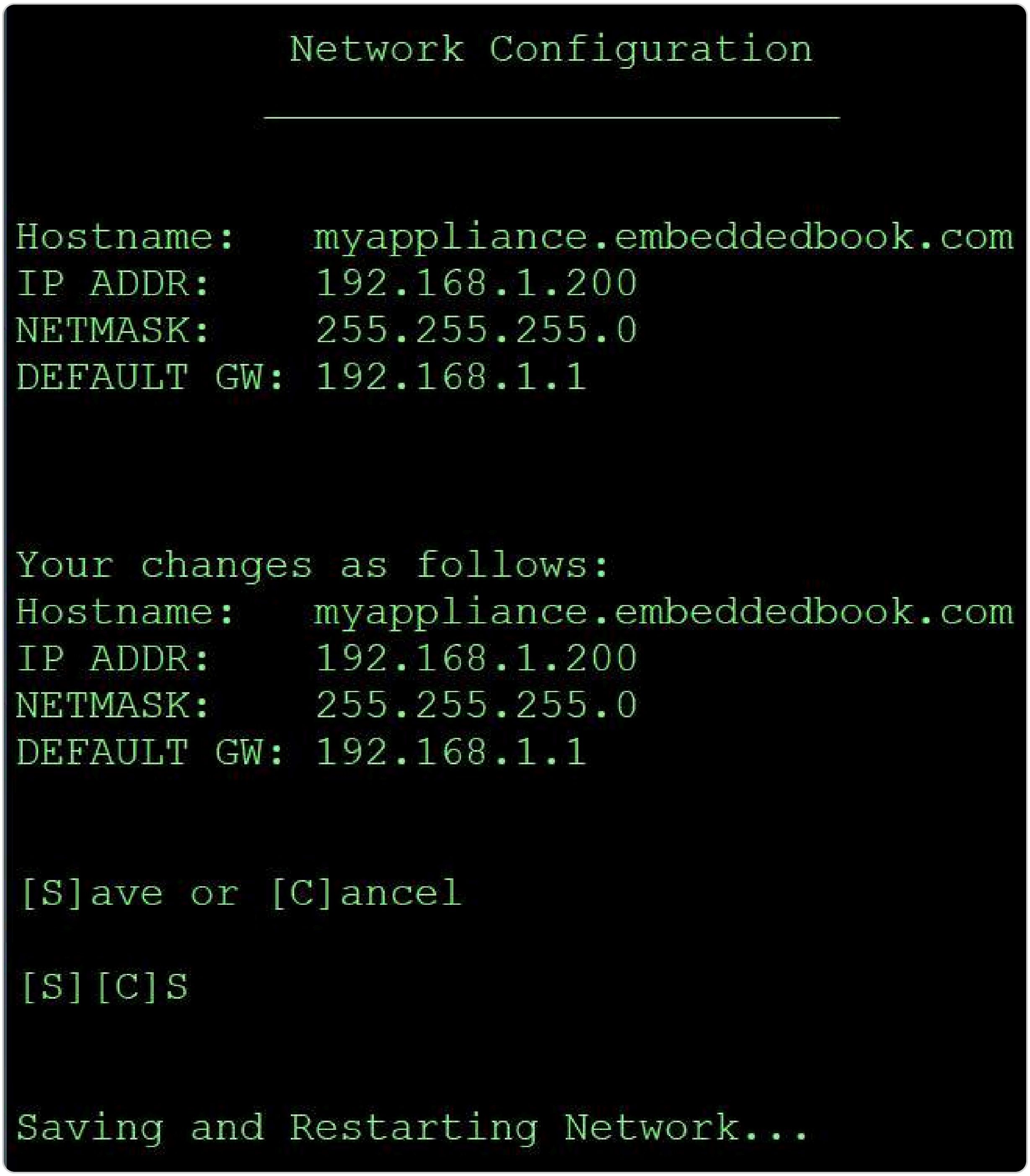
Next, as best practice, we shall always prompt them to confirm (or cancel) at the end before the settings are committed to the running configuration.



```
            Network Configuration
          ______________________________

Hostname:   myappliance.embeddedbook.com
IP ADDR:    192.168.1.200
NETMASK:    255.255.255.0
DEFAULT GW: 192.168.1.1



Your changes as follows:
Hostname:   myappliance.embeddedbook.com
IP ADDR:    192.168.1.200
NETMASK:    255.255.255.0
DEFAULT GW: 192.168.1.1


 [S]ave or [C]ancel

 [S][C]
```

Figure 10.6 – Text UI – network configuration – confirmation

Now your automation behind the scenes is doing the actual configuration work.



```
              Network Configuration
            ______________________


Hostname:   myappliance.embeddedbook.com
IP ADDR:    192.168.1.200
NETMASK:    255.255.255.0
DEFAULT GW: 192.168.1.1



Your changes as follows:
Hostname:   myappliance.embeddedbook.com
IP ADDR:    192.168.1.200
NETMASK:    255.255.255.0
DEFAULT GW: 192.168.1.1


[S]ave or [C]ancel

[S][C]S


Saving and Restarting Network...
```

Figure 10.7 – Text UI – network configuration – execution behind the scenes

So, there you have it – a simple network configuration for a text-based appliance. Try to keep things as simple as possible yet consider adding all facets of your appliance's usage and configuration to such a menu system. Obscure all operating system operations, whilst protecting the system from the end-user as well.

Let's move on to what a simulated minimal web configuration interface may look like.

Initial config – web UI

Web-based interfaces allow for a much more aesthetically pleasing experience but require more programming. You'll still be collecting the same information as the text UI did but instead of instantly calling sudo scripts, the web UI will leverage sudo scripts called by `cgi-bin GET` or `POST` operations. I'm not remotely going to teach you how to program webpages as there are literally hundreds of books that could do a better job at that than I could.

I will, however, mention that for the initial configuration, you should consider leveraging a pre-defined and documented IP address for the new end-user to access your initial setup pages.

In the new few screenshots, we'll be looking at a simulated walk-through of a simplified web UI for the initial appliance configuration.

Let's start with a welcome webpage. Greet your new customer and let them know the system is working! This page politely indicates that we will need to do some basic configuration work before the end-user can begin to consume the services this appliance provides.

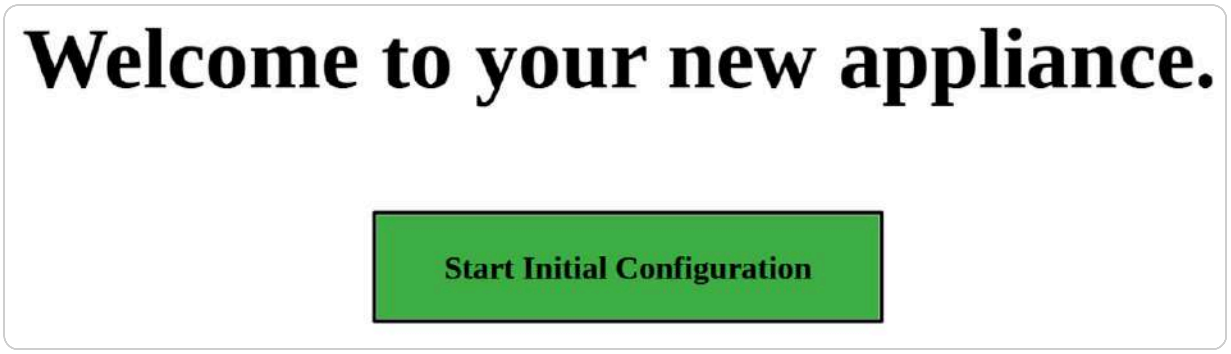

Welcome to your new appliance.

Figure 10.8 – Web UI – welcome screen

From the welcome page, we select **Start Initial Configuration** and enter a simulated main menu with simple navigation selections available on a sidebar. From here, the end-user is presented with several obvious options that should also remain available beyond the initial configuration event.

In this main configuration menu, the end-user will have some key options that should be planned for accordingly – administrative access, user access, network configuration, factory reset, and, of course, a way to process updates.

As the end-user is anxious to get started using this appliance, logical first steps include guiding the end-user to set an admin password and then configure the networking/hostname.

In this example, we'll see what a simplified web-based UI for an appliance's administration main menu should contain at a minimum.

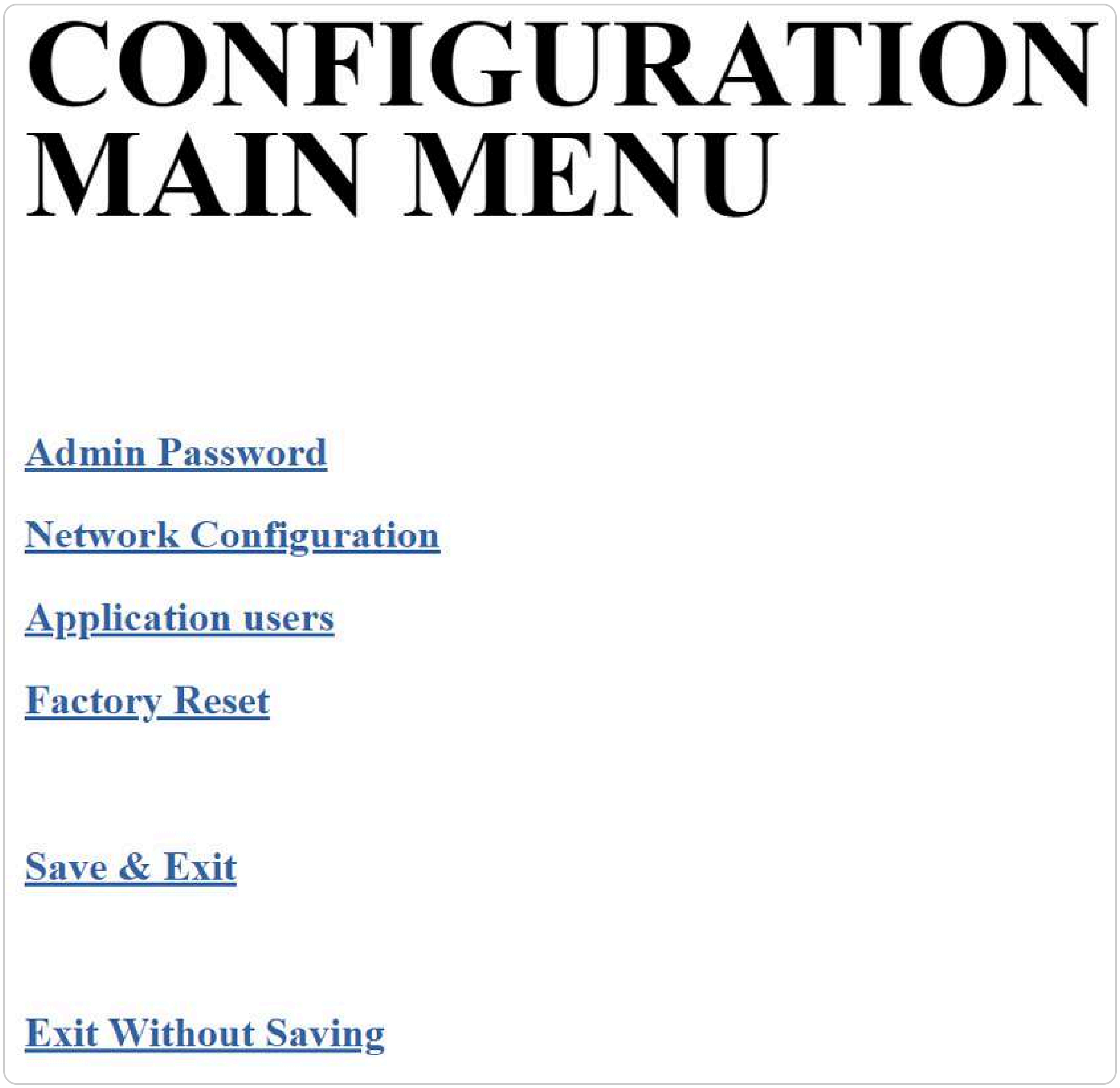


Figure 10.9 – Web UI – main appliance configuration menu

Assume that we had selected **Network Configuration** here and we want to set up the basic networking settings for this appliance. Basic needs for this type of initial configuration should always include not just network configuration but also having the ability for the end-user to assign a proper hostname so that the system can also be referenced by their own DNS systems.

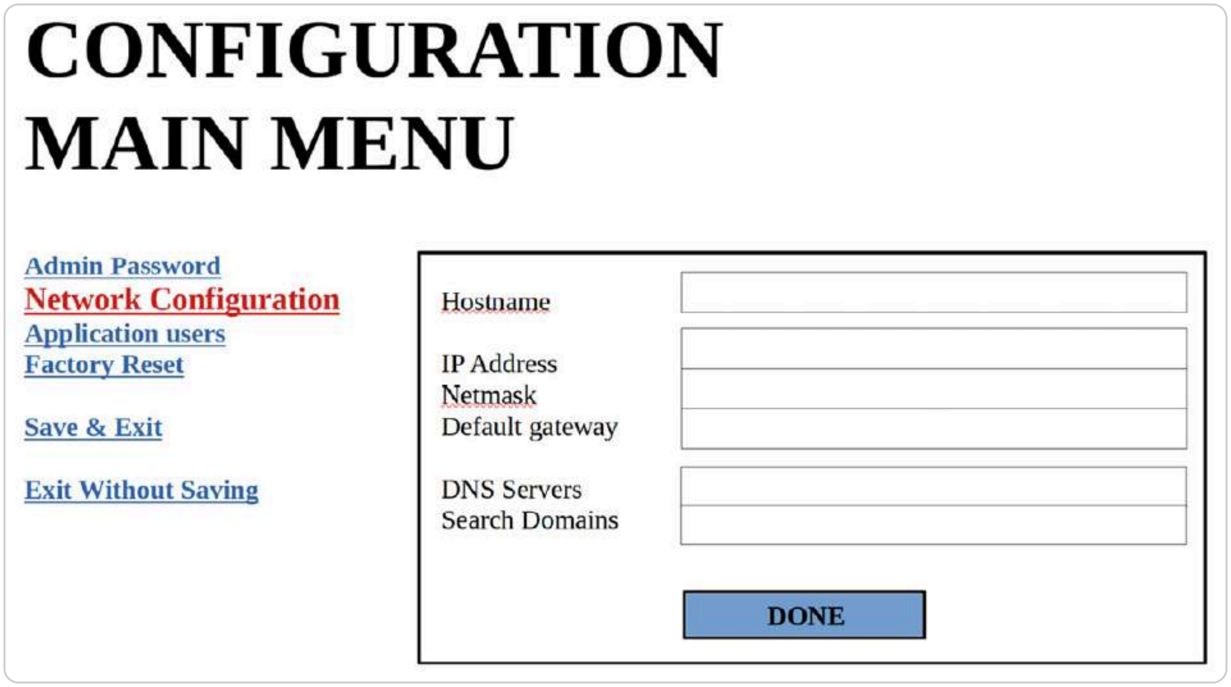


Fig 10.10 – Web UI – network configuration

Now that we've guided our new end-user through a crisp basic appliance configuration process, they can leverage your product for all the great services it provides them. Eventually, as with any computer system nowadays, the appliance will require patching and updates, which leads us to our next section.

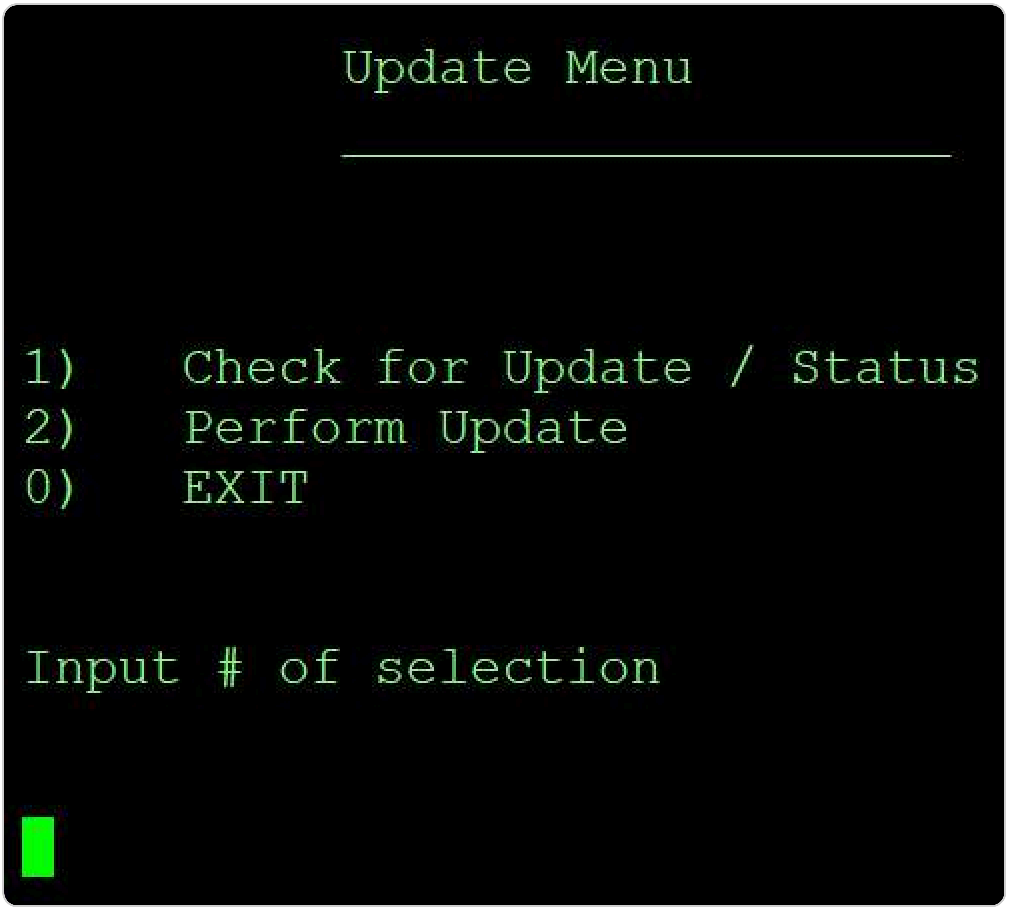
Update controls – text UI

Your appliance's end-user experience would not be complete unless you add in a method for the appliance to get updated. By updates, I refer to operating systems bug fixes and enhancements, application patches and enhancements, new features, and even firmware updates if your solution is physical.

Creating this type of interface also creates a boundary layer from what the end-user can do and what the appliance can do for itself. We must remove any possible need or excuse for any end-user to demand access directly to the operating system itself.

Depending on the use case and deployment methodology, the interface should account for how updates are to be presented. Air-gapped systems will not be able to go out to the internet and check for their updates, so the interface should be able to ingest updates alternatively. USB thumb drives, tarballs, and ISO images are all examples of ways that one could deliver updates to the end-user, however, the end-user would then need to deliver them to the interface and the appliance should process them without the end-user having access to the operating system.

Here's an example of what appliance update controls may look like in a text UI, where the appliance has access to the internet to check for its updates.



```
          Update Menu
          ___________________

1)    Check for Update / Status
2)    Perform Update
0)    EXIT

Input # of selection

█
```

Figure 10.11 – Text UI update menu

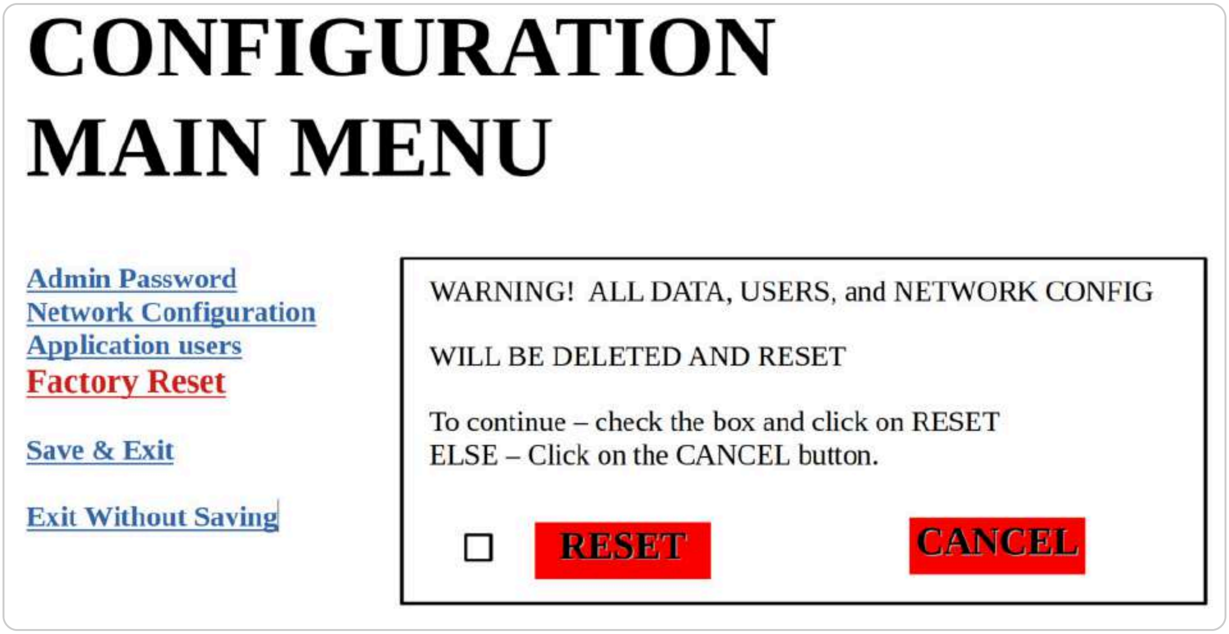
Now that we've addressed the fundamental needs of being able to provide updates to our appliances, we can move on to our next crucial end-user function: appliance resets.

Factory reset controls

Every good appliance should have a way of resetting itself back to its factory-new persona. Creating this experience for the end-users will also make it much simpler for them to reprovision an appliance onto a different network or location within their organization, by removing any stored application data or previous configuration. This type of functionality also gives the customers peace of mind that their possibly sensitive data has been erased if they are shipping a unit back to your company. The next screenshots will show examples of what this could look like on a text UI and webpage.

Let's start with an example web UI reset function page. This page would present a two-step process that ensures they really know that the machine will go back to a brand-new state (all data and configuration erased). To achieve this state, they would first have to check the checkbox in the interface and then press the **RESET** button.

> ***IMPORTANT NOTE***
>
> *I would also recommend having a pop-up window asking them to confirm that yet again if you really want to be user-friendly.*



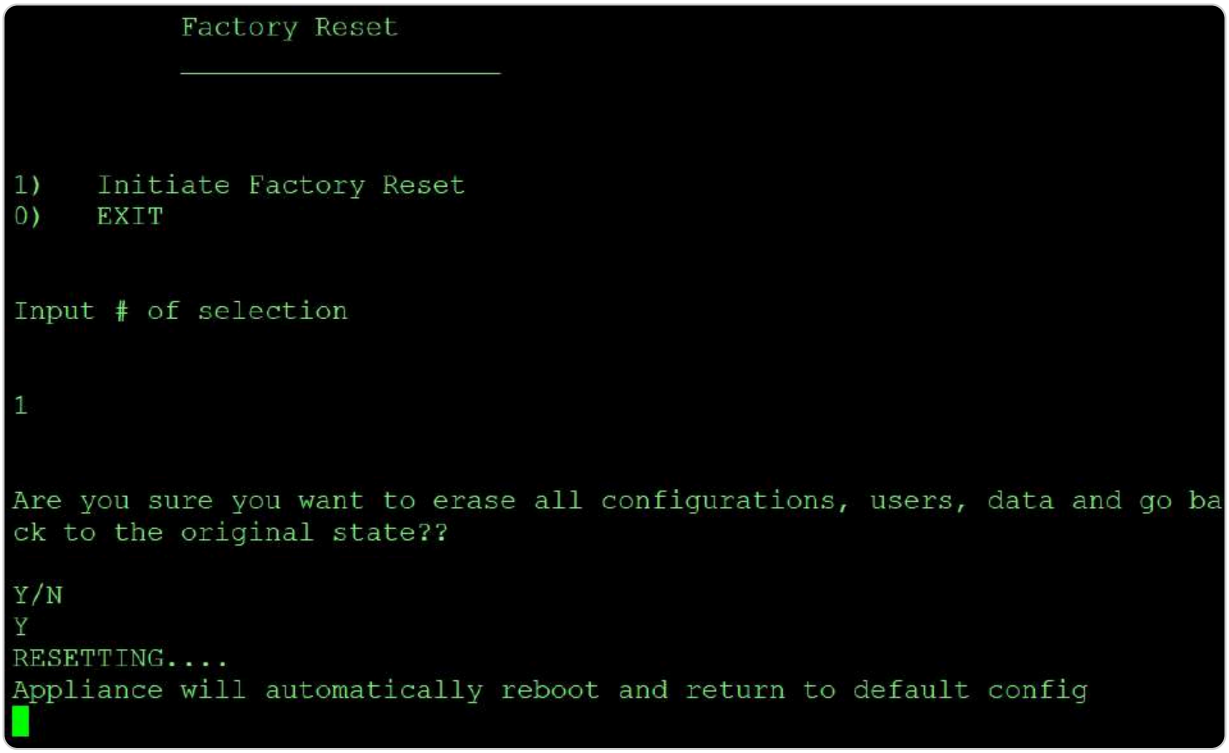
WARNING! ALL DATA, USERS, and NETWORK CONFIG

WILL BE DELETED AND RESET

To continue – check the box and click on RESET
ELSE – Click on the CANCEL button.

Figure 10.12 – Webpage factory reset example

And here's an example reset in a text UI. The end-user is presented with a prompt to confirm that they really want to reset the appliance back to its factory-new configuration and erase all data. Just like with the web example shown previously, they always have the ability to simply cancel the operation.



```
          Factory Reset
          ___________________



1)   Initiate Factory Reset
0)   EXIT


Input # of selection


1


Are you sure you want to erase all configurations, users, data and go ba
ck to the original state??

Y/N
Y
RESETTING....
Appliance will automatically reboot and return to default config
```

Figure 10.13 – Example factory reset in text UI

I trust that these simulations of functions needed in your future UI have been insightful. Yours, of course, will obviously look far more refined and be well in line with your company's branding guidelines for your products. By establishing a secure UI/end-user environment, you've ensured another factor of security for your solution – tamper-proofing. Let's now move on to our end-of-chapter summary.

Summary

Thanks again for driving onward on this journey with me. In this chapter, we've reviewed why locking down our appliances is vital. We have recapped some hardware and storage security options to consider. Best of all, we've identified key ways of ensuring a great user experience while maintaining security by obscuring the operating system access away from the end-user.

In our next chapter, we'll take a deep dive into all the available informational resources that can help you and your team stay ahead of the threat landscape.

Part 3: The Build Chain, Appliance Lifecycle, and Continuous Improvement

In this part, we'll review additional hardening processes for your appliances and explore how to apply some crucial government security standards to systems and how best to capture information from all avenues to continuously improve your appliances throughout their lifecycle.

This part has the following chapters:

- *Chapter 11, Knowing the Threat Landscape – Staying Informed*
- *Chapter 12, Are My Devices' Communications and Interactions Secure?*
- *Chapter 13, Applying Government Security Standards – System Hardening*
- *Chapter 14, Customer and Community Feedback Loops*

11
Knowing the Threat Landscape – Staying Informed

In this chapter, we're going to go on a data-gathering journey, which is a slight departure from the super technical hands-on exercises. Instead, we're going to exercise our minds by enhancing our data collection efforts. This isn't user data or performance data; it's possible threat and prevention data – golden nuggets throughout.

I'm going to take you through a quick personal tour of websites, blogs, and other miscellaneous resources that you'll want to add to your own personal security toolkit. Before I drag you through this jungle, I will give up a couple of disclaimers. Firstly, just because I am mentioning it, doesn't mean I endorse it. You're smart and resourceful. You are capable of making your own decisions on how to process what you're about to learn. Secondly, I accept that my language capabilities are limited; I know I am missing many foreign and non-English sites. It's not that I am being inconclusive, but it's virtually impossible once language barriers get in the way. Good thing that most international tech sites are by nature often in English.

Following the resources, subscribing to the newsletters, and ensuring your vendors are emailing you updates is a huge step in ensuring that you and your team stay informed and prepared. I also recommend that you do more than simply observe; I suggest that you actively participate in making a positive impact within the security community if that's feasible for you.

Please understand that this chapter is meant to guide you to available informational resources (mostly freely available resources). It's my aspiration that this gives you and your team a good starting point of where to gather information in order to make better-informed decisions.

As a companion to this chapter, there will be a cheat sheet with tons of URLs and suggestions that we cover here. It can be found in the book's GitHub repository (https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook/blob/main/Chapter11/Resource_Cheat_Sheet.pdf). It's my plan to keep this cheat sheet as a living document with updates as they become available. So, check the repository regularly for updates.

In this chapter, you will learn about many resources, publicly and commercially available. You will review how to find security data relevant to your appliance and how to apply that information as part of your build chain. I'll also (hopefully) entice you to join in, be active, and participate in the Linux community.

We will have the following main headings in this chapter:

- Navigating the information and disinformation online
- Knowing what vulnerabilities can impact your builds
- Being part of the solution

Navigating the information and disinformation online

As we've all come to know, not all internet sources are equal. Some can be regarded as diamonds in the rough, whereas most websites are generally just *okay*. As I feel compelled to also mention, some websites are just places to be avoided at all costs. Rarely do we know which falls into that category until it's too late. It's safe to say that we've all at least once clicked on a link and, before the page finished loading, said to ourselves, "Oh man, why did I click on that?!" Instantaneous regret. My only point here is for you to properly consider and vet your own information sources.

Over the next few sections, I will be highlighting some excellent resources that you and your team should be taking advantage of. These resources range from government agency compliance sources, open source resources, and commercially available content, along with blogs, newsletters, and video channels. I'm not saying that you need to use all of them. I'm merely offering you a buffet of consumable knowledge that never gets stale. Eat well. Eat often. Sometimes, I recommend that you change it up and try different ones to sample.

As I prefer to remain positive, I have chosen not to dwell on the websites that I perceive as ineffectual or simply flat-out non-productive. Nor do I ever desire to be responsible for ever sending someone to them. Ever. Period. I don't think I could sleep at night if I had done you that disservice. That said, bring your appetite for knowledge, and let's move on to our smorgasbord of security resource delights.

Government resources

Let's start with the usual suspects – the websites that because of your industry, location, or status in the marketplace fall under the umbrella of one or multiple compliance regulations. Finding out exactly what is required can be tedious, and implementing those mandates is even more *fun*.

Don't expect intuitive interfaces, stylish themes, and graphics, or sometimes even ease of navigation. Government websites generally can be a test of one's patience. Yet, they are also the pillar of truth when detailed technical security requirements and configurations are mandated for compliance. Do expect, however, an extreme level of professionalism and technical detail. Ignore these resources at your own peril. The following US government resources are, in my humble opinion, the key starting points for gathering and maintaining information related to security threats that you can apply within your appliance's build chain.

Computer Security Resource Center (CSRC)

This is hosted by NIST. This site can be considered the gateway to all cybersecurity in North America. No other site or organization has the level of detail and depth of information anywhere else in the world.

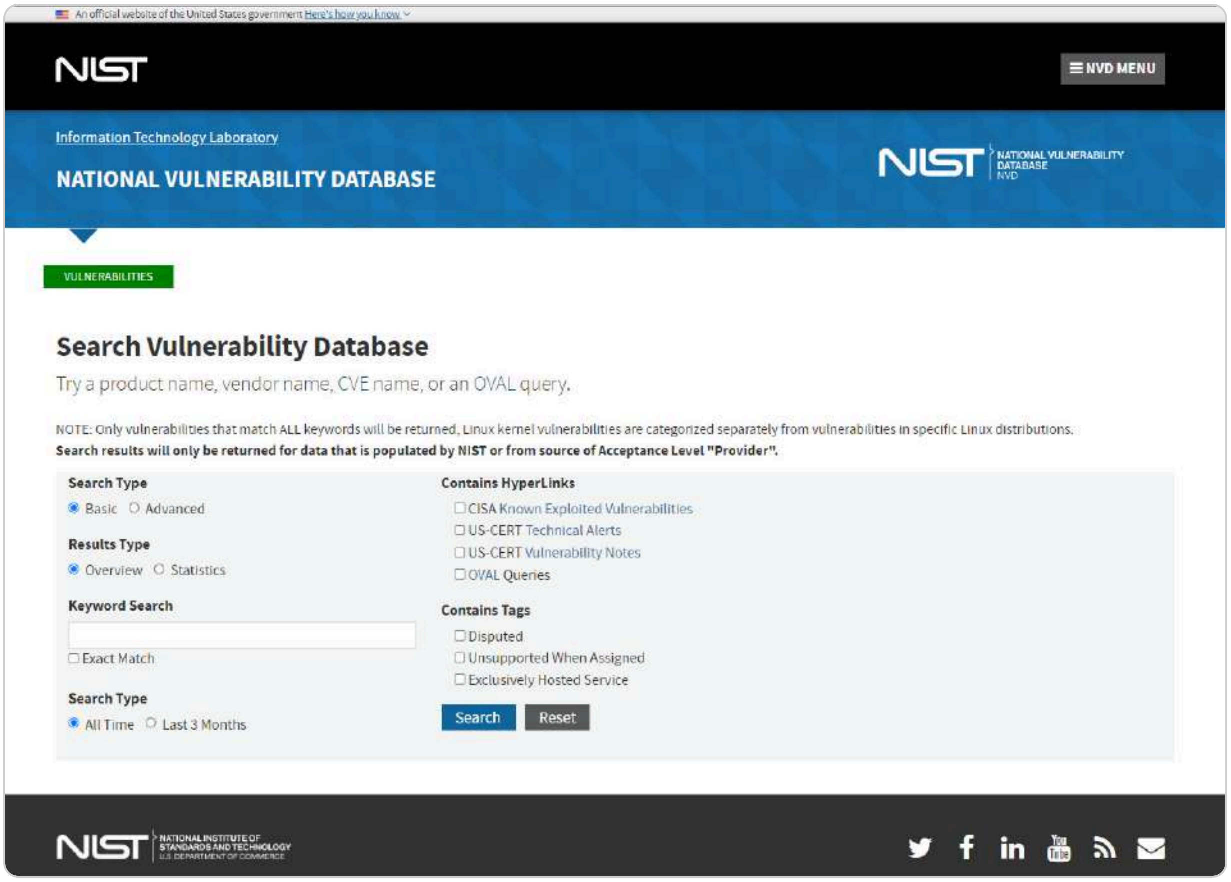
That said, it's easy to get lost here. Why? There's just so much information available and it is constantly changing. Projects, events, publications, and even information on how to properly engage with NIST itself can be found here.

Start here. Take notes, make bookmarks, and plan for follow-ups.

National Vulnerability Database (NVD)

This is also hosted by NIST. This is the home of the CVE database. If it's a risk or vulnerability, this place has it documented. Search here often. It's well maintained and is the pillar of truth for cybersecurity vulnerabilities.

Where this resource becomes valuable is in its searchability. Here's an example:



An official website of the United States government Here's how you know
NVD MENU
Information Technology Laboratory
NATIONAL VULNERABILITY DATABASE
VULNERABILITIES

Search Vulnerability Database

Try a product name, vendor name, CVE name, or an OVAL query.

NOTE: Only vulnerabilities that match ALL keywords will be returned, Linux kernel vulnerabilities are categorized separately from vulnerabilities in specific Linux distributions.
Search results will only be returned for data that is populated by NIST or from source of Acceptance Level "Provider".

Search Type: [✓ Basic] Advanced

Results Type: [✓ Overview] Statistics

Keyword Search

☐ Exact Match

Search Type: [✓ All Time] Last 3 Months

Contains HyperLinks

☐ CISA Known Exploited Vulnerabilities
☐ US-CERT Technical Alerts
☐ US-CERT Vulnerability Notes
☐ OVAL Queries

Contains Tags

☐ Disputed
☐ Unsupported When Assigned
☐ Exclusively Hosted Service

Search
Reset
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY U.S. DEPARTMENT OF COMMERCE

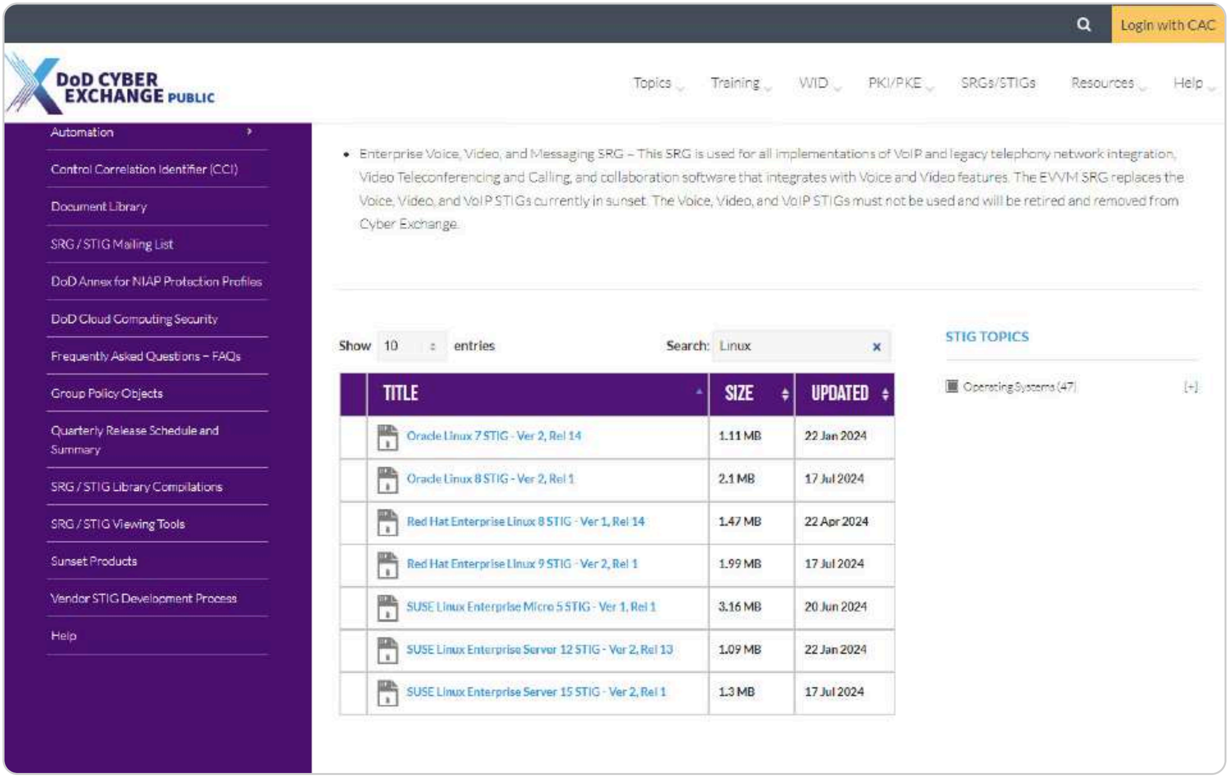
Figure 11.1 – Searching NIST's NVD

It's quite probable that you may very soon find yourself getting intimately acquainted with the search features within the NIST websites. Hopefully, this book's introduction helps guide you to finding the resources that you are seeking. It truly is the nature of the beast. This site will become your lighthouse in the storm, so the sooner you get used to its idiosyncrasies and user-friendliness, the better – the information presented here truly is priceless. I just wish they'd make the UI a little easier on the eyes, if you know what I mean. Let's now progress onward in our journey by learning how and what to configure in the quest to make our systems more secure.

Security Technical Implementation Guides (STIGs)

These were published by the **Defense Information Systems Agency** (**DISA**) for their primary customer, the US DOD. These are published as whole libraries of configurations. DISA touts its mission as, "To conduct DODIN operations for the joint warfighter to enable lethality across all warfighting domains in defense of our nation." Due to its military roots, usage of this website is monitored, and you will be required to consent to your usage being tracked when you go there, even if you do not log in.

Here's an example of the STIGs available for Linux today:


Login with CAC
DoD CYBER EXCHANGE PUBLIC
Topics
Training
WID
PKI/PKE
SRGs/STIGs
Resources
Help
Automation
Control Correlation Identifier (CCI)
Document Library
SRG / STIG Mailing List
DoD Annex for NIAP Protection Profiles
DoD Cloud Computing Security
Frequently Asked Questions – FAQs
Group Policy Objects
Quarterly Release Schedule and Summary
SRG / STIG Library Compilations
SRG / STIG Viewing Tools
Sunset Products
Vendor STIG Development Process
Help
STIG TOPICS
Operating Systems (47)
[+]

- Enterprise Voice, Video, and Messaging SRG – This SRG is used for all implementations of VoIP and legacy telephony network integration, Video Teleconferencing and Calling, and collaboration software that integrates with Voice and Video features. The EVVM SRG replaces the Voice, Video, and VoIP STIGs currently in sunset. The Voice, Video, and VoIP STIGs must not be used and will be retired and removed from Cyber Exchange.

Show 10 entries Search: Linux

TITLE	SIZE	UPDATED
Oracle Linux 7 STIG - Ver 2, Rel 14	1.11 MB	22 Jan 2024
Oracle Linux 8 STIG - Ver 2, Rel 1	2.1 MB	17 Jul 2024
Red Hat Enterprise Linux 8 STIG - Ver 1, Rel 14	1.47 MB	22 Apr 2024
Red Hat Enterprise Linux 9 STIG - Ver 2, Rel 1	1.99 MB	17 Jul 2024
SUSE Linux Enterprise Micro 5 STIG - Ver 1, Rel 1	3.16 MB	20 Jun 2024
SUSE Linux Enterprise Server 12 STIG - Ver 2, Rel 13	1.09 MB	22 Jan 2024
SUSE Linux Enterprise Server 15 STIG - Ver 2, Rel 1	1.3 MB	17 Jul 2024

Figure 11.2 – Linux STIGs available today

To properly view these, you will need to download their **Viewer** tool as well. Leveraging their recommendations will help to ensure that your solution has been configured with the best-intended level of security.

Personally, I have the STIG Viewer installed on multiple systems in my lab and also on my work laptop. I prefer to keep the ability to dive deep into what configurations might be needed readily available to me regardless of where I am at. You may also reach this same conclusion as you progress in your own journey.

Let's now continue our journey by looking at commercially available resources.

Commercial resources

We have to go there. Not everything mentioned in this book is free. This is to be expected, even more so in the enterprise security space. Not all these tools and resources are able to be feasibly run in someone's home lab as cost and hardware requirements might just ultimately make it impossible for most people. That said, all these solution providers do have free publicly available resources that all can take advantage of.

This list of vendors is by no means exhaustive, nor should any listing here be seen as an endorsement. Know the key players. Here are some of them.

Reversing Labs®

Headquartered in Cambridge, Massachusetts, Reversing Labs® brings a unique offering to the table for those with advanced security consciousness. Their flagship product has the ability to check for risks without access to the source code. They also provide a wide range of enterprise security and Bill of Materials offerings.

Qualys®

Headquartered in Foster City, California, Qualys® offers a wide range of products that are tailored to multiple industry verticals, including the government. What they are most known for is their ability to scan and detect risks across the enterprise. Qualys is a huge player in the security industry.

Tenable®

Headquartered in Columbia, Maryland, Tenable® is another security industry juggernaut. They, too, are most known for their security scanning tools, although they have many other products and solutions.

McAfee®

Headquartered in San Jose, California, McAfee® is a well-recognized name in the virus protection space. They also provide many other products and services mostly focused on secure VPN and identity protection.

Norton®

Co-headquartered in Tempe, Arizona and Prague, Czech Republic, Norton® is most known for its antivirus solutions, identity protection, performance management, and VPN solutions. Norton products generally only support Windows, macOS, and Android (which is actually a variant of Linux).

Kaspersky®

Headquartered in London, UK, the Kaspersky® company is another huge player in the antivirus space. It also has products in the identity protection and smart home monitoring space and has extensive Linux support.

Community resources

Both the Linux community and the security community offer the public a wealth of resources online. User groups, newsletters, documentation, best practices guides, and project websites are just the beginning of where you should start gathering data.

These resources cost you nothing but time and effort. In my book, that's time well spent. It's not lost on me as I typed that; that quote is going into my book. Yes, I actually repeated it to myself and laughed out loud. I stand by that position. Engage the community. Be part of the community.

The Center for Internet Security® (CIS®)

CIS Benchmarks are often considered the global go-to for how to configure a secure Linux system. Expect excellent, detailed configuration recommendations that range from easy to complex to apply to your solution. CIS® is a community-driven and supported non-profit organization whose goal is to help the global community build more secure solutions.

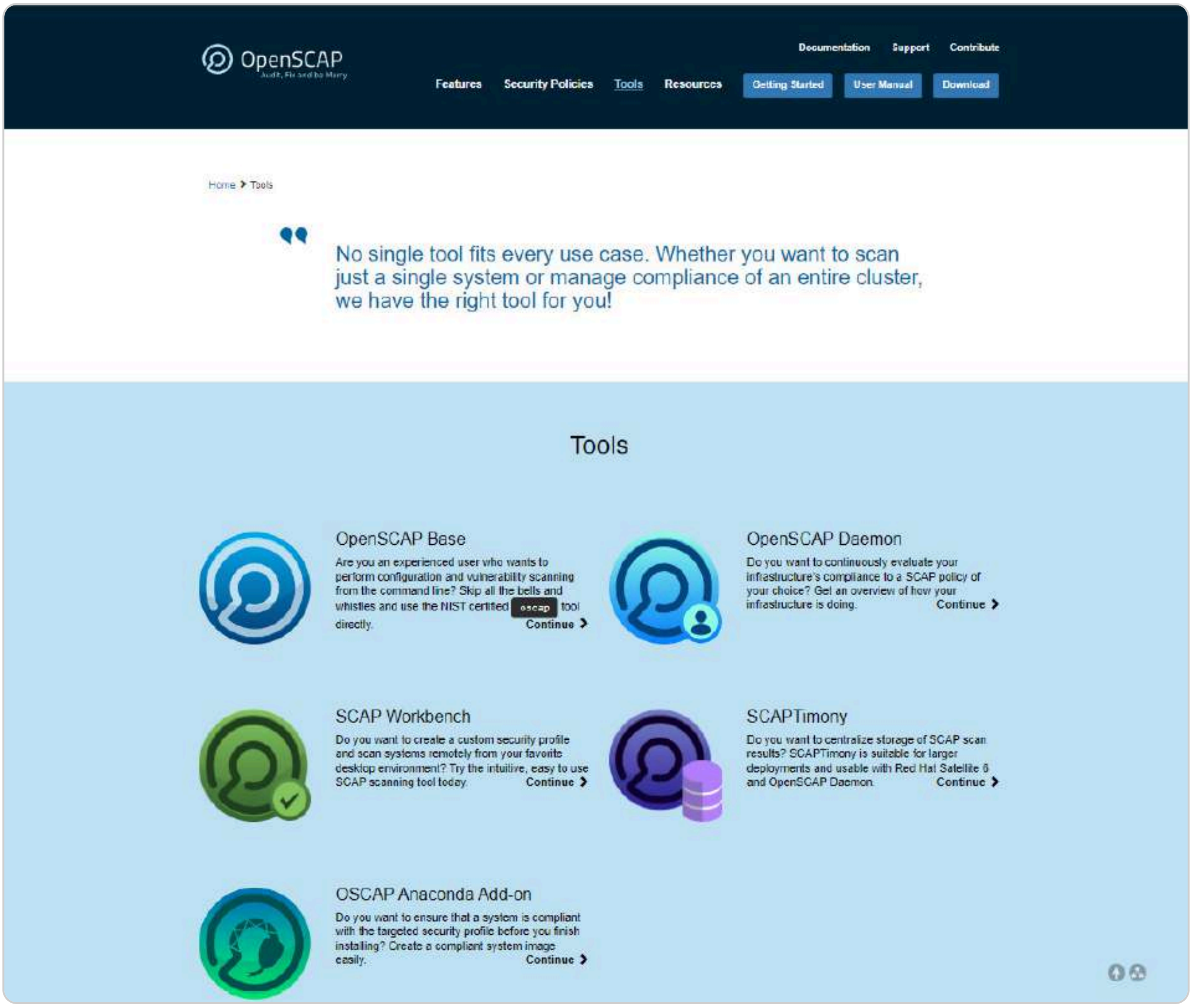
When I first began my journey into Linux security resources, seemingly a lifetime ago, this was my first waypoint. Maybe that's just because I consider myself a community-first kind of engineer. Time spent on this website is quality time. The CIS site is a wealth of information, security benchmarks, configuration recommendations, and even pre-built, ready-to-deploy systems images.

If government compliance is not your mandate, then the resources presented here alone may satisfy most or even all your needs. I make no assumptions on your behalf, so I recommend you form your own opinions and valuations while taking advantage of what CIS has to offer.

OpenSCAP

A key community resource is the OpenSCAP website. This site gives you the tools to scan your systems against industry standards and government or custom policies. It's truly an invaluable tool for your team's build chain. We will be talking more about this tool in chapters to come. Failure to use this tool or something similar may put your product in future peril. Choose wisely. I truly want you to succeed.

Here's a great example of some of the tools available on the OpenSCAP site:



No single tool fits every use case. Whether you want to scan just a single system or manage compliance of an entire cluster, we have the right tool for you!

Tools

OpenSCAP Base

Are you an experienced user who wants to perform configuration and vulnerability scanning from the command line? Skip all the bells and whistles and use the NIST certified `oscap` tool directly. Continue >

OpenSCAP Daemon

Do you want to continuously evaluate your infrastructure's compliance to a SCAP policy of your choice? Get an overview of how your infrastructure is doing. Continue >

SCAP Workbench

Do you want to create a custom security profile and scan systems remotely from your favorite desktop environment? Try the intuitive, easy to use SCAP scanning tool today. Continue >

SCAPTimony

Do you want to centralize storage of SCAP scan results? SCAPTimony is suitable for larger deployments and usable with Red Hat Satellite 6 and OpenSCAP Daemon. Continue >

OSCAP Anaconda Add-on

Do you want to ensure that a system is compliant with the targeted security profile before you finish installing? Create a compliant system image easily. Continue >

Figure 11.3 – Tools from OpenSCAP

Linux user groups

Linux user groups are an amazing way of getting involved with minimal effort. Attending these meetings can be a refreshing escape from marketing and salespeople. Mostly, it's engineers sharing stories with other engineers. I've led many of these over the years and I have found the attendees come from all walks of the technology field. Quite often, though, I find students, IT managers, and even executives quietly attending just to learn more about a new technology in a stress-free environment. Above all other community resources, a user group is the greatest way of connecting with those in the know who are actively involved in technology.

Security user groups

Security user groups (or cybersecurity user groups) are just as educational as their Linux-focused counterparts. They are often led by a product vendor but not always. These groups often tend to be more focused on an industry vertical or specific set of products. I highly recommend joining one or more. The knowledge you gain will be priceless and may help you preemptively prevent a product disaster.

Knowing what tooling your team uses along with what community resources they hope to use in the appliance is key knowledge and a perfect segue to the next section.

Knowing what vulnerabilities can impact your builds

Equally important to the tracking of your product contents for determining what is in each release, tracking said contents versus what vulnerabilities exist also gives you a carved-in-stone list of what to track from a bug list and security vulnerability perspective. These lists are not persistent, as dependencies can change with updates as some packages become deprecated. Tracking ends when your product ends. Security concerns never end. Hackers and bad actors never sleep. Invest in good software and better processes. Always, always, always know the ingredients for your recipe.

In earlier chapters, we reviewed in detail how to create a software manifest of what was in a specific build or release of your product. We've also reviewed hardware implications such as firmware and drivers. This knowledge gives you a virtual checklist of all the things you'll need to check for bugs and security issues along with the fixes for all of them. These lists become your starting point for what may be an arduous and never-ending investigation effort.

The average Linux distribution may have 7,000 or more packages to choose from. Does your security team want to track all 7,000 for you? Do they want you to use such a high number of packages? This is where people tend to rant and rave about minimizing the software footprint (i.e., only installing the bare minimum packages to allow the system to work properly). Once you track this for yourself, you'll appreciate the value of the minimal installation.

Not only does a smaller footprint make supporting the solution easier from an update perspective but it provides fewer avenues for possible attackers to go at your solution. So, I say use whatever you like in the initial prototyping phase but get surgical and exact in removing (or simply not installing) anything not needed. This exercise will bring you closer to your product and assist you further as we delve into keeping tabs on the components of your solution.

Running smart searches based on your components

It could be a full-time job just looking up all the possible issues that may impact on your product, so unless you have an unlimited budget, you and your team will need to work efficiently in this effort. Wherever possible, use automation to confirm that there might be an issue.

This might require you to create some scripts or a database of your components and feed that mechanism all the email updates, release notes, and newsletters you may get from community sources or from your vendors.

Without automation, this can be a truly tedious effort – one where something will eventually slip between the cracks. It happens even to the greatest teams. So, I highly recommend curating this data so your team can review any interesting findings on a regular basis. Knowing a software package has an open CVE attached to it that has yet to be fixed, in my opinion, is critical information. I'm pretty sure that your team would agree.

Sadly, there is no one pillar of truth that can help guide you here. The more sources you get bug and security issue data from, the better. Your vendors' email updates, which you hopefully signed up for, are the first place to start but not the last. Gather data from blogs, newsletters, and government compliance sites, as well. Together, you should be able to create a virtual security weather report for your product and tie that knowledge to your product's roadmap.

Store all the collected data as text. This is where a good set of scripts can take your manifest and see if any of your packages or hardware is impacted. Although it is a complicated process or workflow, you should be able to ultimately create your own database from which to report. A spreadsheet may be easier until you truly identify all aspects of data and data sources that you chose to collate. The CSV format can give you a jumpstart here too. Regardless of the myriad of ways to rummage through this data, creating a concise report at the component level will give you that desired virtual security weather report that you seek. This may have some blind spots because you're building this from known searchable data. Undiscovered zero-day exploits can still hurt us all.

Security scanning tools in the build chain are yet another source of data and a crucial one not to be ignored. Do not rely upon them 100%. Some vendors offer additional products to complement the scanning tools that they sell. These tools, like Tenable's Vulnerability Management suite, can assist your efforts and give you that bird's-eye view you seek.

I find myself in a quandary and dislike mentioning this but here's where I think in the very near future some creative usage of AI might be a good thing. So, indulge me for just a moment. As security experts are developing AI-enhanced security tools today, you can also believe that there are bad actors all over the world looking to leverage AI to take their exploits to a new level. That said, I say stay vigilant, and let's move on to how you can be part of the solution.

Being part of the solution

There are so many ways that you can contribute to the Linux and security communities.

By reading this book, you have already begun that journey if you haven't already dived deep into embedded Linux systems security. Welcome to the party and, again, thanks for buying my book. Don't let this be a single event for you. Check the book's GitHub (https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook) from time to time. It will get updates as things evolve too.

This community is perhaps as old as (or older than) many who are now looking for education, answers, or mentoring. Age has little relevance here, nor should it ever. What matters is the desire to learn or share knowledge and experiences. Security risks are equal opportunity offenders. If you don't prepare, they'll get you. Hence, we have a vibrant and diverse community of engineers, developers, testers, technical writers, and thought leaders. I won't be so pretentious to spell out a plan for how you should or should not get involved or contribute. All I, along with many others in the communities, hope for is that you take that big step to get more involved. In the next sections, you'll see just what I am talking about.

Contribute to the development process

You don't have to be a developer to be able to contribute here. You could volunteer to test early-access code, file bug reports, or even submit a Request For Enhancement (RFE). The feedback you provide will be crucial for those doing development.

Should you choose to be part of that process, there are several Linux communities that you could join. Each of these open source communities have its own process and guidelines for contributing and getting actively involved. Each has its own culture, but all will welcome you aboard.

Below, you'll find some links on how to join in the testing and development effort for several popular community Linux distributions:

- Fedora® (https://docs.fedoraproject.org/en-US/project/join/)
- CentOS Streams® (https://centos.org/)
- Ubuntu® (https://ubuntu.com/community/contribute/ubuntu-development)
- openSUSE® (https://en.opensuse.org/openSUSE:Heroes)

Who knows? Your contributions might just save others a lot of pain and hassle as well as solve your own Linux security issues. Wouldn't that make anyone feel just a little good about being an active community member? Let's move on to other ways to be active in the community.

My personal and work experience over the years has granted me a unique view into many community development projects. I'm also a member of the OpenStack community (also known as the OpenInfra community). For me, choosing a group to get involved with was no easy decision. There are many great groups. What I would recommend is simply to follow your heart. Join a community closest to your favorite Linux distribution. If that doesn't work out for you, at least consider joining a user group focused on the distribution(s) you use professionally, which is a perfect segue to our next section – user groups.

Join a user group

Meetups and **user groups** are the greatest ways of networking within a community, in my opinion. They are devoid of salesy pushes for products and focus on simply sharing creative solutions to real-world technology and security issues.

You don't need to be a rocket scientist or have some fancy title. Neither situation applies to me. I am just an average guy who has been blessed with some rather interesting knowledge and experiences over the expanse of my career. These groups are great for learning or sharing what you've learned. One day, you may even choose to get involved with the group's leadership or present a solution that you and your team devised. I truly love user groups and thoroughly enjoy sharing, when my schedule permits me.

Knowing everything, I think we can all agree, is in actuality totally impossible. For 30 years, I have been in the Linux community, and I still am learning new things all the time. Sometimes it's hard to convey the vastness, depth, and reach of Linux. One can meet, network with, and learn from others who may have focused on aspects of Linux that you may not have (and vice versa – you can help them too). My point is, never think that your experiences do not matter. They do. Sharing your experiences and solutions might just save someone else a truckload of pain.

Joining a user group presents you with a way to grow professionally and all it would ever cost you is your time and attention. Even if you attend and choose just to stay on the sidelines and observe, I assure you that learning opportunities will abound for you. One day, I bet, there will be a discussion, and you (YES, YOU!) will be the one with the answer someone has searched for. Perhaps that situation can also become a catalyst to where you want to participate more and be involved. Organizing, setting up, presenting, and leading a lightning talk on an impromptu technology subject – these are all ways you can get in the game. You may just find that you love user groups too.

Why do we do this? It's simple: sharing is caring… and caring can help build a community. Participate at whatever level you feel comfortable at but participate. The knowledge and networking possibilities are endless.

Leveraging MeetUp

MeetUp is a website that caters to empowering and connecting people of shared interests. Not just technology enthusiasts but also hikers, book lovers, animal activists, cyclists, and so many more interest groups (more than I can think of) can all be found there, and they are looking for you too.

Find a Linux user group or cybersecurity group near you: https://www.meetup.com/. This is an invaluable resource to connect you with hundreds of thousands of like-minded enthusiasts. I encourage you to try out MeetUp: you might just like it. I look forward to seeing you in a Linux meetup soon.

Summary

Thanks for driving onward with me on this amazing journey into what is available in the Linux world for cybersecurity resources and reference materials. As I mentioned earlier, please check the book's GitHub repository often for updates and new information, especially newly updated resources in the cheat sheet.

I trust you have enjoyed this chapter's departure from making you type and test new technologies. I know that the landscape changes from day to day. This is why I implore you to seek out as many trustworthy resources as possible.

We've touched briefly on a great many companies and resources in this chapter. You should now have a firm grasp on some of the most influential public and private resources (beyond your own vendors). Since you already know how to determine what comprises your solution as a whole, you are also now empowered on how to search for risks in those components. In closing, and most importantly, you now know that there's an extensive Linux and security-focused community ready to assist you and welcome you inside. They value your thoughts and feedback. Share and participate. It's truly mutually beneficial.

Let's now move on to our next chapter and return to some serious technical work. Get ready to understand how your appliances and peripheral devices communicate in the next chapter.

12
Are My Devices' Communications and Interactions Secure?

It can be said that the only really secure computer is one that is powered off and locked away in a vault. There's no fun nor usability in that so we must determine appropriate measures to limit exposure to vulnerabilities while still having the system be usable for the mission it was intended for.

In this chapter, you will learn how to determine the use cases and limitations of commonly used external buses for hardware communication. We'll review network security with firewalls. We'll work through some hands-on exercises in locking down your web-based services with SSL certificates. We will dive into the gotchas of legacy hardware and software and close out this chapter with the security validation of your appliance.

"Why is this important to my project?" you may ask... Simply put, you might lock down the system itself incredibly well, however, that system's connections to peripherals or other applications may not exactly be as secure as you might have thought. To be aware is to be forewarned.

The main sections of this chapter are:

- Bus types and issues
- Enhancing security with certificates
- Confirming that your networking is secure
- Limitations of legacy hardware and software
- Validating your solution before shipping

Let's move on to our first section, where we'll take a look at different system buses.

Technical requirements

To complete the exercises in this chapter, you will need root-level access to the web server you built as a custom DNF repository back in *Chapter 5*.

If you have deleted that server/work, you will need to build a new custom repository server as detailed in those previous exercises before beginning this exercise.

Bus types and issues

So, what is a **bus**? Most simply put, a bus is a type of communication channel to and from devices and your system's CPU. With this channel, data signals from (or to) the device can be processed by the CPU. Without this line of communication, the attached device will not function.

A more layman's way of describing a bus would be to break it down in the least technical of terms. Let's look at the forest for the trees. In total, a bus is the culmination of hardware, software, and the cabling required to allow for data transmission. I'll use a non-technical childhood example. Ever take two tin cans and some string to make a communications system? Those two cans were connected by a string that, when tightly stretched, carried your voice (i.e., the data) from one can (peripheral) to the other can (main processor) and was interpreted by the end-user on the receiving end.

Systems buses do this in a vastly more technical manner, but the result is the same. Data travels from a peripheral to the CPU and gets processed. For my super technical readers, please let me go a little deeper. Various buses are in play when sensors, cameras, controls, or other directly attached peripherals are attached to a system and their interactions are transmitted through their connection medium and then, finally, received and processed by the system CPU. There are several standard buses, but in this chapter, we'll cover the most important ones that you may come across while working with embedded Linux systems, and by that, I mean the **CAN bus** and **USB**. In the next few sections, we will focus on system buses (along with standard connection types and protocols) that leverage external connections to your hardware. As we continue, I want you to envision all things that may be able to connect to your solution, with or without your permission. Sometimes, as we previously inferred in the section about childproofing your solution within *Chapter 10*, you must almost assume the end-user is a toddler who might just shove their sandwich into the VCR. I know I'm dating myself by mentioning a VCR but it's to make a point and I hope that point is well received. Plan for the worst, and hope for the best.

What could possibly go wrong? Well, if your system uses an insecure bus or peripheral connection and a bad actor has physical access to said connection, it may be hijacked without your knowing (or worse). When insecure connections are leveraged, physical security must also be part of the design consideration and deployment plan.

Let's also hope that your awesome solution has a custom shell or case that covers up any ports you do not want the end-user to know are there. It's rare when this is an option, but I felt the necessity to mention it. Plus, a custom case also makes a solution look better. Some even have built-in weather resistance. Let your use case guide your design.

IMPORTANT NOTE

> *For an embedded Linux system, we'll assume the case has a level of tamper-proofing even if it's just a warning sticker that states all warranties are void if the case is opened. For this reason, we will not be covering system hardware internals like the well-known* ***PCI bus****... There are countless texts dedicated to that hardware and kernel development, so let's stay focused on the external connections in play.*

That all said, let's move on to a household name. A bus and connection type we all know – **Universal Serial Bus** (**USB**).

USB

Since its initial introduction in 1996, the USB interface has been the worldwide standard connectivity method of choice for virtually all consumer devices. Modern USB ports have such versatility (depending on the device), which is why they are everywhere in daily life.

USB empowers computers to connect to mice, keyboards, storage devices, printers, gaming controllers, authentication keys, and many other types of commercial hardware. The same port can also charge some devices like cell phones, tablets, or portable power units. USB can even be used to add multiple displays to a system.

USB does not protect your data while in transit. Most people choose to encrypt USB-attached storage devices but by default when accessing the data or processing it. That data in transit is not encrypted. This limitation is not limited to Linux systems but all systems leveraging the USB framework.

There are some commercially available solutions to secure this data in transit; however, they are not open source and definitely not free. They are sold as enterprise-grade data loss prevention systems and support multiple platforms (not just Linux). You will need one of these third-party solutions if the securing of your data in transit to USB storage is an issue.

Let's now move on to our next section where we will review USB connections.

USB connectors

The USB family of **connectors** has evolved over the years, but since the beginning and still to this day, USB-A might be the world's most common connection interface ever created. It can be found in household wall outlets, PCs, laptops, external storage devices, on the back of passenger aircraft seats, in the dashboards of cars and recreational vehicles (RVs), solar generators, power banks, and so, so, so much more… (Yeah, I know that's not grammatically correct but I really want to emphasize that USB-A is *E-V-E-R-Y-W-H-E-R-E !!*)

What's ironic about USB is that virtually all the shown connectors are still being actively used today. Granted, there's a difference in the quality of the materials used now vs. the older versions, but that's there to simply allow for better data transfer rates. The cables themselves are not a security issue per se, as the issues reside in the chips and the USB bus itself. Let's take a quick look at those connector types.

From left to right in the following figure, we have USB-B, USB-A, USB-mini, USB-micro, and USB-C:

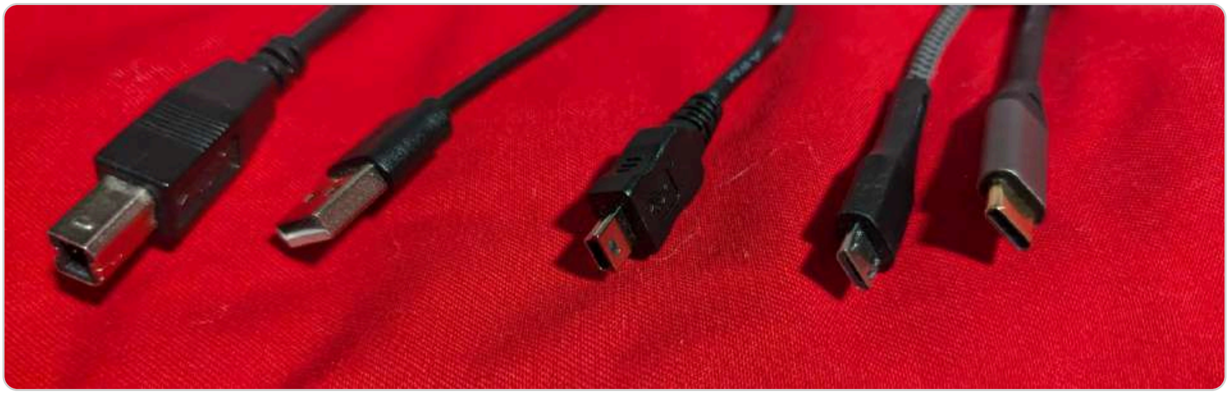


Figure 12.1 – USB connectors over the years

In pondering how to compare the various versions of the USB standard that you will come across, I felt it imperative to throw some critical data in a table format for ease of viewing. Although the connectors may all be compatible, the speeds and throughput of the different versions are vastly and mind-blowingly different.

Here's a table of USB versions, performance, and connectors in use today:

USB Version	**Bandwidth**	**Data Transfer Speed**	**Connectors**
USB4	20 or 40 Gbps	2560–5120 MB/s	C
USB 3.2	20 Gbps	2560 MB/s	C
USB 3.1	10 Gbps	1280 MB/s	A, C
USB 3.0	5 Gbps	640 MB/s	A, B. C
USB 2.0	480 Mbps	60 MB/s	A, B, Mini, Micro
USB 1.1	12 Mbps	1.5 MB/s	A, B, Mini, Micro
USB 1.0	1.5 Mbps	1.5 MB/s	A, B, Mini, Micro

Table 12.1 – Speed and capabilities of different USB versions

As you can see from the above table, the wide range of connections of the USB standard has greatly evolved in the past 30 years. USB has been a game changer and a trendsetter and will continue onward for many more years to come. Let's move on by looking at its predecessor, the serial bus, next.

Serial port

Known for being notoriously insecure, **serial ports** have been providing text terminal access since the old Unix days. There are still many serial devices that are used every day. Keyboards and mice are still quite commonplace (nowadays using USB) and may also be part of your solution. Serial connections are still incredibly common for engineers to use from their laptops to networking equipment such as switches, firewalls, and routers. Today, that often requires an adapter device that connects to an engineer's laptop via USB and then to a serial cable that connects to a serial port on the device to be managed. These devices will automatically establish connections once attached to your Linux system with no effort required from you.

Serial ports (more commonly called terminal ports) can be physical or virtual. Most physical serial connections today are auto-negotiated through the USB bus; however, on older systems, those devices may have used a PS2 connection or the older serial port otherwise known as **DB9**. Serial ports may also be used by industrial automation systems, printers, scanners, and some medical devices. The following figure shows the serial connection female connector (left) and the serial connection male (right):

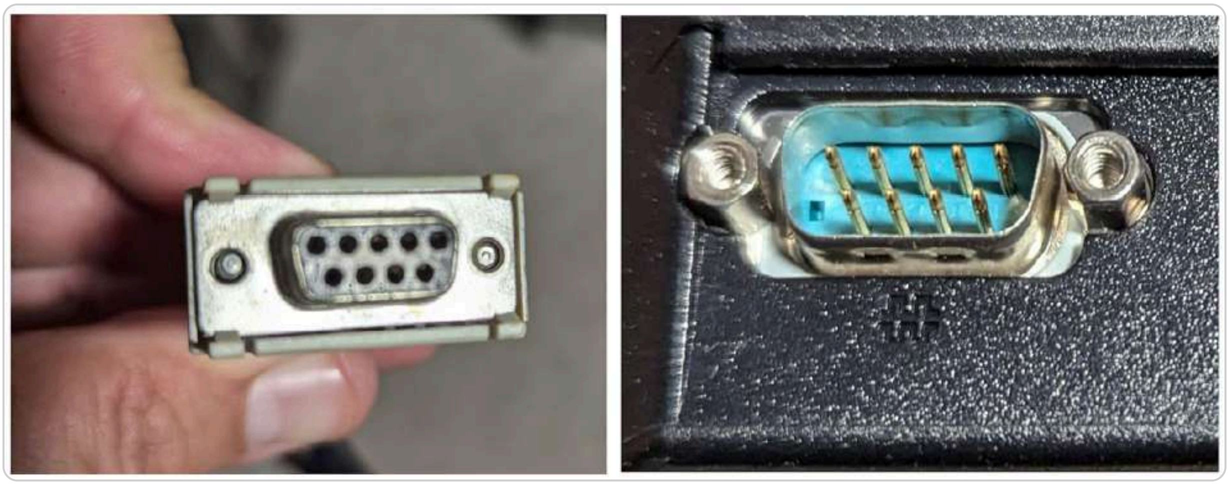


Figure 12.2 – Serial connector examples

Linux systems can also have **virtual serial connections**. It's important to know how your system is mapping these connections. First, why should you care? Well, this directly impacts how many terminal sessions can be directly established with your system concurrently. If your appliance is a console/text-based appliance, this metric is crucial in its management and control. In the simplest terms possible, this metric denotes how many doors can be opened at once. When not crucial to your appliance's user experience, these TTY console sessions should be limited as much as possible without breaking the operation of the appliance.

To get a great sense of just how many TTY sessions your system starts at boot time, here's an example command that will show you how your system has these ports named and mapped. Consider each one an open door that either should be locked, boarded up, or simply guarded.

Let's see how many TTY sessions are possible in our current system:

```
$ ls -l /sys/class/tty/*
```

The output for this one will definitely be lengthy so I have truncated it... What you will see will be an exhaustive list of physical and virtual terminal port possibilities.

```
      ((( output truncated )))
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS4 ->
../../devices/pci0000:00/0000:00:16.3/0000:00:16.3:0/0000:00:16.3:0.0/tty/ttyS4
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS5 ->
../../devices/platform/serial8250/serial8250:0/serial8250:0.5/tty/ttyS5
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS6 ->
../../devices/platform/serial8250/serial8250:0/serial8250:0.6/tty/ttyS6
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS7 ->
../../devices/platform/serial8250/serial8250:0/serial8250:0.7/tty/ttyS7
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS8 ->
../../devices/platform/serial8250/serial8250:0/serial8250:0.8/tty/ttyS8
lrwxrwxrwx. 1 root root 0 Aug 25 09:33 /sys/class/tty/ttyS9 ->
../../devices/platform/serial8250/serial8250:0/serial8250:0.9/tty/ttyS9
```

Knowing how to track these connections is crucial for maintaining device security. Let's move on to yet another bus that also uses the DB9 connector – the CAN bus.

The CAN bus

The **Controller Area Network** (**CAN**) bus is a vehicular standard. Its main purpose is to control communication between **electronic control units** (**ECUs**). It was first introduced back in 1983; however, it still has no standard connector, although the DB9 connector seems to be a de facto standard today.

As cars nowadays are becoming rolling data centers (even the non-self-driving ones), I felt it important to mention this bus type as many manufacturers are jumping to build embedded systems in this evolving space.

However, CAN is not limited to automobiles. CAN bus architectures can be seen deployed in maritime environments, agricultural equipment, and even in large buildings to control elevators, escalators, and other building automation. It's even being leveraged in modern robotics.

Sadly, CAN is not encrypted. Be mindful and be aware. Another problem with CAN is the lack of authentication. Ultimately, this means whoever can access the bus has control of the bus. All security be abandoned. What could possibly go wrong?

There's a whole community around hacking cars' control systems. Most who engage in this journey do so to squeeze a little more performance out of their own vehicle. No malicious intent there, just an engineer's curiosity. That said, a bad actor could easily access a car's locks, GPS records, or controls, or possibly conduct other malicious activities.

Another factor, just like with USB and serial, is that the lack of encryption means any credentials passed across the bus are done in plain text. Anyone watching traffic on the bus can see that data and potentially use it to do less than nice things.

As more and more vehicles rely on electronics to create a better driver's experience, let's hope that the CAN standard itself can get some much-needed security added in its next version.

Let's move on to a key component in setting up encryption – creating certificates.

Enhancing security with certificates

Certificates are used for many facets of secure communications globally. Your hardware and software vendors often digitally sign their drivers and operating system packages. Trusted authorities use certificates to ensure you can safely use DNS and the internet. The list of use cases for certificates is significant. Generally, these security certificates are obtained through a global **certificate authority** (**CA**) organization. Many internet service providers (ISPs) also have the ability to grant their customers an SSL certificate.

These shared system certificates enable things like NSS, TLS, OpenSSL, and applications to have a joint shared source of trust via a system certificate (trust anchor). Not having an SSL certificate will prevent you from using an encrypted web server.

There is also the ability in Linux to create a self-signed certificate. These are generally good enough for internal lab work but are considered insufficient in a production environment and definitely not a good idea for your product.

In thinking of where I can show you an example suitable for lab or demo usage, I came up with this thought, "Where might we consider using a certificate? Your DNF repository server?"

Let's explore that thought with some hands-on exercises.

Exercise 1: Creating a self-signed certificate

In this hands-on exercise, we'll create a self-signed certificate that we could possibly use to encrypt a web server. Let's begin:

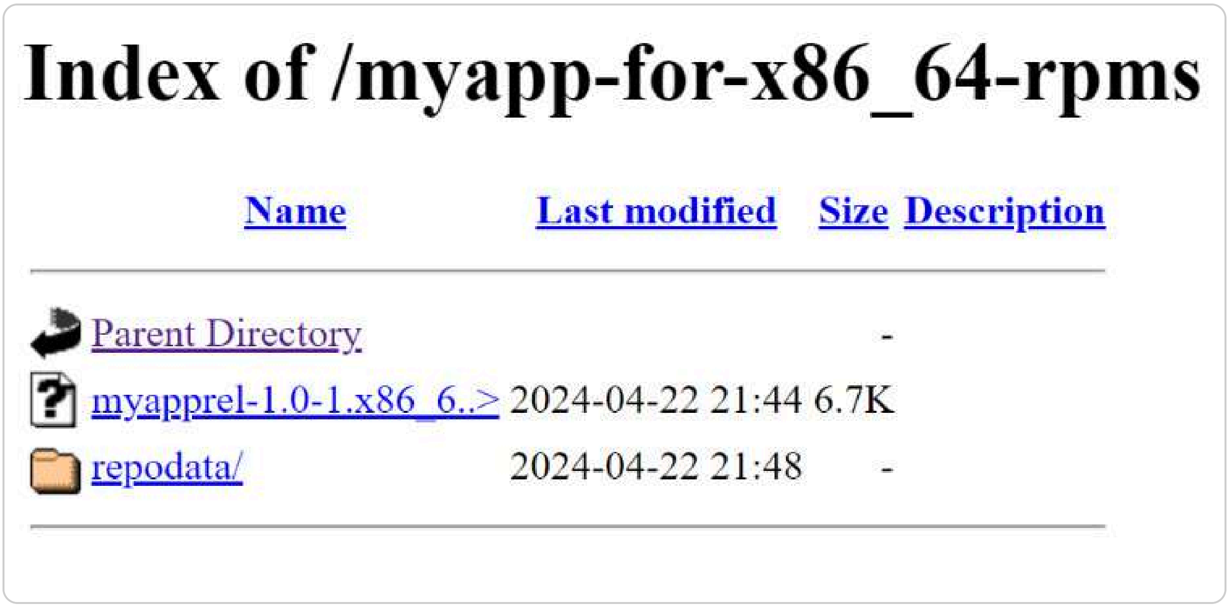
1. First, we'll verify that the web server is hosting our repository as previously configured in *Chapter 5*.

```
$ sudo systemctl status httpd.service
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset:
disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: active (running) since Sun 2024-10-06 17:53:55 EDT; 2min 43s ago
       Docs: man:httpd.service(8)
   Main PID: 1215 (httpd)
     Status: "Total requests: 1; Idle/Busy workers 100/0;Requests/sec: 0.00617;
Bytes served/sec:  12 B/sec"
      Tasks: 178 (limit: 38323)
     Memory: 21.2M (peak: 22.4M)
        CPU: 225ms
CGroup: /system.slice/httpd.service
             ├─1215 /usr/sbin/httpd -DFOREGROUND
             ├─1294 /usr/sbin/httpd -DFOREGROUND
             ├─1295 /usr/sbin/httpd -DFOREGROUND
             ├─1296 /usr/sbin/httpd -DFOREGROUND
             ├─1297 /usr/sbin/httpd -DFOREGROUND
             └─1298 /usr/sbin/httpd -DFOREGROUND
 Oct 06 17:53:55 bm02.local systemd[1]: Starting httpd.service - The Apache HTTP
Server...
Oct 06 17:53:55 bm02.local (httpd)[1215]: httpd.service: Referenced but unset
environment variable evaluates to an >
Oct 06 17:53:55 bm02.local httpd[1215]: Server configured, listening on: port 443,
port 80
Oct 06 17:53:55 bm02.local systemd[1]: Started httpd.service - The Apache HTTP
Server.
```

IMPORTANT NOTE

You'll need to substitute your hostname or IP address that you configured for your system.

2. Confirm that the service is hosting your repository via your web browser.



Index of /myapp-for-x86_64-rpms

```
Name                      Last modified      Size  Description
Parent Directory                              -
myapprel-1.0-1.x86_6..>   2024-04-22 21:44   6.7K
repodata/                 2024-04-22 21:48    -
```

Figure 12.3 – Verifying your repository via a web browser

3. We will now begin setting up the environment by setting a system variable. For this next step, replace **`<hostname>`** with your system's FQDN. As root, execute the following command.

```
# export ssl_name=<hostname>
```

4. Now we'll begin to set up the certificate information by creating our initial **`.pem`** file, which will store your certificate information.

```
# openssl genrsa -out ${ssl_name}.pem 4096
```

5. Let's now verify that the last command generated our new **`.pem`** file.

```
# ls -l *.pem
-rw-------. 1 root root 3272 Aug 25 21:12 bm02.local.pem
```

6. This next command will be a very interactive one. You will be prompted to answer some questions. Substitute your own responses when prompted, as mine are displayed here just for reference. I used **`EmbeddedBook`** as the challenge password at the end. Please feel free to use your own challenge password (but do make a note of it for later). As root, run the following command:

```
# openssl req -new -key ${ssl_name}.pem -out ${ssl_name}.csr
```

Let's see the interactive prompts/output:

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:MA
Locality Name (eg, city) [Default City]:Wilmington
Organization Name (eg, company) [Default Company Ltd]:Embedded Security Book
Organizational Unit Name (eg, section) []:Development
Common Name (eg, your name or your server's hostname) []:bm02.local
Email Address []:admin@bm02.local
```

```
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:EmbeddedBook
An optional company name []:EmbeddedBook
```

7. Let's run a quick command to verify that the last command generated our `.csr` file properly.

   ```
   # ls -l *.csr
   ```

   Output:

   ```
   -rw-r--r--. 1 root root 1850 Aug 25 21:22 bm02.local.csr
   ```

   This CSR file is what you would be sending to your CA if you were procuring an enterprise SSL certificate. We can also leverage this to create a temporary self-signed certificate locally.

8. Let's create a temporary cert that's good for 90 days.

   ```
   # openssl x509 -req -days 90 -in ${ssl_name}.csr \
   -signkey ${ssl_name}.pem \
   -out ${ssl_name}.cert
   ```

   Your output should look something like the following:

   ```
   Certificate request self-signature ok
   subject=C=US, ST=MA, L=Wilmington, O=Embedded Security Book, OU=Development,
   CN=bm02.local, emailAddress=admin@bm02.local
   ```

9. Verify that all three files are there now. Replace `<hostname>` with your system's hostname. Mine is shown just for example purposes.

   ```
   # ls -l *<hostname>*
   ```

   Your output should contain at least the files shown that are identified by your system's hostname.

   ```
   -rw-r--r--. 1 root root 2110 Aug 25 21:30 bm02.local.cert
   -rw-r--r--. 1 root root 1850 Aug 25 21:22 bm02.local.csr
   -rw-------. 1 root root 3272 Aug 25 21:12 bm02.local.pem
   ```

Wow, wasn't that easy?! You've now created your first self-signed server certificate. Please never use that in production unless there is no other way of securing your web service. Now that we've created our certificate, let's put it to good use in our next exercise by attaching it to our web service.

Exercise 2: Adding a certificate to your custom repository server

In this exercise, we'll install your previously created self-signed certificate into the web server you built previously to host your custom DNF repository. Once completed, the web server itself will default to leverage the HTTPS protocol versus the previous default of HTTP. The change in protocol means you'll also need to ensure that the firewall rules have also been updated. Let's get started.

1. As root, install `mod_ssl`.

   ```
   # dnf install -y mod_ssl
   ```

Your output may be lengthy, but it should look somewhat like this – I have truncated the output to save space.

```
((( truncated output )))
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :

         1/1
  Installing       : mod_ssl-1:2.4.62-2.fc40.x86_64

         1/1
  Running scriptlet: mod_ssl-1:2.4.62-2.fc40.x86_64

         1/1
Installed:
  mod_ssl-1:2.4.62-2.fc40.x86_64
Complete!
```

2. We'll stage the requisite files that we created in the last exercise into the appropriate directories as root. Remember to substitute your own **`.cert`** file (mine is shown as an example).

```
#  cp bm02.local.cert /etc/pki/tls/certs/
```

3. Next, we'll copy our **`.pem`** file to the proper directory. Remember to substitute your own **`.pem`** file (mine is shown as an example).

```
# cp bm02.local.pem /etc/pki/tls/certs/
```

4. We'll also copy the **`.pem`** file to an additional directory. Remember to substitute your own **`.pem`** file (mine is shown as an example).

```
# cp bm02.local.pem /etc/pki/tls/private/
```

5. Next, we'll edit the **`/etc/httpd/conf.d/ssl.conf`** file as root. Follow the instructions on which lines to uncomment and edit. When done with the listed changes, save and exit the editor.

```
$  sudo vi /etc/httpd/conf.d/ssl.conf
```

Once in your editor, perform the following edits to the file, and do not forget to save it at the end:

I. Uncomment this line.

```
DocumentRoot "/var/www/html"
```

II. Uncomment and change this line to reflect your hostname: **`443`**.

```
#ServerName www.example.com:443
ServerName bm02.local:443
```

III. Change this line to your own CRT file – mine is shown for example.

```
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateFile /etc/pki/tls/certs/bm02.local.cert
```

IV. Change this line to your own PEM file – mine is shown for example.

```
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
SSLCertificateKeyFile /etc/pki/tls/private/bm02.local.pem
```

V. Uncomment and change this line to your own **`.pem`** file – mine is shown for example.

```
#SSLCertificateChainFile /etc/pki/tls/certs/server-chain.crt
SSLCertificateChainFile /etc/pki/tls/certs/bm02.local.pem
```

VI. Save the file and exit the editor.

6. Now, all that remains is to restart the **`httpd`** service as root and test. Run the following commands as root.

```
# systemctl restart httpd
```

7. Let's validate that the **`httpd`** service is running as expected.

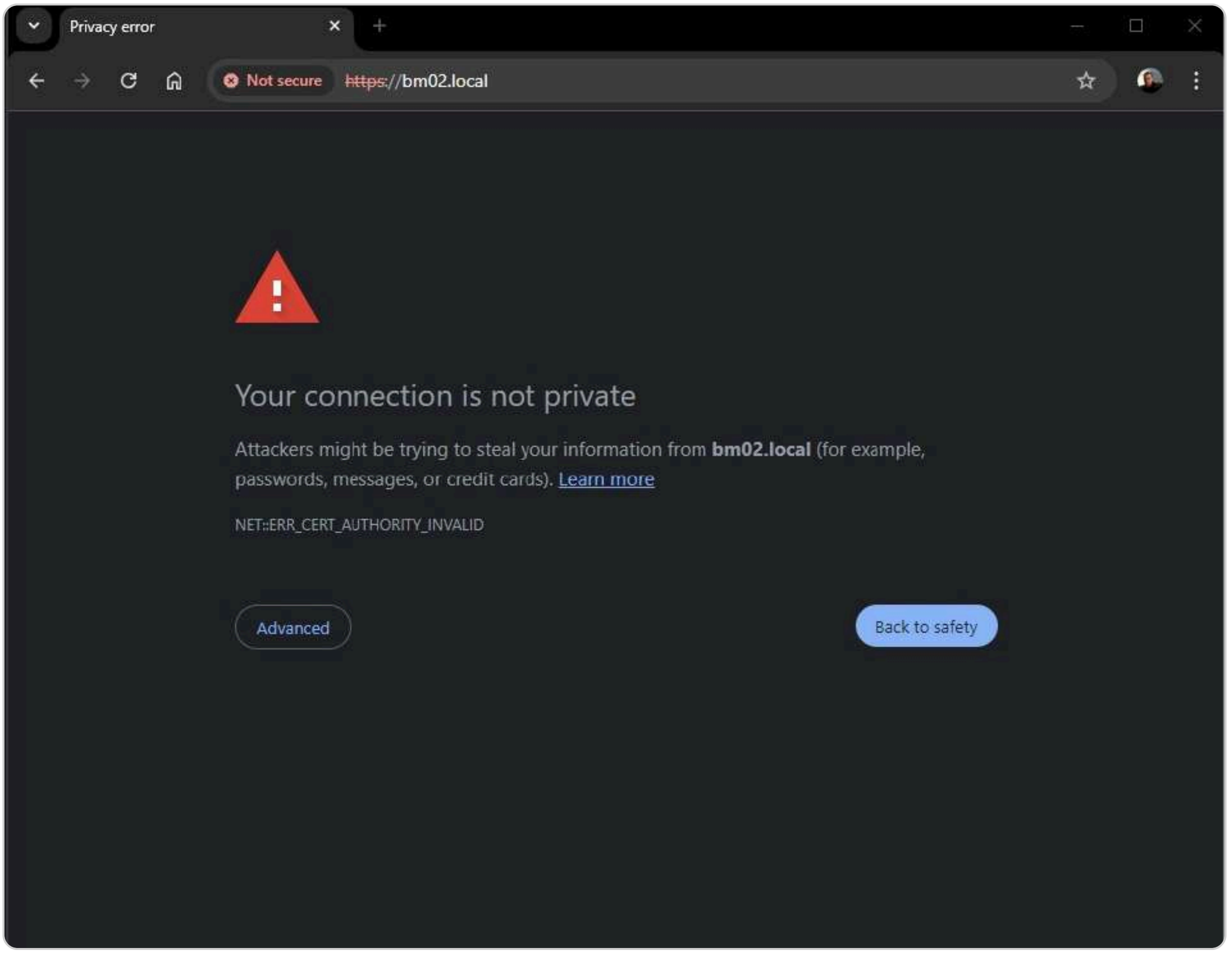
```
# systemctl status httpd
```

Your output should resemble something like this:

```
● httpd.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset:
disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: active (running) since Sun 2024-08-25 22:14:15 EDT; 8s ago
       Docs: man:httpd.service(8)
   Main PID: 4907 (httpd)
     Status: "Started, listening on: port 443, port 80"
      Tasks: 178 (limit: 38323)
     Memory: 16.2M (peak: 17.8M)
        CPU: 115ms
     CGroup: /system.slice/httpd.service
             ├─4907 /usr/sbin/httpd -DFOREGROUND
             ├─4908 /usr/sbin/httpd -DFOREGROUND
             ├─4909 /usr/sbin/httpd -DFOREGROUND
             ├─4910 /usr/sbin/httpd -DFOREGROUND
             ├─4911 /usr/sbin/httpd -DFOREGROUND
             └─4912 /usr/sbin/httpd -DFOREGROUND
Aug 25 22:14:15 bm02.local systemd[1]: Starting httpd.service - The Apache HTTP
Server...
Aug 25 22:14:15 bm02.local (httpd)[4907]: httpd.service: Referenced but unset
environment variable evaluates to an e>
Aug 25 22:14:15 bm02.local httpd[4907]: Server configured, listening on: port 443,
port 80
Aug 25 22:14:15 bm02.local systemd[1]: Started httpd.service - The Apache HTTP
Server.
```

8. Let's open up a browser and, from a different system, go to your newly upgraded server (using your own hostname and **`https`**). Since we have used a self-signed certificate, your browser will definitely not like it, and it will make you confirm that you wish to proceed to that site. Some browsers are more forgiving than others (regardless of the operating system they are run on).

   Confirm in your browser window that you wish to continue by clicking the **Advanced** button and then proceed to the site when prompted. This will be a good test to see that the server itself is running via **`https`** and the self-signed cert.



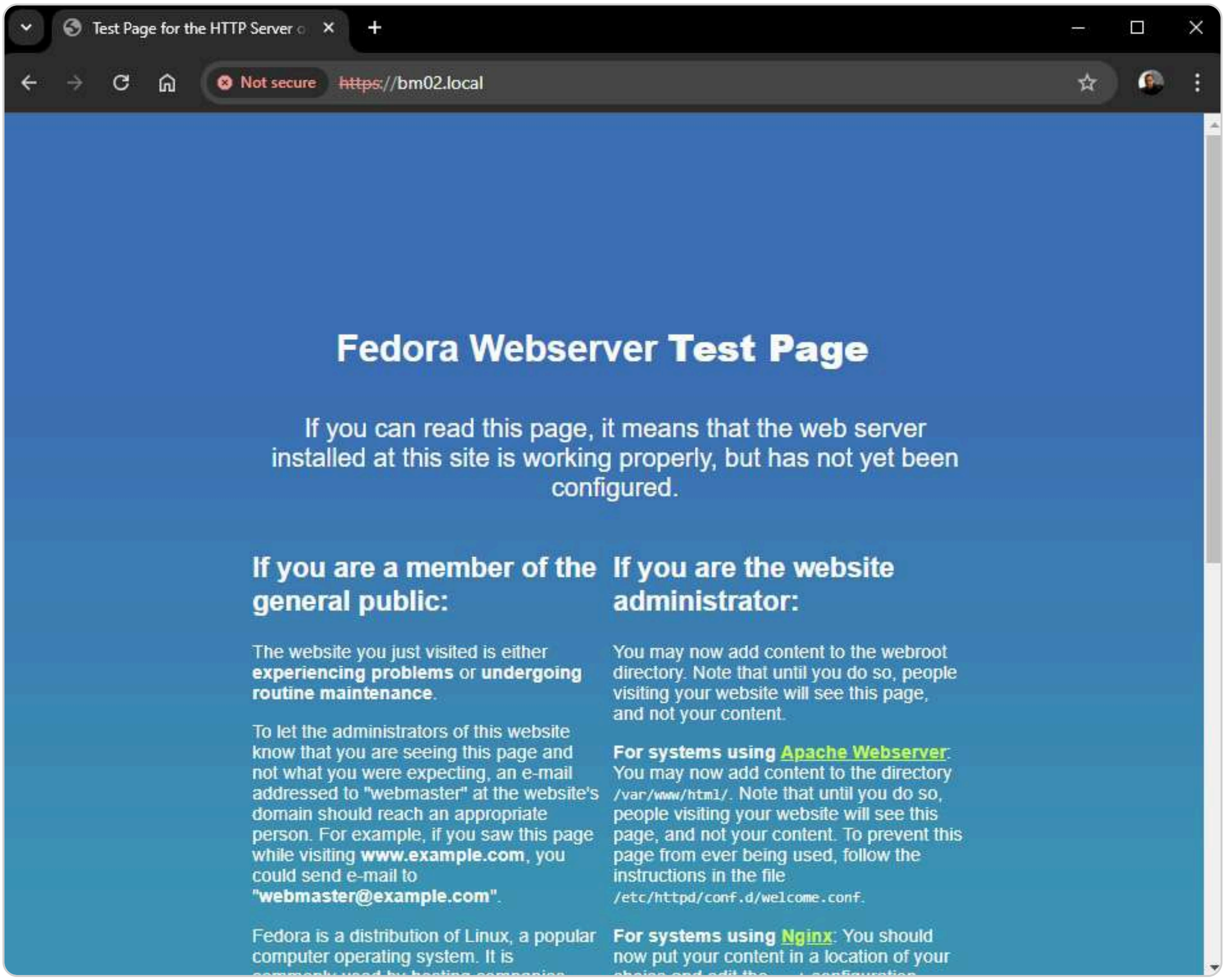
Your connection is not private

Attackers might be trying to steal your information from **bm02.local** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_AUTHORITY_INVALID

Figure 12.4 – SSL warning because of the self-signed certificate

If all has worked as planned, you should have been able to get to your newly encrypted web server and the page should look something like this:



Fedora Webserver Test Page

If you can read this page, it means that the web server installed at this site is working properly, but has not yet been configured.

If you are a member of the general public:

The website you just visited is either **experiencing problems** or **undergoing routine maintenance**.

To let the administrators of this website know that you are seeing this page and not what you were expecting, an e-mail addressed to "webmaster" at the website's domain should reach an appropriate person. For example, if you saw this page while visiting **www.example.com**, you could send e-mail to **"webmaster@example.com"**.

Fedora is a distribution of Linux, a popular computer operating system. It is

If you are the website administrator:

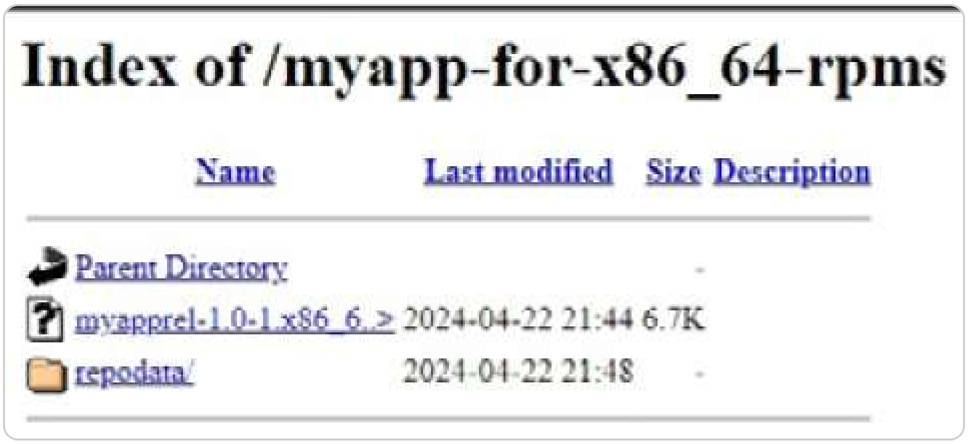
You may now add content to the webroot directory. Note that until you do so, people visiting your website will see this page, and not your content.

For systems using Apache Webserver: You may now add content to the directory `/var/www/html/`. Note that until you do so, people visiting your website will see this page, and not your content. To prevent this page from ever being used, follow the instructions in the file `/etc/httpd/conf.d/welcome.conf`.

For systems using Nginx: You should now put your content in a location of your

Figure 12.5 – HTTPS main test page

9. Now let's test the repository URL. Remember to use HTTPS (not HTTP) in the URL. You may be prompted with the same SSL cert warnings. Drive on and confirm it's OK (substitute your own URL – mine is shown again as an example).



Index of /myapp-for-x86_64-rpms

Name	Last modified	Size	Description
Parent Directory		-	
myapprel-1.0-1.x86_6..>	2024-04-22 21:44	6.7K	
repodata/	2024-04-22 21:48	-	

Figure 12.6 – Our custom DNF repository now uses HTTPS !!!

So, now that we've walked through the creation of a self-signed certificate and then taken it to make our custom DNF repository SSL-enabled, I want to impart to you yet another consideration before we close.

If your solution is going to use HTTPS, please build into your interface a way for the end-user to click a button and have this process automated for them. Even better, add in the option for the appliance to automatically ingest keys and a certificate from a CA in your interface as well. These efforts are in line with the tamper-proofing and user experience topics that were covered in *Chapter 10*. We should always strive to provide our end-users with a positive user experience when using our appliance.

In this section, we secured the HTTPS service with encryption and a certificate. Let's move on to our next section, where we will review tools for securing all of the actual network connections to services and ports of your appliance.

Confirming that your networking is secure

Almost all machines at some level communicate with other machines. In the embedded Linux systems space, this may not always be true. Some appliances are simply standalone solutions that are not connected to any network. In other use cases, the security posture of where they may reside might dictate that they are on a heavily restricted LAN segment that has limited access to other systems and no access whatsoever to the internet or other segments of the enterprise.

Network configuration and securitization are intrinsic to basic Linux systems administration. But we are not designing systems to reside within our own datacenter or network. We are building products that will reside in a customer's ecosystem, whatever level of security that might entail. Our customers have placed a level of trust and responsibility in our hands to ensure that what we deliver to them is already locked down and secure. Oftentimes, more so than what their own limited staff might have had the skills to perform. That's why they are buying your solution.

In the next few blocks, we'll very briefly cover command-line and graphical tools that I truly hope you already know. My reasoning is blunt and simple. Use what works for your team, your environment, and your company's build chain style. I am actually hoping that this section is redundant and unnecessary for you. Even better if it actually helps a little. Let's move on to do a quick review of how we can prepare our appliances (somewhat) for their new homes by using tools to secure what traffic may pass in our next section covering firewalls.

Firewalls

Here's a quick breakdown of some of the major settings groups for Linux firewalls:

- **Connection**: This is a defined *named* network connection.
- **Zone**: A firewalld zone is a predefined level of trust for you to leverage.
- **Interface**: This one is pretty self-explanatory. It's the interface that you plan to apply new settings onto.
- **Services**: These are predefined services that can be made accessible. Having them predefined absolutely helps in the configuration process as memorizing every service to port mapping and whether it's UDP or TCP can be daunting.
- **Ports**: This is a fast way of defining multiple ports or ranges to make them accessible.
- **Configuration**: There are two possible selections for the state of the configuration that you may be editing – **runtime** and **permanent**. Runtime state changes are lost after a reboot or after a restart of the **firewalld** service.
- **IPSets**: These are whitelists or blacklists that you can manage and easily deploy.

There are several different ways of configuring a Linux firewall (firewalld). Let's browse through some examples.

The command line

The command-line tools for managing firewalls are virtually always installed for you when you build a new system. The main tool we'll highlight here is **firewall-cmd**. As you can see from the *massively truncated* listing of all the possible settings, firewall-cmd is a feature-rich toolset. It can easily be scripted as well.

For an example of all the options, one could run the following command in a terminal.

```
$ firewall-cmd --help
```

The output of that command is rather lengthy. I'd recommend that you study it on your own machine, pipe the output to the more or less command, and then page your way through it.

```
Usage: firewall-cmd [OPTIONS...]
General Options
  -h, --help        Prints a short help text and exits
  -V, --version     Print the version string of firewalld
  -q, --quiet       Do not print status messages
Status Options
  --state           Return and print firewalld state
((( output truncated )))
Options to Handle Bindings of Sources
  --list-sources    List sources that are bound to a zone
                    [P] [Z]
  --add-source=<source>[/<mask>]|<MAC>|ipset:<ipset>
                    Bind the source to a zone [P] [Z]
  --change-source=<source>[/<mask>]|<MAC>|ipset:<ipset>
                    Change zone the source is bound to [Z]
  --query-source=<source>[/<mask>]|<MAC>|ipset:<ipset>
                    Query whether the source is bound to a
                    zone [P] [Z]
  --remove-source=<source>[/<mask>]|<MAC>|ipset:<ipset>
                    Remove binding of the source from a
                    zone [P] [Z]
Helper Options
```

```
  --new-helper=<helper> --module=<module> [--family=<family>]
                     Add a new helper [P only]
  --new-helper-from-file=<filename> [--name=<helper>]
                     Add a new helper from file with optional name [P only]
  --delete-helper=<helper>
                     Delete an existing helper [P only]
  --load-helper-defaults=<helper>
                     Load helper default settings [P only]
  --info-helper=<helper> Print information about an helper
(((output truncated)))
Lockdown Options
  --lockdown-on       Enable lockdown.
  --lockdown-off      Disable lockdown.
  --query-lockdown    Query whether lockdown is enabled
Lockdown Whitelist Options
  --list-lockdown-whitelist-commands
                      List all command lines that are on the
                      whitelist [P]
  --add-lockdown-whitelist-command=<command>
                      Add the command to the whitelist [P]
(((output truncated)))
Panic Options
  --panic-on          Enable panic mode
  --panic-off         Disable panic mode
  --query-panic       Query whether panic mode is enabled
```

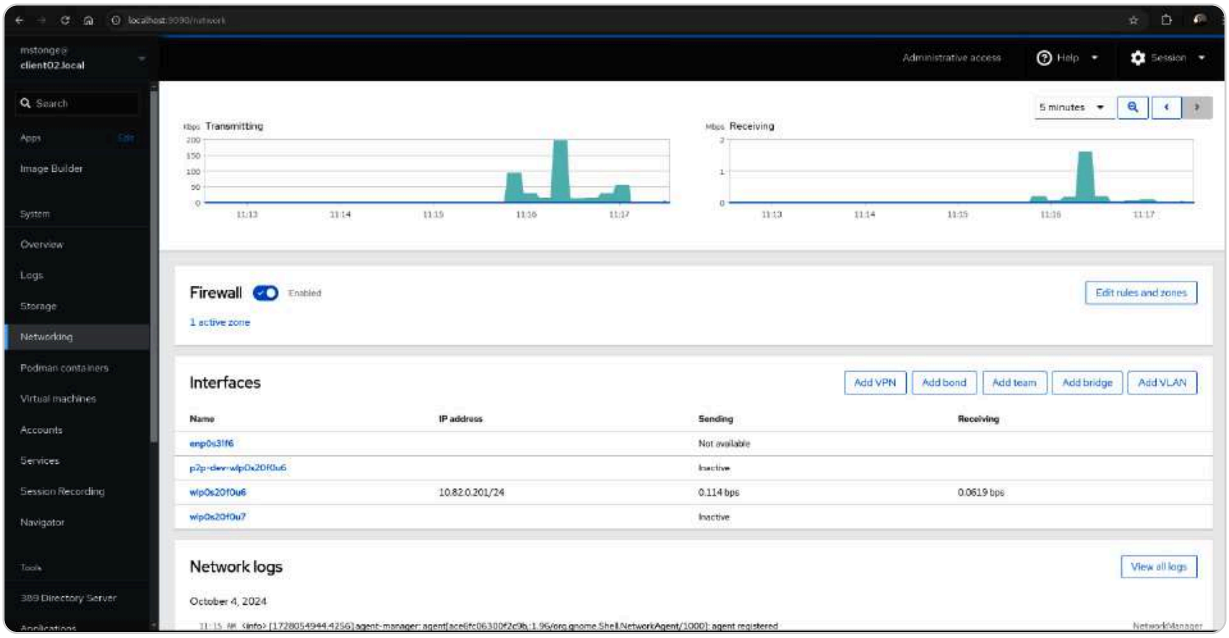
The command line is the greatest and most flexible way to set firewall parameters, but let's be honest, it will take a bit of skill to master it. This methodology is the preferred method by sysadmins around the world as it can be scripted and easily repeated. It's the ease of scripting that will aid your team greatly in making such functionality behind the scenes from the end-users of your solution.

Let's take a look at some other alternative means to configure your firewall.

Web console

The **web console** has been evolving rapidly over the past few years by extending its basic sysadmin functionality to creating systems images or managing containers or virtual machines. The further enhancement of the web console has lowered the bar for admins of other platforms to get comfortable in the management of Linux systems when their skills were rooted in other platforms.

Here's an example screenshot of the web console (aka Cockpit) running on Fedora 40. Under the **Networking** submenu, we can easily modify the firewall rules as well as configure connections and other networking settings.



Interfaces

Name	IP address	Sending	Receiving
enp0s31f6		Not available	
p2p-dev-wlp0s20f0u6		Inactive	
wlp0s20f0u6	10.82.0.201/24	0.114 bps	0.0619 bps
wlp0s20f0u7		Inactive	

Network logs

October 4, 2024

```
11:15 AM <info> [1728054944.4256] agent-manager: agent[ace6fc06300f2c9b,:1.96/org.gnome.Shell.NetworkAgent/1000]: agent registered    NetworkManager
```

Figure 12.7 – Configuring a firewall via the web console

The web console is great for configuring your own systems in a lab or datacenter environment; however, I strongly recommend avoiding this from being used by your end-users on your appliances as it will require administrative privileges to do most tasks… And now we're back to the childproofing discussion again. Let's move on to a quick review of graphical UI tools for your firewall.

Graphical UI-based tools

In Fedora (and most Linux distributions), there are actually multiple graphical firewall configuration tools. Any can be installed via the gnome-software application or via the command line. These tools make configuring and managing the firewall vastly easier than trying to memorize the thousands of options found within the command line.

Please note that when you search for tools to manage your firewall, gnome-software (the app store within the Gnome desktop) also offers up the web console (Cockpit) as a solution.

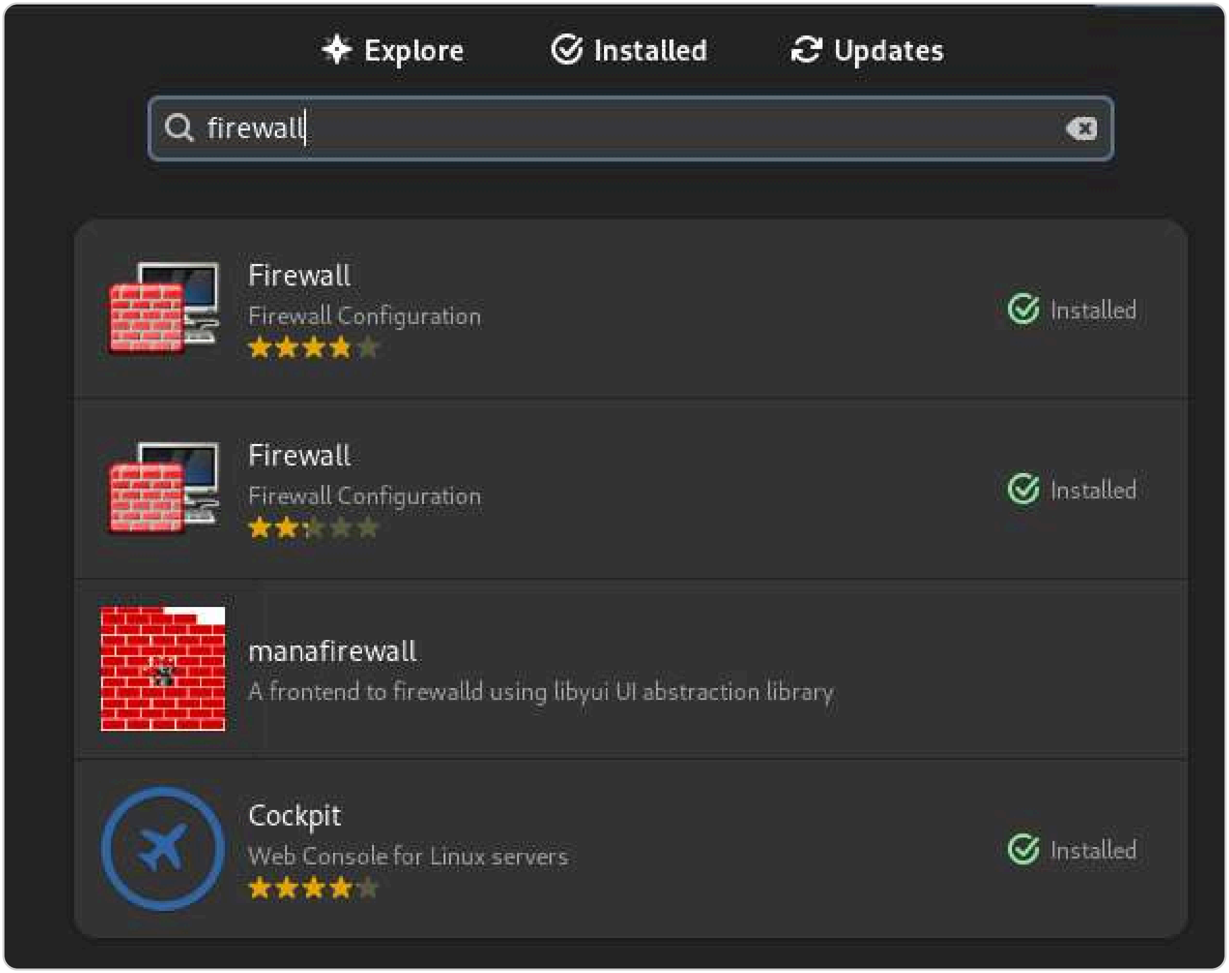
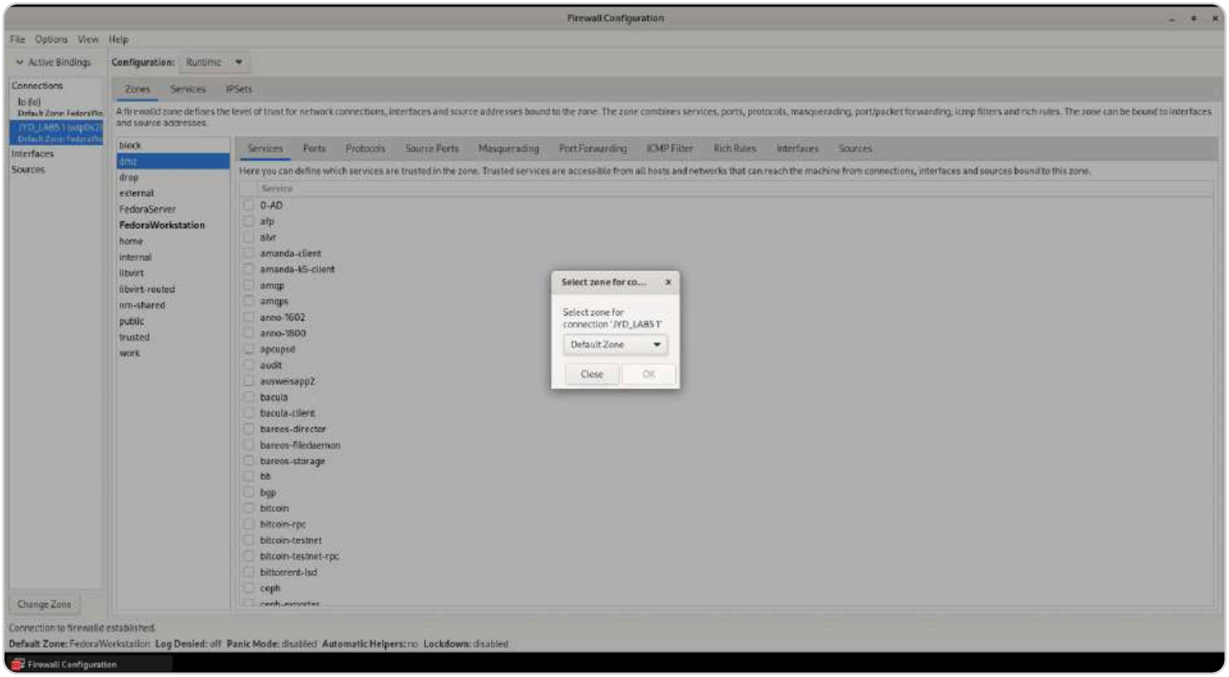


Figure 12.8 – Searching for software via the gnome-software GUI

Here's what the firewall management GUI looks like while running it within the Gnome desktop environment GUI.



A firewalld zone defines the level of trust for network connections, interfaces and source addresses bound to the zone. The zone combines services, ports, protocols, masquerading, port/packet forwarding, icmp filters and rich rules. The zone can be bound to interfaces and source addresses.

block
dmz
drop
external
FedoraServer
FedoraWorkstation
home
internal
libvirt
libvirt-routed
nm-shared
public
trusted
work

Here you can define which services are trusted in the zone. Trusted services are accessible from all hosts and networks that can reach the machine from connections, interfaces and sources bound to this zone.

Service
0-AD
afp
alvr
amanda-client
amanda-k5-client
amqp
amqps
anno-1602
anno-1800
apcupsd
audit
ausweisapp2
bacula
bacula-client
bareos-director
bareos-filedaemon
bareos-storage
bb
bgp
bitcoin
bitcoin-rpc
bitcoin-testnet
bitcoin-testnet-rpc
bittorrent-lsd
ceph
ceph-exporter

Select zone for connection 'JYD_LABS 1'
Default Zone

Connection to firewalld established.
Default Zone: FedoraWorkstation Log Denied: off Panic Mode: disabled Automatic Helpers: no Lockdown: disabled

Figure 12.9 – The firewall GUI

My stance on these GUI-based firewall tools is the same as with the web console. They're great in the datacenter and the lab but I wouldn't give an end-user of an appliance access to them. Ever. Seriously.

Should your appliance provide services that may require the end-user to open additional ports on your appliance, you should automate and build into your interface anything that might possibly require the end-user to have root access – just like we covered in depth in *Chapter 10*. This isn't just tamper-proofing. You should think of it as futureproofing, or even better, enhancing the user experience.

Honestly, it's doubtful anyone will actually care which tools you used or didn't use to configure your appliance. Here's where we shift from configuring the firewall to confirming that what you think you've configured is the actual state of your appliance. This is where I (again) remind you of that old saying, *Trust, but verify*. Trust in your team's efforts but verify that what you think you have configured is what the appliance's state actually is. I greatly recommend automating this where you find it appropriate, but make sure that human hands still touch the appliance, human eyes inspect the appliance, and all is well documented for each unit.

Let's move on to activities that should always be religiously enacted at the end of your build or upgrade cycles. Let's review the validation of your solution.

Limitations of legacy hardware and software

Here’s where I feel compelled to deliver some bad news. You may not always have a choice in which hardware, connectivity methods, or buses your solution leverages. The long-term supportability of your solution may force you and your team to implement something less than ideal in order to maintain backward and forward compatibility (a perfect example is the CAN bus).

Older, legacy hardware wouldn’t be so bad if there wasn’t a common practice for deprecating driver support for extremely old or unsupported chipsets. This means that one could be forced to use unsupported drivers that are not included in your operating system, that may not have been updated for a significant amount of time, or that come from an unreliable source. Unsupported drivers (software) may also mean that there are risks and vulnerabilities that are not addressed. It’s a ticking time bomb.

A compounding problem is that newer operating systems tend to require much newer components. It seems that with each new release of Linux (regardless of distribution), we get vastly more capabilities and functions. These new bells and whistles come with a price tag – newer, more powerful hardware.

Why does using the most current hardware and software matter? Here’s a real-world example: most cars, short of catastrophic accidents, can realistically be on the road for many years and outlive their computer subsystems’ normal lifecycle. That’s just one example among countless possibilities.

Security starts at the design table, but that doesn’t mean that you’ll always have the best components (hardware or software) to choose from. Keep in mind there will be limits.

Validating your solution before shipping

As we have progressed in our journey to secure our solution as much as possible, there are still some critical activities that must be completed before you can confidently ship your solution. These should be considered non-negotiable. Ignore them at your peril.

First is compliance integrations and testing, where you’ll confirm that your appliance meets or exceeds any government or industry standards that may be applicable within the domain it will be used. I feel this is such a crucial subject, that the entire next chapter is dedicated to it entirely.

Secondly, **penetration testing** (aka **pen-testing**) by a third party can give you peace of mind and detailed insights as to anything you may have overlooked. In this scenario, a professional ethical hacker (contractor) would leverage all the tools of the trade in an attempt to gain unauthorized access or degrade the usability of the appliance. Passing this type of testing will be a testament to your team’s success and the brightness of your product’s future.

Let’s move on to our end-of-chapter summary.

Summary

What is attached to your solution and what it communicates with absolutely matters. These communications will also dictate what security concerns need to be addressed proactively. In this chapter, we've toured a key group of buses and connection types that you are most likely to encounter along with their unique use cases. As we progressed, we reviewed a critical set of securitizations – your firewall and encryption for your web traffic. These activities require a special amount of attention to detail. The knowledge gained in this chapter should empower you and your team to prepare for and design for such connectivity risks. We then closed out the chapter with a simple reminder that you may have to worry about forward and backward compatibility, which will impact your security profile.

Again, I'd like to thank you for continuing this journey with me. In the next chapter, we'll be diving headfirst into applying security standards to our solutions.

13
Applying Government Security Standards – System Hardening

Whether your team is trying to give your product a competitive edge over the competition via heightened security or your customer base (i.e., government, military, or other public sector customer) has compliance mandates for all systems that they employ, the application of one or more government standards for security is generally no simple task.

For this level of compliance, you must build a solution based on accepted and certified operating systems. This specific compliance action will obviously take most community distributions out of selection for you, as we discussed in *Chapter 2*. In this space, the list of Linux operating systems is brutally short. You can count the players in this space with the fingers on one hand (and maybe have a finger or two left over). This is not meant to disparage any distribution whether they are community or commercial. The process is lengthy, expensive, and, beyond a shadow of a doubt, possibly quite frustrating for most. Few distributions have the resources to take this journey with NIST.

The good news for you is the list of certified Linux operating systems is expanding, but few can compete with the extensive list of government certifications and standards that are adhered to like **Red Hat® Enterprise Linux®** (**RHEL**) can. It is seen as the de facto standard for Linux in the US government space. The few other players in this space are often clones or derivatives of RHEL, except Ubuntu®, which is the most popular Debian Linux variant.

Another factor in this effort that we will explore here is additional tooling that makes the Herculean effort to comply with these security standards even possible. The **Security Content Automation Protocol** (**SCAP**) can easily be seen as the most common standard for the automated application sets of security measures (AKA SCAP server profiles) to a system or application stack. The actual process of implementing these profiles requires tooling. This is where open source comes to the rescue (again). The OpenSCAP projects provide extensive tooling. Many Linux distributions include these packages within their installers and repositories. SCAP definitions provide an industry standard way of applying security standards and remediations. They are the highway towards compliance regardless of the security standard that you are trying to apply. In this chapter, we'll show you by example how to apply the two most common standards for server security in the US – FIPS 140-3 and the general **STIG** (short for **Secure Technical Implementation Guide**).

In this chapter, the main sections are:

- Adherence to key US government standards

- How do I implement this?
- How do I certify my solution?

Let's move on to our next section, where we will go into some detail as to the details of what you may need to complete the exercises in this chapter.

Technical requirements

For the hands-on exercises in this chapter, you will need two RHEL 9.4 (or newer) machines (physical or virtual) both will be clean installations. We will do one of those installations together as an exercise.

To achieve this, you will need access to a Red Hat® account (developer or production), installation media, Fedora® Media Writer, and the ability to install/reinstall the operating system on the two machines.

Additionally, you will need the ability to download and install additional packages from third-party or government websites. The STIG Viewer tool can be installed on Windows® or Linux systems. You'll need this as well, but it does not have to be installed specifically on your lab machines. Check out the book's GitHub repository for helpful links: https://github.com/PacktPublishing/The-Embedded-Linux-Security-Handbook/blob/main/Chapter11/Resource_Cheat_Sheet.pdf.

Let's move on to the reason *why* adherence to security standards might matter.

Adherence to key US government standards

Many seek to maintain their solutions to a higher standard than what may or may not be required. Many can agree with me on this; I applaud that level of commitment to security and to going above and beyond. In doing so, they can easily set their solutions on a higher pedestal against any of their competitors' solutions. Why? It's because if the solution meets the arduous security standards of the US military or certain three-letter agencies, then it's definitely good enough for them. Well, one would assume such.

Continuing with that thought, I want to highlight by going above and then beyond what is minimally required (or seen as the norm) in a specific industry is yet another excellent method of driving trust within your targeted community base. By raising the bar for yourself and your products, you will ultimately preemptively (and informally) set a new standard for others to attempt to live up to. This will result in easier audits for your customers, who will definitely appreciate a less stressful compliance effort.

Adherence to government standards, when not required specifically, can also give your potential customers more peace of mind in choosing your solution. The customers' auditors may also thank them for executing more due diligence in the selection of your product. It's a win-win for everyone. That extra effort on your teams' parts can go a long way to a wider adoption rate of your products.

Besides trust and peace of mind, having one less system on their network to worry about is also a plus for your customers. I assure you that they will appreciate that too.

Then there's the flip side of the coin. When compliance is mandated due to your customer base's requirements, your team will have its work cut out for them. Regardless of where your product falls in the requirements space, the more secure your product is the better the outcomes will be for all involved.

Let's review an earlier discussion in *Chapter 2*, when we established the single greatest factor: your target customer base's compliance needs. Perhaps your product targets a wide stretch of industry verticals.

If your user base is in the government sector, your product had better bring its *A* game. Failure to comply with the applicable standards imposed upon any system in government infrastructure results in your solution not getting adopted there.

Government is just one vertical that requires such attention to detail. Healthcare and financial services also have their own regulatory security standards (albeit not too different from government/military standards).

Regardless of what verticals and sectors you sell into, casting a wider net compliance-wise is just a smart play. Additionally, I feel remiss if I have failed to mention that there is a massive overlap in many security standards across many industries.

Let's get into more importantly how we can start to apply these standards to our solutions successfully.

How do I implement this?

The answer to the question *How do I implement this?* does not have a single answer. The truthful response is *It's complicated*. Maintaining compliance has many touchpoints. It starts at the design table, where you must identify what standards will impact the overall solution. Some can argue that the significant amounts of research time are the greatest amount of time spent on this effort. I disagree. I believe (and I suspect you will too soon enough) that maintaining compliance from release to release long after the measures have been originally implemented in your product is the true heavy lift.

There's also a catch to all of this. Maintaining adherence is not a one-and-done process. Brace for the upcoming frustration because here it comes. Setting a server profile, whether during installation or later on, is not the end but the beginning. Every time you make a change to the configuration, albeit small or large, you must re-scan and confirm compliance. This part of maintenance and testing can be a rather tedious process, but it IS necessary.

Let's explore, in the next few sections, how we implement and maintain standards compliance throughout the lifecycle of your product.

Implementation of security standards

Security begins at the design table, but the implementation and maintenance of your security measures is where the proverbial rubber hits the road. Paying close attention to the most minute detail could mean the difference that protects your product from being compromised. Why do we do this (as if we need a reminder)? Perpetual risk mitigation.

Our first foray into the implementation of security standards begins with the Linux installers.

The enhancements, to some installers in recent years, do make this process easier on their implementers. But as I have previously stated, it is simply the beginning of a long-term commitment that you and your team have engaged in.

Leveraging RHEL server profiles at installation time

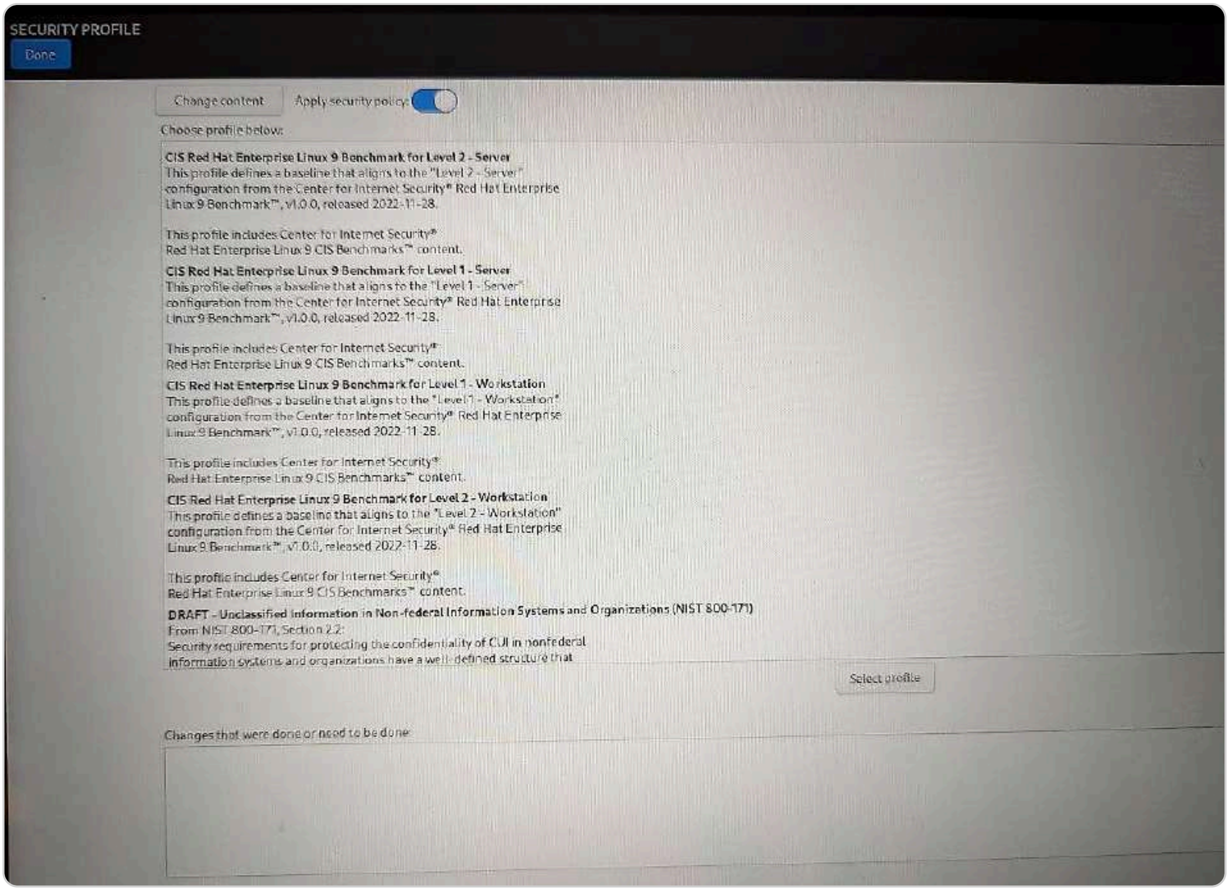
The RHEL installer, Anaconda, has the ability for the end-user to select a predefined set of SCAP Security Guide server profiles. These server profiles represent several governmental agencies and industry standards. Leveraging one of these at installation time is a significant jumpstart to ensuring your system will be compliant with an applicable standard. This doesn't guarantee that, after you have finished setting up your server, it is 100% compliant. You will still need to confirm compliance as part of your QA process later on.

You can select one of the several SCAP profiles (or other standards) in Anaconda during their installation process. Again, this is where I applaud those developers who spent the time to make this arduous process easier for us all.

For those who are curious as to which profiles are included, wait no more. Recently, the master list for publicly shared SCAP profiles for virtually all major Linux distributions was moved to GitHub; it can be found via this link: https://complianceascode.github.io/content-pages/guides/index.html. The list of the SCAP profiles included within the RHEL 9 installer is somewhat extensive. Here's a curtailed list of what profiles are included in the installer:

- Various levels of security (low, medium, high, and enhanced) from the French National Agency for the Security of Information Systems – ANSSI BP-028
- Various levels of **Centro Criptológico Nacional** (**National Cryptologic Center aka CCN**) of Spain defined server security levels
- Various CIS Benchmark levels
- The DRAFT unclassified FIPS standard (NIST 800-171)
- The **Australian Cyber Security Centre** (**ACSC**) Essential Eight
- The ACSC ISM Official
- The **Health Insurance Portability and Accountability Act** (**HIPAA**)
- The Protection Profile for General Purpose Operating Systems
- PCI-DSS version 3.2.1 Control Baseline
- The **Defense Information Systems Agency** (**DISA**) **STIG** for RHEL 9 Server and RHEL 9 Server with GUI

Here's a screenshot taken while browsing the extensive list of available SCAP profiles during a RHEL 9 installation. It's impossible to show them all in a single screenshot.



Choose profile below:

CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server
This profile defines a baseline that aligns to the "Level 2 - Server" configuration from the Center for Internet Security® Red Hat Enterprise Linux 9 Benchmark™, v1.0.0, released 2022-11-28.

This profile includes Center for Internet Security® Red Hat Enterprise Linux 9 CIS Benchmarks™ content.

CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server
This profile defines a baseline that aligns to the "Level 1 - Server" configuration from the Center for Internet Security® Red Hat Enterprise Linux 9 Benchmark™, v1.0.0, released 2022-11-28.

This profile includes Center for Internet Security® Red Hat Enterprise Linux 9 CIS Benchmarks™ content.

CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation
This profile defines a baseline that aligns to the "Level 1 - Workstation" configuration from the Center for Internet Security® Red Hat Enterprise Linux 9 Benchmark™, v1.0.0, released 2022-11-28.

This profile includes Center for Internet Security® Red Hat Enterprise Linux 9 CIS Benchmarks™ content.

CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation
This profile defines a baseline that aligns to the "Level 2 - Workstation" configuration from the Center for Internet Security® Red Hat Enterprise Linux 9 Benchmark™, v1.0.0, released 2022-11-28.

This profile includes Center for Internet Security® Red Hat Enterprise Linux 9 CIS Benchmarks™ content.

DRAFT - Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)
From NIST 800-171, Section 2.2:
Security requirements for protecting the confidentiality of CUI in nonfederal information systems and organizations have a well-defined structure that

Changes that were done or need to be done:

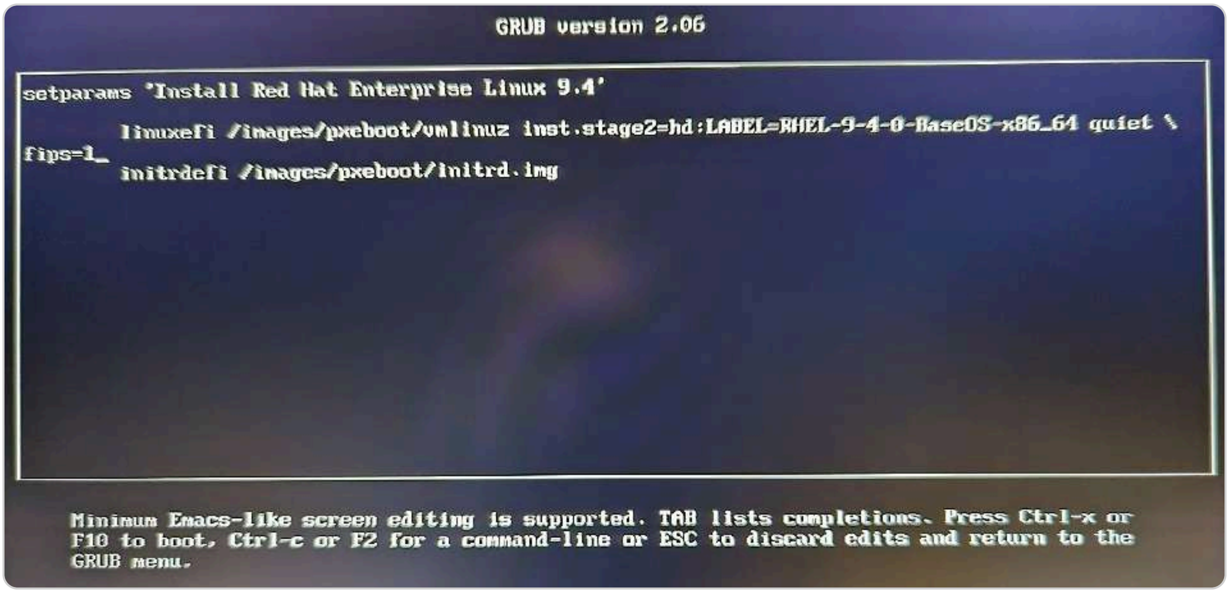
Figure 13.1 – Selecting a server profile in the RHEL installer

Enabling FIPS mode in RHEL

RHEL 9 has been focused on the FIPS 140-3 standard since its initial development began several years ago. Red Hat had determined that attempting to add FIPS 140-3 to RHEL 8 (which is FIPS 140-2 certified) would require too much refactoring and as the operating system was within its final five years of its lifecycle, such an effort was untenable. During the final five years of a RHEL release, the operating system is considered to be in maintenance mode, where new features generally are not introduced but security and bug fixes are still implemented.

Building a FIPS-compliant system starts at installation time by placing the installer into FIPS mode. This simple step is achieved by editing one line during the system's boot process. During installation, append `fips=1` to the kernel line in the installer boot kernel options.

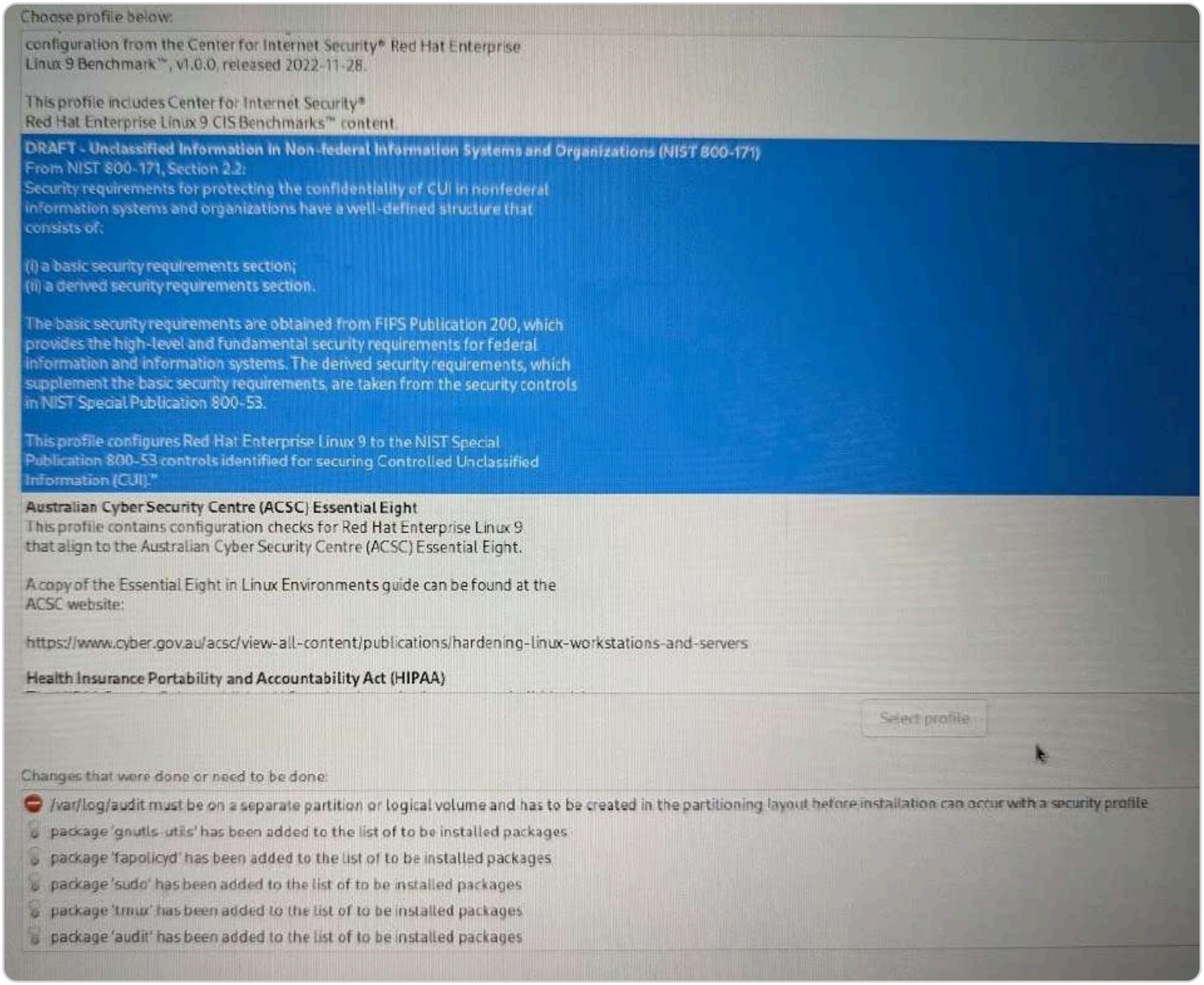
Here's a screenshot of me setting the kernel boot options within GRUB to allow the installer to boot into FIPS mode:



```
                              GRUB version 2.06

setparams 'Install Red Hat Enterprise Linux 9.4'
        linuxefi /images/pxeboot/vmlinuz inst.stage2=hd:LABEL=RHEL-9-4-0-BaseOS-x86_64 quiet \
fips=1_
        initrdefi /images/pxeboot/initrd.img

    Minimum Emacs-like screen editing is supported. TAB lists completions. Press Ctrl-x or
    F10 to boot, Ctrl-c or F2 for a command-line or ESC to discard edits and return to the
    GRUB menu.
```

Figure 13.2 – Configuring the installer to boot in FIPS mode

The next step in getting your system closer to FIPS compliance is to select the FIPS SCAP server profile during the installation process.


Choose profile below:

configuration from the Center for Internet Security® Red Hat Enterprise
Linux 9 Benchmark™, v1.0.0, released 2022-11-28.

This profile includes Center for Internet Security®
Red Hat Enterprise Linux 9 CIS Benchmarks™ content.

DRAFT - Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)
From NIST 800-171, Section 2.2:
Security requirements for protecting the confidentiality of CUI in nonfederal
information systems and organizations have a well-defined structure that
consists of:

(i) a basic security requirements section;
(ii) a derived security requirements section.

The basic security requirements are obtained from FIPS Publication 200, which
provides the high-level and fundamental security requirements for federal
information and information systems. The derived security requirements, which
supplement the basic security requirements, are taken from the security controls
in NIST Special Publication 800-53.

This profile configures Red Hat Enterprise Linux 9 to the NIST Special
Publication 800-53 controls identified for securing Controlled Unclassified
Information (CUI)."

Australian Cyber Security Centre (ACSC) Essential Eight
This profile contains configuration checks for Red Hat Enterprise Linux 9
that align to the Australian Cyber Security Centre (ACSC) Essential Eight.

A copy of the Essential Eight in Linux Environments guide can be found at the
ACSC website:

https://www.cyber.gov.au/acsc/view-all-content/publications/hardening-linux-workstations-and-servers

Health Insurance Portability and Accountability Act (HIPAA)

Changes that were done or need to be done:

/var/log/audit must be on a separate partition or logical volume and has to be created in the partitioning layout before installation can occur with a security profile
package 'gnutls-utils' has been added to the list of to be installed packages
package 'fapolicyd' has been added to the list of to be installed packages
package 'sudo' has been added to the list of to be installed packages
package 'tmux' has been added to the list of to be installed packages
package 'audit' has been added to the list of to be installed packages

Figure 13.3 – Selecting the FIPS server profile during installation

Optionally, if you are like me (detail-oriented) and desire to double-check your work, you could also configure FIPS compliance mode as a post-install process. This same process is needed after any changes to the system have occurred.

Let's see an example of a post-install configuration of enforcing FIPS mode via the command line. This, in itself, is a glaring example of where enterprise distributions (such as RHEL) make the adherence of complex standards easier. In this situation, you would run the following command as root:

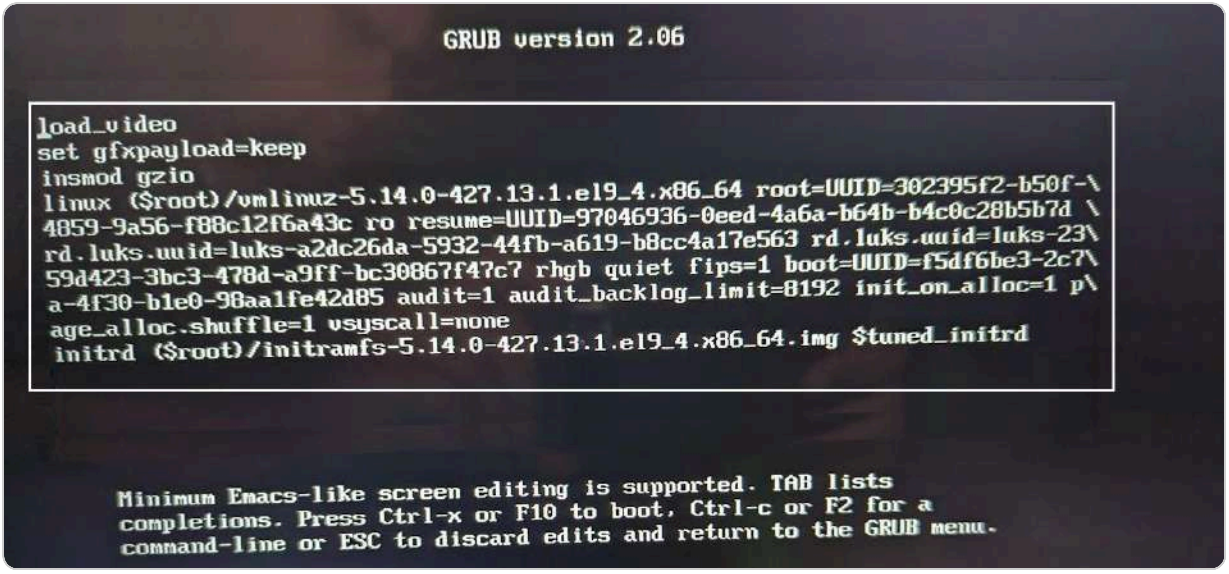
```
 #  fips-mode-setup
```

The output can be extremely lengthy, obscure, and verbose. Feel free to experiment in your lab as you see fit.

Now, we can look at several of the steps we can take to validate FIPS mode before we perform a scan of the system. These are addressed in no particular order, but I'd like to think that the order chosen seems somewhat logical.

Ensuring the kernel boot option is set for FIPS support by checking the kernel's boot options before actually booting Linux. This does not, however, guarantee that all the other configurations required by FIPS 140-3 compliance have been completed. This is simply the start of the journey.

In this pictorial example, you can see the configuration for booting, and it clearly shows kernel options... more specifically, the one we care about, **`fips=1`**:



```
GRUB version 2.06

load_video
set gfxpayload=keep
insmod gzio
linux ($root)/vmlinuz-5.14.0-427.13.1.el9_4.x86_64 root=UUID=302395f2-b50f-\
4859-9a56-f88c12f6a43c ro resume=UUID=97046936-0eed-4a6a-b64b-b4c0c28b5b7d \
rd.luks.uuid=luks-a2dc26da-5932-44fb-a619-b8cc4a17e563 rd.luks.uuid=luks-23\
59d423-3bc3-478d-a9ff-bc30867f47c7 rhgb quiet fips=1 boot=UUID=f5df6be3-2c7\
a-4f30-b1e0-98aa1fe42d85 audit=1 audit_backlog_limit=8192 init_on_alloc=1 p\
age_alloc.shuffle=1 vsyscall=none
initrd ($root)/initramfs-5.14.0-427.13.1.el9_4.x86_64.img $tuned_initrd

Minimum Emacs-like screen editing is supported. TAB lists
completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for a
command-line or ESC to discard edits and return to the GRUB menu.
```

Figure 13.4 – Confirming FIPS mode boot options in GRUB

Here's another example method of ensuring that FIPS mode is enabled on your RHEL server. This will do a ton of heavy lifting in the configuration for you.

You should run the following command to forcibly set the system policy:

```
# update-crypto-policies --set FIPS
```

As you might be curious (or gravely concerned) as to where your system stands in terms of adherence to the FIPS 140-3 standard, we've provided this example walkthrough on how to check. We will ensure that the validation of FIPS mode is completed (regardless of how it was configured previously):

1. First, we'll check to see that FIPS mode is enabled by running the following as root:

   ```
   # fips-mode-setup --check
   ```

   This is where you need to pay close attention to the output as you might just get this as your output – and then you still have more work to do. This is normal. Perfection is difficult to achieve. Your output will most likely be like the following, notifying you that you have more work to accomplish:

```
Installation of FIPS modules is not completed.
FIPS mode is disabled.
```

2. Let's continue down the path assuming that your system needs more work by the fact that you did not get a favorable response to the last command. We can then run this next command and (of course) re-verify that FIPS mode is OK on our host. This may also take a little more time than most commands. Please be patient.

   ```
   #  fips-mode-setup --enable
   ```

   This command will take some time – expect the output to resemble this:

   ```
   Kernel initramdisks are being regenerated. This might take some time.
   Setting system policy to FIPS
   Note: System-wide crypto policies are applied on application start-up.
   It is recommended to restart the system for the change of policies
   to fully take place.
   FIPS mode will be enabled.
   Please reboot the system for the setting to take effect.
   ```

3. OK. Things are now better. Because of the results we received from the last command, we had to do some remediation. That's expected. Let's reboot as recommended now:

   ```
   #  reboot
   ```

4. After the system has rebooted, let's get back to a root prompt and re-run the verification command. Like I continue to say, *trust but verify!*

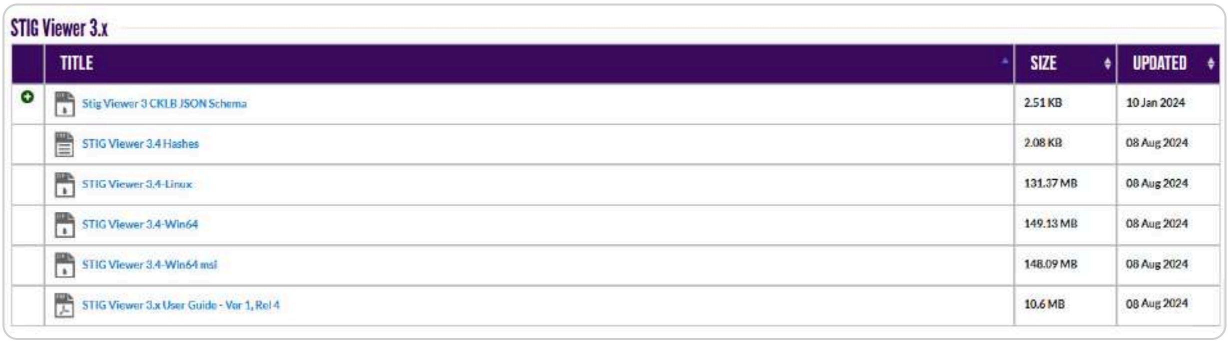
   ```
   #  fips-mode-setup --check
   FIPS mode is enabled.
   ```

We can now breathe a sigh of relief. We have gone through how to implement FIPS mode and test the results. Now, let's move on to another set of security measures that comprises DISA's STIG profiles.

DISA STIG SCAP profiles

Before we dive deep into the setup of STIG compliant systems, let's first get to a point where we can actually make sense of the STIG standard's documentation. This requires you to install a tool from DISA. Yes, you will need another tool just to view the details of the STIGs themselves. The DISA STIG Viewer is a free software tool that runs on Windows and Linux. Sorry everyone – no Mac® support. I'm sure that there's some interesting backstory explaining *why* there's no Mac support, but I do not know it. This tool is well maintained by the security gurus at DISA and is made publicly accessible to all.

First, let's open up a browser and go to the DISA site, then download the free STIG Viewer tool. The tool can be found at: https://public.cyber.mil/stigs/srg-stig-tools/.



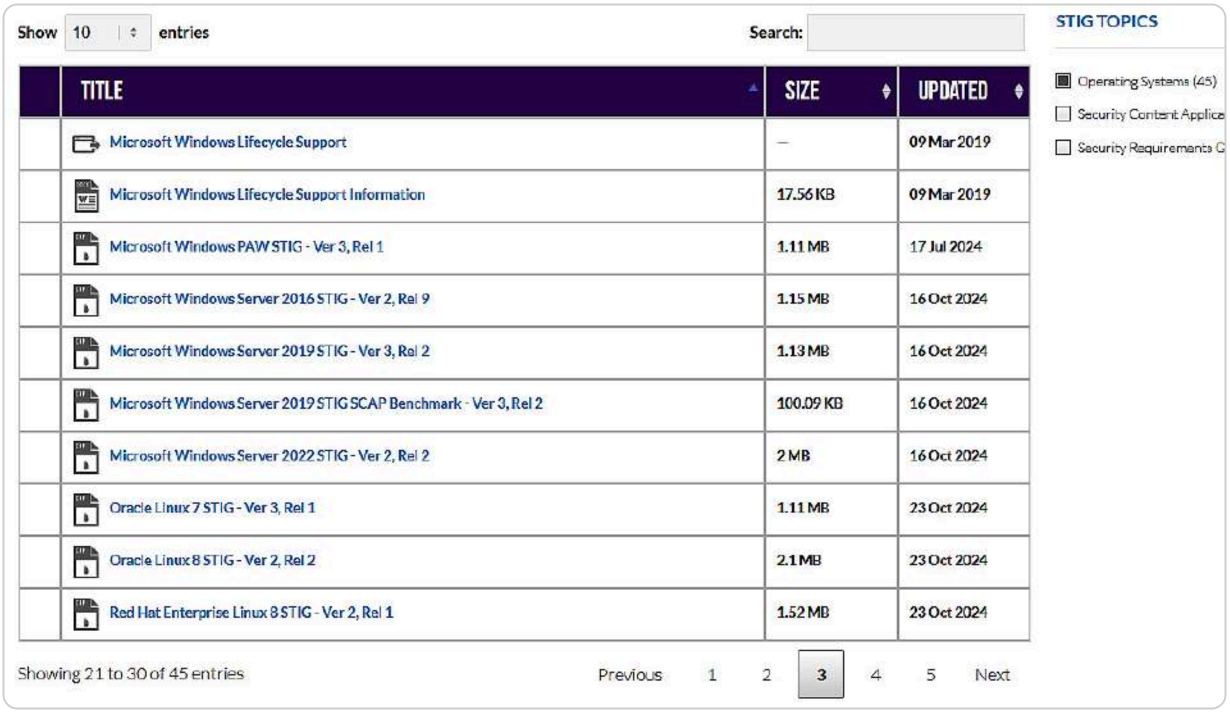
STIG Viewer 3.x

	TITLE	SIZE	UPDATED
●	Stig Viewer 3 CKLB JSON Schema	2.51 KB	10 Jan 2024
	STIG Viewer 3.4 Hashes	2.08 KB	08 Aug 2024
	STIG Viewer 3.4-Linux	131.37 MB	08 Aug 2024
	STIG Viewer 3.4-Win64	149.13 MB	08 Aug 2024
	STIG Viewer 3.4-Win64 msi	148.09 MB	08 Aug 2024
	STIG Viewer 3.x User Guide - Ver 1, Rel 4	10.6 MB	08 Aug 2024

Figure 13.5 – Downloading the STIG Viewer

Now that we have this excellent free tool, let's put it to good use and download the STIG that applies to our example RHEL 9.x lab server. This too is a relatively simple process.

Go to https://public.cyber.mil/stigs/downloads/?_dl_facet_stigs=operating-systems and select the RHEL 9 STIG. You may have to go through several other operating systems' STIG entries to get there (I believe it's on the third page).



Show 10 entries Search:

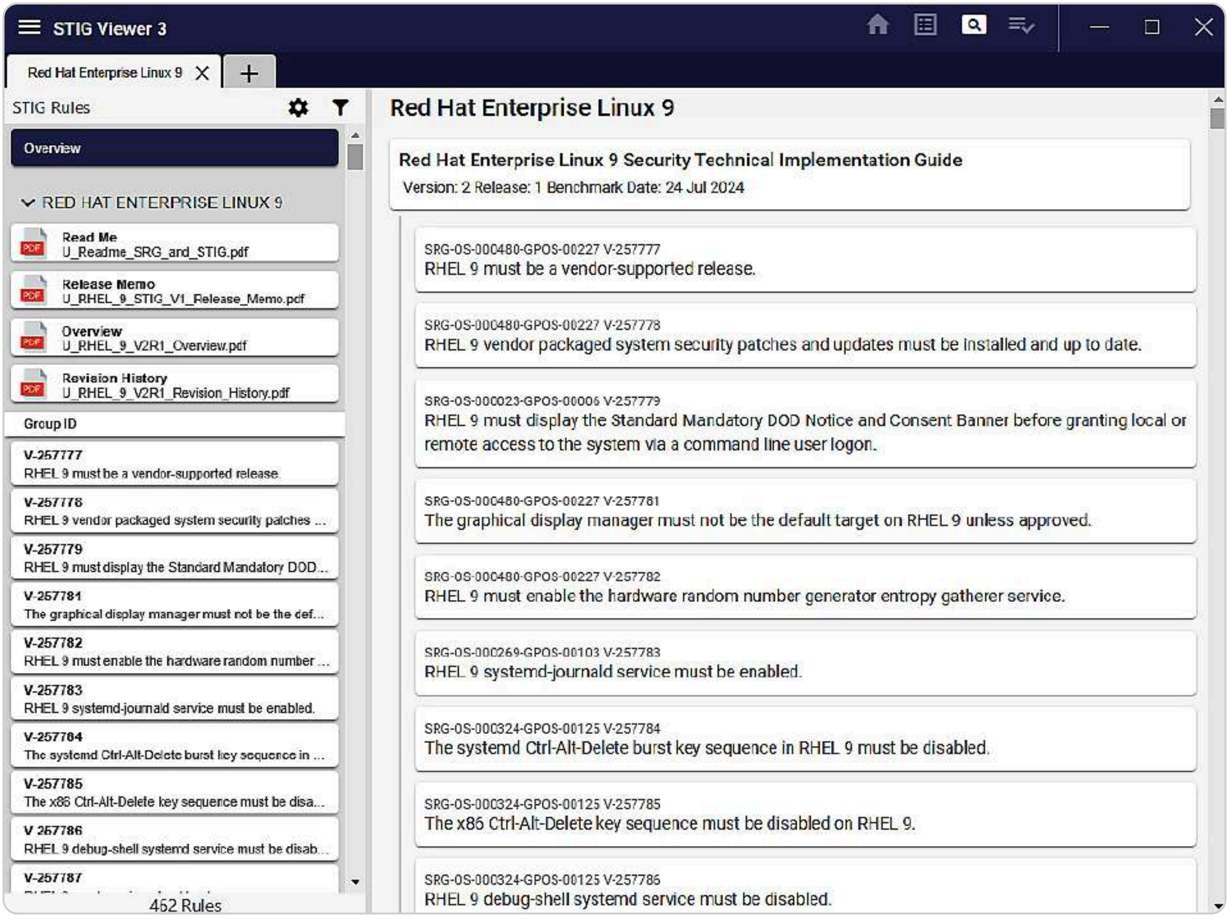
STIG TOPICS
- ▣ Operating Systems (45)
- ☐ Security Content Applica
- ☐ Security Requirements G

	TITLE	SIZE	UPDATED
	Microsoft Windows Lifecycle Support	–	09 Mar 2019
	Microsoft Windows Lifecycle Support Information	17.56 KB	09 Mar 2019
	Microsoft Windows PAW STIG - Ver 3, Rel 1	1.11 MB	17 Jul 2024
	Microsoft Windows Server 2016 STIG - Ver 2, Rel 9	1.15 MB	16 Oct 2024
	Microsoft Windows Server 2019 STIG - Ver 3, Rel 2	1.13 MB	16 Oct 2024
	Microsoft Windows Server 2019 STIG SCAP Benchmark - Ver 3, Rel 2	100.09 KB	16 Oct 2024
	Microsoft Windows Server 2022 STIG - Ver 2, Rel 2	2 MB	16 Oct 2024
	Oracle Linux 7 STIG - Ver 3, Rel 1	1.11 MB	23 Oct 2024
	Oracle Linux 8 STIG - Ver 2, Rel 2	2.1 MB	23 Oct 2024
	Red Hat Enterprise Linux 8 STIG - Ver 2, Rel 1	1.52 MB	23 Oct 2024

Showing 21 to 30 of 45 entries Previous 1 2 3 4 5 Next

Figure 13.6 – Downloading the RHEL 9 STIG SCAP profile

In my lab, I have the STIG Viewer on both my Windows laptop and several of my Linux servers. Here's a screenshot of using the STIG Viewer to drill down into the RHEL 9 STIG library.


STIG Viewer 3
Red Hat Enterprise Linux 9
STIG Rules
Overview
RED HAT ENTERPRISE LINUX 9
Read Me
U_Readme_SRG_and_STIG.pdf
Release Memo
U_RHEL_9_STIG_V1_Release_Memo.pdf
Overview
U_RHEL_9_V2R1_Overview.pdf
Revision History
U_RHEL_9_V2R1_Revision_History.pdf
Group ID
V-257777
RHEL 9 must be a vendor-supported release.
V-257778
RHEL 9 vendor packaged system security patches ...
V-257779
RHEL 9 must display the Standard Mandatory DOD ...
V-257781
The graphical display manager must not be the def...
V-257782
RHEL 9 must enable the hardware random number ...
V-257783
RHEL 9 systemd-journald service must be enabled.
V-257784
The systemd Ctrl-Alt-Delete burst key sequence in ...
V-257785
The x86 Ctrl-Alt-Delete key sequence must be disa...
V 257786
RHEL 9 debug-shell systemd service must be disab...
V-257787
452 Rules

Red Hat Enterprise Linux 9

Red Hat Enterprise Linux 9 Security Technical Implementation Guide
Version: 2 Release: 1 Benchmark Date: 24 Jul 2024

SRG-OS-000480-GPOS-00227 V-257777
RHEL 9 must be a vendor-supported release.

SRG-OS-000480-GPOS-00227 V-257778
RHEL 9 vendor packaged system security patches and updates must be installed and up to date.

SRG-OS-000023-GPOS-00006 V-257779
RHEL 9 must display the Standard Mandatory DOD Notice and Consent Banner before granting local or remote access to the system via a command line user logon.

SRG-OS-000480-GPOS-00227 V-257781
The graphical display manager must not be the default target on RHEL 9 unless approved.

SRG-OS-000480-GPOS-00227 V-257782
RHEL 9 must enable the hardware random number generator entropy gatherer service.

SRG-OS-000269-GPOS-00103 V-257783
RHEL 9 systemd-journald service must be enabled.

SRG-OS-000324-GPOS-00125 V-257784
The systemd Ctrl-Alt-Delete burst key sequence in RHEL 9 must be disabled.

SRG-OS-000324-GPOS-00125 V-257785
The x86 Ctrl-Alt-Delete key sequence must be disabled on RHEL 9.

SRG-OS-000324-GPOS-00125 V-257786
RHEL 9 debug-shell systemd service must be disabled.

Figure 13.7 – Viewing the RHEL 9 STIG Library

This is where I recommend that you spend some time not only getting acclimated with the STIG viewer tool but also using it to examine not just the RHEL 9.x STIG SCAP profile but any others you want to become more intimate with. This tool is priceless.

Now that we've spent time viewing what profiles we could apply to our product, let's move on to the tooling that actually allows us to force our systems' configuration into compliance.

Whichever standards you choose to adhere to during your installation (and configuration processes), you will still need to validate the solution in detail as part of your release processes. Let's take a look at what that may look like.

Validation as part of the QA process

Have I ever said *trust but verify* previously? I bet I have. Yes. I'm saying it again.

Knowing where to get the latest information about how to apply SCAP profiles and scanning is important. Since we are focused on these activities and RHEL, here's the link to Red Hat's documentation on this process: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardening#configuration-compliance-tools-in-rhel_scanning-the-system-for-configuration-compliance-and-vulnerabilities.

> ***TIP***
>
> *You will need an account to log in to view the information.*

Let's move on to an exercise where we'll run our first scan.

Exercise: Installing the OpenSCAP tools and running a scan

During this exercise, you will install the OpenSCAP tools onto one of your lab machines, download a RHEL 9 SCAP profile, and run a security scan. Let's see how to do it:

1. Get the packages installed:

   ```
   $  sudo dnf install -y scap-workbench  \
   openscap-utils openscap-engine-sce \
   openscap-scanner scap-security-guide bzip2
   ```

   The output for this one is exceptionally long; however, I've truncated it to show the important parts of what was installed:

   ```
   Updating Subscription Management repositories.
   Last metadata expiration check: 0:04:16 ago on Wed 16 Oct 2024 04:52:00 AM EDT.
   Package openscap-scanner-1:1.3.10-2.el9_3.x86_64 is already installed.
   Package bzip2-1.0.8-8.el9.x86_64 is already installed.
   Dependencies resolved.
   ==========================================================
        ((( output truncated )))
   Installed products updated.
   Installed:
     adwaita-gtk2-theme-3.28-14.el9.x86_64         gtk2-2.24.33-
   8.el9.x86_64                        ibus-gtk2-1.5.25-5.el9.x86_64
     libcanberra-gtk2-0.30-27.el9.x86_64           openscap-engine-sce-1:1.3.10-
   2.el9_3.x86_64      openscap-utils-1:1.3.10-2.el9_3.x86_64
     openssh-askpass-8.7p1-38.el9_4.4.x86_64       pcre2-utf16-10.40-
   ```

```
5.el9.x86_64                   qt5-qtbase-5.15.9-10.el9_4.x86_64
  qt5-qtbase-common-5.15.9-10.el9_4.noarch      qt5-qtbase-gui-5.15.9-
10.el9_4.x86_64            qt5-qtdeclarative-5.15.9-3.el9.x86_64
  qt5-qtxmlpatterns-5.15.9-2.el9.x86_64         rpmdevtools-9.5-
1.el9.noarch                     scap-workbench-1.2.1-13.el9.x86_64
  xcb-util-image-0.4.0-19.el9.x86_64            xcb-util-keysyms-0.4.0-
17.el9.x86_64             xcb-util-renderutil-0.3.9-20.el9.x86_64
  xcb-util-wm-0.4.1-22.el9.x86_64
Complete!
```

2. Download a general-purpose SCAP profile for RHEL 9 from Red Hat. Just like with the STIG SCAP profile we downloaded earlier, this will be an XML file that will be used by the scanning tool:

```
#  wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2
| bzip2 --decompress > rhel-9.oval.xml
```

The output for this one is rather long too; I have truncated it to save space, leaving only some important informational bits here for you:

```
--2024-10-16 04:59:45--  https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-
9.oval.xml.bz2
Resolving www.redhat.com (www.redhat.com)... 23.37.1.210, 2600:1401:4000:58d::d44,
2600:1401:4000:58b::d44
Connecting to www.redhat.com (www.redhat.com)|23.37.1.210|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://security.access.redhat.com/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2
[following]
--2024-10-16 04:59:45--  https://security.access.redhat.com
/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2
Resolving security.access.redhat.com
(security.access.redhat.com)
... 23.194.190.138, 23.194.190.193,
2600:1401:2000::b819:94a8, ...
Connecting to security.access.redhat.com
(security.access.redhat.com)|23.194.190.138|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 444671 (434K) [application/x-bzip2]
Saving to: 'STDOUT'
-                                  100%[===============
=========
=======================================>] 434.25K  --.-KB/s
    in 0.1s
2024-10-16 04:59:46 (3.26 MB/s) - written to stdout
[444671/444671]
```

3. Run a test scan:

```
$  oscap-ssh <username>@<hostname> <port> oval eval --report
<scan-report.html> <path to rhel-9.oval.xml>
```

IMPORTANT NOTE

The oscap tools (i.e., the **`openscap-utils`** *package) will have to be installed on the target host as well as the host it is executed from. Substitute your own appropriate username hostname and paths to files as specified. Please be aware that the output from this command's execution can be excessively long. If you do not have two machines with the tools installed, you can simply run it against your own user account on localhost. The point of this exercise is simply to demonstrate the process.*

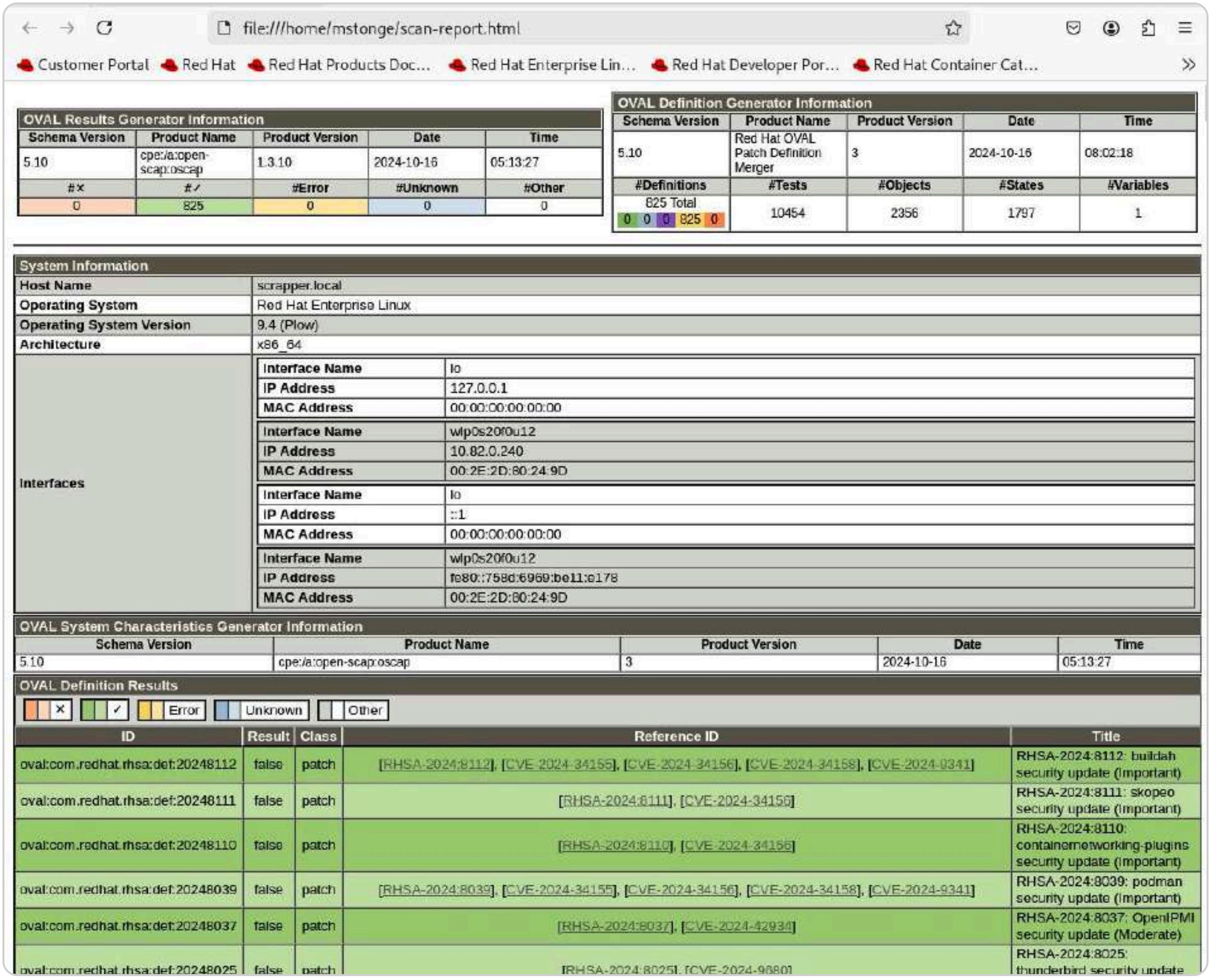
The interactive output for this command is rather lengthy. Again, I have truncated what we are displaying here for you:

```
Connecting to 'mstonge@testmachine' on port '22'...
mstonge@testmachine's password:
Connected!
Copying input file '/home/mstonge/rhel-9.oval.xml' to remote
working directory '/tmp/tmp.10rp1SVS0z'...
rhel-9.oval.xml
                                          100% 9879KB 336.5MB/s
   00:00
Starting the evaluation...
Definition oval:com.redhat.rhsa:def:20248112: false
Definition oval:com.redhat.rhsa:def:20248111: false
     ((( output truncated )))
Definition oval:com.redhat.rhba:def:20228256: false
Definition oval:com.redhat.rhba:def:20228077: false
Definition oval:com.redhat.rhba:def:20225749: false
Definition oval:com.redhat.rhba:def:20223945: false
Definition oval:com.redhat.rhba:def:20223893: false
Evaluation done.
oscap exit code: 0
Copying back requested files...
report.html

      100%  700KB 241.9MB/s   00:00
Removing remote temporary directory...
Disconnecting ssh and removing control ssh socket
 directory...
Exit request sent.
```

4. Next, open a browser and view the report.

TIP

You may wish to copy the report HTML file to a different system for viewing if the graphical desktop is not installed. Review all items. Your output will be different from mine.



file:///home/mstonge/scan-report.html

Customer Portal · Red Hat · Red Hat Products Doc… · Red Hat Enterprise Lin… · Red Hat Developer Por… · Red Hat Container Cat…

OVAL Results Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	cpe:/a:open-scap:oscap	1.3.10	2024-10-16	05:13:27
#×	#✓	#Error	#Unknown	#Other
0	825	0	0	0

OVAL Definition Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	Red Hat OVAL Patch Definition Merger	3	2024-10-16	08:02:18
#Definitions	#Tests	#Objects	#States	#Variables
825 Total 0 0 0 825 0	10454	2356	1797	1

System Information

Host Name	scrapper.local	
Operating System	Red Hat Enterprise Linux	
Operating System Version	9.4 (Plow)	
Architecture	x86_64	
Interfaces	Interface Name	lo
	IP Address	127.0.0.1
	MAC Address	00:00:00:00:00:00
	Interface Name	wlp0s20f0u12
	IP Address	10.82.0.240
	MAC Address	00:2E:2D:80:24:9D
	Interface Name	lo
	IP Address	::1
	MAC Address	00:00:00:00:00:00
	Interface Name	wlp0s20f0u12
	IP Address	fe80::758d:6969:be11:e178
	MAC Address	00:2E:2D:80:24:9D

OVAL System Characteristics Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	cpe:/a:open-scap:oscap	3	2024-10-16	05:13:27

OVAL Definition Results

× ✓ Error Unknown Other

ID	Result	Class	Reference ID	Title
oval:com.redhat.rhsa:def:20248112	false	patch	[RHSA-2024:8112], [CVE-2024-34155], [CVE-2024-34156], [CVE-2024-34158], [CVE-2024-9341]	RHSA-2024:8112: buildah security update (Important)
oval:com.redhat.rhsa:def:20248111	false	patch	[RHSA-2024:8111], [CVE-2024-34156]	RHSA-2024:8111: skopeo security update (Important)
oval:com.redhat.rhsa:def:20248110	false	patch	[RHSA-2024:8110], [CVE-2024-34156]	RHSA-2024:8110: containernetworking-plugins security update (Important)
oval:com.redhat.rhsa:def:20248039	false	patch	[RHSA-2024:8039], [CVE-2024-34155], [CVE-2024-34156], [CVE-2024-34158], [CVE-2024-9341]	RHSA-2024:8039: podman security update (Important)
oval:com.redhat.rhsa:def:20248037	false	patch	[RHSA-2024:8037], [CVE-2024-42934]	RHSA-2024:8037: OpenIPMI security update (Moderate)
oval:com.redhat.rhsa:def:20248025	false	patch	[RHSA-2024:8025], [CVE-2024-9680]	RHSA-2024:8025: thunderbird security update

Figure 13.8 – Viewing the scan report HTML file

Now that you have your first scan report, please take some time to absorb the content and how it is formatted. Make a mental note of any deficiencies it might have found as these are the things we are using this tool to search for. The detailed reporting provided can easily save hours or days of manual checks.

Additionally, I highly recommend keeping these reports as artifacts of your build and testing operations as they will become evidence of your due diligence should you submit your products for government certifications.

In this exercise, we installed our SCAP tools, downloaded a general-purpose SCAP profile from Red Hat, and then executed our first scan. This process is easily repeated with different SCAP profiles that you either obtain from NIST, other third parties, or that you have created yourself. We've just seen a great example of command-line scanning tooling displayed.

Now that you have executed your first scan from the command line, let's move on in our journey by looking at graphical tools to assist with profiles and scanning.

Example: Using the OpenSCAP Workbench

In the previous exercise, we installed several packages. Preemptively, you have already installed the `scap-workbench` and `scap-security-guide` packages. When your scanning system is in graphical UI mode, you'll have the ability to view, edit, and create your own SCAP profiles, run them against hosts, and even automate the remediation of deficiencies found by the scanner.

Let's walk through how we can open and engage with the OpenSCAP workbench in our UI:

1. First, let's open the OpenSCAP Workbench application in the Gnome desktop. Go to **Applications** then select **SCAP Workbench**.

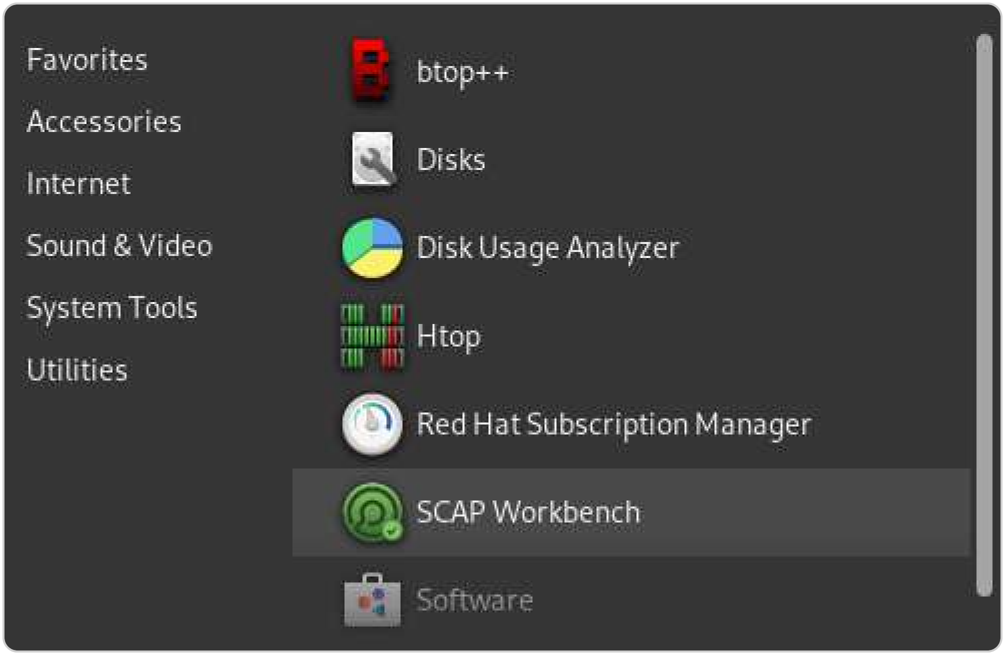
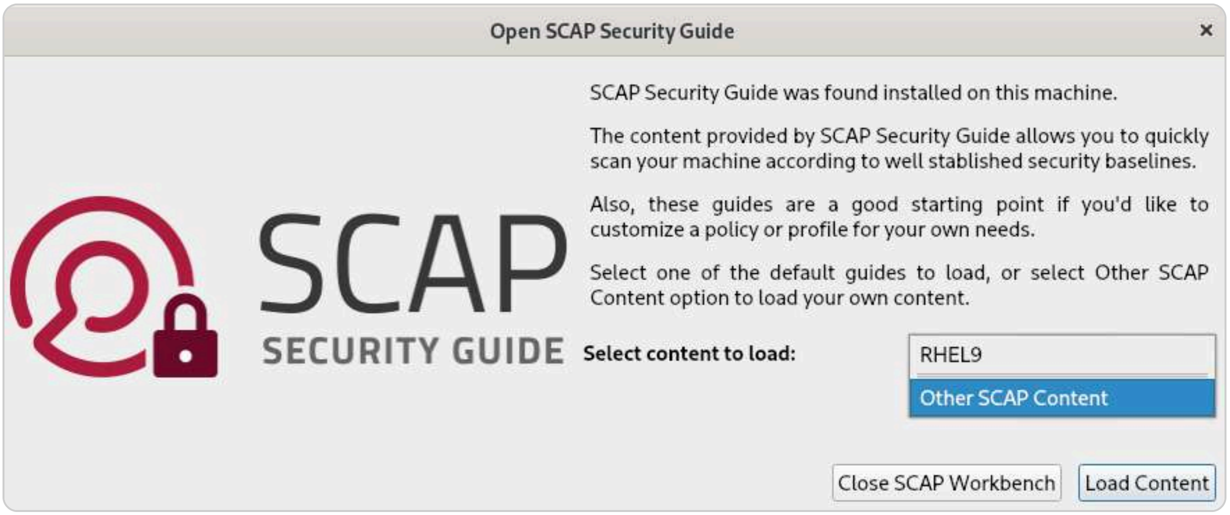


Figure 13.9 – Activating the OpenSCAP Workbench from the Applications Menu in Gnome

2. Once we have started the workbench application, we will first be prompted on which SCAP profile to load. Here, we will select **Other SCAP Content** from the pull-down menu.


Open SCAP Security Guide

SCAP Security Guide was found installed on this machine.

The content provided by SCAP Security Guide allows you to quickly scan your machine according to well stablished security baselines.

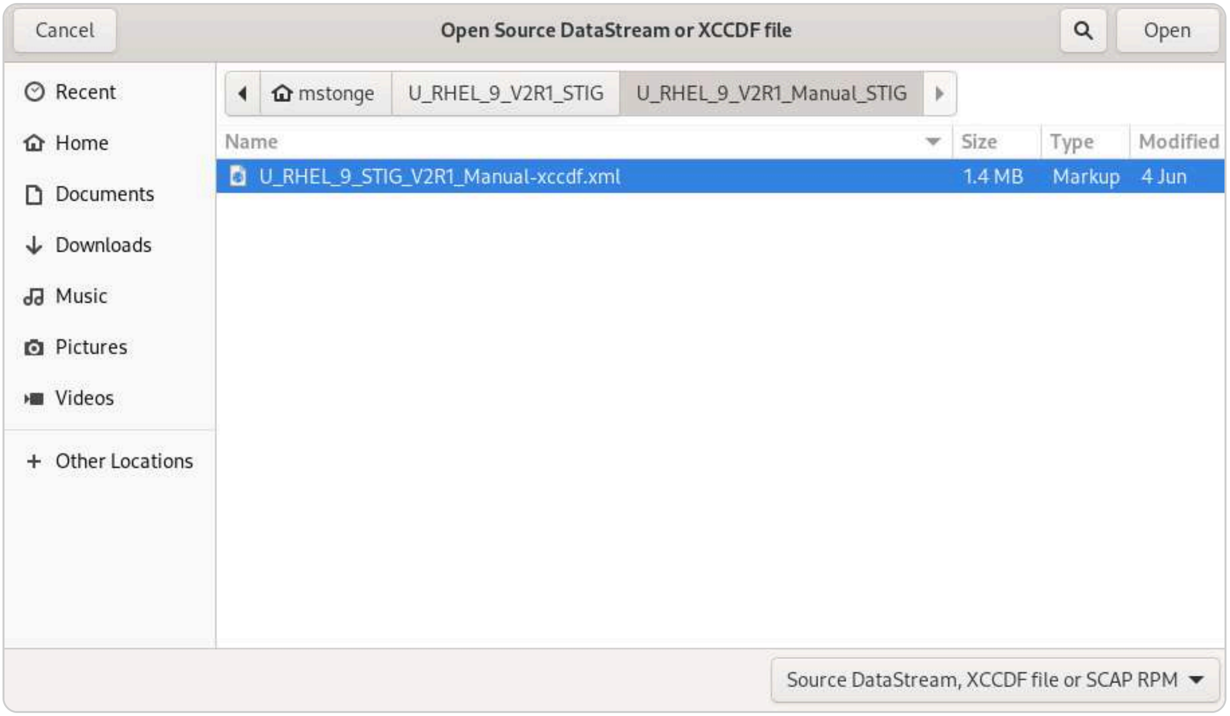
Also, these guides are a good starting point if you'd like to customize a policy or profile for your own needs.

Select one of the default guides to load, or select Other SCAP Content option to load your own content.

Select content to load: RHEL9 / Other SCAP Content

Figure 13.10 – Selecting "Other SCAP content" from the pull-down menu

3. Next, we'll select our previously downloaded RHEL 9 STIG SCAP profile file.


Open Source DataStream or XCCDF file

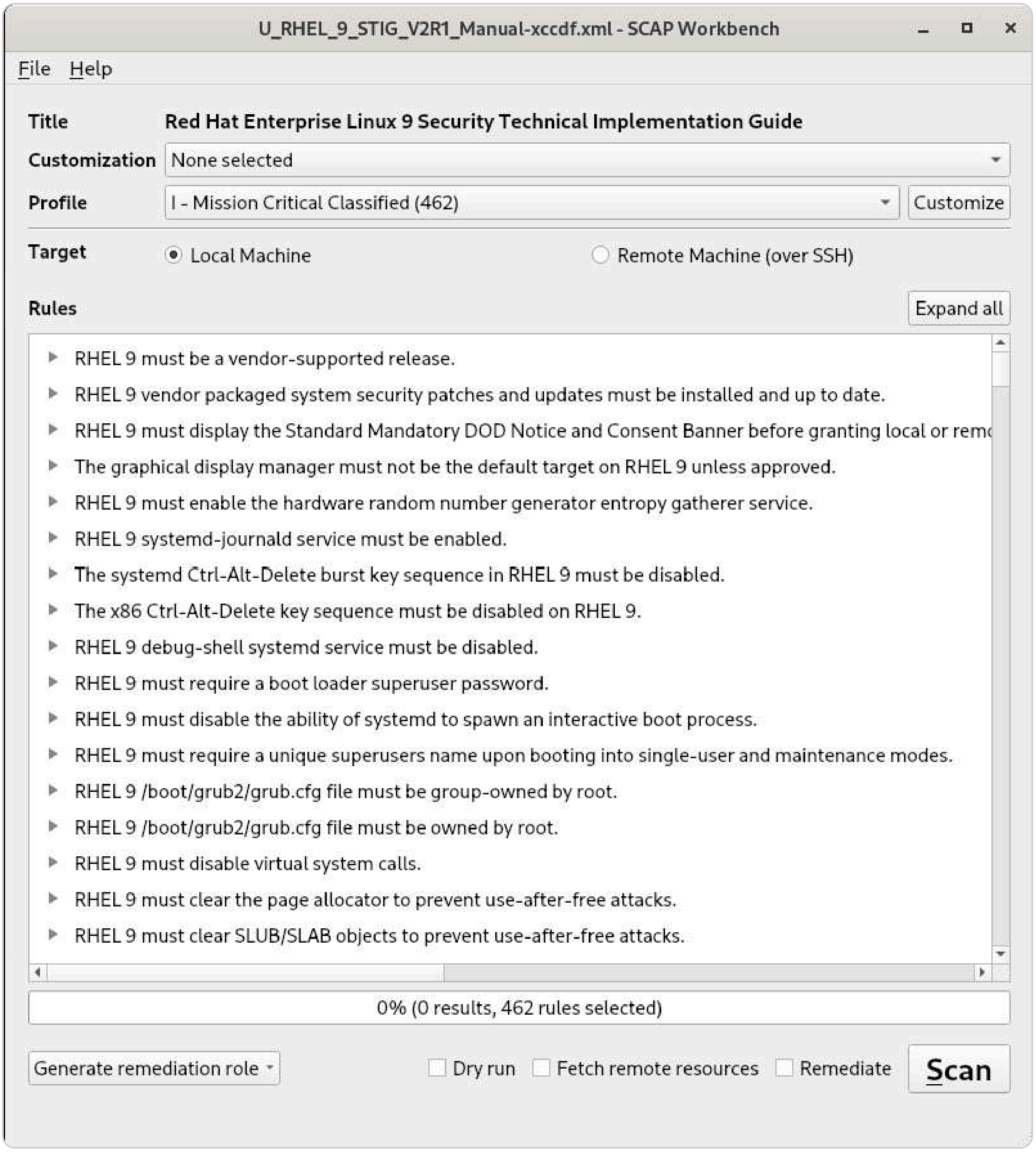
mstonge / U_RHEL_9_V2R1_STIG / U_RHEL_9_V2R1_Manual_STIG

Name	Size	Type	Modified
U_RHEL_9_STIG_V2R1_Manual-xccdf.xml	1.4 MB	Markup	4 Jun

Source DataStream, XCCDF file or SCAP RPM

Figure 13.11 – Selecting the RHEL 9 STIG SCAP profile (downloaded earlier)

Before you decide on your path of execution, here, you can review settings and choose a path forward.


U_RHEL_9_STIG_V2R1_Manual-xccdf.xml - SCAP Workbench

Title: Red Hat Enterprise Linux 9 Security Technical Implementation Guide

Customization: None selected

Profile: I - Mission Critical Classified (462)

Target: (●) Local Machine ( ) Remote Machine (over SSH)

Rules

- RHEL 9 must be a vendor-supported release.
- RHEL 9 vendor packaged system security patches and updates must be installed and up to date.
- RHEL 9 must display the Standard Mandatory DOD Notice and Consent Banner before granting local or rem
- The graphical display manager must not be the default target on RHEL 9 unless approved.
- RHEL 9 must enable the hardware random number generator entropy gatherer service.
- RHEL 9 systemd-journald service must be enabled.
- The systemd Ctrl-Alt-Delete burst key sequence in RHEL 9 must be disabled.
- The x86 Ctrl-Alt-Delete key sequence must be disabled on RHEL 9.
- RHEL 9 debug-shell systemd service must be disabled.
- RHEL 9 must require a boot loader superuser password.
- RHEL 9 must disable the ability of systemd to spawn an interactive boot process.
- RHEL 9 must require a unique superusers name upon booting into single-user and maintenance modes.
- RHEL 9 /boot/grub2/grub.cfg file must be group-owned by root.
- RHEL 9 /boot/grub2/grub.cfg file must be owned by root.
- RHEL 9 must disable virtual system calls.
- RHEL 9 must clear the page allocator to prevent use-after-free attacks.
- RHEL 9 must clear SLUB/SLAB objects to prevent use-after-free attacks.

0% (0 results, 462 rules selected)

Figure 13.12 – Getting ready to execute

From this screen, there are a plethora of actions you can take. You can choose to execute a scan remotely. Others may choose to edit this profile to suit their company's needs. The obvious path is to execute this scan on a host locally or remotely. Here's where I shall also point out that this tool can assist you in generating remediation automation once a scan has been completed.

The OpenSCAP Workbench is a robust and feature-rich tool that I believe deserves more detail than this book can allocate due to space constraints. If you liked this introduction, do not forget to check out the book's GitHub repository for additional content on how to leverage this awesome tool and many other resources.

Let's now move on to our next section, where we will discuss implementing security scans into your CI/CD chain.

Implementation as part of your continuous integration/continuous deployment (CI/CD) process

This is not just a repeat of your QA efforts but a review of which standards and certifications you currently maintain and what efforts you may choose to add to your product or cease to maintain.

Following updates from the agencies that provide the security standards that your appliance must comply with is crucial in keeping abreast of changes or new regulations.

For agile shops, I highly recommend adding these activities to your backlog and ensuring this review takes place at least twice a year. The more often your team performs a review, hopefully, the more security remains in their minds.

Additionally, this is where I would also prescriptively recommend having a distinct level of automation in place that runs these scans against any release candidate host. Which automation tools you select is up to you or whatever your employer has already deemed the corporate standard. Perhaps you prefer scripts. Let's dig into this a little deeper.

In the previous exercises, we downloaded the STIG SCAP profile and a general-purpose RHEL SCAP profile, set up a scanning host, and learned how to execute a scan. To make this a more normalized process, I have some recommendations for your CI/CD chain and your product prototypes.

Next, I'd like to share some lists of recommendations I have for your build chain, tasks to perform on your prototypes, and more.

These are my recommendations for your CI/CD chain:

- Build a permanent host for running these OpenSCAP scans
- Maintain a library of appropriate SCAP profiles
- Automate the scanning activities and distribution of the results reports
- Save your scan results as artifacts with your builds/releases for future reference
- Throw nothing away – maintain an audit trail (always)

These are my recommendations for your prototypes/release candidates:

- Start with the highest level of security possible
- Enable only what sockets, ports, or services are critical to your appliance's functioning
- Have non-root service accounts that can run systemd services and be used for support and scanning activities
- Ensure each host has SSH enabled
- Ensure that the **`openscap-utils`** package is installed

Failure to integrate and automate compliance scanning into your process in the long term (and maybe the short term too) will negatively impact the quality of your product. My final recommendation is not to ignore security concerns during any stage of your product's life cycle. Make it a component of every process and, eventually, it will become second nature for you and your product team.

Let's move on to the trophy phase of your efforts: actually getting your product certified.

How do I certify my solution?

The subject of getting your solution certified probably deserves a whole book on its own. There are so many different types of certifications for different industry and government standards. That said, for our security focus I have chosen to distill this into the holy grail of certifications: the coveted FIPS 140-3 certification with NIST.

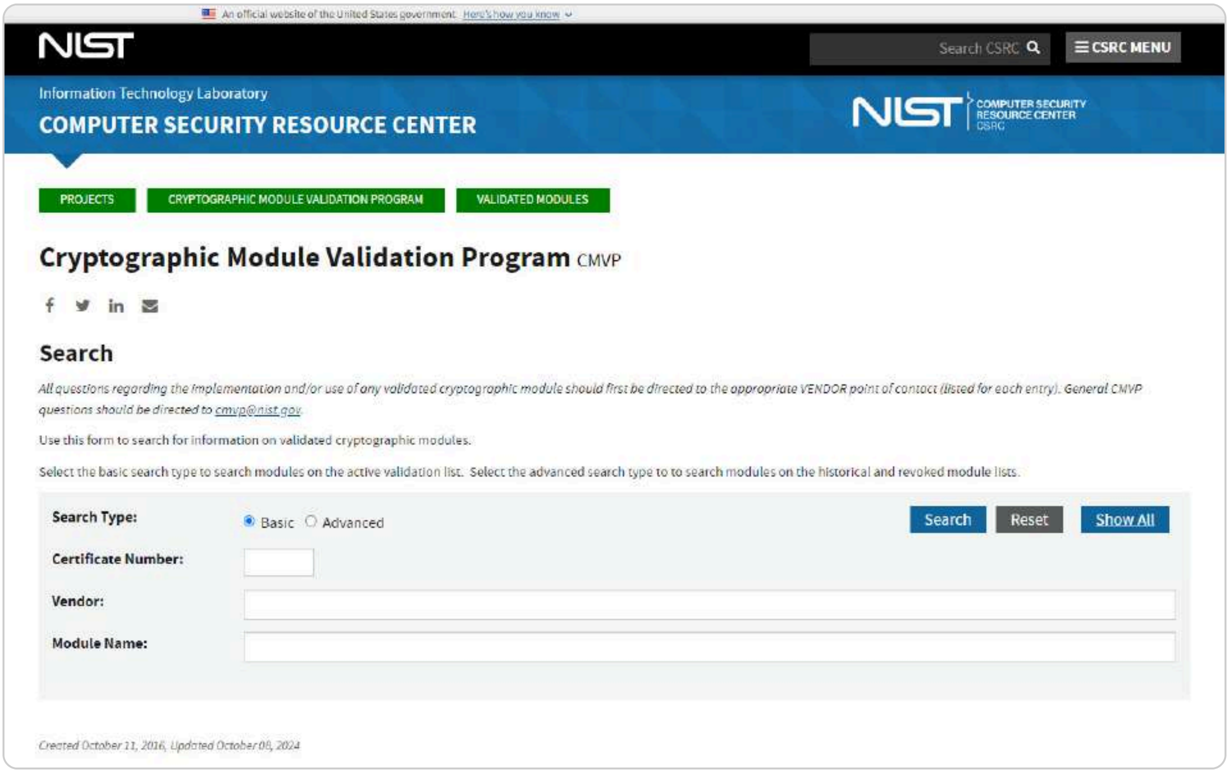
Not to discourage or dissuade you, but I must mention that several companies have built a business model to *help* other businesses navigate this painstaking and lengthy process with the labs and NIST. You can easily find them online with a simple Google search when looking for FIPS certification. Their services will most likely show at the top of all results.

FIPS 140-3 certification is not just a software certification, but a software on specified hardware validation. The testing labs that provide these services to the process use a specific hardware platform for each testing cycle. Which platform that is may vary by the labs themselves.

The process is complex, and not without costs. Depending on the security level of your solution's certification, you could be paying NIST from $1,000 to well over $4,000. This does not include any fees charged by the lab that does the heavy lifting for NIST.

To find a NIST-accredited lab, you need to search the NIST website. Here's a good place to start: https://www-s.nist.gov/niws/index.cfm?event=directory.search#no-back.

Once the lab has tested your product and submitted its findings to NIST, the NIST team then reviews all the documentation and hopefully, eventually, issues a certification. The coveted prize can then be searched for as all validated FIPS solutions (and their statuses) can be found at the NIST site shown in the following screenshot.


An official website of the United States government Here's how you know

Search CSRC

CSRC MENU

Information Technology Laboratory

COMPUTER SECURITY RESOURCE CENTER

PROJECTS | CRYPTOGRAPHIC MODULE VALIDATION PROGRAM | VALIDATED MODULES

Cryptographic Module Validation Program CMVP

Search

All questions regarding the implementation and/or use of any validated cryptographic module should first be directed to the appropriate VENDOR point of contact (listed for each entry). General CMVP questions should be directed to cmvp@nist.gov.

Use this form to search for information on validated cryptographic modules.

Select the basic search type to search modules on the active validation list. Select the advanced search type to to search modules on the historical and revoked module lists.

Search Type: [✓ Basic] Advanced

Certificate Number:

Vendor:

Module Name:

Search | Reset | Show All

Created October 11, 2016, Updated October 08, 2024

Figure 13.13 – Searching for validated/certified modules

FIPS certification re-branding by vendors

The submittal, testing, validation, and certification process can take over two years. For anyone trying to bring a new competitive product to market, this timeframe is unfathomable. Few operating system vendors can lend a hand even if they are certified themselves. Red Hat has a program in which their Embedded Systems partners can leverage their existing certifications for their products that are built upon RHEL. How do I know this? I was one of three founders of the program.

A Red Hat partner choosing to take this path can work with the Red Hat Embedded Team, and they will provide appropriate paperwork (i.e., regarding the existing software certifications) that the partner can then take to NIST and their chosen lab to accelerate the process to get their solution certified. With this certification re-branding documentation, the process, which could have taken over two years, may be shortened to a few weeks or a few months as testing an existing certified module is not necessary. This provides extreme value and a path to revenue for that partner.

Summary

We've come a long way together in this chapter. We have reviewed how adherence to government standards is important regardless of whether or not your industry dictates so. We've also taken a glance at some key standards that transcend many verticals and how to implement the most important ones. Finally, we wrapped up how to certify your appliance with NIST and some of the fun that process will bring you. Your experiences with the examples and exercises should give you newly found confidence that these efforts are not impossible, but very achievable.

Let's move on to our final chapter, where we will discuss our most important resource, our end-users, and how their feedback can assist your processes. We'll also do a full review of the lessons learned throughout the book.

14
Customer and Community Feedback Loops

Welcome to the final chapter, where we will hopefully help you close the infinite loop by guiding you on how to engage your prospective community. Ultimately, it is your customers that drive your solution by having an unfulfilled need. Don't just help them to scratch that itch; help them to achieve great success. Their success becomes yours as well.

The goal of this chapter is not to teach you CI/CD but to enlighten you on the value provided by continuous interaction and feedback from your end-users, executives, and the overall community that leverages your product to solve their problems. You can expect these interactions to provide invaluable insights into use cases, methodologies, and usage patterns that you may not have previously foreseen. The data acquired will fuel future features and innovations, while undoubtedly helping you make your product more secure in the process. This is where your customers return value to the cause.

Your efforts in gathering feedback are the final step in the perpetual **CI/CD loop** before progressing to develop the next update:

1. Plan
2. Build
3. Continuous integration
4. Deploy
5. Continuous feedback

There will be no hands-on exercises in this chapter, but do not think that you'll have no work to do. I will be setting some challenges for you and your team. How you address these challenges could determine the overall success or grievous failure of your project. So, without further ado, let's take those final steps and close the CI/CD loop together.

In this chapter, the main sections are:

- Use case development
- User groups
- Executive roundtables
- Community feedback loops
- Closing the loop: End-of-book summary

Let's now move on to the first section, where we'll review your use cases.

Use case development

Wait, didn't we cover this earlier in the book? Hmmmm. Yes, we did, actually – back in *Chapter 2*. But we just scratched the surface. What you're going to find in this chapter will be far more… prescriptive. A significant part of this chapter will be the flip side of the coin of what I have previously mentioned about engagement.

This will not be redundant. Previously, I tasked you with engaging more with your customers. That is assumed by now. In this chapter, where we are closing the entire CI/CD loop, I am challenging YOU to take the steps beyond basic engagement. In fact, I am recommending that you and your team create your own specific engagement methods with your prospective customer base and, more so, create a community. In other words, don't just join a community – build and lead one!

Here, I am tasking you with truly getting actual information, research, and guidance from those who shall become your customer base. These people are the truest key to getting you to a spot where you can actually define what your future product must do and what security implications must be adhered to, as well as understanding what features they want versus must have.

Use cases themselves will determine how your future user base will leverage your solution. How they consume the services that your appliance provides will drive security measures that must be planned for (UI, user-level access, tamper-proofing, etc.). How you get this information is the challenge this chapter is dedicated to solving.

Additionally, feedback from your users could potentially identify security concerns or missing security measures that your team might not have incorporated into that release of the appliance. Encouraging open feedback with your users is a great source of product improvement recommendations – always. Ignore their feedback at your own peril.

Let's progress to our first method of information gathering – creating and hosting **user group** meetings.

User groups

Earlier in this book, in *Chapter 11*, I suggested that you get involved in security community user groups. I am not deviating from that recommendation. This section is not remotely about that. Here, I am challenging you to find the highest concentration of what could be your user base and create a user group in that area.

This user group will be focused on your product, how it's used, its future, and what they need you to maintain or improve. Create your own community for your products. I cannot emphasize how important this is. The feedback you get from these people will be crucial to the success of your products. If the user base doesn't believe you are progressing in the correct direction, they might just abandon your product.

Learning exactly how your products are deployed and consumed might surprise you. You might find out they are not being used as intended. That in itself could impact the security and integrity of your product. Conversely, you could also come to the conclusion that a function or security precaution may have been overlooked.

You might even get excellent insight from your users on not only how to make their experiences more pleasurable and efficient but also what features could set your product apart from others. Never underestimate the gems user groups could reveal.

How you engage these people will alter how they perceive your product and, ultimately, how they perceive your company as a whole. Engage, be open, and, most of all, open your ears to their thoughts and concerns. These interactions are priceless opportunities to establish trust with your user base by not only letting their voices be heard but also reinforcing that their opinions matter. When they know their opinions matter, they feel like part of the solution themselves and more like a community, rather than just a customer to a vendor.

Here's an example of what a successful user group engagement might look like.

Figure 14.1 – User groups drive customer engagement

These meetings must never be sales-y; in fact, you should keep your sales group away. These groups work best when your users can interact with product management and engineers. *Techies talking to techies* – this works best.

Here is a recommendation. Near the end of your first few meetings, query the audience on their thoughts on the meeting cadence, content they'd like to be covered in future sessions, and the duration of the gatherings (are they too long, too short, or just right?).

One thing I have always found engaging and that leads to success is asking people in the audience to openly share their thoughts on a recent experience that they have had with the product, whether positive or negative. Nothing should be formal. This is a great way to encourage attendees to become more engaged.

Finally, once your user group is somewhat well established, your next mission is to encourage customers and partners to be part of the agenda. Presentations, demos, or just a quick lightning talk performed by either a customer or a partner go a great way toward building your community and securing trust. Let them talk about the good, the bad, and the ugly, along with HOW they solved a problem and the product added value to their organization.

In the next section, we'll review how to engage a different level of leadership – how to get executive sponsorship – within your customer base.

Executive roundtables

While engaging your user base can create a great passion for your product and provide significant insights into usage patterns, creating meaningful relationships with industry executives will drive adoption and visibility and establish a true conduit for your customers to help you drive innovation and greater usability and security in your product.

Having an executive sponsor grants you a *force multiplier*. That executive can drive the rapid adoption of your product. Moreso, they can provide your team with crucial insights into how implementation and integration operations are actually preceding and how your product is perceived.

Executives might also take an interest in liaising or coordinating features with your engineering teams. Their insights on security and compliance could alter your perspective.

Here's what an executive roundtable session might look like for your team.

Figure 14.2 – Executive roundtables drive engagement

Now that you've obtained executive sponsorship and established key leadership channels of communication, let's move on to how to tap into the general community.

Community feedback loops

There are many methods that you can leverage to engage your community. Use all of them: events, user groups, roundtables, surveys, and active research efforts. Earlier in the book, specifically *Chapter 11*, I challenged you to get involved within the Linux and security communities. Here, I am outright challenging you to create your own community.

Once your company starts selling its product, like it or not, your users will find a way to share information informally. These information-sharing activities may start on message boards and evolve naturally into informal groups.

This is your greatest opportunity to shape the evolving community. Host message boards. Create a user group somewhere where there is a high concentration of your users. This might mean creating several unique group meeting environments across multiple geographic regions or cities. Start small where the largest concentration of your users exists and get feedback from them when you decide to expand and create new groups in other locations.

You likely won't be able to address all pockets of users across the globe. You can keep your users engaged with newsletters and surveys. Never expect a 100% response rate to surveys unless something bad has occurred with your product. But these communications are no substitute for direct interactions. The goal here is not simply continuous engagement (which does have some value) but hopefully to encourage a user base to further become a community by having their voices heard and keeping them informed. My personal opinion is that you can actually engage too much. Don't be like a certain unnamed online vendor that asks you to complete a survey for every order. Make your interactions add value to the recipient or do not interact at all.

This is where I recommend hosting events. The target audience for these is everyone. Possible customers, partners, users, executives, and industry-focused media are all your desired targets. Each will provide their own opinions and input. These events can be made up of multiple simultaneous sessions, with different target audiences all at the same time. Product information, training, and panel-based Q&A sessions are the kings here.

Ultimately, I am not speaking about how to market your product here. So many others could do a much better job at discussing that than I could. My point to all of this is to get and stay engaged with your community as a whole and to make it grow positively. Knowing their needs and then addressing them in your solution will continue to drive success and help you achieve a more secure product in the process.

Summary

We've arrived at the end of the trail now. This final chapter has touched on critical aspects of how to gather and ingest feedback from outside your organization. These efforts, often running in parallel to each other, will altogether give your team a virtual weather report of sorts as to exactly how your product is received and perceived and what changes you can make to make gains in security and adoption. They are also a great indication of what the community feels your product got right and you should sustain in future releases... Take these golden nuggets for what they are – an amazing level of insight that you didn't have to pay industry analysts for.

Finally, let's move on to a wrap-up of the book, or the end-of-book summary.

Closing the loop

First, I have to say – thank you for taking this journey with me. Albeit this is the end of the book, but I truly hope that it is just the beginning of a new journey for you, and that that journey has many bright things in store for you and your company. It has been a pleasure and an honor to have been your guide.

For me, this is the end of the book, the end of my narrative, and the end of my security-focused rant, but not the end of the story. I will continue to provide updates to the book's GitHub repository. You should check there periodically to see what new goodies have been added. It could be revised labs, updates, or more examples of embedded Linux awesomeness.

Putting all the pieces together

The intent of this book was to take you, my much-appreciated reader, through a progression of knowledge that you can apply to your projects regardless of the decisions you had to make. Let's go through this progression of best practices, security operations, configurations, and tooling and how we should put them all together.

In the first part of the book – *Introduction to Embedded Systems and Secure Design* – we took a deep introspective journey into embedded Linux systems: what these systems are, who uses them, and the arduous task of applying design criteria.

The key concepts you should now have a firmer grasp of are understanding what an embedded Linux system appliance is, who uses it, and how compliance efforts drive the overall solution, starting with the selection of the hardware and operating system itself.

In the second part of the book – *Design Components* – we took the knowledge gained from understanding our design criteria and applied different security components to our designs. This is where we actually started playing with many technologies in a hands-on manner. We left behind the theoretical mentality of the first section and replaced it with practical application.

The key concepts you should have taken away from this section started with what tools you might need to implement within your build chain, followed by some very detailed and specific security measures to take within your software and hardware configurations, and closed with all the idiosyncrasies induced by tamper-proofing your solution from the end-users themselves.

In the third and final section of the book – *The Build Chain, Appliance Lifecycle, and Continuous Improvement* – we addressed how one can create a sustainable CI/CD chain that doesn't just automate builds and tests. To have the most perfect loop, there must also be human inputs and outputs of this chain. That means you need to create a community for your product/project. The care and feeding of this community are as important as the level of attention and due diligence that you give your product.

The takeaways from this section are enormous. By this point, you should be putting everything together in your head. You now know what resources are at your fingertips to keep you informed and ahead of the game. You also know how to practically test and implement security in your solution. Which brings us to where we are now, in this chapter, where we closed the continuous feedback loop with insights from your community.

We have shared a long journey together, and I'd like to move on to my final challenge for you.

Staying engaged

This should not be the end of the road for you. The single greatest challenge that I can leave you with now is for you to stay engaged, and even more so to stay vigilant. The threat landscape is ever-changing, so you must stay ahead of the curve by staying informed and educated.

Don't just keep yourself engaged. Keep your team engaged. It will continue to take a village to ensure the security of your product. It's not a one-time effort. Don't let your guard down after a couple of successful releases.

Another *don't* – I implore you to heed this warning: "Don't stop learning!" Evolve personally and professionally faster than the threat landscape. Drive your own success. Security is more than a process – it's a state of mind.

Again, thank you for your time, support, and attention. I truly hope this book has helped you.

THE END… or a new beginning? It's your call. What do you plan to do next?

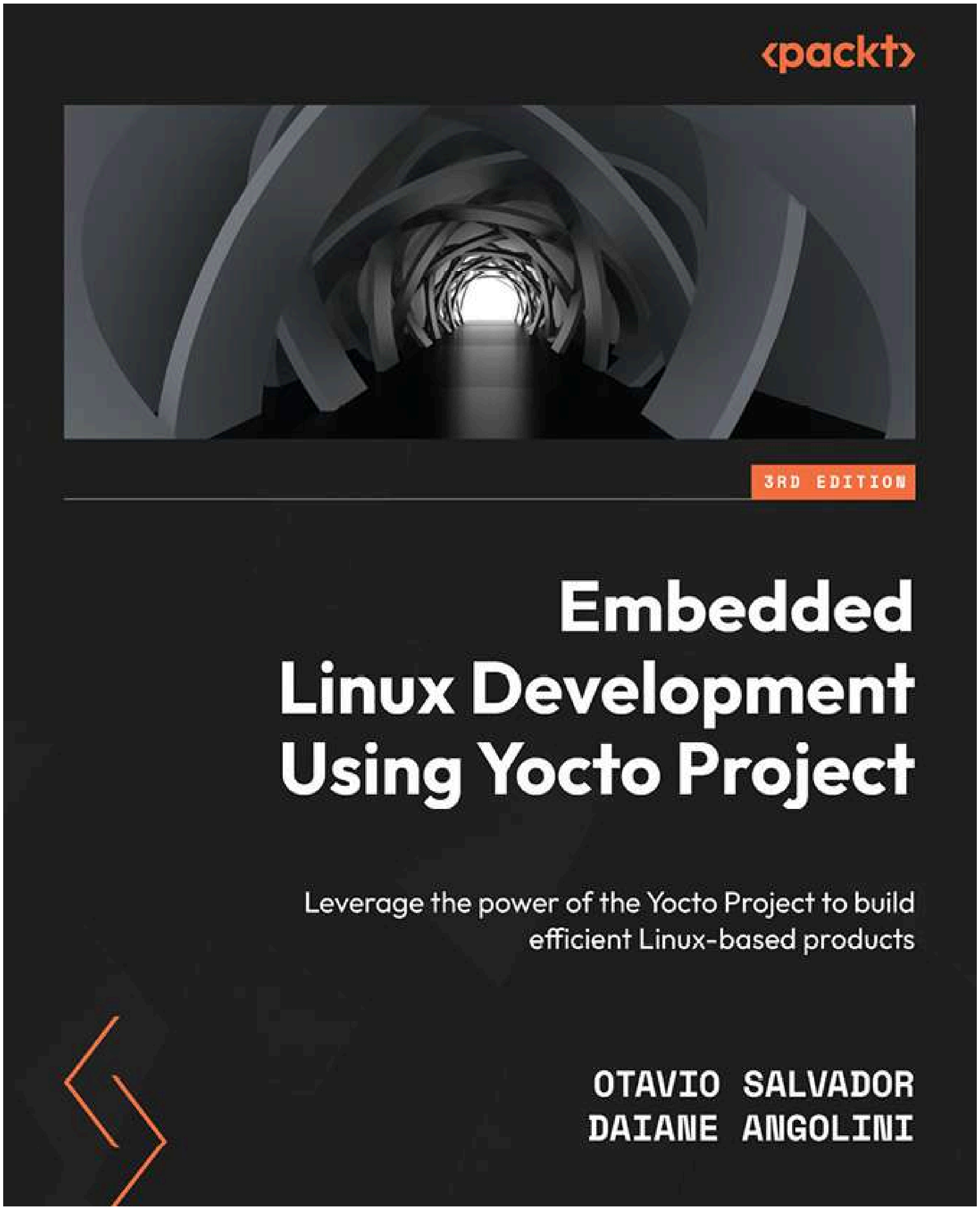
Join our community on Discord

Join our community's Discord space for discussions with the authors and other readers:

https://packt.link/embeddedsystems

Index

As this ebook edition doesn't have fixed pagination, the page numbers below are hyperlinked for reference only, based on the printed edition of this book.



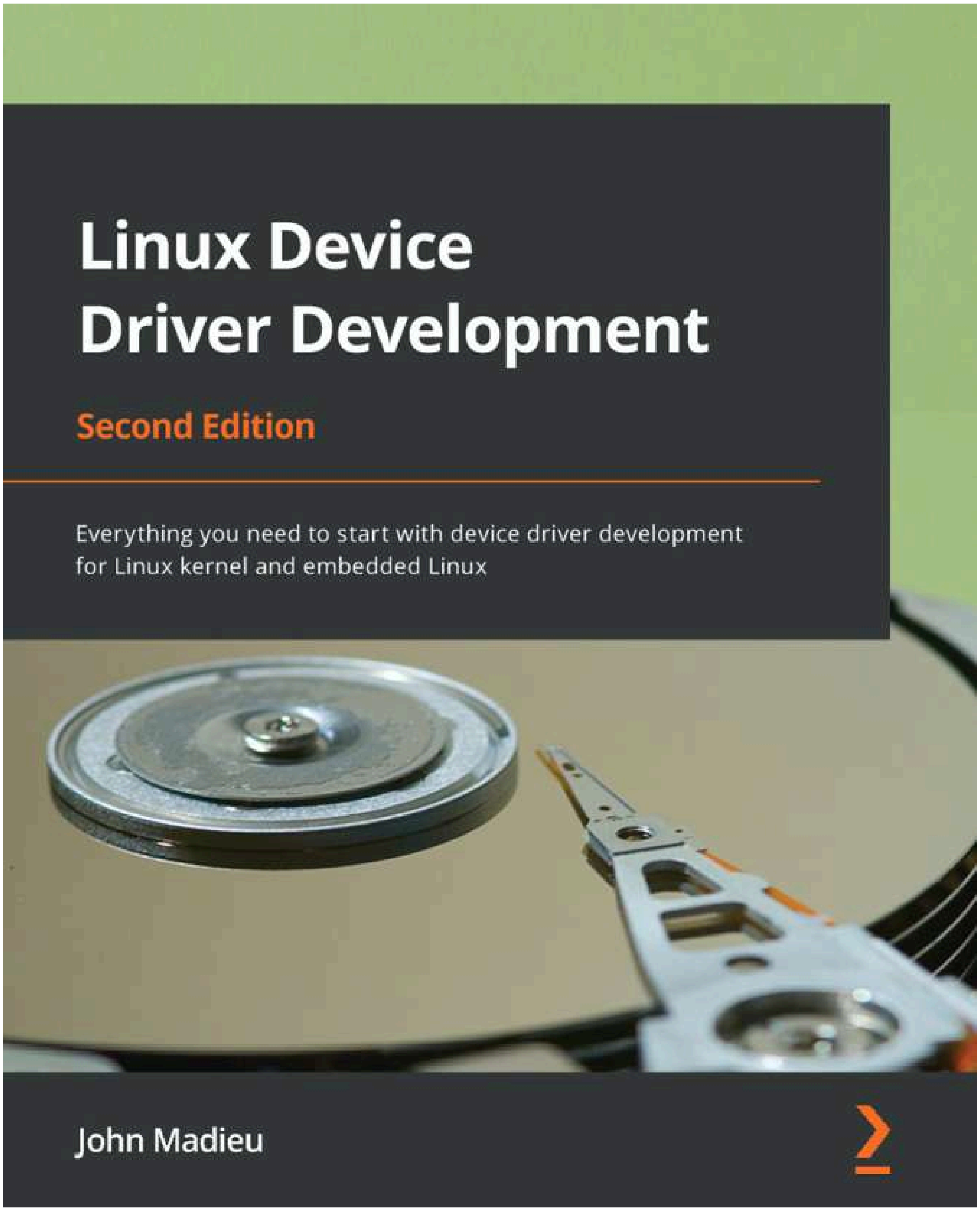
<packt>
3RD EDITION
Embedded Linux Development Using Yocto Project
Leverage the power of the Yocto Project to build efficient Linux-based products
OTAVIO SALVADOR
DAIANE ANGOLINI

Embedded Linux Development Using Yocto Project

Otavio Salvador, Daiane Angolini

ISBN: 978-1-80461-506-5

- Understand the basic Poky workflows concepts along with configuring and preparing the Poky build environment
- Learn with the help of up-to-date examples in the latest version of Yocto Project
- Configure a build server and customize images using Toaster
- Generate images and fit packages into created images using BitBake
- Support the development process by setting up and using Package feeds
- Debug Yocto Project by configuring Poky
- Build an image for the BeagleBone Black, RaspberryPi 4, and Wandboard, and boot it from an SD card


Linux Device Driver Development
Second Edition
Everything you need to start with device driver development for Linux kernel and embedded Linux
John Madieu

Linux Device Driver Development

John Madieu

ISBN: 978-1-80324-006-0

Packt is searching for authors like you

Share Your Thoughts

Download a free PDF copy of this book

Contents

Landmarks