BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF BUSINESS AND MANAGEMENT
FAKULTA PODNIKATELSKÁ

INSTITUTE OF INFORMATICS
ÚSTAV INFORMATIKY

QUALITY ASSURANCE OF RHEL SYSTEMS WITH A HIGH LEVEL OF SECURITY

ZABEZPEČENÍ KVALITY SYSTÉMŮ RHEL S VYSOKOU ÚROVNÍ ZABEZPEČENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. LUKÁŠ JAVORSKÝ**
AUTOR PRÁCE

SUPERVISOR — **Ing. PETR SEDLÁK**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF BUSINESS AND MANAGEMENT

Assignment Master's Thesis

Department:	Institute of Informatics
Student:	**Bc. Lukáš Javorský**
Supervisor:	**Ing. Petr Sedlák**
Academic year:	2023/24
Study programme:	Information Management

Pursuant to Act no. 111/1998 Coll. concerning universities as amended and to the BUT Study Rules, the degree programme supervisor has assigned to you a Master's Thesis entitled:

Quality assurance of RHEL systems with a high level of security

Characteristics of thesis dilemmas:

Introduction
Theoretical background of the work
Analysis of the current state of the art
Own proposals for solutions
Conclusion

Objectives which should be achieve:

The goal is to establish a testing process for Red Hat Enterprise Linux subsystems in accordance with security standards and recommendations, including security recommendations for testing processes.

Basic sources of information:

ČSN EN ISO/IEC 27001 Informační bezpečnost, kybernetická bezpečnost a ochrana soukromí - Systémy managementu informační bezpečnosti - Požadavky, 2023. [Praha]: Česká agentura pro standardizaci.

ČSN EN ISO/IEC 27002 Informační bezpečnost, kybernetická bezpečnost a ochrana soukromí - Opatření informační bezpečnosti, 2023. [Praha]: Česká agentura pro standardizaci.

DOUCEK, Petr; KONEČNÝ, Martin a NOVÁK, Luděk, 2019. Řízení kybernetické bezpečnosti a bezpečnosti informací. Praha: Professional Publishing. ISBN isbn978-80-88260-39-4.

SEDLÁK, Petr a KONEČNÝ, Martin, 2023. Přeměna ISMS v manažerské informatice. Brno: CERM, akademické nakladatelství. ISBN isbn978-80-7623-110-8.

SEDLÁK, Petr a KONEČNÝ, Martin, 2021. Kybernetická (ne)bezpečnost: problematika bezpečnosti v kyberprostoru. Brno: CERM, akademické nakladatelství. ISBN isbn978-80-7623-068-2.

Deadline for submission Master's Thesis is given by the Schedule of the Academic year 2023/24

In Brno dated 4.2.2024

L. S.

doc. Ing. Miloš Koch, CSc. Branch supervisor	doc. Ing. Vojtěch Bartoš, Ph.D. Dean

Abstract

The aim of this thesis is to develop a quality assurance pipeline for RHEL operating systems with security hardening. This thesis is driven by the desire to develop a user-friendly testing environment for quality engineers, allowing them to easily verify the functionality of system packages in security security-hardened testing environment. The achievable outcome was realized by the integration of Red Hat's existing software portfolio, namely Image Builder, and OpenSCAP, resulting in the establishment of pre-hardened virtual machines. Subsequently, integration with Red Hat's test management application Testing Farm, serves to simplify virtual image provisioning and providing an established testing pipeline.

Abstrakt

Cieľom diplomovej práce je vyvinúť proces zabezpečenia kvality pre operačné systémy RHEL s vysokou úrovňou zabezpečenia. Táto práca je vedená snahou vytvoriť používateľsky prívetivé testovacie prostredie pre inžinierov kvality, ktoré im umožní jednoducho overiť funkčnosť systémových balíkov aj v prostredí zabezpečeného testovaného systému. Dosiahnutý výsledok bol realizovaný integráciou existujúceho softvérového portfólia spoločnosti Red Hat, konkrétne aplikáciami Image Builder a OpenSCAP, čo viedlo k vytvoreniu predpripravených zabezpečených virtuálnych strojov. Následne integrácia s aplikáciou na správu testov Testing Farm spoločnosti Red Hat slúži na zjednodušenie poskytovania cloudového virtuálneho stroja a poskytovanie zavedeného testovacieho prostredia.

Keywords

RHEL, Security hardening, Cybersecurity, Security compliance, OpenSCAP, Quality assurance, Testing, Testing Farm, Virtual Machine, Virtual image, Image Builder, Functional testing

Kľúčové slová

RHEL, Bezpečnostný hardening, Kybernetická bezpečnosť, Bezpečnostné normy, OpenSCAP, Zabezpečenie kvality, Testovanie, Testing Farma, Virtuálny počítač, Virtuálny obraz, Image Builder, Funkcionálne testovanie

Bibliographic citation

JAVORSKÝ, Lukáš. *Quality assurance of RHEL systems with a high level of security* [online]. Brno, 2024 [cit. 2024-03-28]. Available at: https://www.vutbr.cz/studenti/zav-prace/detail/158745. Master's thesis. Brno University of Technology, Faculty of Business and Management, Institute of Informatics. Supervisor Ing. Petr Sedlák

Rozšírený abstrakt

Diplomová práca sa zameriava na vývoj komplexného procesu zabezpečenia kvality pre operačné systémy Red Hat Enterprise Linux (RHEL), s dôrazom na dosiahnutie vysokého štandardu bezpečnosti v rámci definovaných bezpečnostných noriem. Hlavným zámerom tejto práce je vytvorenie intuitívneho a používateľsky prívetivého testovacieho prostredia určeného pre inžinierov kvality v rámci firmy Red Hat. Toto prostredie im umožní jednoducho overovať funkčnosť systémových balíkov aj v kontrolovanom a bezpečnom testovacom prostredí, ktoré im bude kedykoľvek k dispozícii. Kvôli jednotlivým požiadavkám inžinierov kvality bolo pre túto prácu vybrané riešenie za pomoci virtuálnych cloudových obrazov (image), ktoré sú hostované v štandartizovanom prostredí pre kontrolu kvality a taktiež pre kontinuálnu integráciu a kontinuálnu distribúciu. Toto prostredie je poskytované Testing Farm infraštruktúrou, ktorá je taktiež spravovaná tímom v rámci firmy a preto je jeho integrácia v rámci diplomovej práce jednoduchšie prevediteľná.

Koncovému riešeniu predchádzala dôkladná analýza možných riešení danej problematiky. Súčasťou tejto analýzy bolo vypracovanie viacero možných riešení a zároveň ich jednotlivé výhody ako aj nevýhody. Následne boli tieto riešenia predstavené zainteresovaným tímom a jedno z nich bolo vybrané ako finálne pre jeho najoptimálnejšie charakteristiky. Po vybraní vhodnej varianty bolo taktiež na mieste oslovenie všetkých tímov, ktoré boli zainteresované vo finálnom produkte. Tieto tímy boli taktiež súčasťou pilotného projektu, ktorý mal za úlohu preukázať funkčnosť a efektivitu vybraného riešenia.

Pre dosiahnutie koncového výsledku tejto snahy boli využité viaceré existujúce softvérové riešenia z portfólia firmy Red Hat. Jednotlivé zabezpečovanie daných virtuálnych operačných systémov bolo spracované pomocou nástrojov, ktoré sú súčasťou OpenSCAP projektu, vedeného tímom RHEL Security Compliance tímu. Generovanie a upravovanie už spomínaných virtuálnych operačných systémov bolo dosiahnuté za pomoci softvérového nástroja Image Builder Composer. Systémová služba poskytovaná týmto nástrojom je schopná vygenerovať virtuálny obraz na základe užívateľských požiadavkov v rámci preddefinovaného štandartizovaného návrhového súboru (blueprint). Tento súbor definuje jednotlivé parametre výsledného obrazu ako napríklad jeho predinštalované softvérové balíky, veľkosti systémových partícií, nastavenie pravidiel kernela alebo nastavenie bezpečnostného štandartu pomocou OpenSCAP profilov definovaných v jeho dátových tokoch (datastream). Výsledný virtuálny obraz je následne integrovaný v rámci spomínanej Testing Farm infraštruktúry, pre jednoduchú distribúciu pomocou cloudového prostredia. Tomuto kroku integrácie predchádzala dohoda o vytvorení štandartizovaného názvoslovia pre virtuálne obrazy s prednastaveným bezpečnostným profilom. V tomto prípade bolo vybrané názvoslovie *RHEL-9.3.0-updates-20240104.37-CIS-HARDENED*, ktorého časti identifikujú jednotlivé parametre konkrétneho virtuálneho obrazu. Prvá časť *RHEL-9.3.0-updates-20240104.37* identifikuje konkrétne použitý compose, ktorý slúži pre distribúciu špecifických verzií softvérových balíkov, čo je potrebné pri kontrole kvality ešte pred samotným vydaním týchto balíkov do produkcie. Druhá časť názvu *CIS-HARDENED* zase identifikuje konkrétny bezpečnostný profil použitý pre zabezpečenie daného virtuálne obrazu. Zvýšenie bezpečnosti týmto profilom môže mať za následok obmedzenie funkčnosti jednotlivých softvérových balíkov. Je preto nevyhnutné otestovať, či obmedzenie tejto funkcionality neovplyvní kritické aspekty systému a či je toto obmedzenie nevyhnutné pre dosiahnutie vyššieho stupňa bezpečnosti.

Nasledujúcim krokom po dokončení implementácie riešenia bola ekonomická a časová evaluácia tohto projektu. Súčasťou ekonomickej sekcie tohto vyhodnotenia bol vypočítaný ukazateľ Return on Security Investment (ROSI), ktorý jasne ukázal, že táto investícia je pre

firmu finančne atraktívna. Z časovej komponenty je taktiež zrejmé, že sa investícia oplatí pre jej časovú efektivitu, v rámci tímov inžinierov kvality. Súčasťou tohto vyhodnotenie je taktiež dôkladná analýza zákazníckeho problému, ktorý bol priamo zapríčinený kombináciou ťažko dostupnej kontrole kvality a zabezpečených systémom pomocou OpenSCAP nástroja. Výsledky analýzy odkrývajú zamaskované dáta, ktoré poskytujú dôležité informácie pre porovnanie nákladov spojených s nápravou vzniknutého problému.

Poslednou časťou diplomovej práce bol návrh budúcich riešení projektu. Keďže riešenie danej problematiky si žiada veľké časové aj ekonomické investície, diplomová práca slúžila ako koncept, ktorého následné výsledky a analýzy budú spracované do finálneho produkčného riešenia. Následné kroky k tomuto riešeniu ako aj návrh pre konkrétne testovacie cykly, v ktorých sa testovanie zabezpečených systémom môže vykonávať, boli taktiež spracované v tejto časti práce.

Quality assurance of RHEL systems with a high level of security

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Ing. Petr Sedlák. The supplementary information was provided by Bc Marek Haičman. I have listed all the literary sources, publications, and other sources, that were used during the preparation of this thesis.

.....
Lukáš Javorský
May 9, 2024

Acknowledgements

I want to express my great gratitude to Bc. Marek Haičman and the Security Compliance team, for guiding me during the whole process, helping me comprehend the fundamentals of this thesis, and always being there when I needed them. Equal appreciation is extended to both Mgr. Miroslav Vadkerti of the Testing Farm team and the entire Image Builder team, whose invaluable assistance greatly helped my understanding and integration of their products within this thesis. I would like to thank Kyle Walker from the Support Team for helping me with the support cases and their analysis on the support level. Finally, I wish to thank my supervisor Ing. Petr Sedlák, who provided me the information about the formal process of the thesis, and all of the academic processes and for being supportive throughout my whole academic years.

Contents

Introduction

Ensuring the quality of the products offered to customers has become one of the crucial elements of current IT workflows. Without *Quality Assurance*, the provider cannot guarantee their product meets all of its customers' expectations. With the introduction of the Agile project management approach, the necessity for testing has skyrocketed. It is an essential part of every project cycle as it helps to discover bugs and limitations before the product is delivered to the actual customers. Without quality assurance, the providers put themselves at risk of losing the trust of their customers, because if they notice even a minor defect in their recently purchased product, they can conclude that the product is of poor overall quality.

Concurrently, as technology continually advances, the demand for cybersecurity (digital asset protection) is experiencing exponential growth as well. In the age of digital transformation, the cybersecurity topic is on top of the list. The fear of being compromised has led the authorities to formulate a set of rules (*Compliance standards*) that, if implemented, will significantly decreases the risk of such unfortunate incidents. *Security Hardening* represents the implementation of such standards and has therefore become mandatory for all companies that are engaged in providing services to governments, financial institutions, defense contractors, healthcare, critical infrastructure, and many others. In other words, there are a lot of subjects that need to comply with the different sets of standards. These standards are then routinely audited, so the companies desire to stay updated. Given the constant and rapid advancements in technology, it becomes vital to regularly update or revise security standards, and at times, generate new ones completely.

For a comprehensive understanding of quality assurance, security hardening, or any other elements covered in this thesis, Chapter 1 contains all the essential theoretical background. It introduces the *Red Hat Enterprise Linux (RHEL)* operating system, which is one that is in the scope of security hardening. It also deeply covers Red Hat's *Image Builder* tool which is adopted for the creation of virtual images that are later utilized in the *Testing Farm* infrastructure for the general accessibility to the quality engineers. This chapter also depicts the *OpenSCAP project*, used for the integration of *Security Hardening* into operating systems by scanning and remediating the risks defined by the set of rules in the *Security Profiles*.

Chapter 2 then covers the initial and very important part of the thorough analysis of the current situation. It describes the preliminary steps that had to be carried out before diving into the proposal and implementation of the selected solution. It also touches on the fundamental utilization of the tools used for the solution, as it is necessary to fully understand the basics which serve as the building blocks for the entire project.

The implementation steps used for the solution are explained in Chapter 3. This chapter covers the overall pipeline designed and implemented as part of this thesis. It deep dives into each of the sub-tasks that collectively contribute to the establishment of the final product.

Chapter 1

Theoretical Background of the Work

1.1 Red Hat Enterprise Linux (RHEL)

Red Hat Enterprise Linux (RHEL) is a Unix-like operating system that is built on the Linux kernel. It is a distribution of Linux, which means it is a collection of software packages, utilities, libraries, and the Linux kernel bundled together to provide a complete operating system. Compared to commonly used Microsoft Windows in personal computers, Linux-based operating systems are used mostly in servers and clouds. The main interface for interacting with Linux distributions is via *terminal user interface* (TUI). [64]

RHEL is built on *open-source* software, which means that the source code of the operating system and many of its components is available to the public and can be modified, distributed, and used by anyone under open-source licenses. It is derived from the Fedora Linux operating system, which is more feature-friendly, but less secure and audited. It is known for its long-term support model. Red Hat offers a predictable and extended lifecycle for RHEL releases, typically with 10 years of support. This is essential for businesses and organizations that require a stable and reliable platform for their critical applications. [64]

RHEL is not a free operating system. It is provided on *a subscription basis*, and users or organizations need to purchase subscriptions to access official updates, security patches, and support services. Red Hat's support services include assistance with deployment, configuration, troubleshooting, and more. The main advantage of the RHEL is that it is designed with a focus on stability and security. Red Hat performs extensive testing and quality assurance to ensure that the operating system is reliable and secure. Security updates are released promptly to address vulnerabilities, making RHEL a trusted platform for critical workloads. [73]

This operating system is often the preferred choice for software vendors and hardware manufacturers, as it is well-certified and compatible with a wide range of hardware and software products. This compatibility ensures that RHEL can run on a variety of systems and is suitable for a diverse range of applications. The government is also a huge user and customer of Red Hat's solutions as they come with a lot of *certifications and standards* that are mandatory in the fields of national subjects. [9]

Figure 1.1 displays what the welcome prompt of the new Red Hat Enterprise Linux version 9 looks like.

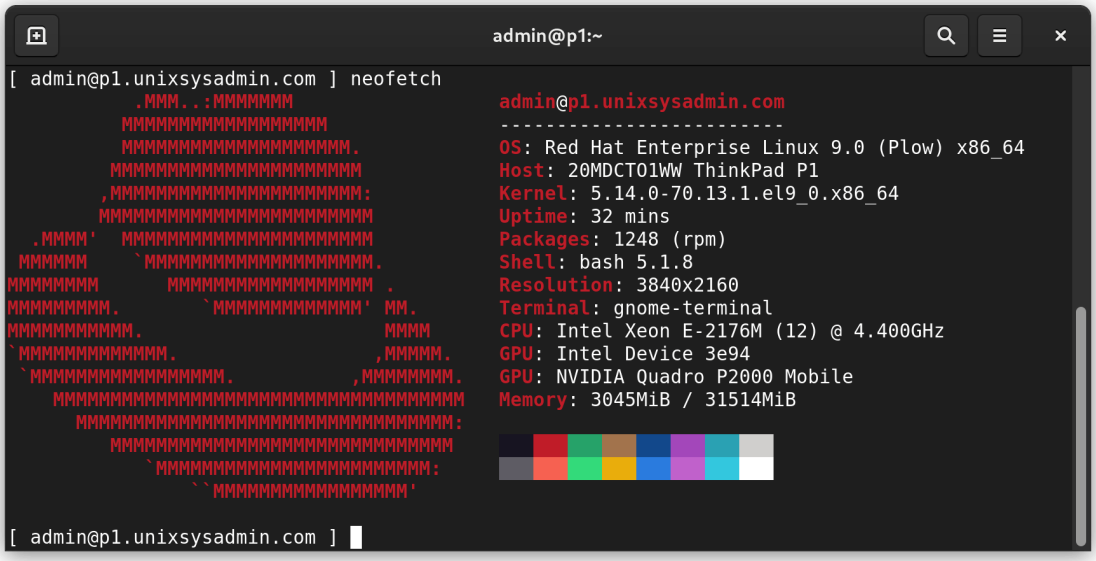

Figure 1.1: Red Hat Enterprise Linux 9 welcome prompt. Image source [85]

1.1.1 Red Hat Package Manager (RPM)

The RPM Package Manager (RPM) is a package management system that runs on Red Hat Enterprise Linux (RHEL), CentOS/CentOS Stream, and Fedora. Maintainers can use RPM to distribute, manage, and update software that they create for any of these operating systems. RPM manages software in the form of *packages* that can be installed, updated, or removed independently of each other, which simplifies the maintenance of an operating system. [60]

An RPM package uses the *.rpm* extension and consists of different files. It can contain *binary files*, also known as executables, *configuration files* which configures the bundled binaries to more specific scenarios, and *documentation files* or *licenses*. The name of the RPM package follows this format *<name>-<version>-<release>.<arch>.rpm*. For example, a bdsync RPM package could have a naming like *bdsync-0.11.1-1.x86_64.rpm* [6]

Source RPMs, or *SRPMs*, serve as a means to package software for RPM-based systems. They encapsulate the source code, patches, auxiliary files required during the build process, and a *specfile*, which provides instructions for generating the RPM package. Essentially, a source RPM captures the state of the source code and patches at the time of RPM build, allowing for consistent and reproducible package creation. The SRPM can be built into a main package and other multiple RPMs also known as *subpackages*. The built process is done in the RPM building software like Koji or Brew. More about these building systems in Section 1.1.2. [48]

1.1.2 Koji/Brew Build Systems

Koji is the software that builds RPM packages for the Fedora project. It utilizes the Mock[1] tool to create *chroot* environments to perform builds. To view logs from the building pro-

[1] https://github.com/rpm-software-management/mock/wiki

cess, Koji offers a web interface[2], which provides multiple filtering mechanisms, that help maintainers to find their own builds. It also allows maintainers to perform actions like *canceling a build* or *resubmitting a failed task*. Koji website utilizes *Kerberos* authentication within its framework to authenticate users. Koji also provides a Command Line Interface (CLI) option, if the maintainers want to automate their work with scripts. The CLI command is called *koji* and is included within the main koji RPM package. [21]

In Koji, there are occasions where it becomes necessary to distinguish between a *package* in general, a specific *build* of a package, and the various RPM files generated during the build process. When accuracy is needed, these definitions should be understood as follows [21]:

- **Package** - The name of a source RPM which refers to the package in general, not any particular build or subpackage. For example: *zlib*, *glibc*, etc.
- **Build** - A particular build of a *package* which refers to the entire build (all architectures and subpackages). For example: *zlib-1.2.11.el*, *glibc-2.3.4-2.19*, etc.
- **RPM** - A particular RPM. A specific architecture and subpackage of the build. For example: *zlib-1.2.11.el.x86_64*, *zlib-devel-1.2.11.el.s390x*, *glibc-2.3.4-2.19.i686*, *glibc-common-2.3.4-2.19.aarch64*. RPM is then more described in Section 1.1.1.

In Koji, packages are organized using tags, which serve as collections analogous to beehives. However, they diverge in several aspects. *Tags* are managed within the database but they are not stored on disk. Each tag has its own list of valid packages that can be inherited. Moreover, package ownership can be set per-tag, further enhancing flexibility. Tag inheritance is highly configurable, offering a more tailored approach. A build target specifies both the location where the package should be built and how it should be tagged. This strategy ensures consistency in target names despite potential alterations in tags across releases. [21]

Brew is a Red Hat's internal instance of the Koji building system, this system is used for building RPM packages or module builds within RHEL distribution.

1.1.3 Dandified Yum (DNF)

Dandified Yum, commonly referred to as DNF, serves as a software package manager designed for RPM-based Linux distributions, like RHEL or Fedora, used for package installation, updates, and package removals. It was initially introduced in Fedora 18 as a testable feature, and since Fedora 22 it has become the default package manager for Fedora. Being the successor to the traditional *yum package manager*, DNF comes with advanced functionalities beyond its predecessor. Some of the features that distinguish DNF from yum are: [18]

- Dependency calculation based on modern dependency-solving technology
- Optimized efficiency in memory-intensive operations.
- Compatibility with both Python 2 and Python 3.
- Comprehensive documentation accessible for Python APIs.

[2] https://koji.fedoraproject.org/koji/

DNF uses hawkey[3] libraries to resolve RPM dependencies when executing queries on client machines. These libraries are built on top of the *libsolv*, a package dependency solver that uses a satisfiability algorithm. Further insights into the algorithm can be explored in the libsolv GitHub repository[4]. [18]

1.2 Quality Assurance

Quality Assurance (QA) is a critical component of any product or service development process, ensuring that the highest standards of quality are consistently met. It involves a systematic and proactive approach to prevent defects and errors by establishing rigorous processes, standards, and methodologies. QA encompasses various activities, including product testing, process auditing, and adherence to industry best practices. By implementing QA, organizations can enhance customer satisfaction, improve product reliability, and reduce the risk of costly recalls or defects, ultimately strengthening their reputation and competitiveness in the market. Continuous improvement and a commitment to QA principles are essential for delivering products and services that meet or exceed customer expectations. [87]

> "*In the broadest sense, quality assurance (QA) is the process of ensuring that a product functions as intended. Does it work properly? Deliver on its promises? Exceed customer expectations? QA is the testing and retesting of the product during development (or updating) to find any flaws and correct them before customers start using it.*" [38]

1.2.1 Quality Assurance Testing

> "*Quality assurance testing is a process that ensures an organization delivers the best products or services possible. QA aims to deliver consistent results through a set of standardized procedures, which means organizations also need to make sure their processes for achieving the desired results hit specific quality benchmarks.*" [1]

QA testing includes process-centric activities focused on the ultimate products. Certain organizations regularly conduct QA testing on existing products to pinpoint and implement fixes based on the reports. While QA testing is commonly linked with software development, various industries employ these methodologies to enhance their results. The QA testing process typically consists of four stages (also known as the PDCA model): [35]

- **Plan:** Develop comprehensive tests that address potential issues and adhere to standards.
- **Do:** Execute tests under various conditions.
- **Check:** Evaluate the product and identify opportunities for improvement.
- **Act:** Address any vulnerabilities revealed during testing.

[3] https://fedoraproject.org/wiki/Features/Hawkey
[4] https://github.com/openSUSE/libsolv

1.2.2 Functional Testing

Functional testing is part of the QA which is described in Section 1.2. In this part, the *QA engineer* determines whether the tested software is performing according to the predetermined requirements. There are multiple types of Functional testing like Unit testing, System testing, Regression testing, Integration testing, and more. [42].

Unit Testing is a software testing technique, that involves the examination of individual units within the software, such as sets of program modules, operational procedures, and usage protocols, to determine whether they are suitable for deployment or not. It is a testing approach in which each independent module is examined by the developer to identify any potential issues. Unit Testing is defined as a crucial phase in the *Software Development Life Cycle (SDLC)*, typically occurring before integration testing. It targets specific components, which can be either individual functions or procedures. Unit testing is a type of testing technique that is usually performed by developers. However, in some cases, Quality Assurance Engineers may also perform unit testing. [22]

Regression testing is a software testing technique that involves re-running tests to ensure that a software application operates as expected following any modifications, updates, revisions, improvements, or optimizations in its code. It is a critical component of the software development life cycle, allowing developers to identify unexpected issues that may arise due to tweaks, enhancements, or extending of the existing codebase. Regression testing provides overall stability to the software application by maintaining a validation on the functionality of the existing features. It is an inevitable step after each code modification, ensuring that the system remains robust and resilient to frequent enhancements. Changes in code may introduce new dependencies, potentially leading to defects or malfunctions within the software. Regression testing serves to mitigate such risks by verifying that previously developed and tested code continues to function correctly when integrated with new features or code changes. [37]

Integration Testing is a software testing method where components of the software are systematically integrated and then tested as a unified group. Usually, these components function correctly when tested individually, but problems may arise when they interact within the integrated system. With integration testing, testers want to find defects that surface due to code conflicts between software modules when they are integrated with each other. Conflicts between software modules happen for many reasons, such as incompatibility between subsystem versions, data format conflict, or different processing logic. Integration testing pinpoints those communication issues between software components. These tests are usually conducted after unit testing and before system testing in the software development process. [39]

System Testing is a level of testing focused on verifying the entire integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Typically, the software is only one element of a broader computer-based system. Ultimately, the software is interfaced with other software or hardware systems. System Testing is defined as a series of different tests whose sole purpose is to exercise the complete computer-based system. [24]

All of the types of functional testing are displayed in Figure 1.2.

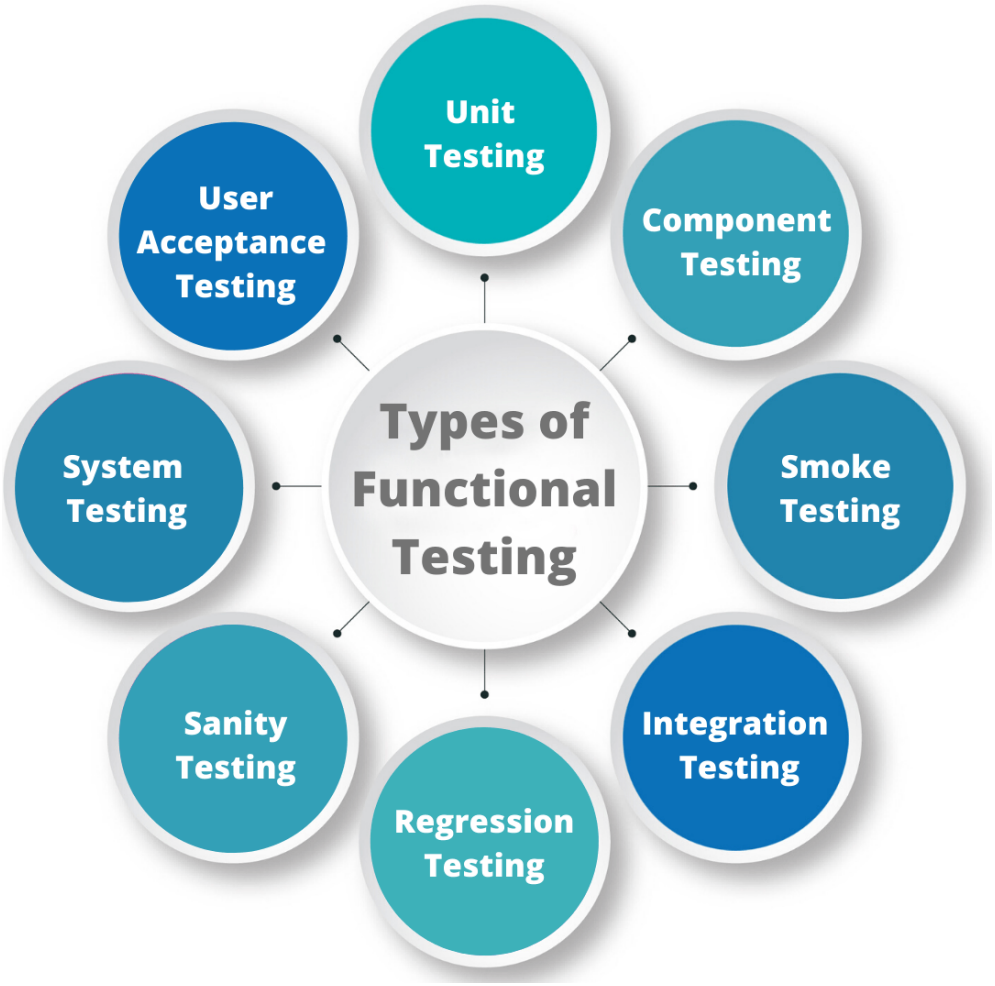

Figure 1.2: Types of functional testing. Image source [42]

This type of testing uses black-box type of testing techniques[5]. The tester uses some or all of the existing types of functional testing to validate the functionality of the testing product. Unlike non-functional testing, functional testing is not testing the quality, security, or performance of the product/application's source code. Functional testing rather focuses on the results of processing and determines whether the product/application satisfies the basic functionality which is described in the documentation. Thus it needs to satisfy the user expectations. [47]

1.2.3 CI/CD

CI/CD is a key component of DevOps[6], which unites development and operations teams, encompassing continuous integration and continuous deployment practices. This approach automates the manual steps traditionally required to move new code from a commit to production, covering build, test (including integration, unit, and regression tests), deploy

[5]In black-box testing, the tester has no knowledge of the internals of the system.

[6]Combination of Development (Dev) and Operations (Ops) to increase the efficiency, speed, and security of software development and delivery

phases, and infrastructure provisioning. The CI/CD pipeline allows development teams to automatically test and deploy code changes, reducing downtime and expediting code releases. [23]

Continuous Integration (CI) involves the automated and regular incorporation of code modifications into a collaborative source code repository. Continuous Delivery and/or Deployment (CD) constitutes a two-step procedure confining the integration, testing, and delivery of code alterations. Continuous Delivery concludes before automatic production deployment, whereas continuous Deployment automatically deploys updates to the production environment. [72]

The internal process of the CI/CD pipeline is animated in Figure 1.3.

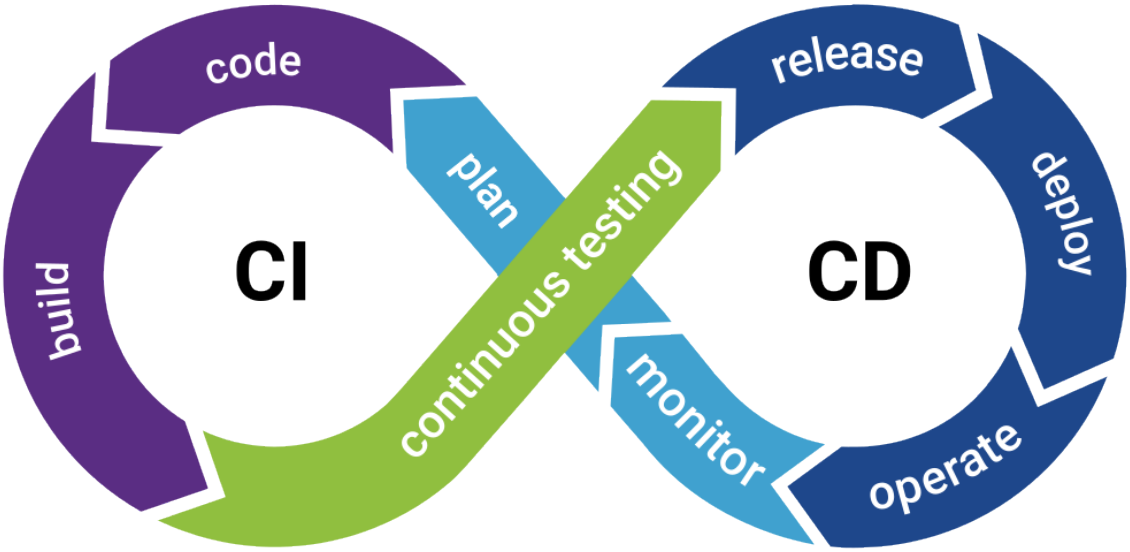

Figure 1.3: Continuous Integration (CI) and Continuous Deployment (CD). Image source [42]

1.2.4 Testing Farm Project

Testing Farm is a testing *System as a Service (SaaS)*[7] for Red Hat internal services, Red Hat Hybrid Cloud services, and open-source projects related to Red Hat products. It is widely used as a test execution back-end of other services or *continuous integration (CI)* systems (CI is further described in Section 1.2.3). Using the well-documented application HTTP *application programmable interface (API)*, Testing Farm can be easily integrated into any service. [83]

The tests that are executed in Testing Farm are being executed via *Test Management Tool (tmt)*[8] that allows users to manage, and execute tests on different environments. Testing Farm is an open-source project which can be used in various ways such as [40]:

- Run tests using the CLI from the local machine to create a testing machine.

[7] SaaS is a cloud computing model that delivers software applications over the internet, allowing users to access and use the software without the need for installation or maintenance.

[8] https://tmt.readthedocs.io/

- Integrate it into the project as a CI check via GitHub Actions[9] or Packit[10].
- Send HTTP requests to the service API with desired requirements.

Testing Farm provides an easy testing environment for *Quality Engineers* (QE). It provides multiple sets of operating systems and their versions, by utilizing generated virtual machine images (covered in Section 1.5.1), that run on their internal servers and users can access them via a *Secure Shell* (SSH) connection.

The use of the Testing Farm within the Red Hat product framework is significant. Its utilization extends to smoothing the Gating process, a pivotal stage in the Continuous Integration and Continuous Delivery (CI/CD) pipeline specifically tailored for RHEL packages. The integration of Testing Farm automation into this workflow, not only boosts efficiency but also acts as a safeguard against potential bugs, thereby promoting a more seamless and resilient software development lifecycle.

Integration into the Package Level Testing was also one of the key areas, where the Testing Farm was needed. However, since Fedora has started using Testing Farm to promote its testing triggered by Merge Requests, its testing environment has been working seamlessly. This also helped to propagate the Testing Farm within the community and the users started to make use of it even more.

Command Line Interface (CLI) tool

Testing Farm provides a simple usage command line interface to interact with Testing Farm infrastructure. This CLI tool can be easily installed following the upstream installation guide[11]. [81]

Once this CLI tool is installed in the system, a user needs to follow the onboarding section [82] from the upstream documentation. This onboarding consists of creating a unique API token that will be used for the authentication within the Testing Farm API endpoints. After getting this API token, it needs to be exported to the working terminal session in which the CLI tool will be used. This can be easily done using an export command like this:

```
export TESTING_FARM_API_TOKEN=<API-token>
```

To provision a Testing Farm image the *testing-farm reserve* `--compose` command needs to be used. This command will reserve the virtual machine with a specified image for the users to test their workloads. The default provision time is 60 minutes, but this can be easily prolonged (the maximum is 12 hours) using the `--duration` command option.

Web-based test result manager

Testing Farm also provides a web-based test result manager. In the website generated for a specific test case, all of the artifacts related to this test case are displayed. They can be further examined by navigating to their details or logs. An example of such a web-based test result is shown in Figure 1.4.

[9] https://github.com/sclorg/testing-farm-as-github-action
[10] https://packit.dev/docs/configuration/upstream/tests
[11] https://gitlab.com/testing-farm/cli/-/blob/main/README.adoc

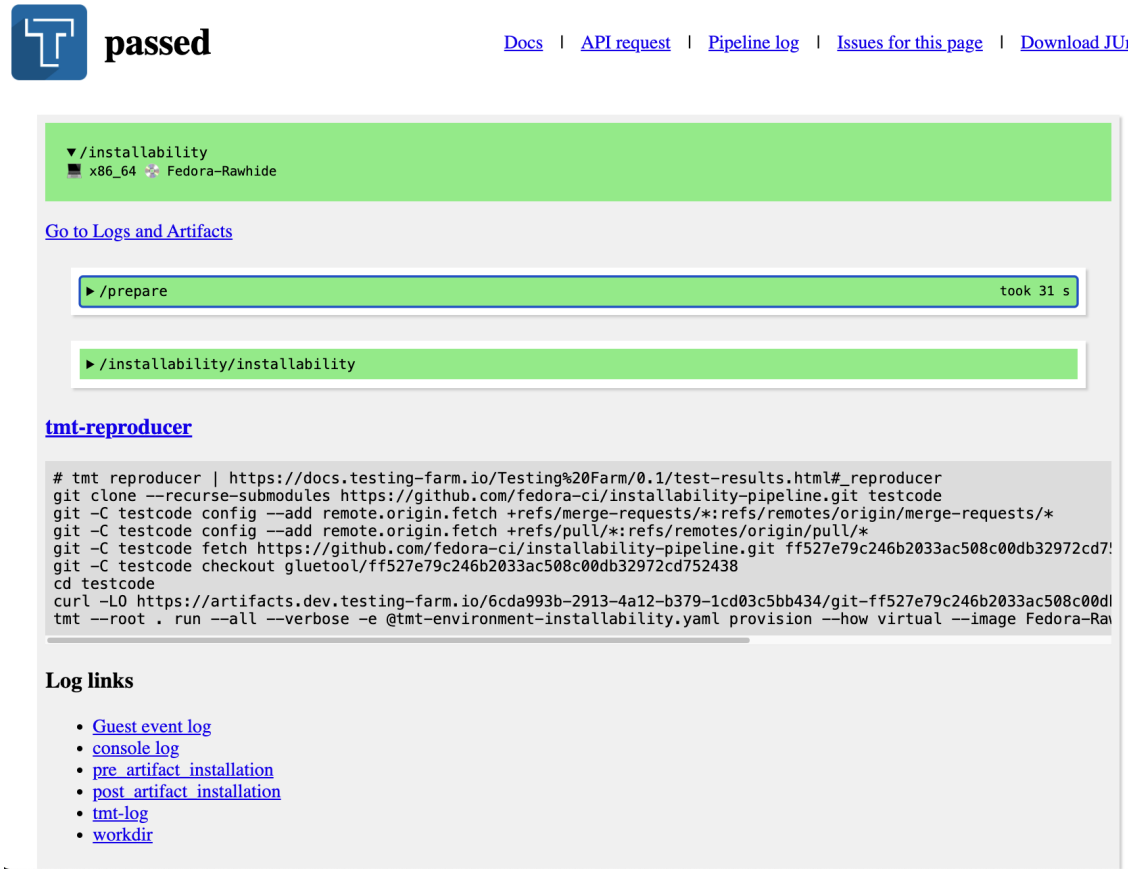

Figure 1.4: Testing Farm web-based test result site

1.3 Security Compliance

The concept of security and compliance used in the same sentence has become a common theme in recent years. The word 'security' specifically in the information technology area brings up several topics, especially the relevant risks that are associated with these topics like *Access security*, *Data security*, *Application security*, *Network security*, *Cyber security*, and so on. However, when users combine security and compliance, the risk of the above-mentioned topics can be quickly remediated. Organizations must implement and maintain some sort of security compliance management system or framework, aligning people, processes, and technology, to survive in today's competitive market and comply with external, and in some cases, regulatory requirements. [75] [89]

1.3.1 RHEL Security Compliance Team at Red Hat

Red Hat's RHEL Security Compliance team is the team responsible for the hardening tools and mechanisms in RHEL. This team develops the OpenSCAP project downstream[12], but they are also highly involved in the upstream[13] development. Overall thesis goals and implementation are conducted with this team's involvement as they are stakeholders for

[12] In Open-Source terminology refers to versions of the software that are derived from the original source. They can also contain some additional system-specific features (not yet accepted by upstream)

[13] Refers to the original source of the software where the core development takes place

this project. The RHEL Security Compliance team also has its Quality Assurance subteam, which will take over the project's maintenance after the final release. [65]

1.3.2 Cybersecurity Standards

A security compliance standard is a specific set of security requirements and controls that must be followed in order to achieve compliance with a particular regulatory requirement. How the standards are to be implemented and what solutions are used to achieve the standard's specifics normally are not part of the standard itself. Companies rather use the Cybersecurity Frameworks (covered in Section 1.3.3, which can be followed to comply with the chosen standard. [13]

These standards are the building blog of the *Information Security Management System* (ISMS). ISMS is a systematic approach to managing sensitive company information to ensure its confidentiality, integrity, and availability. It utilizes these standards with the cybersecurity frameworks to provide a set of policies, processes, and controls designed to identify, manage, and mitigate information security risks. [76]

Here are some examples of the cybersecurity standards:

- **ISO/IEC 27001**: is the world's best-known standard for information security management systems (ISMS). It defines the requirements an ISMS must meet. This standard serves as a tool for risk management, cyber-resilience, and operational excellence. [36]

- **HIPAA** The *Health Insurance Portability and Accountability Act* of 1996 (HIPAA) is a federal law mandating the establishment of national standards to safeguard confidential patient health information, preventing its unauthorized disclosure without the patient's consent or awareness. The US Department of Health and Human Services (HHS) introduced the HIPAA Privacy Rule to enforce the implementation of HIPAA. Additionally, the HIPAA Security Rule is in place to safeguard a specific set of information covered by the Privacy Rule. [50]

- **PCI DSS**: *Payment Card Industry Data Security Standard* (PCI DSS) compliance is obligatory for entities handling cardholder data, whether startups or global enterprises, requiring annual validation as mandated by credit card companies and network agreements. Developed by the PCI Security Standards Council (SSC), these standards aim to fortify the entire payment card ecosystem, applying to merchants and service providers engaged in credit/debit card payment transactions. PCI compliance, mandated by credit card companies, ensures the security of credit card transactions by establishing technical and operational standards for businesses to protect cardholder data. The PCI Security Standards Council oversees the development and management of these compliance standards. [51]

- **FIPS 140-3**: The *Federal Information Processing Standard* (FIPS 140-3) is a cryptographic standard that defines security requirements for cryptographic modules. It presents four progressively advanced levels tailored to address various applications and environments. Encompassing various aspects of the secure design and implementation of a cryptographic module, these include specification; ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electro-

magnetic compatibility (EMI/EMC); self-tests; design assurance; and measures to mitigate other types of attacks. [43]

1.3.3 Cybersecurity Frameworks

> "*A security framework is a set of policies, guidelines, and best practices designed to manage an organization's information security risks.*" [17]

There are a lot of commonly used and very well-known security frameworks, that are widely used in the current era of digital transformation. The creation of frameworks was crucial for organizations that were unable to afford expensive cybersecurity experts to help them secure their digitized assets.

Here is the list of some common cybersecurity frameworks:

- **NIST Cybersecurity Framework (NIST CSF)**: The NIST cybersecurity framework, developed by the *National Institute of Standards and Technology* (NIST), serves as a robust tool for structuring and enhancing cybersecurity programs. It combines guidelines and best practices, which assist organizations in establishing and enhancing their cybersecurity stance. The framework proposes recommendations and standards to empower organizations in recognizing and identifying cyber threats, offering guidance on response strategies, prevention measures, and recovery protocols. [7]
- **CIS Controls Framework**: The CIS Controls Framework, crafted and maintained by the *Center for Internet Security, Inc.* (CIS), is a robust model for advocating cybersecurity best practices. Developed with input from global experts, the framework aids organizations in enhancing threat identification, assessment, and adaptability. Originally released in 2008 by the SANS Institute, it is now managed by the Center for Internet Security and widely adopted under various names like *CIS Critical Security Controls (CSC)* and *SANS Top 20*. The framework defines five crucial areas, addressing learning from attacks, prioritizing security controls, employing standard metrics, emphasizing continuous measurement and mitigation, and leveraging automation for rapid response. With 20 critical security controls, the CIS CSC provides comprehensive best practices for cyber defense teams to identify, confront, and defeat cyber threats. [11]

1.3.4 Security Hardening

The process of security hardening involves organizations examining ways to minimize vulnerabilities in their IT networks and digital environments, enhancing the likelihood of preventing cyberattacks and data breaches. This ongoing effort involves constant monitoring and assessment by a business's IT department and security team, considering network activity, user behaviors, and evolving cybersecurity trends to safeguard digital assets. As cyber threats advance in sophistication, security professionals must continuously adopt new methodologies and technologies. Achieving comprehensive security hardening, safeguarding everything from critical infrastructure to email clients, requires the utilization of a variety of tools, techniques, and processes. Key areas for security hardening are *applications, operating systems, servers, databases, networks,* and *endpoints.* [77]

In the cybersecurity world, the hardening is becoming more and more complicated, and that is why the creation of the security Standards (described in Section 1.3.2) and their congruent security Frameworks (described in Section 1.3.3) was inevitable. Organizations can

simply follow these guidelines which will help them secure (harden) their assets. Figure 1.5 displays what Security Hardening process consists of.

Figure 1.5: Security Hardening process. Image source [79]

1.3.5 Vulnerability Assessment

A vulnerability assessment is a testing procedure used to discover and categorize security defects within a specified time frame. This method utilizes a combination of automated and manual techniques, each with varying degrees of rigor, emphasizing comprehensive coverage. Using a risk-based approach, vulnerability assessments may focus on various technology layers, with host, network, and application layers being the most common targets. The goal is to identify and evaluate the severity of as many security vulnerabilities as possible. [80]

There are several types of vulnerability assessments, including: [34]

- **Host assessment**: Examining critical servers for vulnerabilities, which may be prone to attacks if not adequately tested or if not generated from a tested machine image.
- **Network and wireless assessment**: Evaluating policies and practices aimed at preventing unauthorized access to both private and public networks, along with assessing resources accessible through networks.

- **Database assessment**: Analyzing databases or big data systems to uncover vulnerabilities and misconfigurations, identifying rogue databases or insecure dev/test environments, and categorizing sensitive data across an organization's infrastructure.
- **Application scans**: Detecting security vulnerabilities in web applications and their source code through automated scans on the front-end or static/dynamic analysis of source code.

1.3.6 ComplianceAsCode Project

The *ComplianceAsCode* project[14] was invented to automate the process of Security Hardening which is further detailed in Section 1.3.4.

> "*The purpose of this project is to create security policy content for various platforms - Red Hat Enterprise Linux, Fedora, Ubuntu, Debian, SUSE Linux Enterprise Server (SLES),... - as well as products - Firefox, Chromium, ...*" [15]

The ComplianceAsCode project, previously known as the *SCAP Security Guide*, offers security guidance, baselines, and validation mechanisms utilizing the *Security Content Automation Protocol (SCAP)*. ComplianceAsCode not only provides hardening advice but also establishes connections to compliance requirements, facilitating deployment activities like certification and accreditation. For instance, widely accepted policies like *NIST 800-53* may specify that System Administrators audit "privileged user actions" without defining what constitutes 'privileged actions." ComplianceAsCode serves as a bridge, translating generalized policy requirements into specific implementation guidance in SCAP formats, and promoting automation wherever feasible. The ComplianceAsCode project provides a wide variety of hardening guides and configuration baselines developed by the open-source community, ensuring that the user can choose a security policy that best suits the needs of their organization, regardless of its size.[16]

1.3.7 OpenSCAP Project

Security Content Automation Protocol (SCAP) is a U.S. standard maintained by *National Institute of Standards and Technology* (NIST). The OpenSCAP project is a collection of open-source tools that are focused on implementing this standard. The OpenSCAP ecosystem provides multiple tools aimed at assisting administrators and auditors with the assessment, measurement, and enforcement of security baselines. These tools provide great flexibility and interoperability, reducing the costs of performing security audits. [62]

Individual sets of rules and mitigation strategies (that are part of the profiles) are written in the XCCDF format which is later described in this Section.

OpenSCAP project covers both *Security Compliance* (described in Section 1.3) and the *Vulnerability Assessment* (more in detail in Section 1.3.5) to provide thorough feedback/report on the user's scanned assets.

To make use of the OpenSCAP features, users can install *oscap* command line interface (CLI) tool, which can be used both as a *configuration or vulnerability scanner* and as a *remediation (mitigation) tool*. [53]

The OpenSCAP project offers a comprehensive suite of open-source tools for managing system security and compliance. At its core lies *OpenSCAP Base*, a powerful command-line utility that performs configuration and vulnerability scans, evaluates security content

[14] https://github.com/ComplianceAsCode

and generates reports. Building upon this foundation, the *SCAP Security Guide* provides a wealth of hardening guides and baselines for various systems, allowing users to choose the best fit for their needs. For a user-friendly experience, *SCAP Workbench* offers a graphical interface for tailoring and conducting scans. [63]

oscap Command line tool

OpenSCAP project also provides a Command Line Interface tool called *oscap*. This binary tool can be installed in RPM-based (RPM is described in Section 1.1.1) Linux distributions (like RHEL, Fedora, or CentOS) by installing the *openscap-scanner* RPM package. In order to generate blueprints or evaluate and mitigate the security risks defined in specific OpenSCAP profiles, which are provided by *scap-security-guide* RPM package. These packages can be easily installed using the dnf command below.

```
$ sudo dnf install scap-security-guide openscap-scanner
```

XCCDF standard

The *Extensible Configuration Checklist Description Format* (XCCDF) is a specification language for creating security checklists, benchmarks, and related documents. An XCCDF document represents a structured collection of security configuration rules for some set of target systems. The specification is crafted to facilitate information interchange, document generation, organizational and situational tailoring, automated compliance testing, and compliance scoring. The specification also defines a data model and format for storing the results of benchmark compliance testing. The primary goal of XCCDF is to establish a standardized framework for articulating security checklists, benchmarks, and configuration guidance, promoting the widespread adoption of sound security practices. XCCDF documents are expressed in XML and can be validated using an XML Schema-validating parser. [45]

OVAL

The OVAL acronym stands for *Open Vulnerability and Assessment Language*. OVAL is a declarative language designed for making logical assertions about the state of the system. It is a key component of the SCAP standard and is utilized to describe security vulnerabilities or the preferred configuration of systems. OVAL definitions outline a secure state for various objects within a computer, such as configuration files, file permissions, and processes. OVAL definitions are evaluated using an interpreter called a scanner. The language is structured into three components, aligning with the three phases of the assessment process [41]:

- **OVAL Definitions** – for expressing a specific machine state
- **OVAL System Characteristics** – for representing system information
- **OVAL Results** – schema for reporting the results of an assessment.

CPE

The *Common Platform Enumeration* (CPE) is a structured naming scheme used for categorizing information technology systems, software, and packages. It adheres to the generic

syntax for *Uniform Resource Identifiers* (URI) and includes a formal name format, a method for checking names against a system, and a description format for binding text and tests to a name. In documents such as OVAL and XCCDF, CPE finds application as a means to identify the target platform for checks or definitions. CPE names are stored in Official CPE Dictionary[15]. [44]

ARF

The *Asset Reporting Format* (ARF) is a component of the SCAP standard and serves as a specialized language for expressing information about assets and their relationships with reports. Often referred to as "Result DataStream" ARF acts as a complement to "Source DataStream" simplifying the efficient exchange and interpretation of security-related data. [66]

Security policies

OpenSCAP project includes multiple security hardened policies (also called profiles) that differ in the sets of rules and the security standards (described in Section 1.3.2) they comply with.

Here are some examples of the security policies that are offered by OpenSCAP:

- **Security Technical Implementation Guides (STIG)**: is a comprehensive set of guidelines and configuration standards developed by the *Defense Information Systems Agency* (DISA) to enhance the security of computer systems and networks. STIGs provide detailed recommendations for securing various technology products and systems, including operating systems, applications, and devices. These guidelines are designed to minimize vulnerabilities and ensure compliance with security policies and regulations. STIGs cover a wide range of security controls, addressing aspects such as access controls, audit policies, encryption, and network configurations. Organizations use STIGs as a framework to establish and maintain a robust cybersecurity posture, helping to safeguard sensitive information and mitigate potential risks to their IT infrastructure. [19]

- **Center for Internet Security (CIS)**: provides a range of cybersecurity-related services, with a specific focus on generating benchmarks. These benchmarks, available as complimentary resources for CIS members in PDF format, are not directly compatible with scanning tools but are designed to be human-readable. Organizations use CIS benchmarks as a cybersecurity framework (more in detail in Section 1.3.3) to securely configure their IT services and products. [84]

  CIS benchmarks provide two levels of security settings: [20]

  - **Level-1**: These profiles are generally assigned to surface-level recommendations that can be implemented quickly. Organizations will generally be able to continue normal operations without any interruptions when introducing recommendations of this level.
  - **Level-2**: These profiles are linked to recommendations that deal with areas of significant importance to IT systems and cybersecurity. The recommendations will cover policies and components of IT systems that are vital to cybersecurity

[15] https://nvd.nist.gov/cpe.cfm

within IT infrastructure. Level-2 profiles deal with areas with heightened security considerations or where there is a risk of negative impact on IT systems.

- **ANSSI-BP-028**: *Agence nationale de la sécurité des systèmes d'information* (ANSSI) has published various guides, including ANSSI-BP-028, which provides configuration recommendations for fortifying Linux systems. This document outlines four levels of hardening, each corresponding to the security requirements dictated by the system's applications and workloads. These are the categories ANSSI-BP-028 include: [74]
  - **Minimal**, intended for implementation on every system available.
  - **Intermediary**, generally applies for services protected by multiple layers of higher-level security.
  - **Enhanced**, generally applicable to systems exposed to non-authenticated flows.
  - **High**, designed for systems hosting sensitive data accessible from non-authenticated or poorly controlled networks.

Data Stream

The SCAP data stream, introduced in SCAP version 1.2, is as a file format that represents a bundle of XCCDF, OVAL, and other component files which can be used to define a compliance policy expressed by an XCCDF checklist. This file format includes an index and catalog, enabling the segmentation of the data stream into files based on SCAP components. Utilizing the XML format, the data stream comprises a header with a table of contents and a list of **<ds:component>** elements. Each of these elements represents a SCAP component like XCCDF, OVAL, CPE, and others. The data stream file may contain multiple components of the same type, ensuring comprehensive coverage of security policies required by an organization. Figure 1.6 shows the structure of compliance scanning resources used in the data stream. [56]

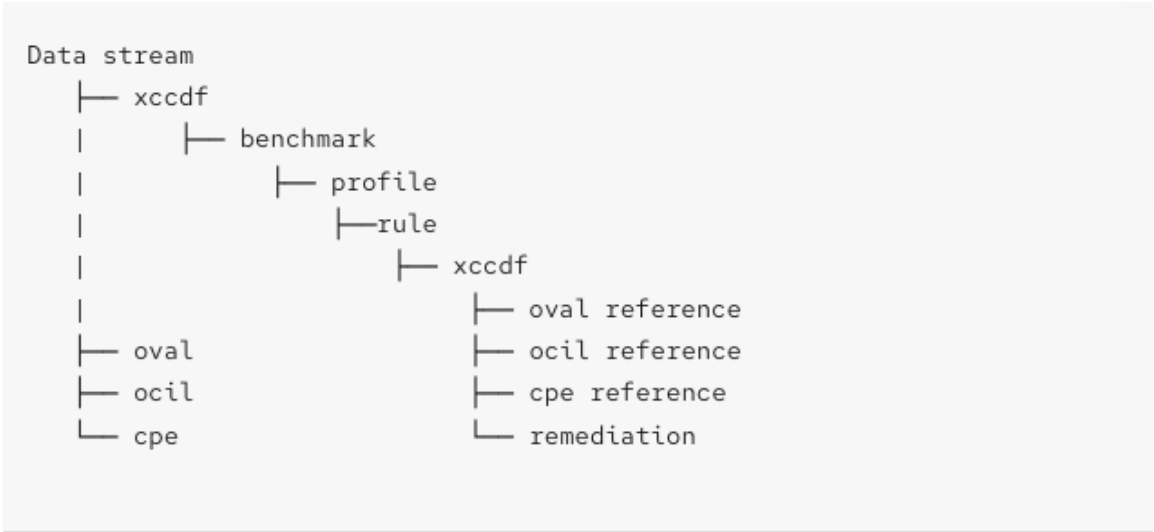

Figure 1.6: Structure of Compliance Scanning Resources. Image source [54]

Profile tailoring

Security policies (profiles) contained in the SCAP Security Guide usually strictly implement requirements of a specified standard (e.g. PCI-DSS or USGCB). However, sometimes the customer may need to adjust the security policy to their specific company needs. Such customization is also sometimes called "*tailoring*". In order to get use of this tailoring, users can make use of the **autotailor**[16] command line interface (CLI) tool. This tool is part of the *openscap-utils* package. The script creates a new file with a new profile with ID in a form **BASE_PROFILE_ID_customized** (this file name is used by default, but it can be changed using the *--new_profile_id* option). Tailoring files can also be easily created using SCAP Workbench[17] which is a GUI variant of the *autotailor* tool. To make use of the tailored profile, it can be added as the value of *--tailoring-file* option in *oscap* CLI tool. [68]

Example usage of the autotailor CLI tool

```
$ autotailor \

--select RULE_ID --unselect RULE_ID --var-value VAR=VALUE \

--output TAILORING_FILE --new_profile_id NEW_PROFILE_ID

DS_FILENAME BASE_PROFILE_ID
```

Generating profile specific blueprints feature

OpenSCAP has the capability to generate remediation in the form of an Image Builder (OSBuild) Blueprint (which will be covered in detail in Section 1.6.1). This remediation serves as a starting point for image creation, typically including only essential configuration elements that would be challenging or impossible to modify after the image is created, such as partitioning or a predefined set of installed packages. [57]

Example generating a blueprint for CIS profile

```
$ oscap xccdf generate fix --profile cis --fix-type blueprint \

/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml > blueprint.toml
```

Evaluation of the tested machine

OpenSCAP scanner tool provides an evaluation feature, which is triggered by the "*oscap xccdf eval*" command. The evaluation can be performed on any supported Security Standard from the OpenSCAP family. In order to evaluate any Security Standard, the *scap-security-guide* RPM package has to be installed. [59]

[16] https://www.mankier.com/8/autotailor
[17] https://www.open-scap.org/tools/scap-workbench/

Generating HTML reports and XML results

Another feature the OpenSCAP scanner tool provides is generating HTML user-friendly human-readable reports, or more advanced XML results, which can be used for some computer processing. First, the user needs to generate the XML results file by using the "*oscap xccdf eval --results results.xml*" command. Once the results file is generated, the HTML report can be generated from the XML results. It can be simply done by "*oscap xccdf generate report --output report.html results.xml*" command. OpenSCAP also provides the possibility to generate the ARF formatted results. To generate such a report, the user needs to first execute the "*oscap xccdf eval --results-arf arf.xml*" command, to generate the XML file with the ARF formatting. Then they need to install the *openscap-report* RPM package, which provides the tools necessary for generating the ARF report from the XML file generated previously. After these steps are completed, the user can generate the human-readable HTML report using the "*oscap xccdf generate report arf.xml > report.html*" command. [59]

1.4 Ansible

Ansible is an open-source IT automation software application, written in *Python* programming language and accessible through the command line interface. Its capabilities extend to system configuration, software deployment, and the orchestration of sophisticated workflows, making it a flexible tool for tasks such as automatic application deployment and system updates. Ansible's main strengths are simplicity and user-friendly interface. It is built with a strong focus on security and reliability, featuring minimal moving parts. Utilizing OpenSSH for transport (with additional transport and pull modes as viable alternatives), Ansible uses a human-readable language that is designed for quick initiation without requiring extensive training. [58]

Red Hat's Ansible software is widely used in automation. Everything from machine deployments to system administrations to network configurations can be automated using Ansible. Ansible uses so-called *playbooks* to configure the automation purpose. [52]

1.4.1 Ansible Playbook

Ansible Playbooks offers a simple, repeatable, and reusable system for configuring and deploying applications across multiple machines. This configuration management tool is particularly well-suited for handling complex applications. If a task needs to be executed with Ansible on multiple occasions, it is best to create a playbook and store it in a source control system. This playbook can then be used to deploy new configurations or verify existing configurations on remote systems. The ansible-examples[18] repository contains numerous playbooks that showcase various useful techniques. Playbooks are written in *YAML* format with a minimalistic syntax. [5]

1.5 Virtual Machines

A Virtual Machine (VM) is a compute resource that uses software instead of a physical computer for program execution and application deployment. On a *physical "host" machine*, one or more *virtual "guest" machines* can operate independently. Each virtual machine runs its own operating system and functions autonomously from the other VMs, even if

[18] https://github.com/ansible/ansible-examples

they all share the same host. For instance, this feature enables the operation of a Linux virtual machine on a physical Windows device. Virtual machines allow businesses to run an operating system that behaves like a completely separate computer in an app window on a desktop. VMs may be deployed to accommodate diverse levels of processing power requirements, to run software compatible with different operating systems, or to safely test applications within a sandboxed environment. Virtual machines have historically been used for server virtualization, which enables IT teams to consolidate their computing resources and enhance efficiency. Moreover, virtual machines can perform specific tasks perceived as too risky to carry out in a host environment, such as accessing virus-infected data or testing operating systems. The isolation of the virtual machine from the rest of the system ensures that the software within the VM cannot compromise the integrity of the host computer. [86]

A VM cannot interact directly with a physical computer. Instead, it relies on a lightweight software layer called a *hypervisor* to coordinate between the virtual machine and the underlying physical hardware. The hypervisor allocates physical computing resources such as processors, memory, and storage to each individual VM. By maintaining a clear separation between each VM, the hypervisor ensures that they operate independently and do not interfere with one another. There are two primary types of hypervisors. [26]

Type 1 hypervisors also known as a *bare-metal hypervisor*, run directly on the physical hardware (usually a server), effectively replacing the need for an operating system. Usually, a separate software product is used to generate and control the VMs on the hypervisor. Various management tools, such as VMware's vSphere, offer functionalities that enable the selection of a guest operating system for installation within the VM. [78]

Type 2 hypervisors run as an application within an existing operating system rather than directly on the hardware. They are primarily used by individual PC users requiring multiple operating systems, such as engineers or security professionals analyzing malware. They often offer toolkits for users to install into the guest OS, enhancing connections between the guest and host OS and allowing actions like cutting and pasting the clipboard between the two. While convenient for end-user productivity, Type 2 hypervisors introduce latency issues due to their reliance on the host OS for resource access, which can affect performance. Additionally, there are security concerns associated with Type 2 hypervisors, particularly if the host OS becomes compromised, as there is the potential for malicious manipulation of any guest OS operating within the hypervisor. [25]

The graphical difference between these two types of hypervisors is displayed in Figure 1.7.

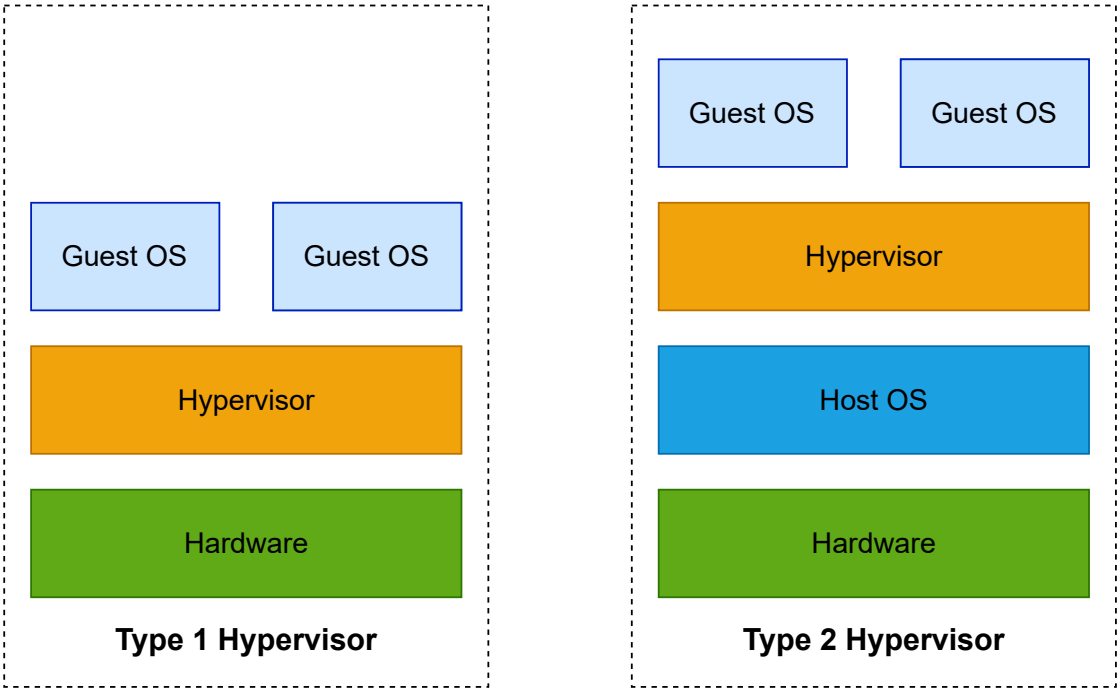

Figure 1.7: Difference between Type 1 and Type 2 Hypervisor

1.5.1 Virtual Machine Images

Virtual machine images refer to pre-configured and encapsulated instances of an operating system, application, or system configuration. These images are created to be deployed on virtualization platforms, allowing organizations to efficiently replicate and distribute software across multiple environments. Virtual machine images serve as templates for virtual machines (VMs) and containers, capturing the entire setup, including the operating system, software, packages, and configuration settings. They provide an efficient way to replicate, scale, and manage computing environments, offering flexibility and consistency in application deployment and systems across various infrastructures. Virtual machine images play a crucial role in modern IT practices, supporting tasks such as software development, testing, and system deployment. Virtual machine images are also the main building block of cloud computing. [46]

QCOW2 Formatted Virtual Machine Storage

QCOW2 serves as a storage format designed for virtual disk images, with QCOW standing for *QEMU Copy On Write*. This format introduces a separation between the physical storage layer and the virtual layer by adding a mapping mechanism between logical and physical blocks. Each logical block is mapped to its corresponding physical offset, which enables storage over-commitment and the creation of virtual machine snapshots, where each QCOW volume only represents changes made to an underlying disk image. In the initial mapping, all logical blocks point to offsets in the backing file or volume. When a virtual machine writes data to a QCOW2 volume after a snapshot, the relevant block is retrieved from the backing volume, updated with new information, and written in a new snapshot QCOW2 volume. Subsequently, the map is updated to point to the new place. [67]

1.5.2 Golden Image

> *"In media production, a gold image is the final cut of an album or film after all edits and mixing have been completed. It's in its final, perfect form–it's gold."* [70]

This concept extends to systems administration, where a golden image refers to a deliberately configured snapshot of a system (server, virtual desktop environment, or even a disk drive) that can be used to deploy new instances. Because this golden image (or sometimes a gold image) is used in network virtualization to create new systems, it is alternatively called a *master image* or *clone image*. Another popular term is a *baseline image*, which can be an illustrative term to frame why golden images are so useful: they create a consistent, reliable baseline for system configuration, which can make it easier to maintain those systems across their life cycle. The main reasons to use golden images in the environment are: [70]

- **Faster deployment** - Using golden images helps to deploy faster in cloud environments, both through scripting and automation.
- **Reduced human error** - According to the IBM Cyber Security Intelligence Index [2], 95% of breaches are caused by human error such as misconfigurations, unpatched systems, or poor access controls. Having a predefined and tested template reduces the likelihood of human error leading to a vulnerable system.
- **Faster patch management and upgrades** - Having defined templates enhanced visibility and monitoring capabilities. This is particularly beneficial as it enables a quick assessment of systems that require a patch or an updated package, and eases the identification of those affected by a security vulnerability.

1.5.3 Amazon Web Services (AWS)

Amazon Web Services (AWS) stands as a comprehensive and continually evolving cloud computing platform provided by Amazon, including a mixture of *infrastructure-as-a-service* (IaaS), *platform-as-a-service* (PaaS), and *software-as-a-service* (SaaS) offerings. This comprehensive suite of AWS services provides organizations with tools such as computational power, database storage, and content delivery services. Originating from the internal infrastructure developed by Amazon.com to manage its online retail operations, AWS initiated its web services in 2002 and introduced its distinctive IaaS services in 2006. AWS was one of the first companies to introduce a *pay-as-you-go* cloud computing model, that scales dynamically to provide users with compute, storage, or throughput as needed. Offering diverse tools and solutions, AWS serves enterprises, software developers, government agencies, educational institutions, non-profits, and private organizations all over the world. [8]

Amazon Machine Images (AMI)

An Amazon Machine Image (AMI) functions as a *golden image* (described in Section 1.5.2) for creating virtual servers, referred to as *EC2* instances, within the *Amazon Web Services* (AWS) environment. These machine images essentially serve as configured templates, with an operating system and other software that determine the user's operating environment. AMI types are classified based on region, operating system, system architecture (32 or 64-bit), launch permissions, and whether they are supported by *Amazon Elastic Block Store*

(EBS)[19] or the instance store. Each AMI includes a template for the root volume specific to a particular instance type, typically containing an operating system, an application server, and relevant applications. Permissions are carefully managed to restrict AMI launches to authorized AWS accounts, and block device mapping ensures the correct volumes are attached to the launched instance. [3] [88]

Amazon EC2 (Elastic Compute Cloud)

Amazon Elastic Compute Cloud (Amazon EC2) is a web-based service designed for businesses to run application programs within the public cloud of *Amazon Web Services (AWS)*. This service allows developers to quickly spin up *virtual machines* (VMs), offering computational power for various IT projects and cloud workloads across AWS's global network of data centers. With the Amazon EC2 web interface or an application programming interface (API), an AWS user has the flexibility to adjust instance capacity as needed, either increasing or decreasing it based on current requirements. A developer can code an application to scale instances automatically with AWS Auto Scaling. Furthermore, developers have the option to define autoscaling policies and groups, providing efficient management of multiple instances simultaneously. [12]

Simple Storage Service (Amazon S3)

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Businesses of various sizes can leverage Amazon S3 to securely store and protect any volume of data for a multitude of purposes. These applications include data lakes, websites, mobile applications, backup and restore operations, archival processes, enterprise applications, IoT devices, and big data analytics. Amazon S3 is equipped with management features, helping users to optimize, organize, and configure access to their data in alignment with specific business, organizational, and compliance requirements. [4]

Amazon S3 operates as an object storage service, distinguishing itself from other cloud computing storage types like *block* and *file* storage. In this service, each object is stored as a file with its metadata, and it is assigned a unique ID number. Applications use this ID number to retrieve objects, in contrast to file and block cloud storage where developers typically access objects through a representational state transfer *REST API*. The Amazon S3 object storage cloud service provides subscribers with access to the same infrastructure used by Amazon for its own websites. [14]

1.6 Image Builder

An image builder is a tool used in system administration to create a copy an exact image of a virtual system or configuration (such as an operating system, server, virtual machine (which are covered in Section 1.5) or container) that can later be used as a base from which developers can build and deploy these systems or customized versions on other machines or platforms, or in other environments. Containers and container images, for example, leverage this approach to transport the application code seamlessly across diverse systems or platforms. By preserving the integrity of the original system, developers can then experiment and enhance the image/copy, keeping the original build components and adding

[19] https://aws.amazon.com/ebs/

features and functionality, learning and making improvements as they go without the danger of jeopardizing the original or its permissions. The image builder streamlines this process, allowing developers to avoid building each new image from scratch. The ability to spin up new instances of systems as needed is an invaluable resource for DevOps, not only saving them the time of building the system copy themselves but providing consistency with each image. This consistency and validation with image creation minimize human errors, thereby boosting confidence in the system and enhancing overall efficiency in the development process. [71]

Image Builder also enables to generate the Amazon Machine Images (AMI) which can be retrieved and utilized by Testing Farm infrastructure. AMI are described in Section 1.5.3.

1.6.1 Blueprints

Blueprints, written in the TOML format, serve as text files describing customizations for the image built by the user. They consist of two main sections: *content* and *customizations*. The content section determines what goes into the image from external sources such as packages, package groups, or containers. Content is defined at the root of the blueprint. Within the customizations section, users specify the components to be included in the image, going beyond the default packages defined in the Content section. This could be details like hostname, SSH keys, additional users and groups, timezone, firewall settings, systemd services, files and directories, installation device, repositories, filesystems, OpenSCAP, and FIPS. [27]

TOML Format

A *Tom's Obvious Minimal Language* (TOML) aims to be a minimal configuration file format prioritizing readability through its straightforward semantics. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages. [49]

Example a Image Builder Blueprint

```
name = "BLUEPRINT-NAME"
description = "LONG FORM DESCRIPTION OF BLUEPRINT"
version = "0.0.1"
modules = []

[[packages]]
name = "PACKAGE1"
version = "*"

[[customizations.filesystem]]
mountpoint = "/var"
minsize = 2147483648

[[customizations.sshkey]]
user = "root"
key = "A SSH KEY FOR ROOT"
```

1.6.2 OSBuild Composer

osbuild-composer is a service designed to generate customized operating system images, currently supporting Fedora and RHEL. These images are compatible with various virtualization software like QEMU, VirtualBox, and VMWare, as well as with cloud computing platforms such as AWS, Microsoft Azure, and Google Cloud Platform. There are two front-ends that users can use for interacting with osbuild-composer: [31]

1. **Cockpit Composer**: The web-based management console Cockpit, includes a UI extension for building operating system artifacts.
2. **Command-line Interface**: The *composer-cli* provides a Linux command line interface (CLI) offering functionality from OSBuild. This CLI is part of the project, which served as a precursor to OSBuild.

The whole OSBuild Composer structure is shown on a simplified block scheme in Figure 1.8.

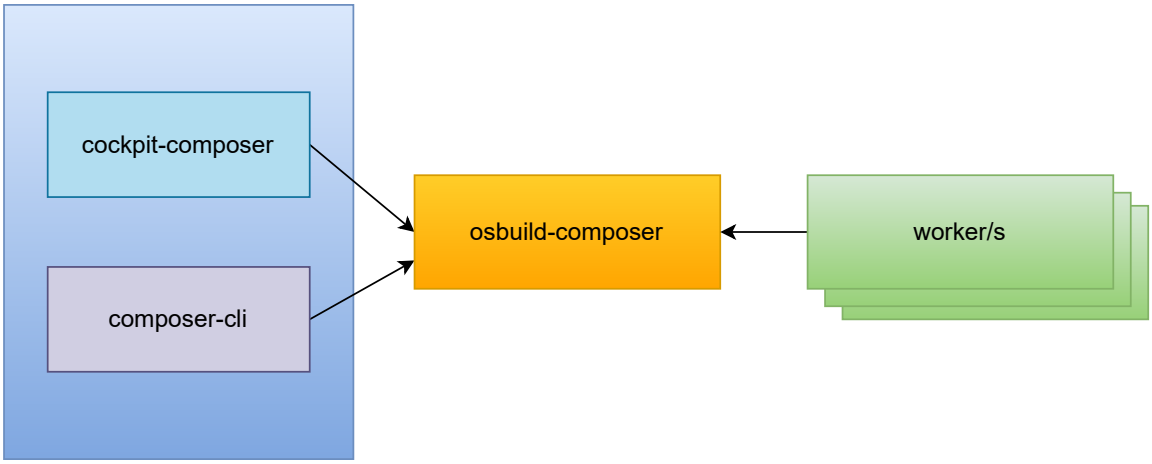

Figure 1.8: OSBuild Composer structure

Composer serves as a middleman linking the workers of *osbuild* with user interfaces such as *cockpit-composer*, *composer-cli*, and others. It defines a framework for high-level image compositions that it can construct. Users can initiate builds of these compositions through various Composer APIs, which subsequently translate the requests into pipeline descriptions for osbuild. The resulting pipeline output is then either returned to the user or uploaded to a target specified by the user. The full infrastructure scheme is shown in Figure 1.9. [33]

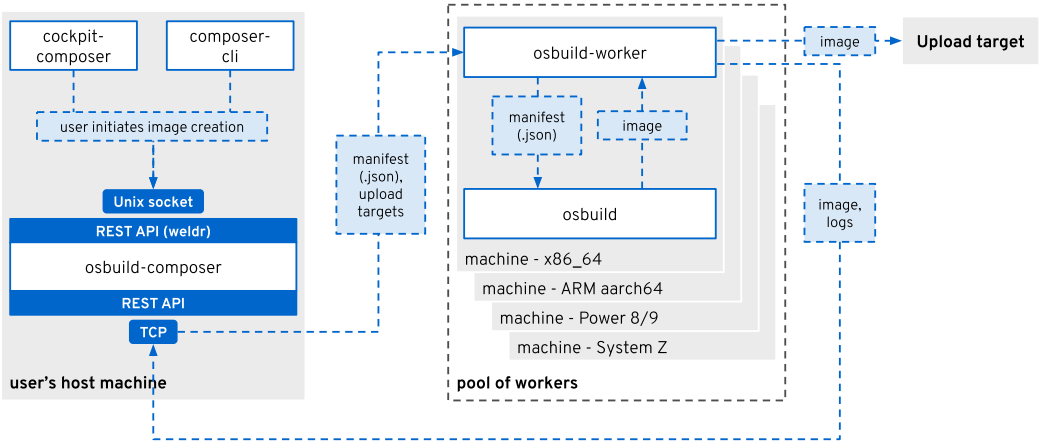

Figure 1.9: Full OSBuild Composer infrastructure. Image source [29]

1.6.3 Image Builder Service

Image Builder is accessible as a managed service as part of *Red Hat's Hybrid Cloud Console*[20]. It is mainly used by Red Hat's customers, but can also be used by anyone with a Red Hat developer license during the development of RHEL and Fedora. This tool is designed to construct personalized images of RHEL and CentOS Stream for many footprints (traditional, ostree-based), targets (Public clouds, Baremetal, etc), and architectures (x86, ARM, IBM). [28]

Image Builder comes with its own online service, that can be utilized via its well-documented HTTP API[21]. Users can make use of it if they do not want to deploy their own local osbuild-composer (described in Section 1.6.2). However, the online service comes with a set of cons as well, the main being that it does not support all of the features that are available from the osbuild-composer due to the fact that it is still under development. It also lacks the newly added features that are introduced to the OSBuild Composer on-premise service, because the composer is used as the testing platform before the finalized features reach the online API-accessible service. For example, the Image Builder service at this time does not support the OpenSCAP tailoring feature, which is required in this thesis's main objective. However, the hosted service comes with a few pros as well. As the on-premise service utilizes the local environment, it is capable of building only the same distribution images. Fortunately, this can be worked around with the utilization of payload repositories that have a whole Section 1.6.4 dedicated to them. The hosted service, on the other hand, provides the possibility to create RHEL/Fedora/CentOS Stream distribution images via the same API, which could help users who need to build multiple images across all distributions. Thus the hosted service removes the obstacle of the need for using the custom payload repositories. Figure 1.10 depicts the architecture of the Image Builder service. [30]

[20] https://console.redhat.com/
[21] https://developers.redhat.com/api-catalog/api/image-builder

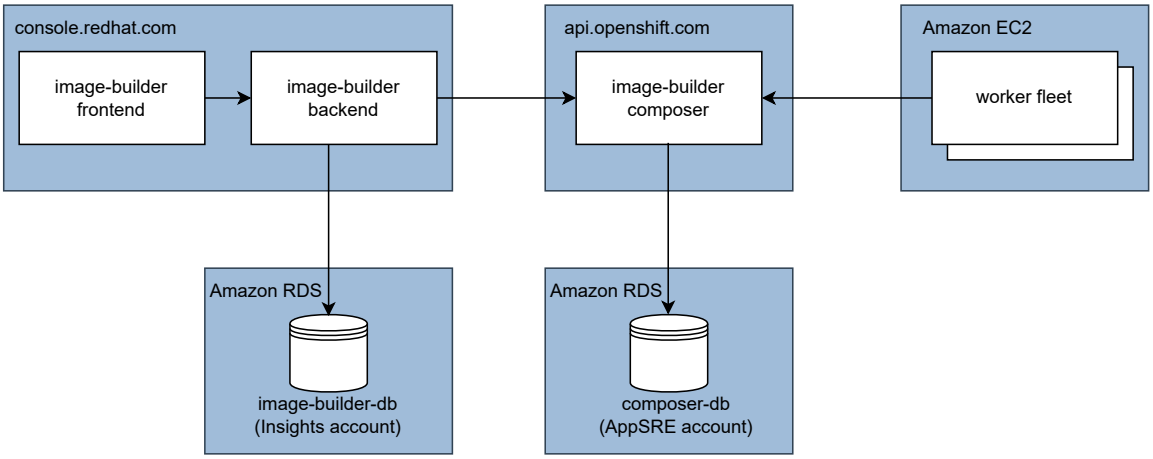

Figure 1.10: Image Builder Service architecture

1.6.4 Payload Repositories

To include a third-party package during the build process, users must activate the appropriate third-party repository as a *payload repository*. However, this does not permanently store the repository configurations in the image, and the repositories will not be accessible to users once the image is compiled. These repositories are stored in the OSBuild specific path */etc/osbuild-composer/reposi-tories* and need to have a specific naming *rhel-9.0.json* (*rhel-90.json* for the osbuild-composer prior to the version 100). [61]

1.6.5 OpenSCAP Integration

The RHEL image builder offers support for OpenSCAP integration, enabling the creation of pre-hardened RHEL images. Through the configuration of a *blueprint* (covered in Section 1.6.1), it is possible to carry out the following tasks: [55]

- Tailor it with a predefined security profile (currently available only for the on-premise image builder.
- Incorporate specific packages or supplementary files.
- Generate a customized RHEL image, prepared for deployment on the selected platform, aligned with customers' environment's requirements.

Red Hat continually updates versions of security hardening profiles, providing options during system building to align with customers' current deployment guidelines. This ensures that customers' systems meet the latest security standards. [55]

This feature is limited to the versions *RHEL-8.7.0* and *RHEL-9.1.0* and all of their subsequent versions. The osbuild stage is actually implemented during the image-building process. This stage incorporates the OpenSCAP tool, which operates on the filesystem tree, evaluating the standard criteria defined by the given profile. The tool then also applies the necessary remediations to the image. This approach allows users to construct a thoroughly hardened image, surpassing the level of hardening achievable by applying remediations directly to a live system. [32]

To use the OpenSCAP profile in the image, the user needs to generate the base blueprint with the filesystem customization (oscap generate tool is more described in Section 1.3.7)

and add the customization section to the blueprint looking similar to the example below. It requires only two values. One for the *OpenSCAP Profile* which is covered in Section 1.3.7, and the second for the desired *Data Stream*, which is also described in Section 1.3.7.

Example of the blueprint customization section for OpenSCAP

```
[customizations.openscap]
profile_id = "xccdf_org.ssgproject.content_profile_standard"
datastream = "/usr/share/xml/scap/ssg/content/ssg-fedora-ds.xml"
```

The number of standards that are supported by Image Builder is also limited. To see all of the supported standards, refer to Table 1.1.

Table 1.1: The list of distro-specific supported Security Standards in Image Builder

Security Standard	**Fedora**	**RHEL 8.7^**	**CS9/RHEL 9.1^**
ANSSI-BP-028 (enhanced)		x	x
ANSSI-BP-028 (high)		x	x
ANSSI-BP-028 (intermediary)		x	x
ANSSI-BP-028 (minimal)		x	x
CIS Level 2 - Server		x	x
CIS Level 1 - Server		x	x
CIS Level 1 - Workstation		x	x
CIS Level 2 - Workstation		x	x
CUI		x	x
Essential Eight		x	x
HIPAA		x	x
ISM Official		x	x
OSPP	x	x	x
PCI-DSS	x	x	x
Standard	x		
DISA STIG		x	x
DISA STIG with GUI		x	x

Chapter 2

Analysis of the Current State

The current solution for the quality assurance of hardened systems is strongly tied to the understanding of the hardening process. This prerequisite means that a lot of quality engineers struggle to get past the barrier of entry, to begin testing their packages. That is why the RHEL Security Compliance team wrote the requirements for this thesis. The selected solution should find the sweet spot between the optimal solution for the users (Quality Engineers) with a minimal cost to the RHEL Security Compliance team (described in Section 1.3.1). The idea is that the final product that will be delivered to the RHEL Security Compliance team will be able to perform a security hardening on a specified RHEL version, and it should be easily distributed within all Quality Engineering teams in Red Hat. Quality engineers from these teams would have access to already pre-hardened systems, and they would only have to worry about the functional testing (described in Section 1.2.2) without the necessity for security hardening know-how.

First, a qualitative analysis had to be carried out on the potential solutions that Red Hat's infrastructure could offer. This analysis identified the first step forward, which was to find the optimal solution by leveraging the feedback from the subject matter experts (Quality Engineers), which is described in Section 2.1.

The next step was to reach out to teams that are critical to this project. As this project is heavily integrated into the existing solutions of Red Hat's portfolio, cooperation with multiple teams was required. Cooperation with teams such as the Testing Farm team, the Image Builder team and the RHEL Security Compliance team were the building blocks of the thesis. In Section 2.2, the process of identifying these teams and collaborating on planning with them is elaborated further.

After having decided on the most favorable option and which teams would have to be involved in the whole project, it had to be also decided which quality engineering teams within Red Hat would be the right ones for the pilot testing of this project. Feedback from these users is crucial for fine-tuning the final testing environment that will be used widely across the organization. More about identifying the right quality engineering team for the pilot project testing and providing feedback is covered in Section 2.3.

RHEL Security Compliance team had a rough idea of how the whole project could work. A simple use-case diagram in Figure 2.1 depicts this imagined workflow.

Figure 2.1: Preliminary idea for the thesis goal

2.1 Finding the Optimal Solution for Testing Machines

After a thorough introduction to the theoretical background of security compliance, the OpenSCAP project, and the main purpose of this thesis, the next step was to brainstorm alternative approaches to the selected solution to the defined problem.

With the internal knowledge of Red Hat's portfolio, and the consultations with the subject matter experts from the Quality Engineering team and RHEL Security Compliance team, four different ideas emerged. The first one was to use the Testing Farm (more about Testing Farm in Section 1.2.4) with the integration of Ansible scripts (described in Section 1.4) to pre-harden the basic image using 'oscap' remediation tool. This approach is more detailed in Section 2.1.1. The second possible solution consisted of not using the Testing Farm architecture, but rather using the internal servers provided to the RHEL Security Compliance team and using the already pre-hardened images from the Image Builder service (described more in detail in Section 1.6.3). To further elaborate on the second possible solution, refer to Section 2.1.2. As for the third possible solution to the thesis's main goal, the idea was to use the Image Builder Service in the same way as in the second solution, although in this case, it could take advantage of the Amazon Machine Images (detailed in Section 1.5.3) building feature implemented in the Image Builder. These images could then be picked up by the Testing Farm Amazon account to provide the provisioning mechanism for the testers. A more detailed explanation of the third potential solution is provided in Section 2.1.3. The last of the fourth potential solutions was to use the Image Builder on-premise OSBuild Composer service (detailed in Section 1.6.2), as it had the latest and greatest features implemented. The provisioning mechanism will take advantage of the Testing Farm infrastructure same as in the third solution. This last solution was also the selected solution due to the limitations of all previous ones. To read more about the fourth and selected proposed solution, refer to Section 2.1.4. At the end of each solution, all discussed advantages and disadvantages were listed, and their evaluation played a crucial role in the final decision-making process.

2.1.1 Testing Farm with Ansible Playbooks

With Testing Farm being one of the standard providers for internal testing at Red Hat, it became one of the primary solutions that appeared to align well with the objectives of the thesis. As the Testing Farm uses the Ansible playbooks for its internal configurations, it could leverage them for security hardening the already integrated RHEL images. However, by following this solution, the entire burden (which comes from maintaining such

a project) would fall on the Testing Farm team. This would also result in the RHEL Security Compliance team not having direct access to the implemented hotfixes and features of the images.

Pros

- **Well-know process** of using the prepare Ansible scripts in Testing Farm infrastructure.
- **Ansible prepare script can be created locally by each user**, and fine-tuned for the specific problem.

Cons

- The entire **responsibility on the Testing Farm team**, with hard reach from the RHEL Security Compliance team.
- Significantly **increased execution time** for the image as the preparation includes hardening that could take several minutes during every single provision.
- **Requires Ansible knowledge** from the Quality Engineers.

The implementation of this solution faced significant challenges due to substantial concerns regarding the inability to share responsibilities between the RHEL Security Compliance team and the Testing Farm team due to the reasons mentioned before. Additionally, a crucial aspect of this solution involved finding an optimal approach with minimal burden on Quality engineers (users). However, it was not possible in this case due to the required Ansible knowledge and extended execution time in testing caused by the non-pre-hardened image. As a result, this solution was deemed impractical.

2.1.2 Compliance Team Infrastructure with Images Generated via Image Builder Service

Since the RHEL Security Compliance team does possess its own testing servers, another idea was to use their already functioning infrastructure and store the virtual machine images (described in detail in Section 1.5.1) in there. These images would have been pre-hardened in the Image Builder tooling, so the execution time is not bottlenecked by the hardening process during the initialization. This would also allow the RHEL Security Compliance team to have complete control of the environment and easy access to the machines in the case of an incident.

Pros

- **Complete control of the testing environment is under the RHEL Security Compliance team**, which will result in faster support time in case of the user's issues.
- **Great cross-team cooperation**, as the Image Builder team and RHEL Security Compliance team have experience with their mutual cross-team cooperation.

Cons

- **Worse scalability** in case of further development of the project due to the small capacity of the RHEL Security Compliance team.
- The RHEL Security Compliance team does not possess the **provisioning mechanism** for a better user experience, which could result in lower engagement.
- Provisioning images across all Red Hat **supported architectures is not possible due to hardware limitations**.

Due to the fact that the RHEL Security Compliance team's internal servers lack hardware for the possibility to provision all Red Hat supported architectures and that they have limited capacity, this solution was also scratched out and could not be used. Although it would mean that future problems with the hardening could be resolved in the shortest time frame, the negatives were far greater than the positives.

2.1.3 Image Builder Hosted Service

One of the ideas was to generate virtual machine images using the Image Builder hosted service. This would ease the image-building time because the RHEL Security Compliance team could simply make use of the Image Builder service API, and request the images straight from it instead of having the on-premise service deployed. Unfortunately, the implementation of this approach is currently blocked by certain limitations in the hosted service.

The Testing Farm was also intended to be used for the image provisioning mechanism in this solution. It has a great integration with the Amazon Simple Storage Service (mentioned in Section 1.5.3) which enables a quick and easy connection between freshly created images from Image Builder service straight into the Testing Farm infrastructure.

Ondrej Budaj from the Image Builder team wrote a blog [10] where he describes how to create simple images in the Image Builder hosted service. Additionally, he highlights the features that enable the creation of Amazon Machine Images along with their subsequent storage on Amazon's S3 servers.

Pros

- **Supported Amazon Machine Images**, which can be later picked by Testing Farm infrastructure with minimal effort.
- **Easy and well-documented HTTP API**, that can be automated and scripted.
- **Omitting the initiation of the on-premise OSBuild Composer** service saves valuable time and resources.

Cons

- **Limitation in tailoring the OpenSCAP profiles**, which are needed for the disabling of root SSH access to the machine (requirement of Testing Farm).
- **Slower feature development** compared to the on-premise OSBuild Composer service.

This solution has many advantages, unfortunately the possibility of tailoring the OpenSCAP profiles was mandatory in order to use the Testing Farm infrastructure. The root SSH access to the machine is required for the initialization of the testing environment, and this step cannot be skipped. The Image Builder team has already planned to release *API version 2.0* which will support this feature, but the *estimated time of arrival* (ETA) is in the 3rd quarter of 2024. This project needs to be finished way before the ETA, so it cannot wait for the new version of the API. In the future, the solution could be re-analyzed and this one could become the most attractive one. The Image Builder team has expressed significant interest in integrating OpenSCAP into their services. They are eager to incorporate all the necessary features requested by the RHEL Security Compliance team. However, in the meantime, this solution has to be postponed.

2.1.4 On-premise OSBuild Composer

The last and selected solution was to change one part of the third idea, and that is to use the on-premise OSBuild Composer instead of using the hosted Image Builder service. The main reason for this is because of the limitation in the OpenSCAP profile tailoring feature in the hosted service. Apart from that, it would use the Testing Farm infrastructure for the provisioning mechanism similar to the solution described in Section 2.1.3. However, the on-premise OSBuild Composer does not have the ability to generate the AMI images, and therefore the QCOW2 images (described in Section 1.5.1) have to be generated and integrated into the Testing Farm.

Pros

- **Feature rich on-premise OSBuild Composer service**, as it gets the latest features implemented first.
- **Good integration with the Testing Farm** provisioning pipeline, which uses the qcow2 images already in their solutions.
- Uses **Red Hat's production grade services**

Cons

- Prerequisite of **initializing the on-premise OSBuild Composer** and the requirement for the RHEL Security Compliance team to understand its fundamental operations.
- Requires **additional storage capacity for each generated variant**, contributing to an incremental demand for storage resources in Testing Farm infrastructure.

Despite the drawbacks of this solution, it was the best solution available at the time of the analysis. Each of these variants was presented to the RHEL Security Compliance team, and after evaluating all of them, this solution was flagged as the *final choice* to be implemented in the thesis. Compared to the others, there were no blockers and the overload on each team was kept optimal and minimal at the same time.

To understand the abstract of this solution, refer to the simple use-case diagram in Figure 2.2.

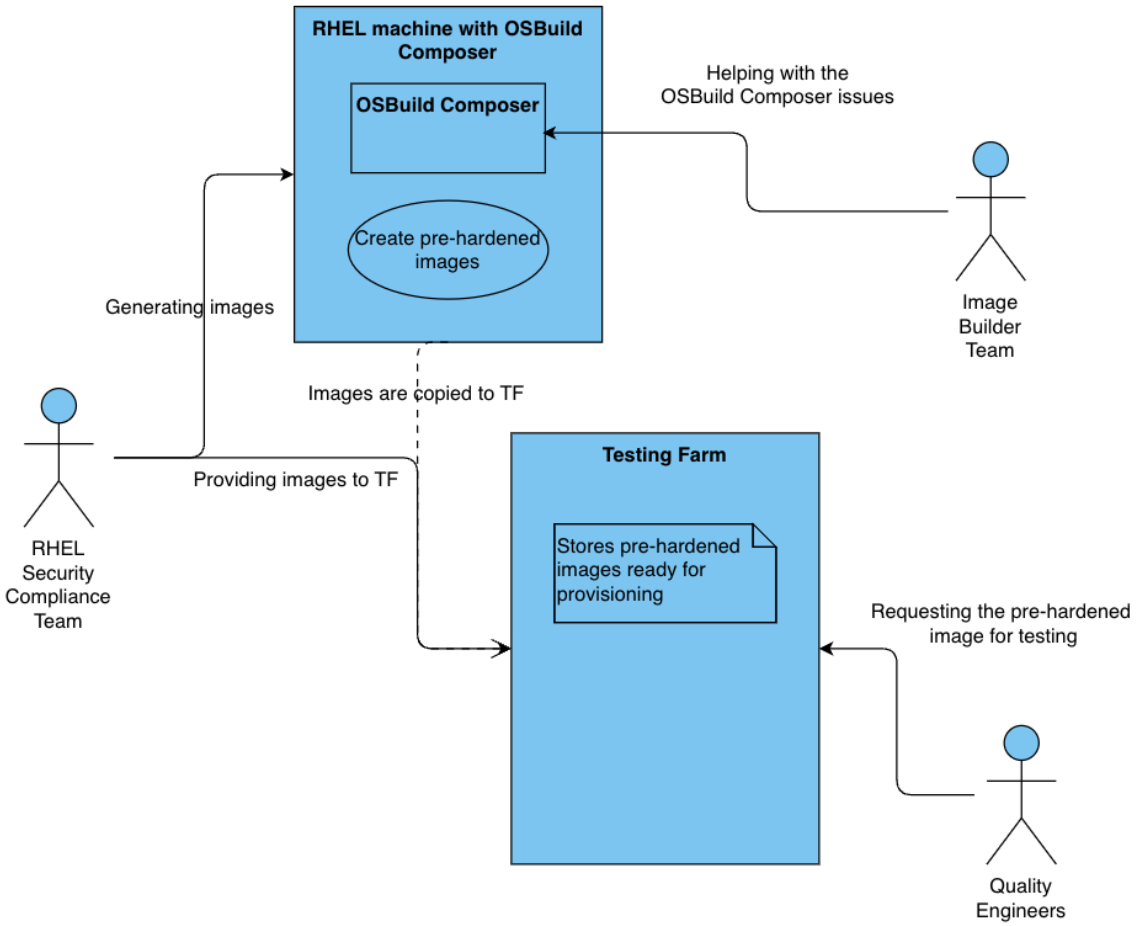

Figure 2.2: Abstracted solution derived for the thesis implementation

2.2 Identifying the Right Teams for Collaboration

After successfully identifying the optimal solution to achieve the thesis's goal, as discussed in Section 2.1, the subsequent step involves initiating communication with the relevant teams. This phase is essential for notifying and engaging the related teams, whose collaboration and contributions are important for the seamless integration of the proposed solution. This strategic interaction aims to ensure a coordinated effort and promote a collaborative environment, thereby improving the overall implementation process and the success of the thesis's objectives.

As indicated by the proposed solution, beyond reaching out to the RHEL Security Compliance team, it is vital to establish contact with other pivotal teams. Specifically, the Image Builder team and the Testing Farm team are identified as the key stakeholders in the implementation process. This strategic outreach to various teams underscores the importance of establishing a cross-functional collaboration that aligns with the broader objectives of the undertaken project. The entirety of Section 2.2.1 is devoted to detailing the approach of involving all specified teams in the concerted effort.

The Image Builder team was important for the whole pre-hardened image generation. The on-premise OSBuild composer service will be utilized for the implementation. In consequence, the team should be made aware of it. This is to ensure their cooperation in identifying the root cause of any bugs or issues that may be encountered. Section 2.2.2 is dedicated to describing how the overall readiness and early problems were analyzed with the Image Builder team, and the mutual cooperation with the underlying issues.

A dedicated focus on the Testing Farm team's involvement is encapsulated in Section 2.2.3. This section is crafted to provide an in-depth exploration of the preparations and considerations specifically tailored to align with the Testing Farm team's requirements. It also covers what initial steps would have to be resolved in order to prepare the underlying infrastructure to be able to handle the pre-hardened images. As these types of images were not considered when the infrastructure was developed, there were few tweaks that had to be done in order to seamlessly provide pre-hardened images to the Quality Engineers using this service.

2.2.1 Initial Meeting (Brainstorming)

Following the preliminary analysis of potential solutions and the selection of the optimal one, the subsequent step involved reaching out to the teams that needed to be involved. The introductory meeting was held in a virtual setting and saw the active participation of seven individuals. Within this diverse group, three representatives that come from the Image Builder team contributed their insights, followed by the valuable perspectives of two representatives from the RHEL Security Compliance team. Additionally, one representative from the Testing Farm team added their unique viewpoint to the collaborative discussions that unfolded during this virtual engagement. The diversity of representation from these distinct teams not only enriched the discussion but also created a comprehensive understanding of the project's reach across various domains within the company.

Most of the discussion also covered the mandatory prerequisite work needed on the side of individual teams, in order to integrate all of these huge pieces of machinery together to complete this goal. While the introductory meeting generated some concerns, it is noteworthy that each concern was thoroughly addressed through collaborative discussions. The participants worked collectively to resolve any uncertainties and ensure that all perspectives were considered. Ultimately, a consensus was reached, affirming that every team member was on board with the project. With all of these approvals and the positive feedback from participants, the project was on to a great start.

The next thing on the "To Do list" was to brainstorm what are the next steps required for smooth project implementation. One of the first steps concluded at the meeting was to start generating the pre-hardened images and find out what could be improved. The Image Builder project faced a challenge in the form of limited user experience feedback, which contributed to an unclear state of readiness. Addressing this gap in user input became crucial to gaining a comprehensive understanding of the project's usability and ensuring that it aligns effectively with end-user expectations. Section 2.2.2 is dedicated to this topic.

2.2.2 Evaluating the Image Builder Readiness

The analysis of Image Builder readiness for implementation of the selected solution was also included as part of the initial analysis for this thesis. This analysis was divided into two parts. The first part evaluated the overall Image Builder readiness, specifically the

OSBuild Composer tool (described in detail in Section 1.6.2). The second part evaluated the readiness of the Image Builder hosted service via its HTTP API (described in Section 1.6.3).

Image Builder OSBuild Composer Readiness

Following the discovery that the Image Builder hosted service lacked support for the OpenSCAP tailoring feature, the next step was to check if the OSBuild composer on-premise service (detailed in Section 1.6.2) offers compatibility with this essential functionality. Fortunately, it was implemented in the OSBuild Composer and it worked as expected. However, implementing this solution would require additional effort and a deeper understanding of using the on-premise service to generate the desired pre-hardened images. Additionally, it is important to note that implementing this solution requires a functioning RHEL machine (for more information about RHEL, refer to Section 1.1) in order to generate RHEL images. This requirement comes from the necessary repository requirement for the specific operating system.

2.2.3 Analysis of Testing Farm Necessary Prerequisite Work

The Testing Farm was not utilizing the images generated by either the Image Builder hosted service or OSBuild composer. This meant that it would be crucial to implement the integration between Testing Farm and Image Builder, to proceed with the selected solution implementation.

The initial step necessitated the establishment of a standardized image naming convention to differentiate pre-hardened images from the existing image repository. This would require the implementation of the internal processes run by the Testing farm, as well as the open-source part.

From the open-source infrastructure[1] the requirement was to implement the pre-artifact-installation, which consist of a several tasks:

- **Enable Beaker-tasks repository** to have available internal beakerlib libraries.
- **Enable Buildroot repository** which provides the crucial components used by the minimal RHEL installation.
- **Enable Code Ready Builder (CRB) repository** which stores packages used for the development by third-party developers.
- **Add RHEL extras updated and nightly repositories** used for extra content, not generally available in standard repositories.
- **Install Koji/Brew** RPM-based build systems, required for the CI/CD tasks by Quality Engineers. To understand the Koji/Brew build systems and RPM, refer to Sections 1.1.2 and 1.1.1 respectively.
- **Add tag repositories**, which can be utilized to install specific packages for a specific Brew/Koji tag.

Upon completion of these implementations, the Testing Farm will achieve efficient provisioning of pre-hardened images generated by the Image Builder services. Also, these images will be ready for the Quality Engineers, minimizing the need for both Testing Farm and OpenSCAP hardening expertise.

[1] https://gitlab.com/testing-farm/infrastructure

2.3 Pilot Project for Gathering User Feedback

After the initial analysis of the project, it was required to incorporate some testing measures. The best way to measure the quality of the product seemed to be the so-called *alpha testing* on the potential users (Quality Engineers). Due to the big interest in such solution within Red Hat's Quality Assurance teams, it was not hard to find volunteers for the pilot project testing. This testing would be implemented in multiple stages and at the end of each phase the feedback would be gathered so the prototype could be tweaked and fixed based on it. As this project is tailored for the Quality Assurance teams, it was crucial to get them aboard as soon as possible. Not only it would capture the attention of the Quality Engineers, but also it would provide the necessary feedback that will help shape the overall user experience.

2.3.1 Selecting QA Teams for the Alpha Testing

As there was widespread interest in this project across several Red Hat teams, it was not hard to identify the alpha testing teams that could be used within the pilot project feedback gathering. This testing would facilitate the gathering of feedback from a diverse and representative cross-section of Red Hat's internal users. This feedback would ultimately contribute to strengthening the reliability and relevance of the project itself.

Systemd Team

Systemd is a system and service manager for Linux widely adopted by Red Hat Enterprise Linux (RHEL) operating system. This makes Systemd team a very valuable part of Red Hat's portfolio, and the requirements for Quality Assurance (more detailed in Section 1.2) are significant. Systemd team also experienced a problematic case that helped them identify how the testing of hardened systems could be improved. It was identified due to the reported regression bug from the customer. This case is described in Section 3.3.3. After resolving this issue the interest in this project became even bigger and team members volunteered in alpha testing.

SELinux Team

SELinux (Security-Enhanced Linux) is a security architecture for Linux systems empowering administrators with greater control over who can access the systems folders and files. Red Hat's blog about SELinux [69] describes this architecture in more detail. Due to the fact that the SELinux team is focused on security, it was without a doubt the first team within Red Hat that was considered to become an alpha testing team. OpenSCAP hardening also modifies the SELinux configurations within its profile rules. Therefore, ensuring the safe handling of these modifications should naturally be a concern for the SELinux team.

Special Projects Team

Special Projects team specializes in the productization and development of security technologies within Red Hat's Engineering teams. They are directly responsible for widely used tools like *syslog*, *sudo*, *usbguard*, or *aide*. They also work on the project *Keylime*, which is the one that is tested on the FIPS-enabled (more about FIPS or other security standards in Section 1.3.2) systems as well. The Special Projects team's desire to leverage security-

hardened images for their testings presents a valuable opportunity for collaboration within the scope of this thesis.

Crypto Team

Given the team's name, their deep involvement in this project comes as no surprise. The Crypto team serves as the cornerstone of modern cybersecurity within the RHEL ecosystem, actively maintaining the cryptographic algorithms that form the foundation of secure communication and data protection. Their expertise extends beyond algorithm development, as they are also responsible for crafting and maintaining the crypto policies that are at the core of the cryptographic subsystems of RHEL. These policies ensure the secure implementation of protocols like *TLS*, *IPSec*, *DNSSec*, *Kerberos*, and many others. The Crypto team's participation in this thesis project was mutually agreed upon from the beginning, reflecting their strong commitment to advancing the field of cryptography and their eagerness to contribute valuable insights and feedback.

Chapter 3

Own Proposal For Solutions and Contribution

Following the comprehensive analysis outlined in Chapter 2, the project was well-positioned to transition into the implementation phase, based on the findings from the analysis itself. This Chapter is divided into multiple sections that are chronologically ordered.

First Section 3.1 mainly focuses on the whole creation process of the testing environment, that would later be used by the Quality Engineers (the users). This section is the biggest part of the implementation process and that is why it is the longest one, to cover all of the details. It covers both the pre-hardened image creation as well as the Testing Farm integration, which serves as the storage and provisioning mechanism for the users. It also covers the automation that was implemented for much easier scalability of the project itself in future use. The automated scripts were introduced both to the Image Builder blueprint creation process as well as to the image building process itself.

Another significant aspect of this project involves establishing a well-documented Quality Assurance (QA) pipeline with the newly introduced security-hardened images. This is crucial for the future development of this project, as it will be transferred to the RHEL Security Compliance team (to read more about this team, refer to Section 1.3.1). Also, it is very important to properly communicate the process with the Quality Engineering team, which will be using it the most. The specific teams in this category were mentioned in Section 2.3. They will be mainly interested in the provisioning mechanism of the Testing Farm, and all of the possibilities of integration in the QA process. All relevant details regarding the QA pipeline and its documentation will be addressed in Section 3.2.

Section 3.3 is focused on the time and economic evaluation from the Quality Engineering point of view, as well as Red Hat's point of view. The first part of this section focuses on the economic point of view of the project itself. The analysis takes into account the time and resources needed for the entire implementation including the involvement of the Testing Farm and Image Builder teams, which were critical for the overall integration of this thesis. One part of this section contains the *Return on Security Investment* (ROSI) calculation that helps determine the overall savings from implementing this solution. The second part of this section establishes a baseline by examining the state of the system before implementing the proposed solution. This is then compared with the state after the successful completion of this thesis and the time saved by this solution is evaluated. This evaluation is done by the combination of quantitative and qualitative analysis with the help of the users (Quality Engineers). Section 3.3.3 also mentions the customer case which could have been prevented

by earlier implementation of this solution. Analysis of this customer case can provide insights into potential future customer evaluations for similar issues.

The concluding Section 3.4 presents projections regarding the key areas of forthcoming advancements in the project. Since the primary aim of this project was meant to be a *proof-of-concept (PoC)* to identify the most effective solutions for the problem and to establish an initial prototype of the pre-hardened virtual image creation process, including its provisioning to Quality Engineering teams. One of the largest changes that will be discussed is the migration of the entire process to the hosted Image Builder service following the implementation of its planned features. Additionally, a proposed testing cycle plan for pre-hardened image functional testing is provided.

3.1 Creating the Testing Environment

This Section describes the whole process of creating a virtual image (refer to Section 1.5.1 to read more about virtual images), hardening it using the OpenSCAP tooling (described in Section 1.3.7) and integrating it into the Testing Farm infrastructure (which is more detailed in Section 1.2.4)

The process of creating the final testing environment is chronologically divided into logically separated subsections. In the first step, the *Image Builder blueprints* (described in Section 1.6.1) had to be generated in order to get some baseline that can be later edited. Section 3.1.1 describes the process of generating the pre-hardened Image Builder blueprints using the *"oscap generate"* command (more detailed in Section 1.3.7). This tool helps to generate some basic configurations that are crucial from the file system's point of view. This tool is still under development, and thus some features related to the hardening process might not be implemented yet. However, it can be worked around with the implementation of a customized Python script. Users can execute only this one single script which generates the missing blueprint customizations, making this blueprint ready to be used.

Additionally, the Testing Farm infrastructure requires a specific set of repositories to be added to the virtual images as well. As the process of adding these repositories can be performed within the Image Builder itself, they have to be added to the blueprint. Testing Farm provides its own scraper tool to generate the repositories, however, the blueprint uses a different syntax language called *Tom's Obvious Minimal Language (TOML)*, which is more detailed in Section 1.6.1, and thus this had to be added to the previously mentioned Python script as well. The whole Section 3.1.2 is dedicated to this part of the implementation process.

To finish the automated blueprint generating process, the scalability feature in the form of changing the blueprint and repositories based on the RHEL (described in Section 1.1), version had to be created. Section 3.1.3 describes how this feature was implemented in the Python script finalizing it to the form where it takes the RHEL compose ID as the argument and generates the final ready-to-be-used blueprint which consists of all of the customizations mentioned earlier.

After having the automation created, the next step was to set the environment for the image-building process. As mentioned in the analysis in Section 2.1.4, the Image Builder Composer service needs to be started in the operating system same as the output image. In this case, it was RHEL version 9. Accordingly, Section 3.1.3 dives deeper into this configuration process, ensuring that the necessary conditions are met for seamless image construction.

Additionally, Section 3.1.4 explains the whole image-building process from the blueprint as the input for the Image Builder Composer, to the qcow2 (more about this image type in Section 1.5.1) image file. This section documents the specific commands utilized for improved replicability.

As the image-building process will be heavily used in the future stages of this project, it was a good thing to automate this process, in order to save the developer's time. Section 3.1.5 is dedicated to this process and explains the scripts that are ready to be used by future developers.

The final step in this process was to integrate the produced virtual image into the Testing Farm infrastructure. It required changes in the existing infrastructure in order to get these images accepted by the machinery and make them ready to be deployed. Section 3.1.6 describes this whole part of the project as well as the contributions made to the Testing Farm project, required for the successful integration. After having the virtual image integrated into the Testing Farm infrastructure, it had to be tested for the hardening rules again. This post-deployment testing served to verify that the image continues to be compliant with the established security policies (that are described in Section 1.3.7). This section details the testing process itself, while the attached figures within this section present the obtained results. This section also describes the brainstorming of the naming convention used for the pre-hardened images produced by Image Builder. It was critical to have this naming convention standardized for the future development of this project.

3.1.1 Generating Blueprints for Pre-Hardened Images

After the initial analysis, which led to the decision that the Image Builder OSBuild on-premise service would be used (Section 2.1) for the virtual image creation process, the first step was to create a *blueprint* (more about blueprints in Section 1.6.1) which would be consumed by the service in order to generate the specified virtual image. The whole blueprint is written in the TOML format as mentioned in the blueprint section earlier.

Several aspects of the blueprint required specification to fulfill the defined criteria:

1. **Basic metadata**: The first section of the blueprint focuses on establishing the basic elements of the virtual image, including its name, description, version, and intended distribution of the underlying operating system.

   In this case, the *name* was picked to meet the naming standards and to contain the information for the automated testing scripts. This specific naming convention is later more detailed as part of Section 3.1.6. The *description* for the image briefly mentions the version of the *RHEL* operating system and the used Security Hardening profile (CIS in this case). As for the *version*, it is the first version of this type of image, so the lowest one was picked. And finally, the *distribution* value was added to specify the desired operating system distribution (with the version included).

   ```
   name = "RHEL-9.3.0-20240302.6-CIS-HARDENED"
   description = "A RHEL-9.3.0 CIS security hardened image"
   version = "0.0.1"
   distro="rhel-93"
   ```

2. **Security profile hardening customizations**: As for the second part of the Image Builder blueprint, it had to be defined which *OpenSCAP security profile* should be used for the actual security hardening process.

This section of the customization requires two key-value pairs. The first one is the *datastream* (detailed in Section 1.3.7) which serves to be used for the individual profiles and their rules for the specific operating system (refer to Table 1.1 for all supported operating systems and their corresponding security profiles). The second key-value pair required by this customization is the *profile ID*. The value of this profile ID serves to specify the exact profile within the datastream. Among the security profiles considered, the CIS security profile was deemed the most suitable for the purposes of this thesis. To read more about CIS security profile refer to Section 1.3.7.

```
[customizations.openscap]
datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml"
profile_id="xccdf_org.ssgproject.content_profile_cis"
```

3. **Testing Farm specific tailoring**: For the last part of the Image Builder blueprint, one of the rules from the CIS security profile had to be disabled. This action is in the OpenSCAP dictionary called *tailoring*, which is also described in Section 1.3.7. As the Testing Farm uses its Amazon machine images (described in Section 1.5.3), the initial preparation scripts have to be executed with the superuser (root) privileges. Also, the individual functional tests executed by Quality Engineers require this access. However, the CIS security profile disables the *Secure Shell (SSH) root login* with its security rules, as such access presents a significant security hazard in the production environments. Fortunately, in this case, it is an internal isolated testing environment, that can have this access enabled without posing any security risk.

   This OpenSCAP tailoring customization in the blueprint requires only one key-value pair. The *unselected* key has to be matched with the exact ID of a rule within the security profile. In this case, the profile ID is *content_rule_sshd_disable_root_login*, with the xccdf prefix (to learn more about XCCDF refer to Section 1.3.7).

```
[customizations.openscap.tailoring]
unselected=[ "xccdf_org.ssgproject.
    content_rule_sshd_disable_root_login" ]
```

Using oscap for Generating Profile Specific Blueprint

The basic required customizations are generated by the command *oscap xccdf generate fix*. This command requires the parameter of *fix-type* which in the case of Image Builder blueprint is called *blueprint*. Also, it requires the profile ID, which in this case should be *cis*. Lastly, it needs the *datastream* path, so it knows the exact rules for the given security profile.

To put it all together, the command that will generate the basic customizations required by the CIS security profile will look like this:

```
oscap xccdf generate fix --fix-type blueprint --profile cis \
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

The output of this command is the individual blueprint customizations, mainly the ones that are at the file system level, and thus have to be defined at the time of the image-building process. The customizations for the CIS security profile are shown in Listing 3.1:

```
[[customizations.filesystem]]
mountpoint = "/home"
size = 1073741824

<< omitted other file system mount points >>

[[packages]]
name = "audit"
version = "*"

<< omitted other packages required for the hardening process >>

[customizations.kernel]
append = "audit_backlog_limit=8192 audit=1"

[customizations.services]
enabled = ["crond","firewalld","systemd-journald","rsyslog","auditd"]
disabled = ["nfs-server","rpcbind","nftables"]
```

Listing 3.1: Image Builder blueprint.toml file snippet

However, the whole integration is still in development which results in some customizations being left out. These customizations had to be added manually, in order to meet the rules within the CIS security profile. The security profile requires the *firewalld* systemd service to be running, however, none of the package-adding parts adds the firewalld package (which was later added to the upstream Git repository in the form of Pull Request[1]). Also, the generated output does not add the main customization part which specifies the OpenSCAP hardening process. Lastly, the tailoring rule had to be added manually as well, because it is specific for this project. The whole manually added part is shown in the code snippet Listing 3.2:

```
[[packages]]
name = "firewalld"
version = "*"

[customizations.openscap]
datastream = "/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml"
profile_id="xccdf_org.ssgproject.content_profile_cis"

[customizations.openscap.tailoring]
unselected=[ "xccdf_org.ssgproject.
           content_rule_sshd_disable_root_login" ]
```

Listing 3.2: Manually added part of the Image Builder blueprint.toml file

There were also parts of the generated blueprint that were not supported by the Image Builder OSBuild service. One of the issues was also reported to the Image Builder project Jira issues[2].

[1] https://github.com/ComplianceAsCode/content/pull/11351
[2] https://issues.redhat.com/browse/RHEL-20021

However, until it is fixed by the Image Builder team, it has to be manually removed, so the image-building process does not fail. The exact lines of the customization part, that had to be removed are listed in Listing 3.3:

```
1 [[customizations.filesystem]]
2 mountpoint = "/dev/shm"
3 size = 2147483648
```

Listing 3.3: Manually removed part of the Image Builder blueprint.toml file

Tailoring the Security Profile

As previously stated, implementing the OpenSCAP security profile tailoring was necessary for this thesis. The main reason was that the Testing Farm requires root access via a Secure Shell (SSH) connection. This requirement is based on the fact that Testing Farm uses cloud-based Amazon Machine Images that have to be accessed remotely. Testing Farm uses this remote access for the post-installation scripts that set the testing environment for the Quality Engineers (users of the Testing Farm).

Also as part of some test suites, root access is required, as only the superuser access can edit the critical system configurations. Disabling this rule through OpenSCAP tailoring is necessary for the execution of certain tests. Consequently, verification of some functional tests would be impossible without this action.

After applying the above-mentioned customizations and virtual image specifications, the OpenSCAP hardening evaluation[3] was conducted at the generated virtual image, and the results are displayed in Figure 3.1. The 16 failing rules are mostly related to the tailored root SSH access, or some beta features, that are in development. All of these failed rules were deeply analyzed and consulted with the RHEL Security Compliance team. The result was that none of these rules could impact the Quality Assurance process and it was safe to use it for testing.

[3] https://ljavorsk.fedorapeople.org/report.html

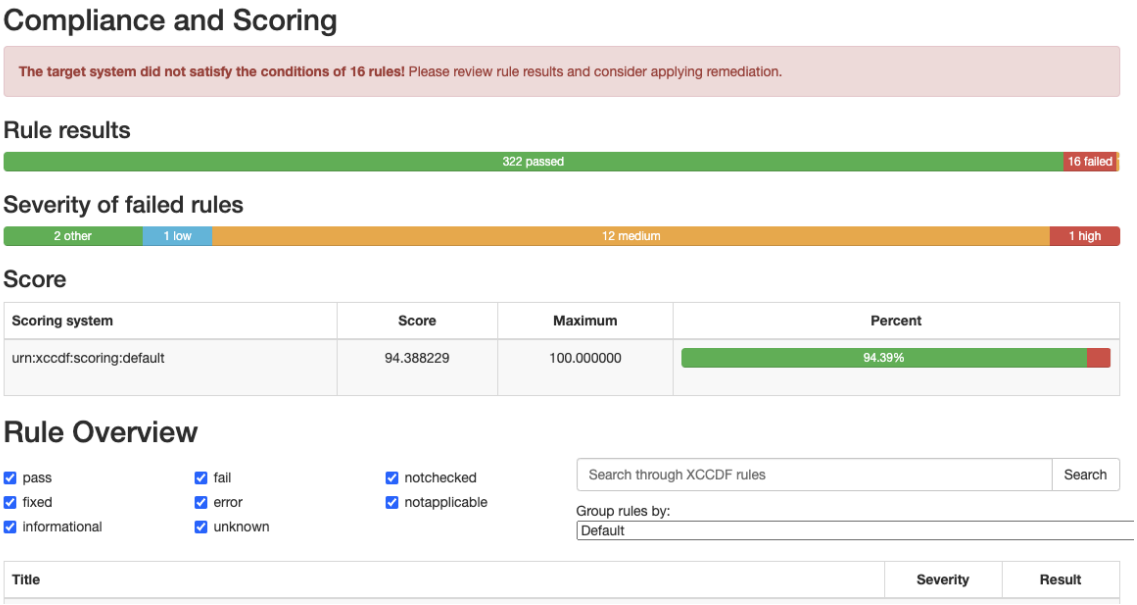

Figure 3.1: OpenSCAP evaluation report after security hardening with the CIS profile

3.1.2 Generating Testing Farm Specific Repositories

Another requirement from the Testing Farm to the virtual images was to add all of the RHEL repositories, which are enabled and ready to be used by the Quality Engineers. This was due to the fact that default RHEL repositories in the image require users to register and subscribe to the RHEL *subscription-manager*[4].

The Testing Farm team has developed a script that generates the Dandified yum (DNF) (more about DNF in Section 1.1.3) repositories for a specified RHEL compose[5] that consists of numerous repositories which contain multiple packages built in the Brew building system (more described in Section 1.1.2). These repositories are then used to install the packages that are in the testing phase and need to be properly tested by Quality Engineers.

To add those repositories to the generated image, these repositories have to be added to the *blueprint.toml* file in the form of the *[[customizations.repositories]]* list. Once the virtual image is loaded, the repositories added to this list will be available for the Quality Engineers in the */etc/yum.repos.d/* directory. In order to test their packages, users will install them using the *dnf install* command with the corresponding package names. DNF will then resolve the *baseurl* values from the repositories and install the packages from the corresponding compose. Such customization is shown in Listing 3.4 (values of the baseurl are obfuscated as they are internal URLs[6]):

[4] https://access.redhat.com/solutions/253273

[5] Composes are individual builds of a system image, based on a specific version of a particular RHEL distribution. Compose as a term refers to the system image, the logs from its creation, inputs, metadata, and the process itself.

[6] Uniform Resource Locator

```
[[customizations.repositories]]
id="rhel-BaseOS"
name="rhel-BaseOS"
filename="rhel-BaseOS"
baseurls=[ "http://some/url/of/RHEL/repository/" ]
enabled=true

[[customizations.repositories]]
id="rhel-AppStream"
name="rhel-AppStream"
filename="rhel-AppStream"
baseurls=[ "http://some/url/of/RHEL/repository/" ]
enabled=true
```

Listing 3.4: Image Builder blueprint repositories customizations required by the Testing Farm

The *[[customizations.repositories]]* list within the configuration file specifies a set of repositories for the operating system. Each entry in this list defines a single repository using four key properties: *id*, *name*, *filename*, and *baseurl*. All four of these properties are mandatory for a repository to be recognized and utilized within the DNF transaction.

In most cases, it is convenient to maintain consistency between the id, name, and filename properties. This means they can all share the same value. This simplifies configuration and aligns with the naming conventions typically used for repositories within RHEL-based distributions. By adopting this approach, it ensures a clear and straightforward mapping between the repository definition and its corresponding files and functionalities.

The above-mentioned lists of repositories in the blueprint file will result in the system repository file with the content detailed in Listing 3.5:

```
[rhel-BaseOS]
name=rhel-BaseOS
baseurl=http://some/url/of/RHEL/repository/
enabled=true

[rhel-AppStream]
name=rhel-AppStream
baseurl=http://some/url/of/RHEL/repository/
enabled=true
```

Listing 3.5: Virtual image custom system repositories

These repositories are intentionally missing the *gpgcheck* value. This is not mandatory and since the packages that are tested within this virtual image come from the trusted internal repositories it can be omitted. Disabling *GNU Privacy Guard* (GPG) key verification by omitting the gpgcheck value offers a potential performance boost within the testing pipeline. This is because DNF skips the additional step of verifying the authenticity of each downloaded package against its corresponding GPG key for every repository. This can lead to a *noticeable reduction in testing execution time*.

Automatic Integration to Image Builder Blueprints

To ensure a smooth handover to the RHEL Security Compliance team, this project should require minimal, ideally zero, manual operations. To help with that, several Python scripts were created as part of this thesis to automate these tasks. One of them is called *repo2blueprint.py* which serves to parse the *rhel.repo* file to the series of customizations lists in the final *blueprint.toml* file that will be served to the Image Builder OSBuild on-premise service.

This script goes through each repository entry, that starts with the [*<repository>*] line and parses its values to the TOML format (more about TOML in Section 1.6.1). Once this script parses all of the repositories and updates the blueprint file, it lets the user know with a short "Repos for blueprint generated successfully" logging message. The functions *parse_repo* and *generate_blueprint* are shown in the code snippet in Listing 3.6. These functions come with the comments and their functionality is self-explanatory from the function name.

```
def parse_repo(repo):
    repos = []
    temp_repo = {}
    for line in repo: # Parse the repository
        if line == "\n" and temp_repo != {}:
            repos.append(temp_repo) # Store repositories to the list
            temp_repo = {}

            continue
        else:
            if line.startswith('#'): # Skip comments
                continue
            elif line.startswith('['): # Parse the repository ID
                id = line.split('[')[1].split(']')[0]
                temp_repo['id'] = id
            elif line.startswith('name'): # Parse the repository attributes
                name = line.split('=')[1].strip()
                temp_repo['name'] = name
            elif line.startswith('url'):
                url = line.split('=')[1].strip()
                temp_repo['url'] = url
            elif line.startswith('baseurl'):
                baseurl = line.split('=')[1].strip()
                temp_repo['baseurl'] = baseurl
            else:
                continue
    return repos

def generate_blueprint(repos):
    with open('blueprint.toml', 'a') as file:
        file.write("\n\n")
        for repo in repos:
            try:
                file.write(f"[[customizations.repositories]]\
                \nid=\"{repo['id']}\"\nname=\"{repo['name']}\"\
                \nfilename=\"{repo['name']}\"\
                \nbaseurls=[ \"{repo['baseurl']}\" ]\
```

```
                \nenabled=true\n\n")
            except Exception as e:
                print(f"Error: Unable to generate blueprint - {e}")
```

Listing 3.6: Automation script repo2blueprint.py code snippet

3.1.3 Automating Blueprint Creation Based on Given Security Profile and Compose ID

The creation of the blueprint is separated into multiple steps as mentioned in Section 3.1.1. Some of them are necessary due to the fact that used features are still in development and they contain some bugs. These bugs were also reported to the upstream and once they are resolved the corresponding tweaks can be removed from the automated Python script.

The steps required for the blueprint generating process are as follows:

1. **Generating blueprint using oscap**: Using the oscap generate command, saves a lot of time with the customizations that are required by the security profile. Also, the name and description had to be edited, so that it is accurate and it refers to the security profile and compose ID used in the blueprint. To view the function that generates the blueprint, refer to Listing 3.7.

```
def generate_blueprint(RHEL_compose_id, security_profile):
    commands = [
        f"yumrepogen/yumrepogen-x86_64 -compose-id={RHEL_compose_id}",
        f"oscap xccdf generate fix --fix-type blueprint --profile {
    security_profile} /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml >
    blueprint.toml"]
    for command in commands:
        subprocess.run(command, shell=True)

    with open('blueprint.toml', 'r') as file:
        lines = file.readlines()
    to_write = []
    remove_after = False
    blueprint_name, blueprint_desc = False, False
    for i in range(len(lines)):
        if "name = " in lines[i] and not blueprint_name:
            blueprint_name = True
            to_write.append(f"name = \"{RHEL_compose_id}-{
    security_profile.upper()}-HARDENED\"\n")
            continue
        if "description = " in lines[i] and not blueprint_desc:
            blueprint_desc = True
            to_write.append(f"description = \"{security_profile.upper()}
    Security Hardened image based from {RHEL_compose_id} compose\"\n")
            continue
```

Listing 3.7: Generating blueprint and editing the metadata

2. **Remove the buggy parts of the pre-generated blueprint**: As the oscap generate feature is still under development, as well as the Image Builder project, sometimes bugs appear. In this case, some parts of the generated blueprint had to be removed as they caused failures within the image creation process. Also, the size of the partitions

had to be shrunk, so the virtual image did not take up too much space. Every part that can be removed after the upstream issues are fixed is marked with the *TODO* keyword, which stands out in any IDE[7]. The code snippet in Listing 3.8 shows how all of this is done in the automated Python script.

```
        # TODO: Can be removed once it is fixed in the oscap generate
        if "mountpoint = \"/dev/shm\"" in lines[i]:
            # Remove last item from to_write
            to_write.pop()
            remove_after = True
            continue
        if "size = " in lines[i]:
            # Change the size of the image as it makes the image too big
            size = int(lines[i].split("=")[1].strip())
            to_write.append(f"size = {str(size/8).split('.')[0]}\n")
            continue
        else:
            if remove_after:
                remove_after = False
                continue
            else:
                to_write.append(lines[i])

    # TODO: Can be removed once it is added to the oscap generate
    customizations_to_add = [
        "[[packages]]\n",
        "name = \"firewalld\"\n",
        "version = \"*\"\n\n",
        "[customizations.openscap]\n",
        "datastream = \"/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
\"\n",
        "profile_id=\"xccdf_org.ssgproject.content_profile_cis\"\n\n",
        "[customizations.openscap.tailoring]\n",
        "unselected=[ \"xccdf_org.ssgproject.
content_rule_sshd_disable_root_login\" ]\n"]

```

Listing 3.8: Automation script repo2blueprint.py code snippet

3. **Integrating the custom repositories from specified compose ID**: This step is required by the Testing Farm, which enables Quality Engineers to access the testing packages that are not shipped to the customers yet.

Setting up the Building Environment

Due to the fact that the only solution available to use (that was analyzed in Section 2.1) is to use the on-premise OSBuild Composer service (detailed in Section 1.6.2) from the Image Builder team, it requires to be run on a RHEL working machine to generate a RHEL image.

Red Hat engineers have a distinct advantage when it comes to building software on RHEL machines. Their access to pre-configured internal machines streamlines the process significantly. However, as the fresh RHEL machines do not come with the pre-installed packages required for the building process, they need to be installed after the machine is

[7]Integrated Development Environment

provided. Fortunately, this preparation step can be easily automated, so it is convenient for the engineers, and also it prevents human error.

The required packages for the building process can be easily installed by executing the following bash command:

```
sudo dnf install -y openscap-scanner scap-security-guide \
osbuild-composer composer-cli
```

To again speed up the preparation time and ease the overload on the users, the function within the automated Python script, that takes care of the whole process was created. This function is being designed with the future in mind, specifically for the RHEL Security Compliance team. As outlined in Section 3.4, this team will take over the project's development, where they can leverage this script as a foundation. The script's modular design and clear documentation will allow them to easily analyze potential improvements and make further customizations to tailor it to their specific security compliance needs.

Adding Specific Payload Repositories for the Image

In order to create an image that will contain packages (kernel and buildroot included) from a specific compose, it is important to add third-party payload repositories (to learn about payload repositories refer to Section 1.6.4) to the Composer service. OSBuild Composer does allow its users to add these repositories into their specified */etc/osbuild-composer/repositories* path[8]. These repositories are then used in the image-building process, where the buildroot or user-specified packages are installed directly from them. This step is critical within this thesis, as the RHEL virtual images will have to be created from a specific compose ID.

3.1.4 Creating VM Images via OSBuild Composer

To create a virtual image, the OSBuild Composer service requires the blueprint.toml file, unlike the JSON structured query within the HTTP API calls to the hosted Image Builder service. The creation of the blueprint is thoroughly described in Section 3.1.1.

After the RHEL building environment is all set up and the OSBuild Composer service is running, the virtual image creation is divided into two steps.

The first step is to *push the blueprint.toml* file to the blueprints database of the OSBuild Composer. In this step, the basic syntax is checked, and if the blueprint does contain some errors, the OSBuild Composer will return an error message and the push action will not execute. Once the blueprint has its syntax checked, and everything is correct, it gets pushed to the database, which can be later listed using *composer-cli blueprints list* command. The name of the blueprint is inherited from the *name* value specified within the blueprint.toml file itself. In order to perform any action on the blueprint, it must be identified by this name.

The second step is to start the image-building process itself. The prerequisite for the building process is to have a blueprint pushed to the composer blueprints database. All of the image operations can be accessed using the *composer-cli compose* command. To build the actual image the *composer-cli compose start <blueprint>* command can be used. This will push the image-building process into action (status with message *RUNNING*), and if there are no semantic errors within the blueprint itself, the image will be successfully built. If there are some semantic errors within the blueprint, the image-building process will end

[8] https://osbuild.org/docs/on-premises/installation/managing-repositories

with the *FAILED* status result. If the blueprint does not have any semantic issues, the status will result in message *FINISHED* and the image can be downloaded.

The list of blueprints pushed to the composer service and the list of the composer image-building processes are displayed in Figure 3.2. In the compose list, the output information also gives the user a unique *image ID*, current *status*, *blueprint* used for the image definition, *version* specified in the blueprint file and the *image type*.

```
[root@vm-10-0-186-115 ~]# composer-cli blueprints list
RHEL-9.3.0-updates-20240104.37-CIS-HARDENED
[root@vm-10-0-186-115 ~]# composer-cli compose list
ID                                   Status    Blueprint
              Version   Type
93baa8b8-fd63-44bb-82c6-099a5853b855   RUNNING   RHEL-9.3.0-updates-20240104.37-
CIS-HARDENED   0.1.69    qcow2
[root@vm-10-0-186-115 ~]#
```

Figure 3.2: Checking pushed blueprint and started image creation

Once the virtual image-building process is finished (status *FINISHED*), the image can be downloaded. To download a virtual image from the Composer service, the *composer-cli compose image <ID> --filename <image_file_name>* can be used. This command will download the specified virtual image and store it with a custom-defined name provided within the *–filename* option. Example of an image download action can be viewed in Figure 3.3.

```
[root@vm-10-0-186-149 ~]# composer-cli compose image 28e0f4e5-f163-4a9a-902f-7f2916ddd942 --filena
me RHEL-9.3.0-updates-20240104.37-CIS-HARDENED.qcow2
RHEL-9.3.0-updates-20240104.37-CIS-HARDENED.qcow2
[root@vm-10-0-186-149 ~]# ls -lah RHEL-9.3.0-updates-20240104.37-CIS-HARDENED.qcow2
-rw-------. 1 root root 850M Mar 15 13:25 RHEL-9.3.0-updates-20240104.37-CIS-HARDENED.qcow2
```

Figure 3.3: Downloading the generated pre-hardened virtual image

While the downloaded virtual image file has a size of *850MB*, the actual space it occupies will be much larger after deployment and filesystem configuration (as defined in the blueprint). Downloaded image now meets the agreed naming convention and can therefore be transferred to the Testing Farm infrastructure. Section 3.1.6 describes the process of deploying this pre-hardened virtual image within the Testing Farm machinery.

3.1.5 Automating the Image Building Process

The process outlined from Section 3.1.1 to Section 3.1.4 is entirely automatable. Since the RHEL Security Compliance team will take this project after the thesis is finished, the best option for them was to have one simple and well-documented script, that will do all the work for them (after providing some arguments to it). Python script was chosen for this task, as it is easy to create custom modules/libraries (which will be needed for the Testing Farm specific tweaks module) and it also handles text parsing very well.

The main script is called *generate_hardened_image.py* and the Testing Farm tweak module is called *testing_farm_tweaks.py*. They are both compressed together with the Testing Farm developed *yumrepogen* binary and *payload-rhel.json* template into one tarball called *generate_hardened_image.tar.gz*. Most of the functions were already mentioned in previous Listings and the main function from the *generate_hardened_image.py* is shown in Listing 3.9.

```
if __name__ == "__main__":
    # Create the parser
    parser = argparse.ArgumentParser(description="Generate the repositories")
    # Parse arguments
    parser.add_argument("-r", "--RHEL-version", help="The RHEL version from
    which you will build your image. Example: 9.3", required=True)
    parser.add_argument("-c", "--RHEL-compose-id", help="The RHEL compose ID
    to generate repositories from. Example: RHEL-9.3.0-updates-20240104.37",
    required=True)
    parser.add_argument("-s", "--security-profile", help="The OpenSCAP
    security profile to use. Example: cis", required=False, default="cis")
    args = parser.parse_args()
    prepare_machine()
    generate_blueprint(args.RHEL_compose_id, args.security_profile)
    repos = parse_repo()
    add_to_blueprint(repos)
    create_payload_repository(args.RHEL_version)
    start_composer_service()
    push_blueprint_to_composer()
    start_image_building_process(args.RHEL_compose_id, args.security_profile)
```

Listing 3.9: Main function from generate_hardened_image.py

Creating the Testing Farm Specific Tweaks Module

Because of the fact that Testing Farm is also a cutting-edge technology, it is still under development and its internals change frequently. As there are few tweaks that are essential to the Testing Farm they may also change with time, so they need to be kept in a separate Python module (file).

To achieve this a new *testing_farm_tweaks.py* Python module had to be created. This file contains all of the Testing Farm specific tweaks to the blueprint file. These can be easily changed in the module in case of changes within the Testing Farm structure, without the need for tempering with the main *generate_hardened_image.py* script.

To import all of the functions from this module a simple import line was added to the main script:

```
from testing_farm_tweaks import parse_repo, add_to_blueprint,\
shrink_filesystem_sizes, create_payload_repository
```

This import ensures that all of the listed functions are defined and implemented in the module and they should be edited there as well.

Executing the Main Script to Generate the Image

As mentioned above, both scripts, *yumrepogen* tool, and *payload-rhel.json* template are compressed into one tarball called *generate_hardened_image.tar.gz*.

This whole tarball needs to be moved to the RHEL building machine that serves to build the image with the installed on-premise OSBuild Composer service. Upon untarring[9] the tarball, the main script can be invoked with the necessary parameters. This will trigger the execution of all prepare actions and initiate the pre-hardened image-building process. Execution of the main script on the fresh RHEL machine can be viewed in Figure 3.4 (part of this script contains internal domains, that had to be blured in this image).

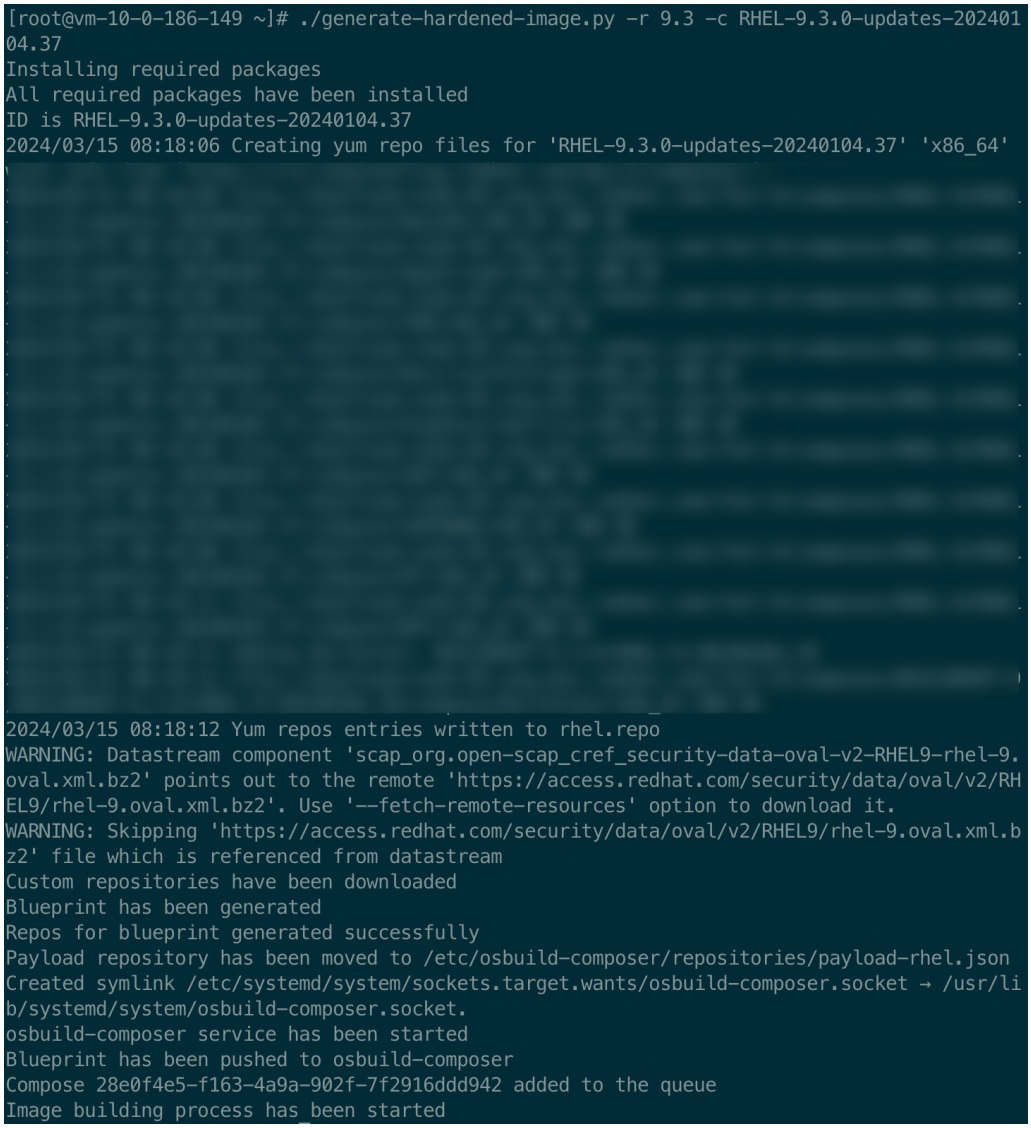

Figure 3.4: Execution of the main generate_hardened_image.py script

This script was called with the arguments *-r 9.3* indicating that the desired image should use the RHEL-9.3 operating system, and the *-c RHEL-9.3.0-updates-20240104.37*, which defines the compose ID that will be used for both payload repositories (described in Section 1.6.4) and installed repositories on the resulting virtual image.

The main script also comes with a user-friendly *--help* argument that helps users understand the arguments required by the script. This help script can be viewed in Figure 3.5.

[9]Similar to unzipping when the file is a zip file

```
❯ ./generate_hardened_image.py --help
usage: generate_hardened_image.py [-h] -r RHEL_VERSION -c RHEL_COMPOSE_ID [-s SECURITY_PROFILE]

Generate the repositories

optional arguments:
  -h, --help            show this help message and exit
  -r RHEL_VERSION, --RHEL-version RHEL_VERSION
                        The RHEL version from which you'll build your image. Example: 9.3
  -c RHEL_COMPOSE_ID, --RHEL-compose-id RHEL_COMPOSE_ID
                        The RHEL compose ID to generate repositories from. Example: RHEL-9.3.0-updates-20240104.37
  -s SECURITY_PROFILE, --security-profile SECURITY_PROFILE
                        The OpenSCAP security profile to use. Example: cis
```

Figure 3.5: Help script provided to understand the required arguments

3.1.6 Integration into the Testing Farm Infrastructure

The final stage of the pre-hardened virtual image supply chain involves its integration into the Testing Farm infrastructure (detailed in Section 1.2.4). This would allow Quality Engineers to easily provision these images for functional testing purposes. As the Testing Farm is a well-established environment for testing within the Red Hat CI/CD pipeline (more described in Section 1.2.3), it does not require any further knowledge, since it is already used widely within testing processes.

Naming Convention

Agreement on a naming convention for the pre-hardened virtual images was established as a prerequisite for their integration. This was done in two iterations as each one of them brought some insight into the problematic naming (examples are used for *RHEL-9.3.0* version and *CIS security profile*):

1. **HARDENED-CIS-RHEL-93**: The problem with this naming was that it did not include information about compose used in the image. This was causing a problem within the automation in Quality Engineering teams that rely on the compose ID within the name of the image. Also, it was not self-explanatory for the users, which packages could have been installed in this image, as the repositories are defined in the compose itself.

2. **RHEL-9.3.0-updates-20240104.37-CIS-HARDENED**: Following the identification of issues during the first iteration, this naming convention was ultimately selected. The compose ID was moved to the beginning of the name to adhere to the standards and the security profile was moved to the end of the image name.

Allowlisting Pre-hardened Images

Once the naming was defined and agreed upon among all the teams involved in this project, it had to be allowlisted in the Testing Farm infrastructure. This was achieved with a series of Merge Requests (MRs) in the Gitlab projects owned by the Testing Farm team. To enable functionality for the specific guest-setup tool, a patch[10] was added to its enabling mechanism. Composer-generated images were further enabled through the implementation of another patch[11].

[10] https://gitlab.com/testing-farm/infrastructure/-/commit/b89c0d9f63ce0e751a7119841c1a
[11] https://gitlab.com/testing-farm/infrastructure/-/merge_requests/425

3.2 Testing Process on the Pre-Hardened Images

This section offers a thorough summary of the testing procedure used on security pre-hardened virtual images. Everything from the provisioning of these images from the infrastructure of the Testing Farm to the outcomes of the real functional testing on particular packages.

Section 3.2.1 describes how the Quality Engineers can easily get the pre-hardened machine ready for testing. Provisioning from Testing Farm includes a post-installation script that is necessary to grant root access for tests requiring superuser privileges. It also covers the basics of how to install and use the Testing Farm command line tool (which is more covered in Section 1.2.4).

Upon successful deployment of the pre-hardened virtual machine, it will undergo an evaluation for security compliance with the CIS security profile (detailed in Section 1.3.7). Section 3.2.2 will go into great detail about the nuances of extracting security compliance results from the OpenSCAP CLI tool, which was also presented in Section 1.3.7. In order to help users gain valuable insights into a system's compliance with a specified security profile, the following section will walk them through the process of utilizing the OpenSCAP command-line interface.

To wrap it all up, Section 3.2.3 will showcase the possibilities for functional testing within the provisioned pre-hardened virtual machine. This section will deep dive into the most critical aspect of the Quality Engineers' workflow, as they are the primary beneficiaries of this functional testing methodology.

3.2.1 Provisioning the Pre-Hardened Images via Testing Farm

There are several methods available for provisioning an image from Testing Farm infrastructure. However, under the hood, all of them use the standard API calls defined by the Testing Farm upstream API documentation[12].

The approach used in the following example makes use of the CLI tool provided by the Testing Farm team itself, which is detailed in Section 1.2.4. To provision the previously created security-hardened virtual image, users need to call this command:

```
testing-farm reserve --compose RHEL-9.3.0-updates-20240104.37-CIS-HARDENED\
--post-install-script="$(curl https://gitlab.com/testing-farm/\
infrastructure/-/raw/testing-farm/ranch/public/citool-config/\
post-install-scripts/enable-root.sh?ref_type=heads)"
```

This command also specifies the *post-install script*[13] which creates SSH keys for the root SSH connection, that is highly used during the functional testing. Since the root SSH connection was not blocked by the hardening process (due to the tailoring customization described in Section 3.1.1), the post-install script can easily temper with the SSH configuration to enable these keys.

Upon completion of the provisioning process initiated by this command, a root SSH connection to the pre-hardened virtual machine becomes available to the user. Figure 3.6 shows how this connection looks from the user's point of view in the terminal.

[12] https://api.testing-farm.io/

[13] https://gitlab.com/testing-farm/infrastructure/-/tree/testing-farm/ranch/public/citool-config/post-install-scripts

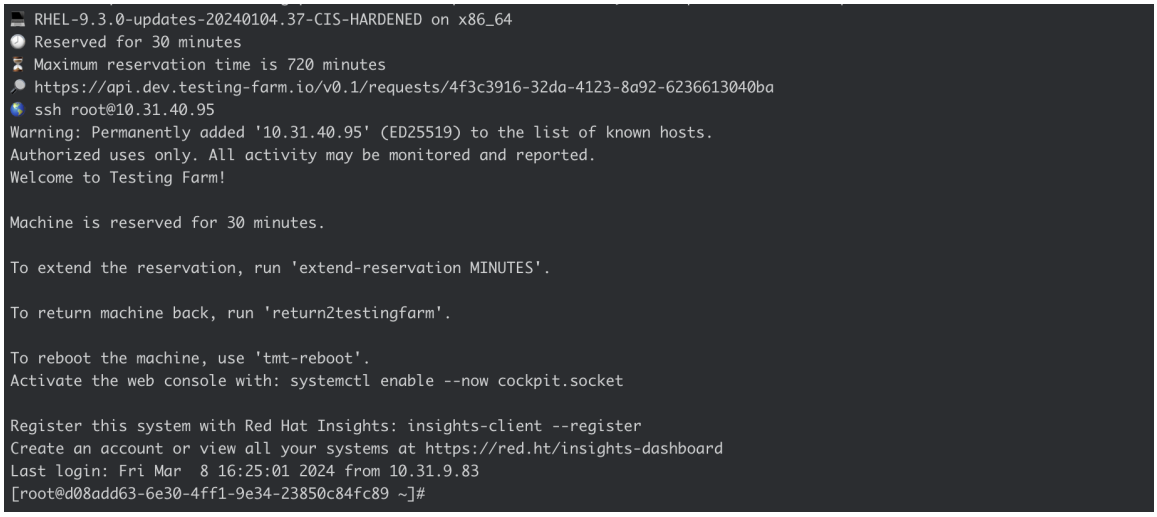

Figure 3.6: Provisioning of the pre-hardened RHEL virtual image hosted in Testing Farm

3.2.2 Verifying the Pilot Project Security Compliance Results

Testing the security compliance of the chosen security profile (CIS in this case) was necessary following the successful provisioning of the newly created pre-hardened virtual machine. In order to evaluate the overall system compliance the *oscap CLI tool* can be used. In this specific example with CIS security profile, the following command was executed, to evaluate the compliance:

```
oscap xccdf eval --profile cis --results /tmp/results.xml\
--report /tmp/report.html\
/usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml
```

To understand this command more, refer to Section 1.3.7. This command generates a human-readable HTML report named *report.html* which can be opened and viewed with any web browser. The specific part for this command is the *--profile* option, which specifies the CIS security profile used for the hardening of the image. The second argument of this command specifies the *datastream* (further detailed in Section 1.3.7) used for the chosen operating system and the OpenSCAP security profiles.

Report from this particular pre-hardened virtual machine was already shown in Figure 3.1. As previously stated, every test that failed was examined, and since the tailoring and other Testing Farm-specific configurations were the reason for the individual failures, none of them were critical for the hardening process.

3.2.3 Testing Possibilities

Following verification of security compliance and confirmation of the virtual machine's proper operation, this section will outline the testing options available for images provided by the Testing Farm. There are multiple possibilities for how the Testing Farm virtual machines can be used for testing purposes (most common ones are described in Section 1.2.4).

Example of Testing Zlib Regression Test on Pre-Hardened Image

The easiest way to functional test a package is to use tests[14] written by Red Hat Quality Engineers. These tests are designed for the Test Management Tool (TMT), which is used within Red Hat CI/CD pipelines. One example testing case of a simple regression zlib test[15] is shown in Figure 3.7.

```
::   Setup
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:: [ 11:57:26 ] :: [   PASS   ] :: Command 'yum remove minizip-devel -y' (Expected 0,1,
got 0)
:: [ 11:57:26 ] :: [   PASS   ] :: Creating tmp directory (Expected 0, got 0)
:: [ 11:57:26 ] :: [   PASS   ] :: Command 'tar -C /tmp/tmp.WPtKpQdaU4 -xzf minizip.tar.
gz' (Expected 0, got 0)
:: [ 11:57:26 ] :: [   PASS   ] :: Command 'pushd /tmp/tmp.WPtKpQdaU4' (Expected 0, got
0)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Duration: 1s
::   Assertions: 4 good, 0 bad
::   RESULT: PASS (Setup)


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Test
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Duration: 0s
::   Assertions: 0 good, 0 bad
::   RESULT: PASS (Test)


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Cleanup
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:: [ 11:57:26 ] :: [   PASS   ] :: Command 'popd' (Expected 0, got 0)
:: [ 11:57:26 ] :: [   PASS   ] :: Removing tmp directory (Expected 0, got 0)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Duration: 0s
::   Assertions: 2 good, 0 bad
::   RESULT: PASS (Cleanup)


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   unknown
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:: [ 11:57:26 ] :: [   LOG    ] :: JOURNAL XML: /var/tmp/beakerlib-T6PqeSl/journal.xml
:: [ 11:57:26 ] :: [   LOG    ] :: JOURNAL TXT: /var/tmp/beakerlib-T6PqeSl/journal.txt
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::   Duration: 1s
::   Phases: 3 good, 0 bad
::   OVERALL RESULT: PASS (unknown)

[root@5e2f692f-53fa-4d7e-a636-b74571979a31 bz690835-Please-enable-minizip-in-zlib-spec-f
ile]#
```

Figure 3.7: Testing case of a zlib regression test on pre-hardened image

3.3 Economic Evaluation

This section will address the economic as well as the time-based evaluation for this thesis implementation. Specifically, Section 3.3.1 explains the *Return on Security Investment (ROSI)* ratio of the monetary benefits of a security investment to its cost. It will contain all

[14] https://gitlab.com/redhat/centos-stream/tests/

[15] https://gitlab.com/redhat/centos-stream/tests/zlib/-/tree/main/Regression/bz690835-Please-enable-minizip-in-zlib-spec-file?ref_type=heads

of the investments from the people's perspective as well as from the hardware maintenance investment.

Section 3.3.2 will cover how the barriers of entry for Quality Engineers (QEs) were eliminated by implementing this solution. It will also cover approximately how much time it saves to have the virtual machines pre-hardened as the hardening process does take quite a significant time. On the other hand, it will also cover how much time it will cost the RHEL Security Compliance team to maintain this solution.

Lastly, Section 3.3.3 will cover the real customer case that could have been prevented by this solution in Red Hat's pipeline. The costs of the employees and procedures that were spent in order to resolve and correct this case were also assessed, and the section includes obfuscated but comparable costs.

3.3.1 Return on Security Investment (ROSI)

The economic evaluation of such an investment can be addressed through the application of *Return on Security Investment* (ROSI). It is calculated to measure the financial benefit of cybersecurity initiatives. It compares the cost of security measures with the potential financial losses prevented by those measures. This allows organizations to assess the value of their cybersecurity investments. The mathematical equation for the ROSI is shown in Formula 3.1.

$$ROSI = \frac{Monetary\ loss\ reduction - Cost\ of\ the\ solution}{Cost\ of\ the\ solution} \tag{3.1}$$

The individual parts of the ROSI formula are:

- **Monetary loss reduction**: represents the total expected financial loss from security incidents per year. It is estimated by multiplying the likelihood (annual rate of occurrence) of an incident by the average cost (single loss expectancy) of a single incident.
- **Cost of the solution**: includes all expenses associated with implementing and maintaining the security measures, such as software licenses, hardware upgrades, and employee training.

Using the data from customer cases and the existing Jira reports that are related to Security Compliance testing, the following calculation presented in Equation 3.2 was made. The original numbers had to be obfuscated so the result would not leak any sensitive information. The *Cost of the solution* was calculated from the allocated Mandays (MDs) for the solution analysis and implementation and the maintenance cost for the implemented solution.

$$ROSI = \frac{1\ 040\ 500 - 155\ 000}{155\ 000} \approx 5.713 \tag{3.2}$$

The result of *5.713* indicates that the security investment is cost-effective, as the reduction in monetary losses outweighs the solution's cost. This means that for every dollar spent invested into the Quality Assurance of RHEL Security Hardened systems, the organization can expect $5.713 in financial benefits. Inspiration for the numbers used in the equation is partly based on the customer case later described in Section 3.3.3 and partly on the analysis from Red Hat's internal sources.

3.3.2 Time-Based Evaluation

In addition to other evaluation methods, a time-based assessment can be conducted to estimate the overall impact of this project implementation on the Quality Engineering (QE) team. This metric passively calculates the net time saved by the QEs. *Positive* values represent time reductions achieved through the final implementation, such as the elimination of tasks like security hardening the virtual testing machines. Vice versa, *negative* values reflect the time invested into analyzing, implementing, and maintaining the newly created solution of pre-hardened virtual machines provisioned using Testing Farm infrastructure. This time-based evaluation provides valuable insights into the overall efficiency gains or losses brought by the implementation. Specifically, Table 3.1 shows how many MDs were spent on the analysis and implementation.

Table 3.1: Initial time-based evaluation of the implemented solution

Type of action	**Spent MDs**
Analysis	6
Meetings and Discussions	4
Implementation	24
Fine-Tunning	3
Grand Total	37

Table 3.2 shows how many MDs are saved each quarter with the implemented solution. The individual actions are eliminated as the pre-hardened virtual machines do not require any action from Quality Engineers. Also, the maintenance cost is integrated into these data (red negative number), as it must be calculated as well. Again, the data are obfuscated for privacy reasons.

Table 3.2: Quaterly time-based evaluation of the implemented solution

Type of action	**Saved MDs**
Maintenance	-4
Training/Education in OpenSCAP hardening	12
Security hardening in testing machines	4
Support from the RHEL Security Compliance team	4
Grand Total	16

It is clear from the above Tables that after three quarters, the MDs that will be saved begin to exceed the initial implementation costs. Thus, starting with the fourth quarter, roughly 18 MDs of the Quality Engineers' capacity are saved every other quarter. Nevertheless, these data were collected on the small sample of the Quality Engineers that were selected for the pilot project in Section 2.3.1. The rate at which cybersecurity awareness is growing will probably result in higher savings for quality engineering teams.

3.3.3 Case 1 Example

Unfortunately, one customer case already happened before the solution could have been fully implemented and turned over to the RHEL Security Compliance team. The customer

issue is fully described in the reported Jira issue[16]. The problem was discovered on the *systemd* package, which is tempered with during the security hardening process.

Several more problems that were connected to the primary one were brought about by this customer case. Since the problem affected such a widely used package, it was reported by several customers. Before the cause of the problem was determined, the impacted team had to conduct a thorough analysis of the root cause of this issue.

The below statistics, displayed in Table 3.3, are based off of Salesforce Case information as linked to the primary Knowledge-Centered Support (KCS)[17]

Table 3.3: Number of related KCS and Jira issues linked to the customer case

Issue Link Type	**Count**
Jira issue	40
KCS case	120

The reason behind such a large number of linked issues is that the teams traditionally link these issues together so the original case is not lost and thus it may look like there were a lot of issues, but it is not that many.

Severity and Count of the Customer Cases

Figure 3.8 displays how the severity is distributed between the filled issues. Looking at the distribution of cases, the majority of them skew towards Severity 1 (*Urgent*) and 2 (*High*). This is understandable as the particular regression results in a system failing to reboot following a routine update, thereby halting production.

Additionally, this signified that there was a sense of urgency injected within the teams and a collective push for resolution and support toward addressing the issue at hand. Red Hat's Severity guidelines[18] also describes the individual Severity numbers in detail. Regarding the individual severities, Red Hat engineers are tied to the offered Production Support Terms of Service[19] also known as *Service Level Agreements* (SLAs).

[16] https://issues.redhat.com/browse/RHEL-17164
[17] https://access.redhat.com/solutions/7045909
[18] https://access.redhat.com/support/policy/severity
[19] https://access.redhat.com/support/offerings/production/sla

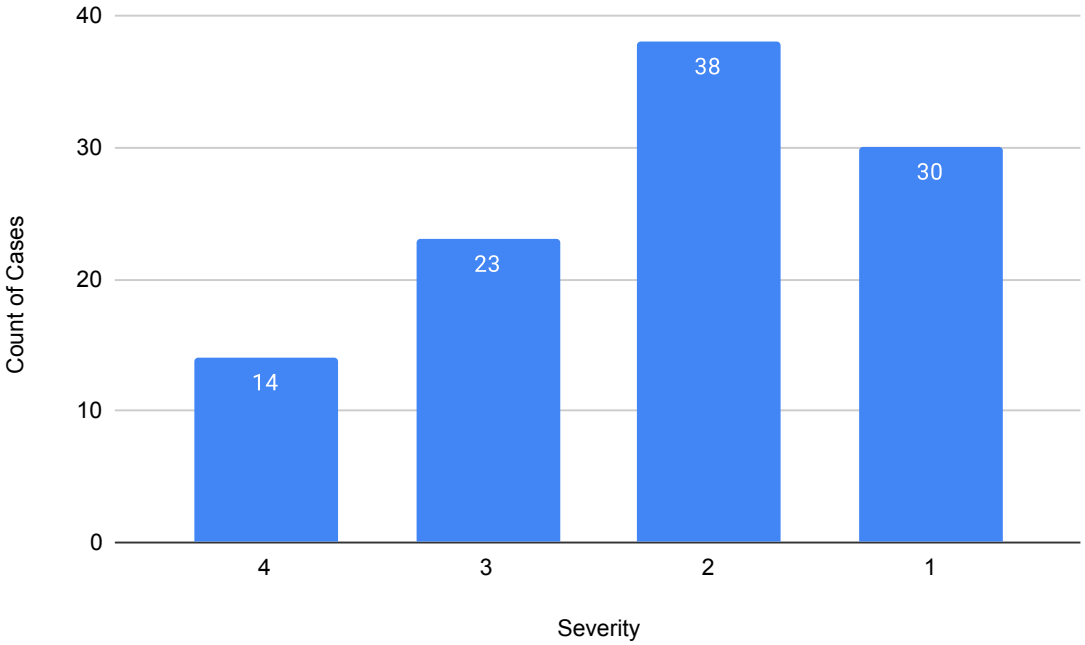

Figure 3.8: Severity of the individual reported issues and their count

Volume of Individual Reported Cases Over Time

The next graph displayed in Figure 3.9 shows the impact of the regression caused by the increasing volume of the reported issues. The impact is immediately seen with a large spike in case volume. However, the effects are seen well after the resolution is available. The time when the fix was deployed to production is shown by the red dashed line.

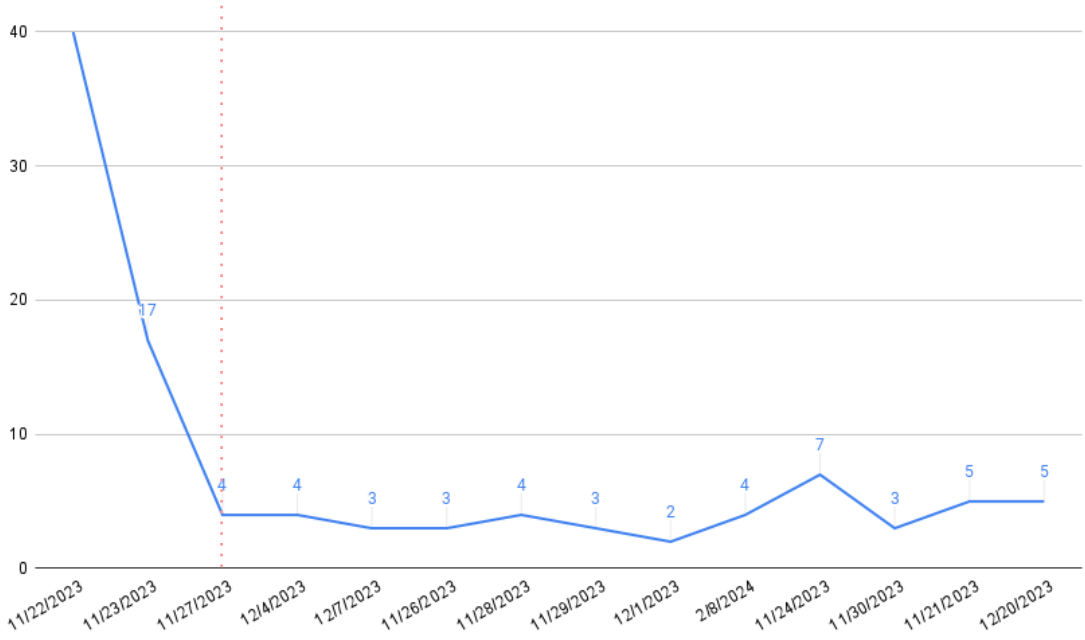

Figure 3.9: Graph depicting the volume of reported issues over time

The reasoning behind these numbers can be influenced by a number of factors. For example, some installations are behind distribution networks that could pick up the initial regression immediately, and not capture the subsequent fix for some time due to unfortunate timing. Or, end users also tend to practice poor "reboot discipline" following updates. It can be presumed that some of the delays in case creation are due to the systems being upgraded to include the regressed systemd build, but had not yet seen the failure state as the systems had not yet been rebooted.

Time to Closure/Resolution

The graph shown in Figure 3.10 displays the number of days required to resolve an individual customer case. The X-axis represents the number of days the case remained open and the Y-axis represents the total number of cases within that interval.

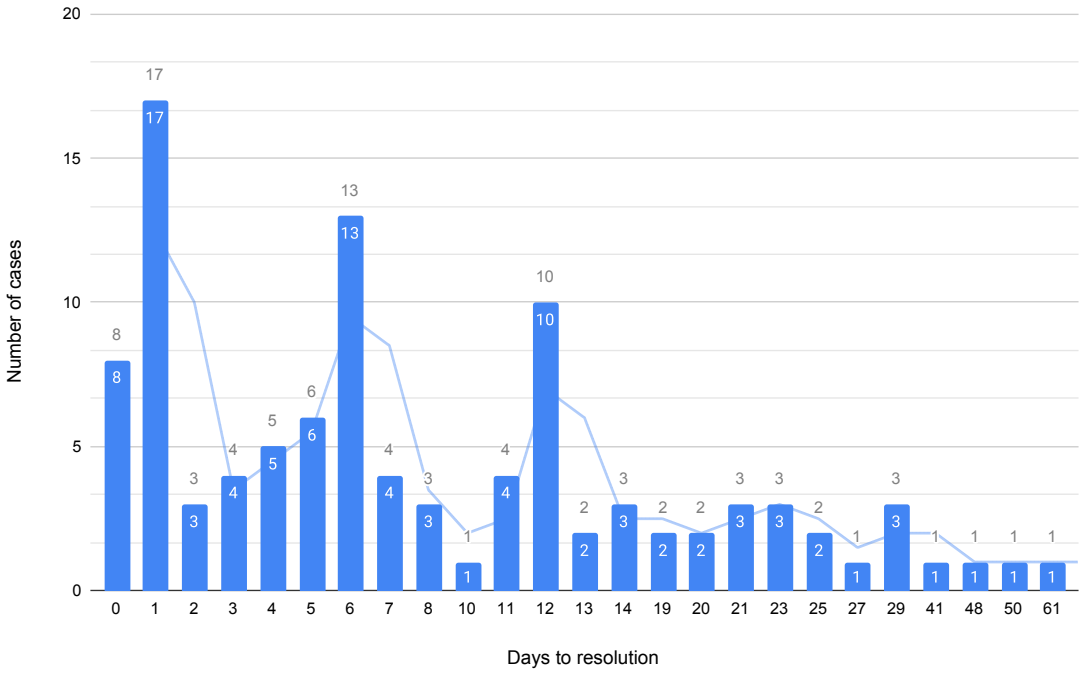

Figure 3.10: Time to closure graph for customer cases related to the regression issue

The majority of cases were resolved within the first one to two weeks. Yet, the distribution of case duration does indicate the general difficulty of bringing attention to an issue and figuring out it is linked to the resolution which was pushed to production within two days.

Case Complexities

Case complexity is predicated/measured with the assumption shown in Equation 3.3.

$$Comments = Complexity,\ Complexity = Effort,\ Effort = Cost \tag{3.3}$$

When considering the impact on the customer from a complexity perspective, a surprisingly

high proportion of problems still need significant work to fix. Figure 3.11 shows how the case complexity was distributed within the filled cases.

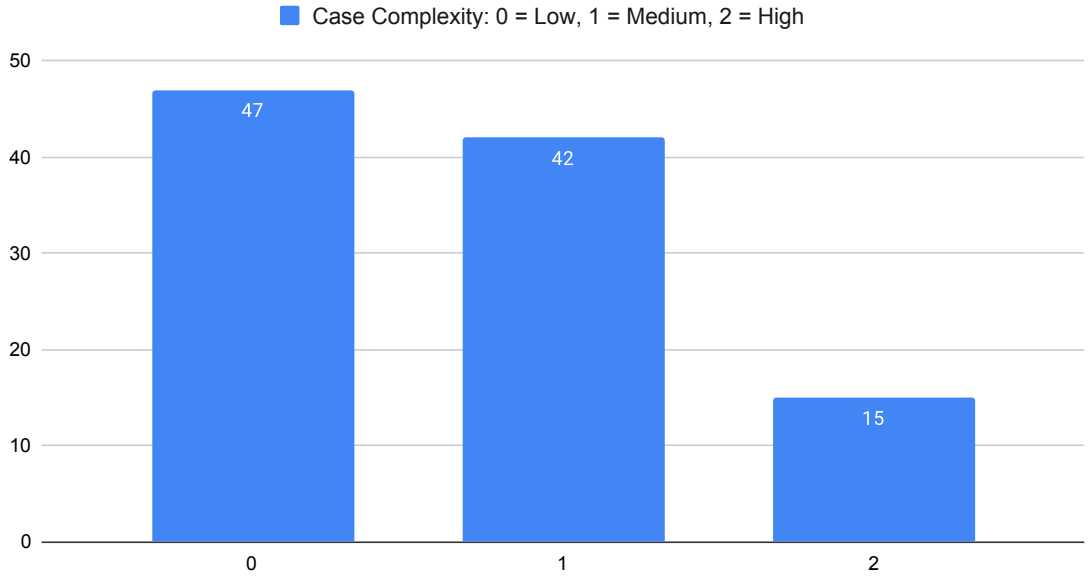

Figure 3.11: Customer case complexity perspective

Ideally, the entire case population would have been *Low* complexity. However, there is a reasonably large burden that the Support Teams bear in guiding customers through the troubleshooting and resolution process.

Cost Analysis

Since the evaluation of the cost for each case is a difficult task, and also the data had to be obfuscated for obvious reasons, these estimates are only exemplary. Table 3.4 shows these obfuscated numbers of cases and their average costs based on the *support costs* and *engineering costs*. The total estimated cost to support delivery for this particular issue is shown in Table 3.5.

Table 3.4: Summary of Reported Customer Cases for this Issue

Month	Complexity	Cases	Sum of Comments	Average Comments	Average Case Cost
Dec 2023	0	5	15	3	$128.52
	1	10	30	3	$346.95
	2	5	60	12	$1,000.15
Nov 2023	0	40	120	3	$220.13
	1	30	180	6	$300.32
	2	14	120	8.6	$1,329.15
Grand Total		**104**	**525**	**5.9**	**$554.20**

Table 3.5: Total Estimated Cost to Support Delivery

Month	Complexity	Cases	Average Case Cost	Total Case Cost
Dec 2023	0	5	$128.52	$642.60
	1	10	$346.95	$3,469.50
	2	5	$1,000.15	$5,000.75
Nov 2023	0	40	$220.13	$8,805.20
	1	30	$300.32	$9,009.60
	2	14	$1,329.15	$18,608.10
Grand Total		104	$554.20	**$45,535.75**

Note that the above totals are a very rough (also obfuscated) estimate and subject to a large number of confounding variables and will underrepresent the cost in a number of ways. However, some cases from the overall population were mis-identified as various different products and versions were affected by this issue as well. For example, a Red Hat Satellite system failing to boot would be represented as a "Satellite" installation issue, though the failure exists in the underlying Red Hat Enterprise Linux installation base.

3.4 Future Development

The project's future trajectory beyond the proof-of-concept stage is described in this section. Although the project has successfully shown that it is feasible, it is important to remember that the tools and infrastructure that are being used are still in the stages of development for this particular purpose. Therefore, it is essential to complete the ongoing development of these foundational elements before full implementation can be realized.

Use of the API Staging Service for Easier Scalability

As mentioned earlier in Section 2.1.4 , the current solution utilizes the on-premise OSBuild Composer service. This solution is fully featured, however it also lacks the easily accessible interface. Image Builder team also provides the hosted service (described in Section 2.1.3), unfortunately it lacks the newest features due to the fact that this service needs to go through a lot of testing and stabilization phases which delays the new features.

Nevertheless, the new features requests like the OpenSCAP autotailor[20], allowing users to customize the AMI (described in Section 1.5.3) name[21], or the requested feature to support a specific compose ID payload repositories[22] are being worked on and integrated. With these implemented, the hosted API Image Builder service could be utilized for the further pre-hardened image creation.

More Accurate Testing Cycle

As was mentioned in the beginning of Chapter 2, the current quality assurance of the hardened images requires a lot of knowledge and time of the Quality Engineers. With the proposed solution this time can be highly reduced (more detailed in Section 3.3.2).

[20] https://github.com/osbuild/images/pull/43
[21] https://github.com/osbuild/osbuild-composer/pull/3886
[22] https://issues.redhat.com/browse/COMPOSER-2174

However, this can be reduced even further, when the right testing cycle is selected for these type of tests. One of the proposed testing cycles for the RHEL Security Compliance team was the *Comprehensive Test Cycles* (CTC). These cycles would help teams to identify the regressions or newly created issues far before they reach a production builds.

It is crucial to note that these proposals might change once the necessary tools are in place to support the desired features. The Quality Assurance of the pre-hardened images could then be handled in a more optimized manner.

Conclusion

Research conducted in this thesis was mainly focused on the more optimized and better scalable solution to the current Quality Assurance of the security hardened Red Hat Enterprise Linux (RHEL) systems. Currently available solution is more time consuming and also requires a deep knowledge of the OpenSCAP security hardening which creates a big barrier of entry for the quality engineers.

From the comprehensive analysis, the best variants for the used solution were selected and each of them was consulted and reviewed with the RHEL Security Compliance team as a stakeholder for this project. The best proposed solution that met all of the criteria was to utilize the Red Hat's Image Builder Composer project for creating the pre-hardened images so that the quality engineers do not have to worry about the hardening process. It also leverages the Testing Farm project, which is used for virtual image provisioning and is standardized approach within the Red Hat's infrastructure. With this solution, individual quality engineers can simply request the pre-hardened virtual image from the Testing Farm infrastructure and proceed with their typical testing tasks.

Significant part of this thesis was the proposal and the implementation. During this phase, new problems occurred that needed to be resolved. For example, the naming convention for the pre-hardened virtual images had to be agreed upon with both the Quality Engineering teams and the Testing Farm team, as these names had to be allowlisted in many automation scripts in order to work properly.

According to the thesis's time evaluation, this implementation begins to save the quality engineers' work after just three quarters, saving roughly 16 mandays of capacity every quarter. In addition, from the economic point of view, the return on security investment resulted into 5,713, indicating that maintaining this solution makes financial sense. A detailed inspection of the customer case reported for the missed regression bug in the hardened image was carried out to support these data. The individual graphs from this analysis demonstrate how this solution would save a substantial amount of engineering and support capacity and make it far simpler to identify and rectify such regressions before they are pushed to the production builds.

Since this thesis was intended to serve as a proof-of-concept, it was necessary to analyze and propose future development plans as well. These primarily address improved scalability across various RHEL versions and security profiles used for hardening. Utilizing the Image Builder Hosted service and integrating the image sharing pipeline using the Amazon S3 storage service are recommended as the next steps to accomplish this.

Bibliography

[1] 3Pillar. *What Is Quality Assurance Testing?* January 2020. Available at: https://www.3pillarglobal.com/insights/what-is-qa-in-software-testing/. [cit. 2023-11-29].

[2] Abraham, S. *Top 5 Cybersecurity Breaches Due to Human Error* [online]. March 2022. Available at: https://threatcop.com/blog/top-5-cyber-attacks-and-security-breaches-due-to-human-error/. [cit. 2024-01-15].

[3] Amazon Web Service. *Amazon Machine Images (AMI)* [online]. Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html. [cit. 2024-01-16].

[4] Amazon Web Service. *What is Amazon S3?* [online]. Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. [cit. 2024-01-17].

[5] Ansible. *Ansible playbooks* [online]. Available at: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html. [cit. 2024-01-26].

[6] Bajrami, V. *How to create a Linux RPM package* [online]. November 2020. Available at: https://www.redhat.com/sysadmin/create-rpm-package. [cit. 2024-02-07].

[7] Balbix. *What is the NIST Cybersecurity Framework?* [online]. Available at: https://www.balbix.com/insights/nist-cybersecurity-framework/. [cit. 2024-01-07].

[8] Barney, N. and Gillis, A. S. *Amazon Web Services (AWS)* [online]. Available at: https://www.techtarget.com/searchaws/definition/Amazon-Web-Services. [cit. 2024-01-16].

[9] Bilginc. *What is Red Hat Enterprise Linux (RHEL)?* Available at: https://bilginc.com/en/blog/what-is-red-hat-enterprise-linux-rhel-5914/. [cit. 2023-11-05].

[10] Budai, O. *Using hosted image builder via its API* [online]. Available at: https://www.redhat.com/en/blog/using-hosted-image-builder-its-api. [cit. 2024-01-16].

[11] bugcrowd. *CIS Controls Framework (Center for Internet Security)* [online]. Available at: https://www.bugcrowd.com/glossary/cis-controls-framework-center-for-internet-security/. [cit. 2024-01-07].

[12] Carty, D. *Amazon EC2 (Elastic Compute Cloud)* [online]. Available at: https://www.techtarget.com/searchaws/definition/Amazon-Elastic-Compute-Cloud-Amazon-EC2. [cit. 2024-01-16].

[13] CGI Inc.. Understanding Cybersecurity Standards. *CGI* [online]. april 2019. Available at: https://www.cgi.com/sites/default/files/2019-08/cgi-understanding-cybersecurity-standards-white-paper.pdf, [cit. 2023-12-07].

[14] Chai, W. *Amazon Simple Storage Service (Amazon S3)* [online]. Available at: https://www.techtarget.com/searchaws/definition/Amazon-Simple-Storage-Service-Amazon-S3. [cit. 2024-01-17].

[15] ComplianceAsCode. *ComplianceAsCode* [online]. Available at: https://github.com/ComplianceAsCode/content?tab=readme-ov-file#welcome. [cit. 2024-01-07].

[16] ComplianceAsCode. *ComplianceAsCode Introduction* [online]. Available at: https://complianceascode.readthedocs.io/en/latest/manual/user/01_introduction.html. [cit. 2024-01-07].

[17] Dalao, K. *Understanding IT security frameworks: Types and examples* [online]. March 2023. Available at: https://www.onetrust.com/blog/security-framework-types/. [cit. 2023-12-07].

[18] Das, A. *A quick guide to DNF for yum users* [online]. August 2018. Available at: https://opensource.com/article/18/8/guide-yum-dnf. [cit. 2024-03-03].

[19] DoD Cyber Exchange. *Security Technical Implementation Guides (STIGs)* [online]. Available at: https://public.cyber.mil/stigs/. [cit. 2024-01-12].

[20] Donohue, J. *CIS compliance: What it is & how to comply with CIS benchmarks* [online]. Available at: https://www.diligent.com/resources/blog/what-is-cis-compliance. [cit. 2024-01-12].

[21] Fedora. *Koji HOWTO* [online]. Available at: https://docs.pagure.org/koji/HOWTO/#introduction. [cit. 2024-02-07].

[22] Geeksforgeeks. *Unit Testing – Software Testing* [online]. November 2023. Available at: https://www.geeksforgeeks.org/unit-testing-software-testing/. [cit. 2024-02-12].

[23] Gitlab. *CI/CD explained* [online]. Available at: https://about.gitlab.com/topics/ci-cd/. [cit. 2024-01-07].

[24] Hamilton, T. *What is System Testing?* [online]. January 2024. Available at: https://www.guru99.com/system-testing.html. [cit. 2024-02-12].

[25] IBM. *What are hypervisors?* [online]. Available at: https://www.ibm.com/topics/hypervisors. [cit. 2024-01-15].

[26] IBM. *What are virtual machines (VMs)?* [online]. Available at: https://www.ibm.com/topics/virtual-machines. [cit. 2024-01-15].

[27] Image Builder Team. *Blueprint Reference* [online]. Available at: https://www.osbuild.org/guides/image-builder-on-premises/blueprint-reference.html#fips. [cit. 2024-01-18].

[28] Image Builder Team. *Build Reliable Operating System Images* [online]. Available at: https://www.osbuild.org/guides/introduction.html. [cit. 2024-01-20].

[29] Image Builder Team. *Developer Guide* [online]. Available at: https://www.osbuild.org/guides/developer-guide/index.html. [cit. 2024-01-19].

[30] Image Builder Team. *Image Builder* [online]. Available at: https://github.com/osbuild/image-builder/blob/main/README.md. [cit. 2024-01-20].

[31] Image Builder Team. *Image Builder on premises* [online]. Available at: https://www.osbuild.org/guides/image-builder-on-premises/image-builder-on-premises.html. [cit. 2024-01-18].

[32] Image Builder Team. *OpenSCAP Remediation* [online]. Available at: https://www.osbuild.org/guides/image-builder-on-premises/oscap-remediation.html. [cit. 2024-01-22].

[33] Image Builder Team. *OSBuild Composer* [online]. Available at: https://github.com/osbuild/osbuild-composer/blob/7bcf8e5942e81f7bbf20da79902c2dc8466de657/README.md. [cit. 2024-01-19].

[34] Imperva. *What is Vulnerability Assessment* [online]. Available at: https://www.imperva.com/learn/application-security/vulnerability-assessment/. [cit. 2024-01-11].

[35] Indeed Editorial Team. *What Is Quality Assurance Testing? (With Types and Benefits)*. December 2022. Available at: https://www.indeed.com/career-advice/finding-a-job/what-is-quality-assurance-testing. [cit. 2023-11-29].

[36] ISO org.. *What is ISO/IEC 27001?* [online]. October 2022. Available at: https://www.iso.org/standard/27001. [cit. 2023-12-07].

[37] Kanade, V. *What Is Regression Testing? Definition, Techniques, and Tools* [online]. October 2022. Available at: https://www.spiceworks.com/tech/devops/articles/what-is-regression-testing/. [cit. 2024-02-12].

[38] Kaplan, Z. *What Is QA (Quality Assurance)?* March 2023. Available at: https://www.theforage.com/blog/skills/quality-assurance. [cit. 2023-11-05].

[39] Katalon. *What is Integration Testing? Definition, How-to, Examples* [online]. Available at: https://katalon.com/resources-center/blog/integration-testing. [cit. 2024-02-12].

[40] Kornel, D. and Stejskal, J. *How Testing Farm makes testing your upstream project easier* [online]. August 2023. Available at: https://developers.redhat.com/articles/2023/08/17/how-testing-farm-makes-testing-your-upstream-project-easier. [cit. 2023-11-29].

[41] MITRE. *Open Vulnerability and Assessment Language* [online]. Available at: https://oval.mitre.org/. [cit. 2024-01-14].

[42] Munawar, S. *8 Functional Testing Types Explained With Examples*. August 2021. Available at: https://novateus.com/blog/8-functional-testing-types-explained-with-examples/. [cit. 2023-11-29].

[43] NIST. *FIPS 140-2* [online]. Available at: https://csrc.nist.gov/pubs/fips/140-2/upd2/final. [cit. 2024-01-07].

[44] NIST. *Official Common Platform Enumeration (CPE) Dictionary* [online]. Available at: https://nvd.nist.gov/products/cpe. [cit. 2024-01-14].

[45] NIST. *Vulnerability Assessment* [online]. April 2023. Available at: https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/xccdf. [cit. 2024-01-11].

[46] Openstack. *Virtual Machine Image Guide* [online]. Available at: https://docs.openstack.org/image-guide/introduction.html. [cit. 2024-01-16].

[47] Opentext. *What is Functional Testing?* Available at: https://www.opentext.com/what-is/functional-testing. [cit. 2023-11-29].

[48] Packagecloud Ops. *Working with Source RPMs* [online]. April 2015. Available at: https://blog.packagecloud.io/working-with-source-rpms/. [cit. 2024-02-12].

[49] Preston Werner, T. *TOML [Tom's Obvious Minimal Language]* [online]. Available at: https://toml.io/en/. [cit. 2024-01-18].

[50] Public Health Infrastructure Center. *Health Insurance Portability and Accountability Act of 1996 (HIPAA)* [online]. June 2022. Available at: https://www.cdc.gov/phlp/publications/topic/hipaa.html. [cit. 2024-01-07].

[51] Rane, S. *What are the 12 requirements of PCI DSS Compliance?* [online]. Available at: https://www.controlcase.com/what-are-the-12-requirements-of-pci-dss-compliance/. [cit. 2024-01-07].

[52] Red Hat. *Automation for everyone* [online]. Available at: https://www.ansible.com/. [cit. 2024-01-15].

[53] Red Hat. *Basic oscap Usage* [online]. Available at: https://static.open-scap.org/openscap-1.2/oscap_user_manual.html#_basic_oscap_usage. [cit. 2024-01-14].

[54] Red Hat. *Configuration Compliance Scanning* [online]. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/configuration-compliance-scanning_scanning-the-system-for-configuration-compliance-and-vulnerabilities. [cit. 2024-05-01].

[55] Red Hat. *Creating pre-hardened images with RHEL image builder OpenSCAP integration* [online]. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/composing_a_customized_rhel_system_image/assembly_creating-pre-hardened-images-with-image-builder-openscap-integration_composing-a-customized-rhel-system-image. [cit. 2024-01-22].

[56] Red Hat. *The Data Stream Format* [online]. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/sect-scap_format-data_stream. [cit. 2024-01-14].

[57] Red Hat. *Generating Image Builder Blueprints* [online]. Available at: https://static.open-scap.org/openscap-1.3/oscap_user_manual.html#_generating_image_builder_blueprints. [cit. 2024-02-04].

[58] Red Hat. *How Ansible works* [online]. Available at: https://www.ansible.com/overview/how-ansible-works. [cit. 2024-01-15].

[59] Red Hat. *How to Evaluate DISA STIG* [online]. Available at: https://static.open-scap.org/openscap-1.2/oscap_user_manual.html#_how_to_evaluate_disa_stig. [cit. 2024-02-17].

[60] Red Hat. *Introduction to RPM* [online]. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/packaging_and_distributing_software/introduction-to-rpm_packaging-and-distributing-software. [cit. 2024-02-07].

[61] Red Hat. *Managing repositories* [online]. Available at: https://osbuild.org/docs/on-premises/installation/managing-repositories. [cit. 2024-03-24].

[62] Red Hat. *OpenSCAP* [online]. Available at: https://www.open-scap.org/. [cit. 2024-01-11].

[63] Red Hat. *OpenSCAP* [online]. Available at: https://www.open-scap.org/. [cit. 2024-02-17].

[64] Red Hat. *Red Hat Enterprise Linux*. Available at: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux. [cit. 2023-11-05].

[65] Red Hat. *Red Hat Enterprise Linux security and compliance* [online]. Available at: https://www.redhat.com/en/resources/enterprise-linux-security-compliance-brief. [cit. 2024-02-19].

[66] Red Hat. *SCAP Components* [online]. Available at: https://www.open-scap.org/features/scap-components/. [cit. 2024-02-18].

[67] Red Hat. *Storage Formats for Virtual Disk Images* [online]. Available at: https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.0/html/technical_reference/qcow2. [cit. 2024-01-16].

[68] Red Hat. *Tailoring* [online]. Available at: https://static.open-scap.org/openscap-1.3/oscap_user_manual.html#_tailoring. [cit. 2024-01-14].

[69] Red Hat. *What is SELinux?* [online]. August 2019. Available at: https://www.redhat.com/en/topics/linux/what-is-selinux. [cit. 2024-02-11].

[70] Red Hat. *What is a golden image?* [online]. August 2022. Available at: https://www.redhat.com/en/topics/linux/what-is-a-golden-image. [cit. 2024-01-15].

[71] Red Hat. *What is an image builder* [online]. December 2022. Available at: https://www.redhat.com/en/topics/linux/what-is-an-image-builder. [cit. 2024-01-15].

[72] Red Hat. *What is CI/CD?* [online]. May 2022. Available at: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [cit. 2024-01-07].

[73] Rouse, M. *Red Hat Enterprise Linux*. February 2017. Available at: https://www.techopedia.com/definition/15777/red-hat-enterprise-linux-rhel. [cit. 2023-11-07].

[74] Sato, W. *Implementing ANSSI security recommendations for RHEL 7 and 8* [online]. April 2021. Available at: https://www.redhat.com/en/blog/implementing-anssi-security-recommendations-rhel-7-and-8. [cit. 2024-01-12].

[75] Scytale. *Security Compliance* [online]. Available at: https://scytale.ai/glossary/security-compliance/. [cit. 2023-12-07].

[76] Sedlák, P. and Konečný, M. 2023. *Přeměna ISMS v manažerské informatice*. Brno: CERM, Akademické nakladatelství. ISBN 978-80-7623-110-8.

[77] Shidlovskaya, A. *A Guide to Security Hardening* [online]. May 2023. Available at: https://www.cobalt.io/blog/guide-to-security-hardening. [cit. 2024-01-07].

[78] Simic, S. *What is a Hypervisor? Types of Hypervisors 1 & 2* [online]. September 2022. Available at: https://phoenixnap.com/kb/what-is-hypervisor-type-1-2. [cit. 2024-01-15].

[79] Sosecure. *Security Hardening Services* [online]. Available at: https://www.sosecure.co.th/security-hardening-services/. [cit. 2024-03-18].

[80] Synopsys. *Vulnerability Assessment* [online]. Available at: https://www.synopsys.com/glossary/what-is-vulnerability-assessment.html. [cit. 2024-01-11].

[81] Testing Farm Contributors. *Command Line Interface* [online]. Available at: https://docs.testing-farm.io/Testing%20Farm/0.1/cli.html. [cit. 2024-03-27].

[82] Testing Farm Contributors. *Onboarding* [online]. Available at: https://docs.testing-farm.io/Testing%20Farm/0.1/onboarding.html. [cit. 2024-03-28].

[83] Testing Farm Contributors. *Testing Farm* [online]. Available at: https://docs.testing-farm.io/Testing%20Farm/0.1/index.html. [cit. 2023-11-29].

[84] Tougne, J.-S. *Center for Internet Security (CIS) compliance in Red Hat Enterprise Linux using OpenSCAP* [online]. December 2020. Available at: https://www.redhat.com/en/blog/center-internet-security-cis-compliance-red-hat-enterprise-linux-using-openscap. [cit. 2024-01-12].

[85] unixsysadmin. *What's New in RHEL 9*. October 2022. Available at: https://www.unixsysadmin.com/rhel-9-resources/. [cit. 2023-11-05].

[86] VMWare. *What is a virtual machine?* [online]. Available at: https://www.vmware.com/topics/glossary/content/virtual-machine.html. [cit. 2024-01-15].

[87] Westland, J. *The Quality Assurance Process: Roles, Methods & Tools*. December 2022. Available at: https://www.projectmanager.com/blog/quality-assurance-and-testing. [cit. 2023-11-07].

[88] Wigmore, I. *Amazon Machine Image (AMI)* [online]. Available at: https://www.techtarget.com/searchaws/definition/Amazon-Machine-Image-AMI. [cit. 2024-01-16].

[89] ČSN EN ISO/IEC 27002. *Informační bezpečnost, kybernetická bezpečnost a ochrana soukromí - Opatření informační bezpečnosti*, 2023. [Praha]: Česká agentura pro standardizaci.

List of Figures

List of Tables

List of Listings