Red Hat Enterprise Linux 8

Security hardening

Securing Red Hat Enterprise Linux 8

Red Hat Enterprise Linux 8 Security hardening

Securing Red Hat Enterprise Linux 8

Legal Notice

Abstract

This title assists users and administrators in learning the processes and practices of securing workstations and servers against local and remote intrusion, exploitation, and malicious activity. Focused on Red Hat Enterprise Linux but detailing concepts and techniques valid for all Linux systems, this guide details the planning and the tools involved in creating a secured computing environment for the data center, workplace, and home. With proper administrative knowledge, vigilance, and tools, systems running Linux can be both fully functional and secured from most common intrusion and exploit methods.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For simple comments on specific passages:

  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.

  2. Use your mouse cursor to highlight the part of text that you want to comment on.

  3. Click the **Add Feedback** pop-up that appears below the highlighted text.

  4. Follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla ticket:

  1. Go to the Bugzilla website.

  2. As the Component, use **Documentation**.

  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.

  4. Click **Submit Bug**.

CHAPTER 1. OVERVIEW OF SECURITY HARDENING IN RHEL

Due to the increased reliance on powerful, networked computers to help run businesses and keep track of our personal information, entire industries have been formed around the practice of network and computer security. Enterprises have solicited the knowledge and skills of security experts to properly audit systems and tailor solutions to fit the operating requirements of their organization. Because most organizations are increasingly dynamic in nature, their workers are accessing critical company IT resources locally and remotely, hence the need for secure computing environments has become more pronounced.

Unfortunately, many organizations, as well as individual users, regard security as more of an afterthought, a process that is overlooked in favor of increased power, productivity, convenience, ease of use, and budgetary concerns. Proper security implementation is often enacted postmortem — after an unauthorized intrusion has already occurred. Taking the correct measures prior to connecting a site to an untrusted network, such as the Internet, is an effective means of thwarting many attempts at intrusion.

1.1. WHAT IS COMPUTER SECURITY?

Computer security is a general term that covers a wide area of computing and information processing. Industries that depend on computer systems and networks to conduct daily business transactions and access critical information regard their data as an important part of their overall assets. Several terms and metrics have entered our daily business vocabulary, such as total cost of ownership (TCO), return on investment (ROI), and quality of service (QoS). Using these metrics, industries can calculate aspects such as data integrity and high-availability (HA) as part of their planning and process management costs. In some industries, such as electronic commerce, the availability and trustworthiness of data can mean the difference between success and failure.

1.2. STANDARDIZING SECURITY

Enterprises in every industry rely on regulations and rules that are set by standards-making bodies such as the American Medical Association (AMA) or the Institute of Electrical and Electronics Engineers (IEEE). The same concepts hold true for information security. Many security consultants and vendors agree upon the standard security model known as CIA, or *Confidentiality, Integrity, and Availability*. This three-tiered model is a generally accepted component to assessing risks of sensitive information and establishing security policy. The following describes the CIA model in further detail:

- Confidentiality — Sensitive information must be available only to a set of pre-defined individuals. Unauthorized transmission and usage of information should be restricted. For example, confidentiality of information ensures that a customer's personal or financial information is not obtained by an unauthorized individual for malicious purposes such as identity theft or credit fraud.

- Integrity — Information should not be altered in ways that render it incomplete or incorrect. Unauthorized users should be restricted from the ability to modify or destroy sensitive information.

- Availability — Information should be accessible to authorized users any time that it is needed. Availability is a warranty that information can be obtained with an agreed-upon frequency and timeliness. This is often measured in terms of percentages and agreed to formally in Service Level Agreements (SLAs) used by network service providers and their enterprise clients.

1.3. CRYPTOGRAPHIC SOFTWARE AND CERTIFICATIONS

Red Hat Enterprise Linux undergoes several security certifications, such as **FIPS 140-2** or **Common Criteria** (CC), to ensure that industry best practices are followed.

The RHEL 8 core crypto components Knowledgebase article provides an overview of the Red Hat Enterprise Linux 8 core crypto components, documenting which are they, how are they selected, how are they integrated into the operating system, how do they support hardware security modules and smart cards, and how do crypto certifications apply to them.

1.4. SECURITY CONTROLS

Computer security is often divided into three distinct main categories, commonly referred to as **controls**:

- Physical

- Technical

- Administrative

These three broad categories define the main objectives of proper security implementation. Within these controls are sub-categories that further detail the controls and how to implement them.

1.4.1. Physical controls

Physical control is the implementation of security measures in a defined structure used to deter or prevent unauthorized access to sensitive material. Examples of physical controls are:

- Closed-circuit surveillance cameras

- Motion or thermal alarm systems

- Security guards

- Picture IDs

- Locked and dead-bolted steel doors

- Biometrics (includes fingerprint, voice, face, iris, handwriting, and other automated methods used to recognize individuals)

1.4.2. Technical controls

Technical controls use technology as a basis for controlling the access and usage of sensitive data throughout a physical structure and over a network. Technical controls are far-reaching in scope and encompass such technologies as:

- Encryption

- Smart cards

- Network authentication

- Access control lists (ACLs)

- File integrity auditing software

1.4.3. Administrative controls

Administrative controls define the human factors of security. They involve all levels of personnel within an organization and determine which users have access to what resources and information by such means as:

- Training and awareness

- Disaster preparedness and recovery plans

- Personnel recruitment and separation strategies

- Personnel registration and accounting

1.5. VULNERABILITY ASSESSMENT

Given time, resources, and motivation, an attacker can break into nearly any system. All of the security procedures and technologies currently available cannot guarantee that any systems are completely safe from intrusion. Routers help secure gateways to the Internet. Firewalls help secure the edge of the network. Virtual Private Networks safely pass data in an encrypted stream. Intrusion detection systems warn you of malicious activity. However, the success of each of these technologies is dependent upon a number of variables, including:

- The expertise of the staff responsible for configuring, monitoring, and maintaining the technologies.

- The ability to patch and update services and kernels quickly and efficiently.

- The ability of those responsible to keep constant vigilance over the network.

Given the dynamic state of data systems and technologies, securing corporate resources can be quite complex. Due to this complexity, it is often difficult to find expert resources for all of your systems. While it is possible to have personnel knowledgeable in many areas of information security at a high level, it is difficult to retain staff who are experts in more than a few subject areas. This is mainly because each subject area of information security requires constant attention and focus. Information security does not stand still.

A vulnerability assessment is an internal audit of your network and system security; the results of which indicate the confidentiality, integrity, and availability of your network. Typically, vulnerability assessment starts with a reconnaissance phase, during which important data regarding the target systems and resources is gathered. This phase leads to the system readiness phase, whereby the target is essentially checked for all known vulnerabilities. The readiness phase culminates in the reporting phase, where the findings are classified into categories of high, medium, and low risk; and methods for improving the security (or mitigating the risk of vulnerability) of the target are discussed

If you were to perform a vulnerability assessment of your home, you would likely check each door to your home to see if they are closed and locked. You would also check every window, making sure that they closed completely and latch correctly. This same concept applies to systems, networks, and electronic data. Malicious users are the thieves and vandals of your data. Focus on their tools, mentality, and motivations, and you can then react swiftly to their actions.

1.5.1. Defining assessment and testing

Vulnerability assessments may be broken down into one of two types: *outside looking in* and *inside looking around*.

When performing an outside-looking-in vulnerability assessment, you are attempting to compromise your systems from the outside. Being external to your company provides you with the cracker's point of view. You see what a cracker sees – publicly-routable IP addresses, systems on your *DMZ*, external interfaces of your firewall, and more. DMZ stands for "demilitarized zone", which corresponds to a computer or small subnetwork that sits between a trusted internal network, such as a corporate private LAN, and an untrusted external network, such as the public Internet. Typically, the DMZ contains devices accessible to Internet traffic, such as web (HTTP) servers, FTP servers, SMTP (e-mail) servers and DNS servers.

When you perform an inside-looking-around vulnerability assessment, you are at an advantage since you are internal and your status is elevated to trusted. This is the point of view you and your co-workers have once logged on to your systems. You see print servers, file servers, databases, and other resources.

There are striking distinctions between the two types of vulnerability assessments. Being internal to your company gives you more privileges than an outsider. In most organizations, security is configured to keep intruders out. Very little is done to secure the internals of the organization (such as departmental firewalls, user-level access controls, and authentication procedures for internal resources). Typically, there are many more resources when looking around inside as most systems are internal to a company. Once you are outside the company, your status is untrusted. The systems and resources available to you externally are usually very limited.

Consider the difference between vulnerability assessments and *penetration tests*. Think of a vulnerability assessment as the first step to a penetration test. The information gleaned from the assessment is used for testing. Whereas the assessment is undertaken to check for holes and potential vulnerabilities, the penetration testing actually attempts to exploit the findings.

Assessing network infrastructure is a dynamic process. Security, both information and physical, is dynamic. Performing an assessment shows an overview, which can turn up false positives and false negatives. A false positive is a result, where the tool finds vulnerabilities which in reality do not exist. A false negative is when it omits actual vulnerabilities.

Security administrators are only as good as the tools they use and the knowledge they retain. Take any of the assessment tools currently available, run them against your system, and it is almost a guarantee that there are some false positives. Whether by program fault or user error, the result is the same. The tool may find false positives, or, even worse, false negatives.

Now that the difference between a vulnerability assessment and a penetration test is defined, take the findings of the assessment and review them carefully before conducting a penetration test as part of your new best practices approach.

> **WARNING**
>
> Do not attempt to exploit vulnerabilities on production systems. Doing so can have adverse effects on productivity and efficiency of your systems and network.

The following list examines some of the benefits of performing vulnerability assessments.

- Creates proactive focus on information security.

- Finds potential exploits before crackers find them.

- Results in systems being kept up to date and patched.

- Promotes growth and aids in developing staff expertise.

- Abates financial loss and negative publicity.

1.5.2. Establishing a methodology for vulnerability assessment

To aid in the selection of tools for a vulnerability assessment, it is helpful to establish a vulnerability assessment methodology. Unfortunately, there is no predefined or industry approved methodology at this time; however, common sense and best practices can act as a sufficient guide.

What is the target? Are we looking at one server, or are we looking at our entire network and everything within the network? Are we external or internal to the company? The answers to these questions are important as they help determine not only which tools to select but also the manner in which they are used.

To learn more about establishing methodologies, see the following website:

- https://www.owasp.org/ — **The Open Web Application Security Project**

1.5.3. Vulnerability assessment tools

An assessment can start by using some form of an information-gathering tool. When assessing the entire network, map the layout first to find the hosts that are running. Once located, examine each host individually. Focusing on these hosts requires another set of tools. Knowing which tools to use may be the most crucial step in finding vulnerabilities.

The following tools are just a small sampling of the available tools:

- **Nmap** is a popular tool that can be used to find host systems and open ports on those systems. To install **Nmap** from the **AppStream** repository, enter the **yum install nmap** command as the **root** user. See the **nmap(1)** man page for more information.

- The tools from the **OpenSCAP** suite, such as the **oscap** command-line utility and the **scap-workbench** graphical utility, provides a fully automated compliance audit. See Scanning the system for security compliance and vulnerabilities for more information.

- Advanced Intrusion Detection Environment (**AIDE**) is a utility that creates a database of files on the system, and then uses that database to ensure file integrity and detect system intrusions. See Checking integrity with AIDE for more information.

1.6. SECURITY THREATS

1.6.1. Threats to network security

Bad practices when configuring the following aspects of a network can increase the risk of an attack.

Insecure architectures

A misconfigured network is a primary entry point for unauthorized users. Leaving a trust-based, open local network vulnerable to the highly-insecure Internet is much like leaving a door ajar in a crime-ridden neighborhood — nothing may happen for an arbitrary amount of time, but someone exploits the opportunity *eventually*.

Broadcast networks

System administrators often fail to realize the importance of networking hardware in their security

schemes. Simple hardware, such as hubs and routers, relies on the broadcast or non-switched principle; that is, whenever a node transmits data across the network to a recipient node, the hub or router sends a broadcast of the data packets until the recipient node receives and processes the data. This method is the most vulnerable to address resolution protocol (*ARP*) or media access control (*MAC*) address spoofing by both outside intruders and unauthorized users on local hosts.

Centralized servers

Another potential networking pitfall is the use of centralized computing. A common cost-cutting measure for many businesses is to consolidate all services to a single powerful machine. This can be convenient as it is easier to manage and costs considerably less than multiple-server configurations. However, a centralized server introduces a single point of failure on the network. If the central server is compromised, it may render the network completely useless or worse, prone to data manipulation or theft. In these situations, a central server becomes an open door that allows access to the entire network.

1.6.2. Threats to server security

Server security is as important as network security because servers often hold a great deal of an organization's vital information. If a server is compromised, all of its contents may become available for the cracker to steal or manipulate at will. The following sections detail some of the main issues.

Unused services and open ports

A full installation of Red Hat Enterprise Linux 8 contains more than 1000 applications and library packages. However, most server administrators do not opt to install every single package in the distribution, preferring instead to install a base installation of packages, including several server applications.

A common occurrence among system administrators is to install the operating system without paying attention to what programs are actually being installed. This can be problematic because unneeded services may be installed, configured with the default settings, and possibly turned on. This can cause unwanted services, such as Telnet, DHCP, or DNS, to run on a server or workstation without the administrator realizing it, which in turn can cause unwanted traffic to the server or even a potential pathway into the system for crackers.

Unpatched services

Most server applications that are included in a default installation are solid, thoroughly tested pieces of software. Having been in use in production environments for many years, their code has been thoroughly refined and many of the bugs have been found and fixed.

However, there is no such thing as perfect software and there is always room for further refinement. Moreover, newer software is often not as rigorously tested as one might expect, because of its recent arrival to production environments or because it may not be as popular as other server software.

Developers and system administrators often find exploitable bugs in server applications and publish the information on bug tracking and security-related websites such as the Bugtraq mailing list (http://www.securityfocus.com) or the Computer Emergency Response Team (CERT) website (http://www.cert.org). Although these mechanisms are an effective way of alerting the community to security vulnerabilities, it is up to system administrators to patch their systems promptly. This is particularly true because crackers have access to these same vulnerability tracking services and will use the information to crack unpatched systems whenever they can. Good system administration requires vigilance, constant bug tracking, and proper system maintenance to ensure a more secure computing environment.

Inattentive administration

Administrators who fail to patch their systems are one of the greatest threats to server security. This applies as much to inexperienced administrators as it does to overconfident or amotivated administrators.

Some administrators fail to patch their servers and workstations, while others fail to watch log messages from the system kernel or network traffic. Another common error is when default passwords or keys to services are left unchanged. For example, some databases have default administration passwords because the database developers assume that the system administrator changes these passwords immediately after installation. If a database administrator fails to change this password, even an inexperienced cracker can use a widely-known default password to gain administrative privileges to the database. These are only a few examples of how inattentive administration can lead to compromised servers.

Inherently insecure services

Even the most vigilant organization can fall victim to vulnerabilities if the network services they choose are inherently insecure. For instance, there are many services developed under the assumption that they are used over trusted networks; however, this assumption fails as soon as the service becomes available over the Internet — which is itself inherently untrusted.

One category of insecure network services are those that require unencrypted user names and passwords for authentication. Telnet and FTP are two such services. If packet sniffing software is monitoring traffic between the remote user and such a service user names and passwords can be easily intercepted.

Inherently, such services can also more easily fall prey to what the security industry terms the *man-in-the-middle* attack. In this type of attack, a cracker redirects network traffic by tricking a cracked name server on the network to point to his machine instead of the intended server. Once someone opens a remote session to the server, the attacker's machine acts as an invisible conduit, sitting quietly between the remote service and the unsuspecting user capturing information. In this way a cracker can gather administrative passwords and raw data without the server or the user realizing it.

Another category of insecure services include network file systems and information services such as NFS or NIS, which are developed explicitly for LAN usage but are, unfortunately, extended to include WANs (for remote users). NFS does not, by default, have any authentication or security mechanisms configured to prevent a cracker from mounting the NFS share and accessing anything contained therein. NIS, as well, has vital information that must be known by every computer on a network, including passwords and file permissions, within a plain text ASCII or DBM (ASCII-derived) database. A cracker who gains access to this database can then access every user account on a network, including the administrator's account.

By default, Red Hat Enterprise Linux 8 is released with all such services turned off. However, since administrators often find themselves forced to use these services, careful configuration is critical.

1.6.3. Threats to workstation and home PC security

Workstations and home PCs may not be as prone to attack as networks or servers, but because they often contain sensitive data, such as credit card information, they are targeted by system crackers. Workstations can also be co-opted without the user's knowledge and used by attackers as "bot" machines in coordinated attacks. For these reasons, knowing the vulnerabilities of a workstation can save users the headache of reinstalling the operating system, or worse, recovering from data theft.

Bad passwords

Bad passwords are one of the easiest ways for an attacker to gain access to a system.

Vulnerable client applications

Although an administrator may have a fully secure and patched server, that does not mean remote users are secure when accessing it. For instance, if the server offers Telnet or FTP services over a public network, an attacker can capture the plain text user names and passwords as they pass over the network, and then use the account information to access the remote user's workstation.

Even when using secure protocols, such as SSH, a remote user may be vulnerable to certain attacks if they do not keep their client applications updated. For instance, SSH protocol version 1 clients are vulnerable to an X-forwarding attack from malicious SSH servers. Once connected to the server, the attacker can quietly capture any keystrokes and mouse clicks made by the client over the network. This problem was fixed in the SSH version 2 protocol, but it is up to the user to keep track of what applications have such vulnerabilities and update them as necessary.

1.7. COMMON EXPLOITS AND ATTACKS

Table 1.1, "Common exploits" details some of the most common exploits and entry points used by intruders to access organizational network resources. Key to these common exploits are the explanations of how they are performed and how administrators can properly safeguard their network against such attacks.

Table 1.1. Common exploits

Exploit	Description	Notes
Null or default passwords	Leaving administrative passwords blank or using a default password set by the product vendor. This is most common in hardware such as routers and firewalls, but some services that run on Linux can contain default administrator passwords as well (though Red Hat Enterprise Linux 8 does not ship with them).	Commonly associated with networking hardware such as routers, firewalls, VPNs, and network attached storage (NAS) appliances. Common in many legacy operating systems, especially those that bundle services (such as UNIX and Windows.) Administrators sometimes create privileged user accounts in a rush and leave the password null, creating a perfect entry point for malicious users who discover the account.
Default shared keys	Secure services sometimes package default security keys for development or evaluation testing purposes. If these keys are left unchanged and are placed in a production environment on the Internet, **all** users with the same default keys have access to that shared-key resource, and any sensitive information that it contains.	Most common in wireless access points and preconfigured secure server appliances.

Exploit	Description	Notes
IP spoofing	A remote machine acts as a node on your local network, finds vulnerabilities with your servers, and installs a backdoor program or Trojan horse to gain control over your network resources.	Spoofing is quite difficult as it involves the attacker predicting TCP/IP sequence numbers to coordinate a connection to target systems, but several tools are available to assist crackers in performing such a vulnerability. Depends on target system running services (such as **rsh**, **telnet**, FTP and others) that use *source-based* authentication techniques, which are not recommended when compared to PKI or other forms of encrypted authentication used in **ssh** or SSL/TLS.
Eavesdropping	Collecting data that passes between two active nodes on a network by eavesdropping on the connection between the two nodes.	This type of attack works mostly with plain text transmission protocols such as Telnet, FTP, and HTTP transfers. Remote attacker must have access to a compromised system on a LAN in order to perform such an attack; usually the cracker has used an active attack (such as IP spoofing or man-in-the-middle) to compromise a system on the LAN. Preventative measures include services with cryptographic key exchange, one-time passwords, or encrypted authentication to prevent password snooping; strong encryption during transmission is also advised.

Exploit	Description	Notes
Service vulnerabilities	An attacker finds a flaw or loophole in a service run over the Internet; through this vulnerability, the attacker compromises the entire system and any data that it may hold, and could possibly compromise other systems on the network.	HTTP-based services such as CGI are vulnerable to remote command execution and even interactive shell access. Even if the HTTP service runs as a non-privileged user such as "nobody", information such as configuration files and network maps can be read, or the attacker can start a denial of service attack which drains system resources or renders it unavailable to other users. Services sometimes can have vulnerabilities that go unnoticed during development and testing; these vulnerabilities (such as *buffer overflows*, where attackers crash a service using arbitrary values that fill the memory buffer of an application, giving the attacker an interactive command prompt from which they may execute arbitrary commands) can give complete administrative control to an attacker. Administrators should make sure that services do not run as the root user, and should stay vigilant of patches and errata updates for applications from vendors or security organizations such as CERT and CVE.

Exploit	Description	Notes
Application vulnerabilities	Attackers find faults in desktop and workstation applications (such as email clients) and execute arbitrary code, implant Trojan horses for future compromise, or crash systems. Further exploitation can occur if the compromised workstation has administrative privileges on the rest of the network.	Workstations and desktops are more prone to exploitation as workers do not have the expertise or experience to prevent or detect a compromise; it is imperative to inform individuals of the risks they are taking when they install unauthorized software or open unsolicited email attachments.

Safeguards can be implemented such that email client software does not automatically open or execute attachments. Additionally, the automatic update of workstation software using Red Hat Network; or other system management services can alleviate the burdens of multi-seat security deployments. |
| Denial of Service (DoS) attacks | Attacker or group of attackers coordinate against an organization's network or server resources by sending unauthorized packets to the target host (either server, router, or workstation). This forces the resource to become unavailable to legitimate users. | The most reported DoS case in the US occurred in 2000. Several highly-trafficked commercial and government sites were rendered unavailable by a coordinated ping flood attack using several compromised systems with high bandwidth connections acting as *zombies*, or redirected broadcast nodes.

Source packets are usually forged (as well as rebroadcast), making investigation as to the true source of the attack difficult.

Advances in ingress filtering (IETF rfc2267) using **iptables** and Network Intrusion Detection Systems such as **snort** assist administrators in tracking down and preventing distributed DoS attacks. |

CHAPTER 2. SECURING RHEL DURING INSTALLATION

Security begins even before you start the installation of Red Hat Enterprise Linux. Configuring your system securely from the beginning makes it easier to implement additional security settings later.

2.1. BIOS AND UEFI SECURITY

Password protection for the BIOS (or BIOS equivalent) and the boot loader can prevent unauthorized users who have physical access to systems from booting using removable media or obtaining root privileges through single user mode. The security measures you should take to protect against such attacks depends both on the sensitivity of the information on the workstation and the location of the machine.

For example, if a machine is used in a trade show and contains no sensitive information, then it may not be critical to prevent such attacks. However, if an employee's laptop with private, unencrypted SSH keys for the corporate network is left unattended at that same trade show, it could lead to a major security breach with ramifications for the entire company.

If the workstation is located in a place where only authorized or trusted people have access, however, then securing the BIOS or the boot loader may not be necessary.

2.1.1. BIOS passwords

The two primary reasons for password protecting the BIOS of a computer are[1]:

1. **Preventing changes to BIOS settings** — If an intruder has access to the BIOS, they can set it to boot from a CD-ROM or a flash drive. This makes it possible for them to enter rescue mode or single user mode, which in turn allows them to start arbitrary processes on the system or copy sensitive data.

2. **Preventing system booting** — Some BIOSes allow password protection of the boot process. When activated, an attacker is forced to enter a password before the BIOS launches the boot loader.

Because the methods for setting a BIOS password vary between computer manufacturers, consult the computer's manual for specific instructions.

If you forget the BIOS password, it can either be reset with jumpers on the motherboard or by disconnecting the CMOS battery. For this reason, it is good practice to lock the computer case if possible. However, consult the manual for the computer or motherboard before attempting to disconnect the CMOS battery.

2.1.2. Non-BIOS-based systems security

Other systems and architectures use different programs to perform low-level tasks roughly equivalent to those of the BIOS on x86 systems. For example, the *Unified Extensible Firmware Interface* (*UEFI*) shell.

For instructions on password protecting BIOS-like programs, see the manufacturer's instructions.

2.2. DISK PARTITIONING

Red Hat recommends creating separate partitions for the **/boot**, /, **/home**, **/tmp**, and **/var/tmp/** directories.

/boot

> This partition is the first partition that is read by the system during boot up. The boot loader and kernel images that are used to boot your system into Red Hat Enterprise Linux 8 are stored in this partition. This partition should not be encrypted. If this partition is included in / and that partition is encrypted or otherwise becomes unavailable then your system is not able to boot.

/home

> When user data (**/home**) is stored in / instead of in a separate partition, the partition can fill up causing the operating system to become unstable. Also, when upgrading your system to the next version of Red Hat Enterprise Linux 8 it is a lot easier when you can keep your data in the **/home** partition as it is not be overwritten during installation. If the root partition (/) becomes corrupt your data could be lost forever. By using a separate partition there is slightly more protection against data loss. You can also target this partition for frequent backups.

/tmp and **/var/tmp/**

> Both the **/tmp** and **/var/tmp/** directories are used to store data that does not need to be stored for a long period of time. However, if a lot of data floods one of these directories it can consume all of your storage space. If this happens and these directories are stored within / then your system could become unstable and crash. For this reason, moving these directories into their own partitions is a good idea.

> **NOTE**
>
> During the installation process, you have an option to encrypt partitions. You must supply a passphrase. This passphrase serves as a key to unlock the bulk encryption key, which is used to secure the partition's data.

2.3. RESTRICTING NETWORK CONNECTIVITY DURING THE INSTALLATION PROCESS

When installing Red Hat Enterprise Linux 8, the installation medium represents a snapshot of the system at a particular time. Because of this, it may not be up-to-date with the latest security fixes and may be vulnerable to certain issues that were fixed only after the system provided by the installation medium was released.

When installing a potentially vulnerable operating system, always limit exposure only to the closest necessary network zone. The safest choice is the "no network" zone, which means to leave your machine disconnected during the installation process. In some cases, a LAN or intranet connection is sufficient while the Internet connection is the riskiest. To follow the best security practices, choose the closest zone with your repository while installing Red Hat Enterprise Linux 8 from a network.

2.4. INSTALLING THE MINIMUM AMOUNT OF PACKAGES REQUIRED

It is best practice to install only the packages you will use because each piece of software on your computer could possibly contain a vulnerability. If you are installing from the DVD media, take the opportunity to select exactly what packages you want to install during the installation. If you find you need another package, you can always add it to the system later.

2.5. POST-INSTALLATION PROCEDURES

The following steps are the security-related procedures that should be performed immediately after installation of Red Hat Enterprise Linux 8.

- Update your system. Enter the following command as root:

```
# yum update
```

- Even though the firewall service, **firewalld**, is automatically enabled with the installation of Red Hat Enterprise Linux, there are scenarios where it might be explicitly disabled, for example in the kickstart configuration. In such a case, it is recommended to consider re-enabling the firewall. To start **firewalld** enter the following commands as root:

```
# systemctl start firewalld
# systemctl enable firewalld
```

- To enhance security, disable services you do not need. For example, if there are no printers installed on your computer, disable the **cups** service using the following command:

```
# systemctl disable cups
```

To review active services, enter the following command:

```
$ systemctl list-units | grep service
```

---

[1] Since system BIOSes differ between manufacturers, some may not support password protection of either type, while others may support one type but not the other.

CHAPTER 3. SECURING SERVICES

It is important in an organization to monitor the active network services that are important to administers and Linux system admins. Red Hat Enterprise Linux 8 supports many network servers. When a network service is running on a machine, daemon keeps listening for connections on the network ports. These daemons can lead to any kind of attach. As a result, the services needs to be secured in order to prevent any mishappenings. This chapter helps you secure different services.

3.1. SECURING RPCBIND

The **rpcbind** service is a dynamic port assignment daemon for Remote Procedure Calls (RPC) services such as Network Information Service (NIS) and Network File Sharing (NFS). Because it has weak authentication mechanisms and can assign a wide range of ports for the services it controls, it is important to secure the **rpcbind** service.

You can secure the **rpcbind** service by adding firewall rules to the server. You can restrict access to all networks and define specific exceptions using the firewall rules.

> **NOTE**
>
> - The **rpcbind** service is required by **NFSv2** and **NFSv3** servers and you should secure the **rpcbind** service when you are working on it.
>
> - **NFSv4** does not require the **rpcbind** service to listen on the network.

Procedure

- Following are examples for the **firewalld** commands:

    - Limit TCP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

        ```
        # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24" invert="True" drop'
        ```

    - Limit TCP connection and accept packages only from local host via the **111** port:

        ```
        # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept'
        ```

    - Limit UDP connection and accept packages only from the **192.168.0.0/24** host via the **111** port:

        ```
        # firewall-cmd --add-rich-rule='rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24" invert="True" drop'
        ```

> **NOTE**
>
> - To make the firewall settings permanent, use the **--permanent** option when adding firewall rules.
>
> - Reload firewall to accept the new rules using **# firewall-cmd --reload** command.

Verification steps

- Verify the firewall rules:

  ```
  # firewall-cmd --list-rich-rule
  rule family="ipv4" port port="111" protocol="tcp" source address="192.168.0.0/24"
  invert="True" drop
  rule family="ipv4" port port="111" protocol="tcp" source address="127.0.0.1" accept
  rule family="ipv4" port port="111" protocol="udp" source address="192.168.0.0/24"
  invert="True" drop
  ```

Additional resources

- To learn about **NFSv4-only** Server, see Configuring an NFSv4-only Server section.

- To learn more about firewall rules, see Chapter 5 Using and configuring firewalld .

3.2. SECURING RPC.MOUNTD

The **rpc.mountd** daemon implements the server side of the NFS mount protocol. The NFS mount
protocol is used by NFS version 2 (RFC 1904) and NFS version 3 (RFC 1813).

You can secure the **rpc.mountd** service by adding firewall rules to the server. You can restrict access to
all networks and define specific exceptions using the firewall rules.

Procedure

- Following are examples for the firewalld commands:

  - Accept **mountd** connections from the **192.168.0.0/24** host:

    ```
    # firewall-cmd --add-rich-rule 'rule family="ipv4" service name="mountd" source
    address="192.168.0.0/24" invert="True" drop'
    ```

  - Accept **mountd** connections from the local host:

    ```
    # firewall-cmd --add-rich-rule 'rule family="ipv4" source address="127.0.0.1" service
    name="mountd" accept'
    ```

> **NOTE**
>
> - To make the firewall settings permanent, use the **--permanent** option
>   when adding firewall rules.
>
> - Reload firewall to accept the new rules using the **# firewall-cmd --reload**
>   command.

Verification steps

- Verify the firewall rules:

  ```
  # firewall-cmd --list-rich-rule
  rule family="ipv4" service name="mountd" source address="192.168.0.0/24" invert="True"
  drop
  ```

```
rule family="ipv4" source address="127.0.0.1" service name="mountd" accept
```

Additional resources

- To learn more about firewall rules, see Chapter 5 Using and configuring firewalld .

CHAPTER 4. INSTALLING A RHEL 8 SYSTEM WITH FIPS MODE ENABLED

To enable the cryptographic module self-checks mandated by the Federal Information Processing Standard (FIPS) Publication 140-2, you have to operate RHEL 8 in FIPS mode.

You can achieve this by:

- Starting the installation in FIPS mode.

- Switching the system into FIPS mode after the installation.

To avoid cryptographic key material regeneration and reevaluation of the compliance of the resulting system associated with converting already deployed systems, Red Hat recommends starting the installation in FIPS mode.

4.1. FEDERAL INFORMATION PROCESSING STANDARD (FIPS)

The Federal Information Processing Standard (FIPS) Publication 140-2 is a computer security standard developed by the U.S. Government and industry working group to validate the quality of cryptographic modules. See the official FIPS publications at NIST Computer Security Resource Center .

The FIPS 140-2 standard ensures that cryptographic tools implement their algorithms correctly. One of the mechanisms for that is runtime self-checks. See the full FIPS 140-2 standard at FIPS PUB 140-2 for further details and other specifications of the FIPS standard.

To learn about compliance requirements, see the Red Hat Government Standards page.

4.2. INSTALLING THE SYSTEM WITH FIPS MODE ENABLED

To enable the cryptographic module self-checks mandated by the Federal Information Processing Standard (FIPS) Publication 140-2, enable FIPS mode during the system installation.

> **IMPORTANT**
>
> Red Hat recommends installing Red Hat Enterprise Linux 8 with FIPS mode enabled, as opposed to enabling FIPS mode later. Enabling FIPS mode during the installation ensures that the system generates all keys with FIPS-approved algorithms and continuous monitoring tests in place.

Procedure

- Add the **fips=1** option to the kernel command line during the system installation. During the software selection stage, do not install any third-party software.

After the installation, the system starts in FIPS mode automatically.

Verification

- After the system starts, check that FIPS mode is enabled:

  ```
  $ fips-mode-setup --check
  FIPS mode is enabled.
  ```

Additional resources

- Editing boot options section in the Performing an advanced RHEL installation

4.3. ADDITIONAL RESOURCES

- Switching the system to FIPS mode

- Enabling FIPS mode in a container

- List of RHEL 8 applications using cryptography that is not compliant with FIPS 140-2

CHAPTER 5. USING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES

Crypto policies is a system component that configures the core cryptographic subsystems, covering the TLS, IPSec, SSH, DNSSec, and Kerberos protocols. It provides a small set of policies, which the administrator can select.

5.1. SYSTEM-WIDE CRYPTOGRAPHIC POLICIES

Once a system-wide policy is set up, applications in RHEL follow it and refuse to use algorithms and protocols that do not meet the policy, unless you explicitly request the application to do so. That is, the policy applies to the default behavior of applications when running with the system-provided configuration but you can override it if required so.

Red Hat Enterprise Linux 8 contains the following policy levels:

DEFAULT	The default system-wide cryptographic policy level offers secure settings for current threat models. It allows the TLS 1.2 and 1.3 protocols, as well as the IKEv2 and SSH2 protocols. The RSA keys and Diffie-Hellman parameters are accepted if they are at least 2048 bits long.
LEGACY	This policy ensures maximum compatibility with Red Hat Enterprise Linux 5 and earlier; it is less secure due to an increased attack surface. In addition to the **DEFAULT** level algorithms and protocols, it includes support for the TLS 1.0 and 1.1 protocols. The algorithms DSA, 3DES, and RC4 are allowed, while RSA keys and Diffie-Hellman parameters are accepted if they are at least 1023 bits long.
FUTURE	A conservative security level that is believed to withstand any near-term future attacks. This level does not allow the use of SHA-1 in signature algorithms. The RSA keys and Diffie-Hellman parameters are accepted if they are at least 3072 bits long.
FIPS	A policy level that conforms with the FIPS 140-2 requirements. This is used internally by the **fips-mode-setup** tool, which switches the RHEL system into FIPS mode.

Red Hat continuously adjusts all policy levels so that all libraries, except when using the LEGACY policy, provide secure defaults. Even though the LEGACY profile does not provide secure defaults, it does not include any algorithms that are easily exploitable. As such, the set of enabled algorithms or acceptable key sizes in any provided policy may change during the lifetime of the Red Hat Enterprise Linux 8.

Such changes reflect new security standards and new security research. If you must ensure interoperability with a specific system for the whole lifetime of Red Hat Enterprise Linux 8, you should opt-out from cryptographic-policies for components that interact with that system.
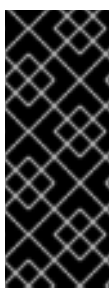
> **IMPORTANT**
>
> Because a cryptographic key used by a certificate on the Customer Portal API does not meet the requirements by the **FUTURE** system-wide cryptographic policy, the **redhat-support-tool** utility does not work with this policy level at the moment.
>
> To work around this problem, use the **DEFAULT** crypto policy while connecting to the Customer Portal API.

> **NOTE**
>
> The specific algorithms and ciphers described in the policy levels as allowed are available only if an application supports them.

Tool for managing crypto policies

To view or change the current system-wide cryptographic policy, use the **update-crypto-policies** tool, for example:

```
$ update-crypto-policies --show
DEFAULT
# update-crypto-policies --set FUTURE
Setting system policy to FUTURE
```

To ensure that the change of the cryptographic policy is applied, restart the system.

Strong crypto defaults by removing insecure cipher suites and protocols

The following list contains cipher suites and protocols removed from the core cryptographic libraries in Red Hat Enterprise Linux 8. They are not present in the sources, or their support is disabled during the build, so applications cannot use them.

- DES (since RHEL 7)

- All export grade cipher suites (since RHEL 7)

- MD5 in signatures (since RHEL 7)

- SSLv2 (since RHEL 7)

- SSLv3 (since RHEL 8)

- All ECC curves < 224 bits (since RHEL 6)

- All binary field ECC curves (since RHEL 6)

Cipher suites and protocols disabled in all policy levels

The following cipher suites and protocols are disabled in all crypto policy levels. They can be enabled only by an explicit configuration of individual applications.

- DH with parameters < 1024 bits

- RSA with key size < 1024 bits

- Camellia

- ARIA

- SEED

- IDEA

- Integrity-only cipher suites

- TLS CBC mode cipher suites using SHA-384 HMAC

- AES-CCM8

- All ECC curves incompatible with TLS 1.3, including secp256k1

- IKEv1 (since RHEL 8)

Cipher suites and protocols enabled in the crypto-policies levels

The following table shows the enabled cipher suites and protocols in all four crypto-policies levels.

	LEGACY	DEFAULT	FIPS	FUTURE
IKEv1	no	no	no	no
3DES	yes	no	no	no
RC4	yes	no	no	no
DH	min. 1024-bit	min. 2048-bit	min. 2048-bit	min. 3072-bit
RSA	min. 1024-bit	min. 2048-bit	min. 2048-bit	min. 3072-bit
DSA	yes	no	no	no
TLS v1.0	yes	no	no	no
TLS v1.1	yes	no	no	no
SHA-1 in digital signatures	yes	yes	no	no
CBC mode ciphers	yes	yes	yes	no[a]
Symmetric ciphers with keys < 256 bits	yes	yes	yes	no
SHA-1 and SHA-224 signatures in certificates	yes	yes	yes	no

[a] CBC ciphers are disabled for TLS. In a non-TLS scenario, **AES-128-CBC** is disabled but **AES-256-CBC** is enabled. To disable also **AES-256-CBC**, apply a custom subpolicy.

Additional resources

- **update-crypto-policies(8)** man page

5.2. SWITCHING THE SYSTEM-WIDE CRYPTOGRAPHIC POLICY TO MODE COMPATIBLE WITH EARLIER RELEASES

The default system-wide cryptographic policy in Red Hat Enterprise Linux 8 does not allow communication using older, insecure protocols. For environments that require to be compatible with Red Hat Enterprise Linux 5 and in some cases also with earlier releases, the less secure **LEGACY** policy level is available.

> ⚠️ **WARNING**
>
> Switching to the **LEGACY** policy level results in a less secure system and applications.

Procedure

1. To switch the system-wide cryptographic policy to the **LEGACY** level, enter the following command as **root**:

   ```
   # update-crypto-policies --set LEGACY
   Setting system policy to LEGACY
   ```

Additional resources

- For the list of available cryptographic policy levels, see the **update-crypto-policies(8)** man page.

5.3. SWITCHING THE SYSTEM TO FIPS MODE

The system-wide cryptographic policies contain a policy level that enables cryptographic modules self-checks in accordance with the requirements by the Federal Information Processing Standard (FIPS) Publication 140-2. The **fips-mode-setup** tool that enables or disables FIPS mode internally uses the **FIPS** system-wide cryptographic policy level.

> **IMPORTANT**
>
> Red Hat recommends installing Red Hat Enterprise Linux 8 with FIPS mode enabled, as opposed to enabling FIPS mode later. Enabling FIPS mode during the installation ensures that the system generates all keys with FIPS-approved algorithms and continuous monitoring tests in place.

Procedure

1. To switch the system to FIPS mode in RHEL 8:

   ```
   # fips-mode-setup --enable
   Setting system policy to FIPS
   FIPS mode will be enabled.
   Please reboot the system for the setting to take effect.
   ```

2. Restart your system to allow the kernel to switch to FIPS mode:

   ```
   # reboot
   ```

Verification

1. After the restart, you can check the current state of FIPS mode:

   ```
   # fips-mode-setup --check
   FIPS mode is enabled.
   ```

Additional resources

- **fips-mode-setup(8)** man page

- List of RHEL 8 applications using cryptography that are not compliant with FIPS 140-2

- Security Requirements for Cryptographic Modules on the National Institute of Standards and Technology (NIST) web site.

5.4. ENABLING FIPS MODE IN A CONTAINER

In RHEL 8.3 and later versions, you do not need to manually enable cryptographic modules self-checks in accordance with the requirements by Federal Information Processing Standard (FIPS) Publication 140-2. On systems with FIPS mode enabled, the **podman** utility automatically configures containers to FIPS mode.
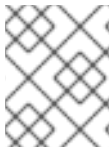
> **NOTE**
>
> In RHEL 8, the **fips-mode-setup** command does not work correctly in containers, and it cannot be used to enable or check FIPS mode in this scenario.

5.4.1. Enabling FIPS mode in a container in RHEL 8.2

In RHEL 8.2 and later versions, you can manually switch a container to FIPS mode by using only a single command in the container. Note that the host system must be in FIPS mode, see Switching the system to FIPS mode.

```
# mount --bind /usr/share/crypto-policies/back-ends/FIPS /etc/crypto-policies/back-ends
```

5.4.2. Enabling FIPS mode in a container in RHEL 8.1 and earlier

In RHEL 8.1 and earlier versions, to enable cryptographic modules self-checks in accordance with the requirements by Federal Information Processing Standard (FIPS) Publication 140-2 in a container:

Prerequisites

- The host system must be in FIPS mode, see Switching the system to FIPS mode .

Procedure

1. Mount the **/etc/system-fips** file on the container from the host.

2. Set the FIPS cryptographic policy level in the container:

```
$ update-crypto-policies --set FIPS
```

5.5. LIST OF RHEL APPLICATIONS USING CRYPTOGRAPHY THAT IS NOT COMPLIANT WITH FIPS 140-2

Red Hat recommends to utilize libraries from the core crypto components set, as they are guaranteed to pass all relevant crypto certifications, such as FIPS 140-2, and also follow the RHEL system-wide crypto policies.

See the RHEL 8 core crypto components article for an overview of the RHEL 8 core crypto components, the information on how are they selected, how are they integrated into the operating system, how do they support hardware security modules and smart cards, and how do crypto certifications apply to them.

In addition to the following table, in some RHEL 8 Z-stream releases (for example, 8.1.1), the Firefox browser packages have been updated, and they contain a separate copy of the NSS cryptography library. This way, Red Hat wants to avoid the disruption of rebasing such a low-level component in a patch release. As a result, these Firefox packages do not use a FIPS 140-2-validated module.

Table 5.1. List of RHEL 8 applications using cryptography that is not compliant with FIPS 140-2

Application	Details
FreeRADIUS	The RADIUS protocol uses MD5
ghostscript	Custom cryptogtaphy implementation (MD5, RC4, SHA-2, AES) to encrypt and decrypt documents
Grafana	Cryptographic implementations from the Golang **x/crypto** module (Ed25519, CBC, OCFB, ...)
ipxe	Crypto stack for TLS is compiled in, however, it is unused
libica	Software fallbacks for various algorithms such as RSA and ECDH through CPACF instructions
Ovmf (UEFI firmware), Edk2, shim	Full crypto stack (an embedded copy of the OpenSSL library)
perl-Digest-HMAC	HMAC, HMAC-SHA1, HMAC-MD5
perl-Digest-SHA	SHA-1, SHA-224, ...
pidgin	DES, RC4
qatengine	Mixed hardware and software implementation of cryptographic primitives (RSA, EC, DH, AES, ...)
samba[a]	AES, DES, RC4

Application	Details
valgrind	AES, hashes[b]

[a] Starting with RHEL 8.3, samba uses FIPS-compliant cryptography.

[b] Re-implements in software hardware-offload operations, such as AES-NI.

5.6. EXCLUDING AN APPLICATION FROM FOLLOWING SYSTEM-WIDE CRYPTO POLICIES

You can customize cryptographic settings used by your application preferably by configuring supported cipher suites and protocols directly in the application.

You can also remove a symlink related to your application from the **/etc/crypto-policies/back-ends** directory and replace it with your customized cryptographic settings. This configuration prevents the use of system-wide cryptographic policies for applications that use the excluded back end. Furthermore, this modification is not supported by Red Hat.

5.6.1. Examples of opting out of system-wide crypto policies

wget

To customize cryptographic settings used by the **wget** network downloader, use **--secure-protocol** and **--ciphers** options. For example:

```
$ wget --secure-protocol=TLSv1_1 --ciphers="SECURE128" https://example.com
```

See the HTTPS (SSL/TLS) Options section of the **wget(1)** man page for more information.

curl

To specify ciphers used by the **curl** tool, use the **--ciphers** option and provide a colon-separated list of ciphers as a value. For example:

```
$ curl https://example.com --ciphers '@SECLEVEL=0:DES-CBC3-SHA:RSA-DES-CBC3-SHA'
```

See the **curl(1)** man page for more information.

Firefox

Even though you cannot opt out of system-wide cryptographic policies in the **Firefox** web browser, you can further restrict supported ciphers and TLS versions in Firefox's Configuration Editor. Type **about:config** in the address bar and change the value of the **security.tls.version.min** option as required. Setting **security.tls.version.min** to **1** allows TLS 1.0 as the minimum required, **security.tls.version.min 2** enables TLS 1.1, and so on.

OpenSSH

To opt out of the system-wide crypto policies for your **OpenSSH** server, uncomment the line with the **CRYPTO_POLICY=** variable in the /etc/sysconfig/sshd file. After this change, values that you specify in the **Ciphers**, **MACs**, **KexAlgoritms**, and **GSSAPIKexAlgorithms** sections in the /etc/ssh/sshd_config file are not overridden. See the **sshd_config(5)** man page for more information.

To opt out of system-wide crypto policies for your **OpenSSH** client, perform one of the following tasks:

- For a given user, override the global **ssh_config** with a user-specific configuration in the ~/**.ssh/config** file.

- For the entire system, specify the crypto policy in a drop-in configuration file located in the /**etc/ssh/ssh_config.d/** directory, with a two-digit number prefix smaller than 50, so that it lexicographically precedes the **50-redhat.conf** file, and with a **.conf** suffix, for example, **49-crypto-policy-override.conf**.

See the **ssh_config(5)** man page for more information.

Libreswan

See the Configuring IPsec connections that opt out of the system-wide crypto policies in the Securing networks document for detailed information.

Additional resources

- **update-crypto-policies(8)** man page

5.7. CUSTOMIZING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES WITH POLICY MODIFIERS

Use this procedure to adjust certain algorithms or protocols of any system-wide cryptographic policy level or a full custom policy.
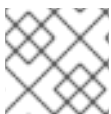
> **NOTE**
>
> Customization of system-wide cryptographic policies is available from RHEL 8.2.

Procedure

1. Checkout to the /**etc/crypto-policies/policies/modules/** directory:

   ```
   # cd /etc/crypto-policies/policies/modules/
   ```

2. Create policy modules for your adjustments, for example:

   ```
   # touch MYCRYPTO1.pmod
   # touch NO-AES128.pmod
   ```
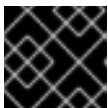
   > **IMPORTANT**
   >
   > Use upper-case letters in file names of policy modules.

3. Open the policy modules in a text editor of your choice and insert options that modify the system-wide cryptographic policy, for example:

   ```
   # vi MYCRYPTO1.pmod
   ```

   ```
   sha1_in_certs = 0
   min_rsa_size = 3072
   ```

```
# vi NO-AES128.pmod
```

```
cipher = -AES-128-GCM -AES-128-CCM -AES-128-CTR -AES-128-CBC
```

4. Save the changes in the module files.

5. Apply your policy adjustments to the **DEFAULT** system-wide cryptographic policy level:

```
# update-crypto-policies --set DEFAULT:MYCRYPTO1:NO-AES128
```

6. To make your cryptographic settings effective for already running services and applications, restart the system:

```
# reboot
```

Additional resources

- **Custom Policies** section in the **update-crypto-policies(8)** man page and the **Crypto Policy Definition Format** section in the **crypto-policies(7)** man page

- How to customize crypto policies in RHEL 8.2 Red Hat blog article

5.8. DISABLING SHA-1 BY CUSTOMIZING A SYSTEM-WIDE CRYPTOGRAPHIC POLICY

Because the SHA-1 hash function has an inherently weak design, and advancing cryptanalysis has made it vulnerable to attacks, RHEL 8 does not use SHA-1 by default. Nevertheless, some third party applications, for example public signatures, still use SHA-1. To disable the use of SHA-1 in signature algorithms on your system, you can use the **NO-SHA1** policy module.
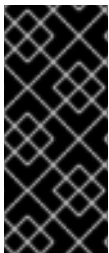
IMPORTANT

The **NO-SHA1** policy module disables the SHA-1 hash function only in signatures and not elsewhere. In particular, the **NO-SHA1** module still allows the use of SHA-1 with hash-based message authentication codes (HMAC). This is because HMAC security properties do not rely on collision resistance of the corresponding hash function, and therefore the recent attacks on SHA-1 have a significantly lower impact on the use of SHA-1 for HMAC.

NOTE

The module for disabling SHA-1 is available from RHEL 8.3. Customization of system-wide cryptographic policies is available from RHEL 8.2.

Procedure

1. Apply your policy adjustments to the **DEFAULT** system-wide cryptographic policy level:

```
# update-crypto-policies --set DEFAULT:NO-SHA1
```

2. To make your cryptographic settings effective for already running services and applications, restart the system:

```
# reboot
```

Additional resources

- **Custom Policies** section in the **update-crypto-policies(8)** man page.

- The **Crypto Policy Definition Format** section in the **crypto-policies(7)** man page.

- How to customize crypto policies in RHEL 8.2 Red Hat blog article.

5.9. CREATING AND SETTING A CUSTOM SYSTEM-WIDE CRYPTOGRAPHIC POLICY

The following steps demonstrate customizing the system-wide cryptographic policies by a complete policy file.
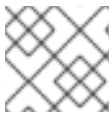
NOTE

Customization of system-wide cryptographic policies is available from RHEL 8.2.

Procedure

1. Create a policy file for your customizations:

   ```
   # cd /etc/crypto-policies/policies/
   # touch MYPOLICY.pol
   ```

   Alternatively, start by copying one of the four predefined policy levels:

   ```
   # cp /usr/share/crypto-policies/policies/DEFAULT.pol /etc/crypto-policies/policies/MYPOLICY.pol
   ```

2. Edit the file with your custom cryptographic policy in a text editor of your choice to fit your requirements, for example:

   ```
   # vi /etc/crypto-policies/policies/MYPOLICY.pol
   ```

3. Switch the system-wide cryptographic policy to your custom level:

   ```
   # update-crypto-policies --set MYPOLICY
   ```

4. To make your cryptographic settings effective for already running services and applications, restart the system:

   ```
   # reboot
   ```

Additional resources

- **Custom Policies** section in the **update-crypto-policies(8)** man page and the **Crypto Policy Definition Format** section in the **crypto-policies(7)** man page

- How to customize crypto policies in RHEL 8.2 Red Hat blog article

5.10. RELATED INFORMATION

- System-wide crypto policies in RHEL 8 and Strong crypto defaults in RHEL 8 and deprecation of weak crypto algorithms Knowledgebase articles

CHAPTER 6. SETTING A CUSTOM CRYPTOGRAPHIC POLICY ACROSS SYSTEMS

As an administrator, you can use the Crypto Policies System Role on RHEL to quickly and consistently configure custom cryptographic policies across many different systems using Red Hat Ansible Automation Platform.

6.1. CRYPTO POLICIES SYSTEM ROLE VARIABLES AND FACTS

In a Crypto Policies System Role playbook, you can define the parameters for the crypto policies configuration file according to your preferences and limitations.

If you do not configure any variables, the system role does not configure the system and only reports the facts.

Selected variables for the Crypto Policies System Role

crypto_policies_policy

Determines the cryptographic policy level the system role applies to the managed nodes. For details about the different crypto policy levels, see System-wide cryptographic policies .

crypto_policies_reload

If set to **yes**, the affected services, currently the **ipsec**, **bind**, and **sshd** services, reload after applying a crypto policy. Defaults to **yes**.

crypto_policies_reboot_ok

If set to **yes**, and a reboot is necessary after the system role changes the crypto policy, it sets **crypto_policies_reboot_required** to **yes**. Defaults to **no**.

Facts set by the Crypto Policies System Role

crypto_policies_active

Lists the currently selected policy.

crypto_policies_available_policies

Lists all available policy levels available on the system.

crypto_policies_available_modules

Lists all available subpolicy modules available on the system.

Additional resources

* Creating and setting a custom system-wide cryptographic policy .

6.2. SETTING A CUSTOM CRYPTOGRAPHIC POLICY USING THE CRYPTO POLICIES SYSTEM ROLE

You can use the Crypto Policies System Role to configure a large number of managed nodes consistently from a single control node.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the Crypto Policies System Role.

- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Engine configures other systems.
  On the control node:

  - Red Hat Ansible Engine is installed

  - The **rhel-system-roles** package is installed

  - An inventory file which lists the managed nodes.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

   ```
   ---
   - hosts: all
     tasks:
     - name: Configure crypto policies
       include_role:
         name: linux-system-roles.crypto_policies
       vars:
         - crypto_policies_policy: FUTURE
         - crypto_policies_reboot_ok: true
   ```

   You can replace the *FUTURE* value with your preferred crypto policy, for example: **DEFAULT**, **LEGACY**, and **FIPS:OSPP**.

   The **crypto_policies_reboot_ok: true** variable causes the system to reboot after the system role changes the crypto policy.

   For more details, see Crypto Policies System Role variables and facts .

2. Optional: Verify playbook syntax.

   ```
   # ansible-playbook --syntax-check playbook.yml
   ```

3. Run the playbook on your inventory file:

   ```
   # ansible-playbook -i inventory_file playbook.yml
   ```

Verification

1. On the control node, create another playbook named, for example, ***verify_playbook.yml***:

   ```
   - hosts: all
     tasks:
     - name: Verify active crypto policy
       include_role:
         name: linux-system-roles.crypto_policies

     - debug:
         var: crypto_policies_active
   ```

This playbook does not change any configurations on the system, only reports the active policy on the managed nodes.

2. Run the playbook on the same inventory file:

```
# ansible-playbook -i inventory_file verify_playbook.yml

TASK [debug] **************************
ok: [host] => {
    "crypto_policies_active": "FUTURE"
}
```

The **"crypto_policies_active":** variable shows the policy active on the managed node.

6.3. ADDITIONAL RESOURCES

- **/usr/share/ansible/roles/rhel-system-roles.crypto_policies/README.md** file.

- **ansible-playbook(1)** man page.

- Installing RHEL System Roles .

- Applying a system role .

CHAPTER 7. CONFIGURING APPLICATIONS TO USE CRYPTOGRAPHIC HARDWARE THROUGH PKCS #11

Separating parts of your secret information on dedicated cryptographic devices, such as smart cards and cryptographic tokens for end-user authentication and hardware security modules (HSM) for server applications, provides an additional layer of security. In Red Hat Enterprise Linux 8, support for cryptographic hardware through the PKCS #11 API is consistent across different applications, and the isolation of secrets on cryptographic hardware is not a complicated task.

7.1. CRYPTOGRAPHIC HARDWARE SUPPORT THROUGH PKCS #11

PKCS #11 (Public-Key Cryptography Standard) defines an application programming interface (API) to cryptographic devices that hold cryptographic information and perform cryptographic functions. These devices are called tokens, and they can be implemented in a hardware or software form.

A PKCS #11 token can store various object types including a certificate; a data object; and a public, private, or secret key. These objects are uniquely identifiable through the PKCS #11 URI scheme.

A PKCS #11 URI is a standard way to identify a specific object in a PKCS #11 module according to the object attributes. This enables you to configure all libraries and applications with the same configuration string in the form of a URI.

Red Hat Enterprise Linux 8 provides the OpenSC PKCS #11 driver for smart cards by default. However, hardware tokens and HSMs can have their own PKCS #11 modules that do not have their counterpart in Red Hat Enterprise Linux. You can register such PKCS #11 modules with the **p11-kit** tool, which acts as a wrapper over the registered smart card drivers in the system.

To make your own PKCS #11 module work on the system, add a new text file to the **/etc/pkcs11/modules/** directory

You can add your own PKCS #11 module into the system by creating a new text file in the **/etc/pkcs11/modules/** directory. For example, the OpenSC configuration file in **p11-kit** looks as follows:

```
$ cat /usr/share/p11-kit/modules/opensc.module
module: opensc-pkcs11.so
```

Additional resources

- Consistent PKCS #11 support in Red Hat Enterprise Linux 8

- The PKCS #11 URI Scheme

- Controlling access to smart cards

7.2. USING SSH KEYS STORED ON A SMART CARD

Red Hat Enterprise Linux enables you to use RSA and ECDSA keys stored on a smart card on OpenSSH clients. Use this procedure to enable authentication using a smart card instead of using a password.

Prerequisites

- On the client side, the **opensc** package is installed and the **pcscd** service is running.

Procedure

1. List all keys provided by the OpenSC PKCS #11 module including their PKCS #11 URIs and save the output to the *keys.pub* file:

   ```
   $ ssh-keygen -D pkcs11: > keys.pub
   $ ssh-keygen -D pkcs11:
   ssh-rsa AAAAB3NzaC1yc2E...KKZMzcQZzx
   pkcs11:id=%02;object=SIGN%20pubkey;token=SSH%20key;manufacturer=piv_II?module-
   path=/usr/lib64/pkcs11/opensc-pkcs11.so
   ecdsa-sha2-nistp256 AAA...J0hkYnnsM=
   pkcs11:id=%01;object=PIV%20AUTH%20pubkey;token=SSH%20key;manufacturer=piv_II?
   module-path=/usr/lib64/pkcs11/opensc-pkcs11.so
   ```

2. To enable authentication using a smart card on a remote server (*example.com*), transfer the public key to the remote server. Use the **ssh-copy-id** command with *keys.pub* created in the previous step:

   ```
   $ ssh-copy-id -f -i keys.pub username@example.com
   ```

3. To connect to *example.com* using the ECDSA key from the output of the **ssh-keygen -D** command in step 1, you can use just a subset of the URI, which uniquely references your key, for example:

   ```
   $ ssh -i "pkcs11:id=%01?module-path=/usr/lib64/pkcs11/opensc-pkcs11.so" example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

4. You can use the same URI string in the **~/.ssh/config** file to make the configuration permanent:

   ```
   $ cat ~/.ssh/config
   IdentityFile "pkcs11:id=%01?module-path=/usr/lib64/pkcs11/opensc-pkcs11.so"
   $ ssh example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

   Because OpenSSH uses the **p11-kit-proxy** wrapper and the OpenSC PKCS #11 module is registered to PKCS#11 Kit, you can simplify the previous commands:

   ```
   $ ssh -i "pkcs11:id=%01" example.com
   Enter PIN for 'SSH key':
   [example.com] $
   ```

If you skip the **id=** part of a PKCS #11 URI, OpenSSH loads all keys that are available in the proxy module. This can reduce the amount of typing required:

```
$ ssh -i pkcs11: example.com
Enter PIN for 'SSH key':
[example.com] $
```

Additional resources

- Fedora 28: Better smart card support in OpenSSH .

- **p11-kit(8)**, **opensc.conf(5)**, **pcscd(8)**, **ssh(1)**, and **ssh-keygen(1)** man pages.

7.3. CONFIGURING APPLICATIONS TO AUTHENTICATE USING CERTIFICATES FROM SMART CARDS

- The **wget** network downloader enables you to specify PKCS #11 URIs instead of paths to locally stored private keys, and thus simplifies creating scripts for tasks that require safely stored private keys and certificates. For example:

  ```
  $ wget --private-key 'pkcs11:token=softhsm;id=%01;type=private?pin-value=111111' --certificate 'pkcs11:token=softhsm;id=%01;type=cert' https://example.com/
  ```

  See the **wget(1)** man page for more information.

- Specifying PKCS #11 URI for use by the **curl** tool is analogous:

  ```
  $ curl --key 'pkcs11:token=softhsm;id=%01;type=private?pin-value=111111' --cert 'pkcs11:token=softhsm;id=%01;type=cert' https://example.com/
  ```

  See the **curl(1)** man page for more information.

- The **Firefox** web browser automatically loads the **p11-kit-proxy** module. This means that every supported smart card in the system is automatically detected. For using TLS client authentication, no additional setup is required and keys from a smart card are automatically used when a server requests them.

Using PKCS #11 URIs in custom applications

If your application uses the **GnuTLS** or **NSS** library, support for PKCS #11 URIs is ensured by their built-in support for PKCS #11. Also, applications relying on the **OpenSSL** library can access cryptographic hardware modules thanks to the **openssl-pkcs11** engine.

With applications that require working with private keys on smart cards and that do not use **NSS**, **GnuTLS**, and **OpenSSL**, use **p11-kit** to implement registering PKCS #11 modules.

Additional resources

- **p11-kit(8)** man page.

7.4. USING HSMS PROTECTING PRIVATE KEYS IN APACHE

The **Apache** HTTP server can work with private keys stored on hardware security modules (HSMs), which helps to prevent the keys' disclosure and man-in-the-middle attacks. Note that this usually requires high-performance HSMs for busy servers.

For secure communication in the form of the HTTPS protocol, the **Apache** HTTP server (**httpd**) uses the OpenSSL library. OpenSSL does not support PKCS #11 natively. To utilize HSMs, you have to install the **openssl-pkcs11** package, which provides access to PKCS #11 modules through the engine interface. You can use a PKCS #11 URI instead of a regular file name to specify a server key and a certificate in the **/etc/httpd/conf.d/ssl.conf** configuration file, for example:

```
SSLCertificateFile    "pkcs11:id=%01;token=softhsm;type=cert"
SSLCertificateKeyFile "pkcs11:id=%01;token=softhsm;type=private?pin-value=111111"
```

Install the **httpd-manual** package to obtain complete documentation for the **Apache** HTTP Server, including TLS configuration. The directives available in the **/etc/httpd/conf.d/ssl.conf** configuration file are described in detail in /usr/share/httpd/manual/mod/mod_ssl.html.

7.5. USING HSMS PROTECTING PRIVATE KEYS IN NGINX

The **Nginx** HTTP server can work with private keys stored on hardware security modules (HSMs), which helps to prevent the keys' disclosure and man-in-the-middle attacks. Note that this usually requires high-performance HSMs for busy servers.

Because **Nginx** also uses the OpenSSL for cryptographic operations, support for PKCS #11 must go through the **openssl-pkcs11** engine. **Nginx** currently supports only loading private keys from an HSM, and a certificate must be provided separately as a regular file. Modify the **ssl_certificate** and **ssl_certificate_key** options in the **server** section of the **/etc/nginx/nginx.conf** configuration file:

```
ssl_certificate     /path/to/cert.pem
ssl_certificate_key "engine:pkcs11:pkcs11:token=softhsm;id=%01;type=private?pin-value=111111";
```

Note that the **engine:pkcs11:** prefix is needed for the PKCS #11 URI in the **Nginx** configuration file. This is because the other **pkcs11** prefix refers to the engine name.

7.6. RELATED INFORMATION

- **pkcs11.conf(5)** man page.

CHAPTER 8. USING SHARED SYSTEM CERTIFICATES

The shared system certificates storage enables NSS, GnuTLS, OpenSSL, and Java to share a default source for retrieving system certificate anchors and block-list information. By default, the trust store contains the Mozilla CA list, including positive and negative trust. The system allows updating the core Mozilla CA list or choosing another certificate list.

8.1. THE SYSTEM-WIDE TRUST STORE

In Red Hat Enterprise Linux, the consolidated system-wide trust store is located in the **/etc/pki/ca-trust/** and **/usr/share/pki/ca-trust-source/** directories. The trust settings in **/usr/share/pki/ca-trust-source/** are processed with lower priority than settings in **/etc/pki/ca-trust/**.

Certificate files are treated depending on the subdirectory they are installed to the following directories:

- for trust anchors

    - **/usr/share/pki/ca-trust-source/anchors/** or

    - **/etc/pki/ca-trust/source/anchors/**

- for distrusted certificates

    - **/usr/share/pki/ca-trust-source/blacklist/** or

    - **/etc/pki/ca-trust/source/blacklist/**

- for certificates in the extended BEGIN TRUSTED file format

    - **/usr/share/pki/ca-trust-source/** or

    - **/etc/pki/ca-trust/source/**

> **NOTE**
>
> In a hierarchical cryptographic system, a trust anchor is an authoritative entity which other parties consider being trustworthy. In the X.509 architecture, a root certificate is a trust anchor from which a chain of trust is derived. To enable chain validation, the trusting party must have access to the trust anchor first.

8.2. ADDING NEW CERTIFICATES

To acknowledge applications on your system with a new source of trust, add the corresponding certificate to the system-wide store, and use the **update-ca-trust** command.

Prerequisites

- The **ca-certificates** package is present on the system.

Procedure

1. To add a certificate in the simple PEM or DER file formats to the list of CAs trusted on the system, copy the certificate file to the **/usr/share/pki/ca-trust-source/anchors/** or **/etc/pki/ca-trust/source/anchors/** directory, for example:

```
# cp ~/certificate-trust-examples/Cert-trust-test-ca.pem /usr/share/pki/ca-trust-
source/anchors/
```

2. To update the system-wide trust store configuration, use the **update-ca-trust** command:

```
# update-ca-trust
```
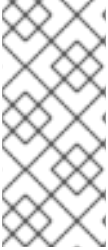
> **NOTE**
>
> While the Firefox browser is able to use an added certificate without executing **update-ca-trust**, Red Hat recommends to use the **update-ca-trust** command after a CA change. Also note that browsers, such as Firefox, Epiphany, or Chromium, cache files, and you might have to clear browser's cache or restart your browser to load the current system certificates configuration.

8.3. MANAGING TRUSTED SYSTEM CERTIFICATES

The **trust** command provides a convenient way for managing certificates in the shared system-wide trust store.

- To list, extract, add, remove, or change trust anchors, use the **trust** command. To see the built-in help for this command, enter it without any arguments or with the **--help** directive:

```
$ trust
usage: trust command <args>...

Common trust commands are:
  list           List trust or certificates
  extract        Extract certificates and trust
  extract-compat   Extract trust compatibility bundles
  anchor         Add, remove, change trust anchors
  dump           Dump trust objects in internal format

See 'trust <command> --help' for more information
```

- To list all system trust anchors and certificates, use the **trust list** command:

```
$ trust list
pkcs11:id=%d2%87%b4%e3%df%37%27%93%55%f6%56%ea%81%e5%36%cc%8c%1e%3
f%bd;type=cert
    type: certificate
    label: ACCVRAIZ1
    trust: anchor
    category: authority

pkcs11:id=%a6%b3%e1%2b%2b%49%b6%d7%73%a1%aa%94%f5%01%e7%73%65%4c%
ac%50;type=cert
    type: certificate
    label: ACEDICOM Root
    trust: anchor
    category: authority
...
```

- To store a trust anchor into the system-wide trust store, use the **trust anchor** sub-command and specify a path to a certificate. Replace *path.to/certificate.crt* by a path to your certificate and its file name:

  ```
  # trust anchor path.to/certificate.crt
  ```

- To remove a certificate, use either a path to a certificate or an ID of a certificate:

  ```
  # trust anchor --remove path.to/certificate.crt
  # trust anchor --remove "pkcs11:id=%AA%BB%CC%DD%EE;type=cert"
  ```

Additional resources

- All sub-commands of the **trust** commands offer a detailed built-in help, for example:

  ```
  $ trust list --help
  usage: trust list --filter=<what>

    --filter=<what>     filter of what to export
                    ca-anchors        certificate anchors
  ...
    --purpose=<usage>   limit to certificates usable for the purpose
                    server-auth       for authenticating servers
  ...
  ```

8.4. ADDITIONAL RESOURCES

- **update-ca-trust(8)** and **trust(1)** man pages

CHAPTER 9. SCANNING THE SYSTEM FOR CONFIGURATION COMPLIANCE AND VULNERABILITIES

A compliance audit is a process of determining whether a given object follows all the rules specified in a compliance policy. The compliance policy is defined by security professionals who specify the required settings, often in the form of a checklist, that a computing environment should use.

Compliance policies can vary substantially across organizations and even across different systems within the same organization. Differences among these policies are based on the purpose of each system and its importance for the organization. Custom software settings and deployment characteristics also raise a need for custom policy checklists.

9.1. CONFIGURATION COMPLIANCE TOOLS IN RHEL

Red Hat Enterprise Linux provides tools that enable you to perform a fully automated compliance audit. These tools are based on the Security Content Automation Protocol (SCAP) standard and are designed for automated tailoring of compliance policies.

- **SCAP Workbench** - The **scap-workbench** graphical utility is designed to perform configuration and vulnerability scans on a single local or remote system. You can also use it to generate security reports based on these scans and evaluations.

- **OpenSCAP** - The **OpenSCAP** library, with the accompanying **oscap** command-line utility, is designed to perform configuration and vulnerability scans on a local system, to validate configuration compliance content, and to generate reports and guides based on these scans and evaluations.

- **SCAP Security Guide (SSG)** - The **scap-security-guide** package provides the latest collection of security policies for Linux systems. The guidance consists of a catalog of practical hardening advice, linked to government requirements where applicable. The project bridges the gap between generalized policy requirements and specific implementation guidelines.

- **Script Check Engine (SCE)** - SCE is an extension to the SCAP protocol that enables administrators to write their security content using a scripting language, such as Bash, Python, and Ruby. The SCE extension is provided in the **openscap-engine-sce** package. The SCE itself is not part of the SCAP standard.

To perform automated compliance audits on multiple systems remotely, you can use the OpenSCAP solution for Red Hat Satellite.

Additional resources

- **oscap(8)**, **scap-workbench(8)**, and **scap-security-guide(8)** man pages

- Red Hat Security Demos: Creating Customized Security Policy Content to Automate Security Compliance

- Red Hat Security Demos: Defend Yourself with RHEL Security Technologies

- Security Compliance Management in the Administering Red Hat Satellite Guide .

9.2. VULNERABILITY SCANNING

9.2.1. Red Hat Security Advisories OVAL feed

Red Hat Enterprise Linux security auditing capabilities are based on the Security Content Automation Protocol (SCAP) standard. SCAP is a multi-purpose framework of specifications that supports automated configuration, vulnerability and patch checking, technical control compliance activities, and security measurement.

SCAP specifications create an ecosystem where the format of security content is well-known and standardized although the implementation of the scanner or policy editor is not mandated. This enables organizations to build their security policy (SCAP content) once, no matter how many security vendors they employ.

The Open Vulnerability Assessment Language (OVAL) is the essential and oldest component of SCAP. Unlike other tools and custom scripts, OVAL describes a required state of resources in a declarative manner. OVAL code is never executed directly but using an OVAL interpreter tool called scanner. The declarative nature of OVAL ensures that the state of the assessed system is not accidentally modified.

Like all other SCAP components, OVAL is based on XML. The SCAP standard defines several document formats. Each of them includes a different kind of information and serves a different purpose.

Red Hat Product Security helps customers evaluate and manage risk by tracking and investigating all security issues affecting Red Hat customers. It provides timely and concise patches and security advisories on the Red Hat Customer Portal. Red Hat creates and supports OVAL patch definitions, providing machine-readable versions of our security advisories.

Because of differences between platforms, versions, and other factors, Red Hat Product Security qualitative severity ratings of vulnerabilities do not directly align with the Common Vulnerability Scoring System (CVSS) baseline ratings provided by third parties. Therefore, we recommend that you use the RHSA OVAL definitions instead of those provided by third parties.

The RHSA OVAL definitions are available individually and as a complete package, and are updated within an hour of a new security advisory being made available on the Red Hat Customer Portal.

Each OVAL patch definition maps one-to-one to a Red Hat Security Advisory (RHSA). Because an RHSA can contain fixes for multiple vulnerabilities, each vulnerability is listed separately by its Common Vulnerabilities and Exposures (CVE) name and has a link to its entry in our public bug database.

The RHSA OVAL definitions are designed to check for vulnerable versions of RPM packages installed on a system. It is possible to extend these definitions to include further checks, for example, to find out if the packages are being used in a vulnerable configuration. These definitions are designed to cover software and updates shipped by Red Hat. Additional definitions are required to detect the patch status of third-party software.

Additional resources

- Red Hat and OVAL compatibility

- Red Hat and CVE compatibility

- Notifications and Advisories in the Product Security Overview

- Security Data Metrics

9.2.2. Scanning the system for vulnerabilities

The **oscap** command-line utility enables you to scan local systems, validate configuration compliance content, and generate reports and guides based on these scans and evaluations. This utility serves as a front end to the OpenSCAP library and groups its functionalities to modules (sub-commands) based on

the type of SCAP content it processes.

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. Install the **openscap-scanner** and **bzip2** packages:

   ```
   # yum install openscap-scanner bzip2
   ```

2. Download the latest RHSA OVAL definitions for your system:

   ```
   # wget -O - https://www.redhat.com/security/data/oval/v2/RHEL8/rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.oval.xml
   ```

3. Scan the system for vulnerabilities and save results to the *vulnerability.html* file:

   ```
   # oscap oval eval --report vulnerability.html rhel-8.oval.xml
   ```

Verification

1. Check the results in a browser of your choice, for example:

   ```
   $ firefox vulnerability.html &
   ```

Additional resources

- **oscap(8)** man page

- [Red Hat OVAL definitions](#)

9.2.3. Scanning remote systems for vulnerabilities

You can check also remote systems for vulnerabilities with the OpenSCAP scanner using the **oscap-ssh** tool over the SSH protocol.

Prerequisites

- The **AppStream** repository is enabled.

- The **openscap-scanner** package is installed on the remote systems.

- The SSH server is running on the remote systems.

Procedure

1. Install the **openscap-utils** and **bzip2** packages:

   ```
   # yum install openscap-utils bzip2
   ```

2. Download the latest RHSA OVAL definitions for your system:

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL8/rhel-8.oval.xml.bz2 | bzip2 --decompress > rhel-8.oval.xml
```

3. Scan a remote system with the *machine1* host name, SSH running on port 22, and the *joesec* user name for vulnerabilities and save results to the *remote-vulnerability.html* file:

```
# oscap-ssh joesec@machine1 22 oval eval --report remote-vulnerability.html rhel-8.oval.xml
```

Additional resources

- **oscap-ssh(8)**

- Red Hat OVAL definitions

9.3. CONFIGURATION COMPLIANCE SCANNING

9.3.1. Configuration compliance in RHEL 8

You can use configuration compliance scanning to conform to a baseline defined by a specific organization. For example, if you work with the US government, you might have to comply with the Operating System Protection Profile (OSPP), and if you are a payment processor, you might have to be compliant with the Payment Card Industry Data Security Standard (PCI-DSS). You can also perform configuration compliance scanning to harden your system security.

Red Hat recommends you follow the Security Content Automation Protocol (SCAP) content provided in the SCAP Security Guide package because it is in line with Red Hat best practices for affected components.

The SCAP Security Guide package provides content which conforms to the SCAP 1.2 and SCAP 1.3 standards. The **openscap scanner** utility is compatible with both SCAP 1.2 and SCAP 1.3 content provided in the SCAP Security Guide package.

> **IMPORTANT**
>
> Performing a configuration compliance scanning does not guarantee the system is compliant.

The SCAP Security Guide suite provides profiles for several platforms in a form of data stream documents. A data stream is a file that contains definitions, benchmarks, profiles, and individual rules. Each rule specifies the applicability and requirements for compliance. RHEL 8 provides several profiles for compliance with security policies. In addition to the industry standard, Red Hat data streams also contain information for remediation of failed rules.

Structure of compliance scanning resources

```
Data stream
  ├── xccdf
  │    ├── benchmark
  │    │    ├── profile
  │    │    │    ├──rule reference
  │    │    │    └──variable
```

```
|              ├── rule
|                   ├── human readable data
|                   ├── oval reference
├── oval            ├── ocil reference
├── ocil            ├── cpe reference
└── cpe             └── remediation
```
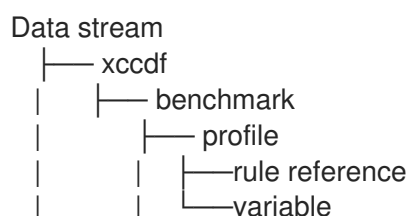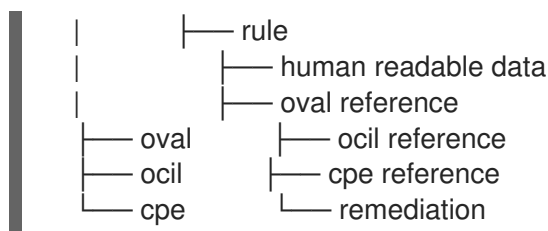
A profile is a set of rules based on a security policy, such as OSPP, PCI-DSS, and Health Insurance Portability and Accountability Act (HIPAA). This enables you to audit the system in an automated way for compliance with security standards.

You can modify (tailor) a profile to customize certain rules, for example, password length. For more information on profile tailoring, see Customizing a security profile with SCAP Workbench .

9.3.2. Possible results of an OpenSCAP scan

Depending on various properties of your system and the data stream and profile applied to an OpenSCAP scan, each rule may produce a specific result. This is a list of possible results with brief explanations of what they mean.

Table 9.1. Possible results of an OpenSCAP scan

Result	Explanation
Pass	The scan did not find any conflicts with this rule.
Fail	The scan found a conflict with this rule.
Not checked	OpenSCAP does not perform an automatic evaluation of this rule. Check whether your system conforms to this rule manually.
Not applicable	This rule does not apply to the current configuration.
Not selected	This rule is not part of the profile. OpenSCAP does not evaluate this rule and does not display these rules in the results.
Error	The scan encountered an error. For additional information, you can enter the **oscap** command with the **--verbose DEVEL** option. Consider opening a bug report.
Unknown	The scan encountered an unexpected situation. For additional information, you can enter the **oscap** command with the `**--verbose DEVEL** option. Consider opening a bug report.

9.3.3. Viewing profiles for configuration compliance

Before you decide to use profiles for scanning or remediation, you can list them and check their detailed descriptions using the **oscap info** sub-command.

Prerequisites

- The **openscap-scanner** and **scap-security-guide** packages are installed.

Procedure

1. List all available files with security compliance profiles provided by the SCAP Security Guide project:

   ```
   $ ls /usr/share/xml/scap/ssg/content/
   ssg-firefox-cpe-dictionary.xml  ssg-rhel6-ocil.xml
   ssg-firefox-cpe-oval.xml        ssg-rhel6-oval.xml
   ...
   ssg-rhel6-ds-1.2.xml         ssg-rhel8-oval.xml
   ssg-rhel8-ds.xml            ssg-rhel8-xccdf.xml
   ...
   ```

2. Display detailed information about a selected data stream using the **oscap info** sub-command. XML files containing data streams are indicated by the **-ds** string in their names. In the **Profiles** section, you can find a list of available profiles and their IDs:

   ```
   $ oscap info /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ...
   Profiles:
     Title: Health Insurance Portability and Accountability Act (HIPAA)
       Id: xccdf_org.ssgproject.content_profile_hipaa
     Title: PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8
       Id: xccdf_org.ssgproject.content_profile_pci-dss
     Title: OSPP - Protection Profile for General Purpose Operating Systems
       Id: xccdf_org.ssgproject.content_profile_ospp
   ...
   ```

3. Select a profile from the data-stream file and display additional details about the selected profile. To do so, use **oscap info** with the **--profile** option followed by the last section of the ID displayed in the output of the previous command. For example, the ID of the HIPPA profile is: **xccdf_org.ssgproject.content_profile_hipaa**, and the value for the **--profile** option is **hipaa**:

   ```
   $ oscap info --profile hipaa /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ...
   Profile
    Title: Health Insurance Portability and Accountability Act (HIPAA)

    Description: The HIPAA Security Rule establishes U.S. national standards to protect
   individuals' electronic personal health information that is created, received, used, or
   maintained by a covered entity.
    ...
   ```

Additional resources

- **scap-security-guide(8)** man page

9.3.4. Assessing configuration compliance with a specific baseline

To determine whether your system conforms to a specific baseline, follow these steps.

Prerequisites

- The **openscap-scanner** and **scap-security-guide** packages are installed

- You know the ID of the profile within the baseline with which the system should comply. To find the ID, see Viewing Profiles for Configuration Compliance.

Procedure

1. Evaluate the compliance of the system with the selected profile and save the scan results in the report.html HTML file, for example:

   ```
   $ sudo oscap xccdf eval --report report.html --profile hipaa
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

2. Optional: Scan a remote system with the **machine1** host name, SSH running on port **22**, and the **joesec** user name for compliance and save results to the **remote-report.html** file:

   ```
   $ oscap-ssh joesec@machine1 22 xccdf eval --report remote_report.html --profile hipaa
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

Additional resources

- **scap-security-guide(8)** man page

- **SCAP Security Guide** documentation in the **file:///usr/share/doc/scap-security-guide/** directory

- Guide to the Secure Configuration of Red Hat Enterprise Linux 8 installed with the **scap-security-guide-doc** package

9.4. REMEDIATING THE SYSTEM TO ALIGN WITH A SPECIFIC BASELINE

Use this procedure to remediate the RHEL 8 system to align with a specific baseline. This example uses the Health Insurance Portability and Accountability Act (HIPAA) profile.

> **WARNING**
>
> If not used carefully, running the system evaluation with the **Remediate** option enabled might render the system non-functional. Red Hat does not provide any automated method to revert changes made by security-hardening remediations. Remediations are supported on RHEL systems in the default configuration. If your system has been altered after the installation, running remediation might not make it compliant with the required security profile.

Prerequisites

- The **scap-security-guide** package is installed on your RHEL 8 system.

Procedure

1. Use the **oscap** command with the **--remediate** option:

   > $ sudo oscap xccdf eval --profile hipaa --remediate /usr/share/xml/scap/ssg/content/ssg-
   > rhel8-ds.xml

2. Restart your system.

Verification

1. Evaluate compliance of the system with the HIPAA profile, and save scan results in the **hipaa_report.html** file:

   > $ oscap xccdf eval --report *hipaa_report.html* --profile hipaa
   > /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml

Additional resources

- **scap-security-guide(8)** and **oscap(8)** man pages

9.5. REMEDIATING THE SYSTEM TO ALIGN WITH A SPECIFIC BASELINE USING THE SSG ANSIBLE PLAYBOOK

Use this procedure to remediate your system with a specific baseline using the Ansible playbook file from the SCAP Security Guide project. This example uses the Health Insurance Portability and Accountability Act (HIPAA) profile.

> **WARNING**
>
> If not used carefully, running the system evaluation with the **Remediate** option enabled might render the system non-functional. Red Hat does not provide any automated method to revert changes made by security-hardening remediations. Remediations are supported on RHEL systems in the default configuration. If your system has been altered after the installation, running remediation might not make it compliant with the required security profile.

Prerequisites

- The **scap-security-guide** package is installed on your RHEL 8 system.

- The **ansible** package is installed. See the Ansible Installation Guide for more information.

Procedure

1. Remediate your system to align with HIPAA using Ansible:

   > # ansible-playbook -i localhost, -c local /usr/share/scap-security-guide/ansible/rhel8-
   > playbook-hipaa.yml

2. Restart the system.

Verification

1. Evaluate compliance of the system with the HIPAA profile, and save scan results in the **hipaa_report.html** file:

   ```
   # oscap xccdf eval --profile hipaa --report hipaa_report.html
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

Additional resources

- **scap-security-guide(8)** and **oscap(8)** man pages

- [Ansible Documentation](#)

9.6. CREATING A REMEDIATION ANSIBLE PLAYBOOK TO ALIGN THE SYSTEM WITH A SPECIFIC BASELINE

You can create an Ansible playbook containing only the remediations that are required to align your system with a specific baseline. This example uses the Health Insurance Portability and Accountability Act (HIPAA) profile. With this procedure, you create a smaller playbook that does not cover already satisfied requirements. By following these steps, you do not modify your system in any way, you only prepare a file for later application.

Prerequisites

- The **scap-security-guide** package is installed on your RHEL 8 system.

Procedure

1. Scan the system and save the results:

   ```
   # oscap xccdf eval --profile hipaa --results hipaa-results.xml
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

2. Generate an Ansible playbook based on the file generated in the previous step:

   ```
   # oscap xccdf generate fix --fix-type ansible --output hipaa-remediations.yml hipaa-results.xml
   ```

3. The **hipaa-remediations.yml** file contains Ansible remediations for rules that failed during the scan performed in step 1. After reviewing this generated file, you can apply it with the **ansible-playbook hipaa-remediations.yml** command.

Verification

1. In a text editor of your choice, review that the **hipaa-remediations.yml** file contains rules that failed in the scan performed in step 1.

Additional resources

- **scap-security-guide(8)** and **oscap(8)** man pages

- [Ansible Documentation](#)

9.7. CREATING A REMEDIATION BASH SCRIPT FOR A LATER APPLICATION

Use this procedure to create a Bash script containing remediations that align your system with a security profile such as HIPAA. Using the following steps, you do not do any modifications to your system, you only prepare a file for later application.

Prerequisites

- The **scap-security-guide** package is installed on your RHEL 8 system.

Procedure

1. Use the **oscap** command to scan the system and to save the results to an XML file. In the following example, **oscap** evaluates the system against the **hipaa** profile:

   ```
   # oscap xccdf eval --profile hipaa --results hipaa-results.xml
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

2. Generate a Bash script based on the results file generated in the previous step:

   ```
   # oscap xccdf generate fix --profile hipaa --fix-type bash --output hipaa-remediations.sh
   hipaa-results.xml
   ```

3. The **hipaa-remediations.sh** file contains remediations for rules that failed during the scan performed in step 1. After reviewing this generated file, you can apply it with the **./hipaa-remediations.sh** command when you are in the same directory as this file.

Verification

1. In a text editor of your choice, review that the **hipaa-remediations.sh** file contains rules that failed in the scan performed in step 1.

Additional resources

- **scap-security-guide(8)**, **oscap(8)**, and **bash(1)** man pages

9.8. SCANNING THE SYSTEM WITH A CUSTOMIZED PROFILE USING SCAP WORKBENCH

SCAP Workbench, which is contained in the **scap-workbench** package, is a graphical utility that enables users to perform configuration and vulnerability scans on a single local or a remote system, perform remediation of the system, and generate reports based on scan evaluations. Note that **SCAP Workbench** has limited functionality compared with the **oscap** command-line utility. **SCAP Workbench** processes security content in the form of data-stream files.

9.8.1. Using SCAP Workbench to scan and remediate the system

To evaluate your system against the selected security policy, use the following procedure.

Prerequisites

- The **scap-workbench** package is installed on your system.

Procedure

1. To run **SCAP Workbench** from the **GNOME Classic** desktop environment, press the **Super** key to enter the **Activities Overview**, type **scap-workbench**, and then press **Enter**. Alternatively, use:

   ```
   $ scap-workbench &
   ```

2. Select a security policy using either of the following options:

   - **Load Content** button on the starting window

   - **Open content from SCAP Security Guide**

   - **Open Other Content** in the **File** menu, and search the respective XCCDF, SCAP RPM, or data stream file.
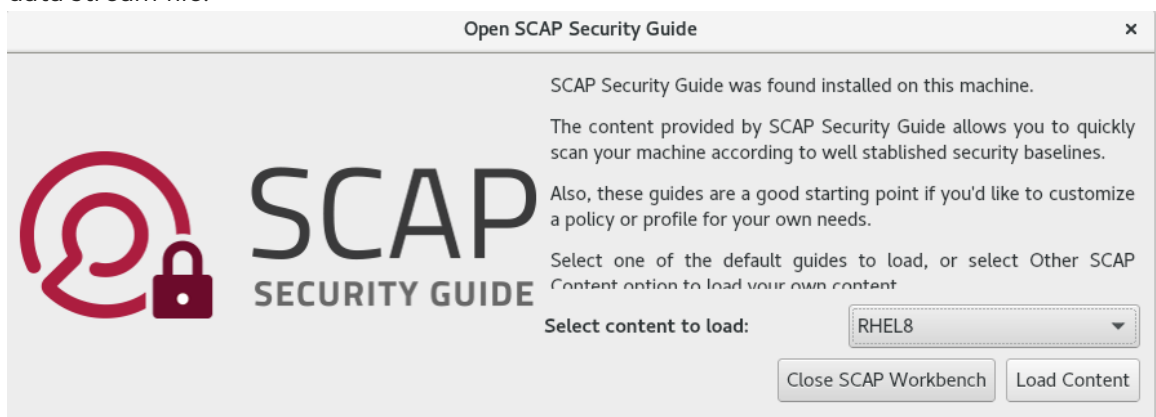
3. You can allow automatic correction of the system configuration by selecting the **Remediate** check box. With this option enabled, **SCAP Workbench** attempts to change the system configuration in accordance with the security rules applied by the policy. This process should fix the related checks that fail during the system scan.

> **WARNING**
>
> If not used carefully, running the system evaluation with the **Remediate** option enabled might render the system non-functional. Red Hat does not provide any automated method to revert changes made by security-hardening remediations. Remediations are supported on RHEL systems in the default configuration. If your system has been altered after the installation, running remediation might not make it compliant with the required security profile.

4. Scan your system with the selected profile by clicking the **Scan** button.
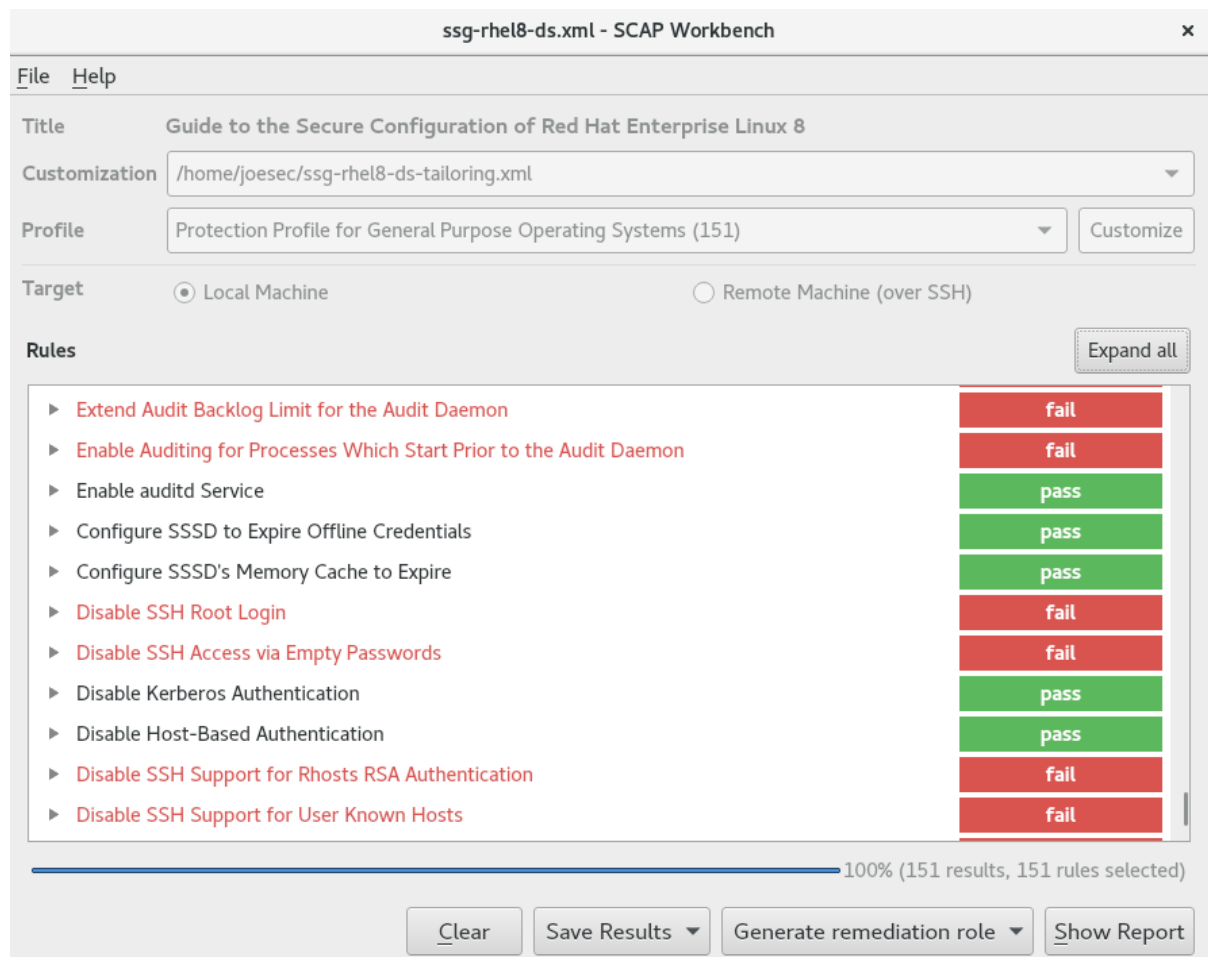
5. To store the scan results in form of an XCCDF, ARF, or HTML file, click the **Save Results** combo box. Choose the **HTML Report** option to generate the scan report in human-readable format. The XCCDF and ARF (data stream) formats are suitable for further automatic processing. You can repeatedly choose all three options.

6. To export results-based remediations to a file, use the **Generate remediation role** pop-up menu.

9.8.2. Customizing a security profile with SCAP Workbench

You can customize a security profile by changing parameters in certain rules (for example, minimum password length), removing rules that you cover in a different way, and selecting additional rules, to implement internal policies. You cannot define new rules by customizing a profile.

The following procedure demonstrates the use of **SCAP Workbench** for customizing (tailoring) a profile. You can also save the tailored profile for use with the **oscap** command-line utility.

Prerequisites

- The **scap-workbench** package is installed on your system.

Procedure

1. Run **SCAP Workbench**, and select the profile to customize by using either **Open content from SCAP Security Guide** or **Open Other Content** in the **File** menu.

2. To adjust the selected security profile according to your needs, click the **Customize** button.

This opens the new Customization window that enables you to modify the currently selected profile without changing the original data stream file. Choose a new profile ID.
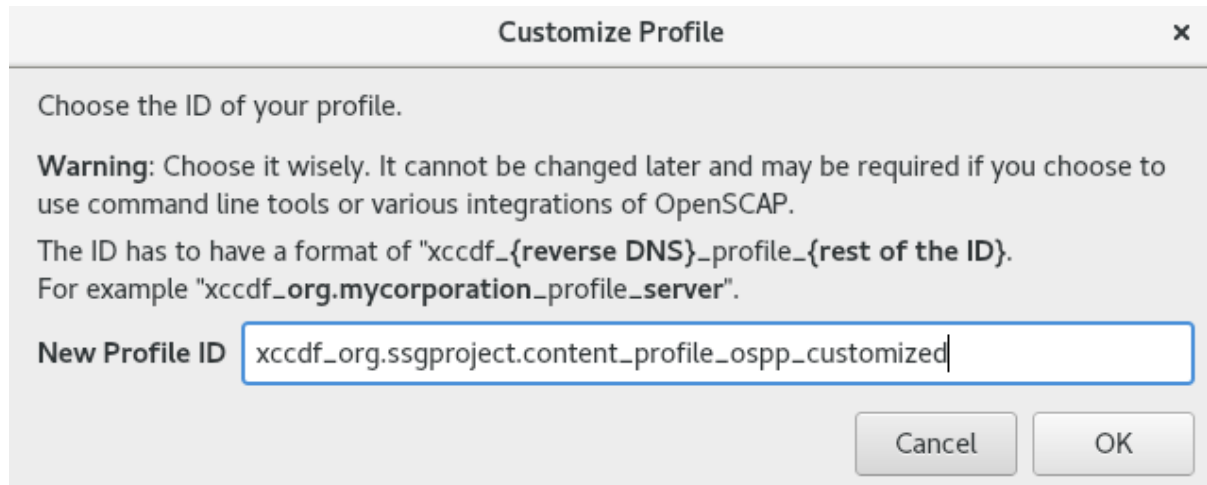
3. Find a rule to modify using either the tree structure with rules organized into logical groups or the **Search** field.

4. Include or exclude rules using check boxes in the tree structure, or modify values in rules where applicable.
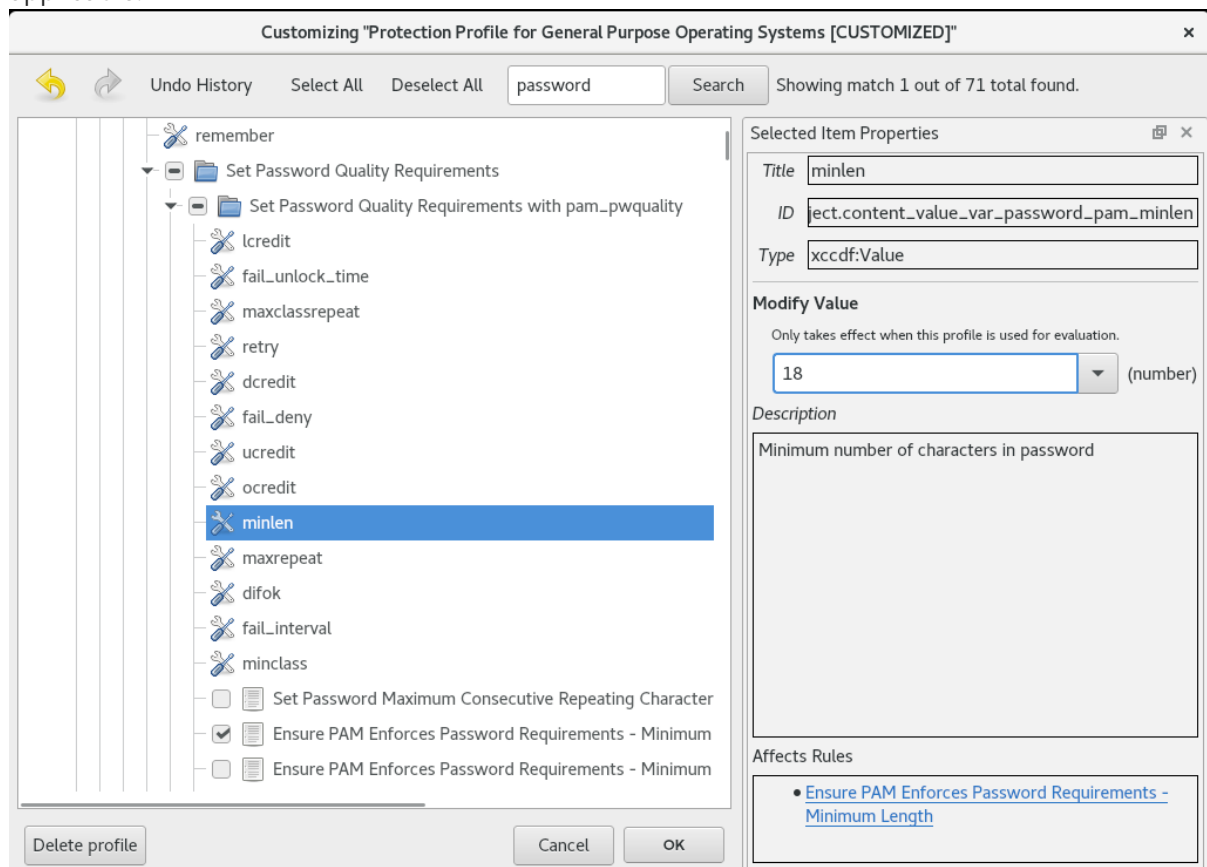
5. Confirm the changes by clicking the **OK** button.

6. To store your changes permanently, use one of the following options:

   - Save a customization file separately by using **Save Customization Only** in the **File** menu.

   - Save all security content at once by **Save All** in the **File** menu.
     If you select the **Into a directory** option, **SCAP Workbench** saves both the data stream file and the customization file to the specified location. You can use this as a backup solution.

By selecting the **As RPM** option, you can instruct **SCAP Workbench** to create an RPM package containing the data stream file and the customization file. This is useful for distributing the security content to systems that cannot be scanned remotely, and for delivering the content for further processing.

> **NOTE**
>
> Because **SCAP Workbench** does not support results-based remediations for tailored profiles, use the exported remediations with the **oscap** command-line utility.

9.8.3. Related information

- **scap-workbench(8)** man page

- SCAP Workbench User Manual

- Deploy customized SCAP policies with Satellite 6.x - a Knowledge Base article on tailoring scripts

9.9. DEPLOYING SYSTEMS THAT ARE COMPLIANT WITH A SECURITY PROFILE IMMEDIATELY AFTER AN INSTALLATION

You can use the OpenSCAP suite to deploy RHEL systems that are compliant with a security profile, such as OSPP, PCI-DSS, and HIPAA profile, immediately after the installation process. Using this deployment method, you can apply specific rules that cannot be applied later using remediation scripts, for example, a rule for password strength and partitioning.

9.9.1. Deploying baseline-compliant RHEL systems using the graphical installation

Use this procedure to deploy a RHEL system that is aligned with a specific baseline. This example uses Protection Profile for General Purpose Operating System (OSPP).

Prerequisites

- You have booted into the **graphical** installation program. Note that the **OSCAP Anaconda Add-on** does not support text-only installation.

- You have accessed the **Installation Summary** window.

Procedure

1. From the **Installation Summary** window, click **Software Selection**. The **Software Selection** window opens.

2. From the **Base Environment** pane, select the **Server** environment. You can select only one base environment.

> **WARNING**
>
> Do not use the **Server with GUI** base environment if you want to deploy a compliant system. Security profiles provided as part of the **SCAP Security Guide** may not be compatible with the extended package set of **Server with GUI**. For more information, see, for example, BZ#1648162, BZ#1787156, or BZ#1816199.

3. Click **Done** to apply the setting and return to the **Installation Summary** window.

4. Click **Security Policy**. The **Security Policy** window opens.

5. To enable security policies on the system, toggle the **Apply security policy** switch to **ON**.

6. Select **Protection Profile for General Purpose Operating Systems** from the profile pane.

7. Click **Select Profile** to confirm the selection.

8. Confirm the changes in the **Changes that were done or need to be done** pane that is displayed at the bottom of the window. Complete any remaining manual changes.

9. Because OSPP has strict partitioning requirements that must be met, create separate partitions for **/boot**, **/home**, **/var**, **/var/log**, **/var/tmp**, and **/var/log/audit**.

10. Complete the graphical installation process.

> **NOTE**
>
> The graphical installation program automatically creates a corresponding Kickstart file after a successful installation. You can use the **/root/anaconda-ks.cfg** file to automatically install OSPP-compliant systems.

Verification

1. To check the current status of the system after installation is complete, reboot the system and start a new scan:

```
# oscap xccdf eval --profile ospp --report eval_postinstall_report.html
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

Additional resources

- For more details on partitioning, see Configuring manual partitioning.

9.9.2. Deploying baseline-compliant RHEL systems using Kickstart

Use this procedure to deploy RHEL systems that are aligned with a specific baseline. This example uses Protection Profile for General Purpose Operating System (OSPP).

Prerequisites

- The **scap-security-guide** package is installed on your RHEL 8 system.

Procedure

1. Open the **/usr/share/scap-security-guide/kickstarts/ssg-rhel8-ospp-ks.cfg** Kickstart file in an editor of your choice.

2. Update the partitioning scheme to fit your configuration requirements. For OSPP compliance, the separate partitions for **/boot**, **/home**, **/var**, **/var/log**, **/var/tmp**, and **/var/log/audit** must be preserved, and you can only change the size of the partitions.

> **⚠ WARNING**
>
> Because the **OSCAP Anaconda Addon** plugin does not support text-only installation, do not use the **text** option in your Kickstart file. For more information, see RHBZ#1674001.

3. Start a Kickstart installation as described in Performing an automated installation using Kickstart.

> **IMPORTANT**
>
> Passwords in the hash form cannot be checked for OSPP requirements.

Verification

1. To check the current status of the system after installation is complete, reboot the system and start a new scan:

   ```
   # oscap xccdf eval --profile ospp --report eval_postinstall_report.html
   /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
   ```

Additional resources

- For more details, see the OSCAP Anaconda Addon project page.

9.10. SCANNING CONTAINER AND CONTAINER IMAGES FOR VULNERABILITIES

Use this procedure to find security vulnerabilities in a container or a container image.

> **NOTE**
>
> The **oscap-podman** command is available from RHEL 8.2. For RHEL 8.1 and 8.0, use the workaround described in the Using OpenSCAP for scanning containers in RHEL 8 Knowledgebase article.

Prerequisites

- The **openscap-utils** package is installed.

Procedure

1. Download the latest RHSA OVAL definitions for your system:

   > # wget -O - https://www.redhat.com/security/data/oval/v2/RHEL8/rhel-8.oval.xml.bz2 | bzip2 -
   > -decompress > rhel-8.oval.xml

2. Get the ID of a container or a container image, for example:

   > # podman images
   > REPOSITORY   TAG   IMAGE ID   CREATED   SIZE
   > registry.access.redhat.com/ubi8/ubi   latest   096cae65a207   7 weeks ago   239 MB

3. Scan the container or the container image for vulnerabilities and save results to the
   vulnerability.html file:

   > # oscap-podman *096cae65a207* oval eval --report *vulnerability.html* rhel-8.oval.xml

   Note that the **oscap-podman** command requires root privileges, and the ID of a container is the
   first argument.

Verification

1. Check the results in a browser of your choice, for example:

   > $ firefox *vulnerability.html* &

Additional resources

- For more information, see the **oscap-podman(8)** and **oscap(8)** man pages.

9.11. ASSESSING SECURITY COMPLIANCE OF A CONTAINER OR A CONTAINER IMAGE WITH A SPECIFIC BASELINE

Follow these steps to assess compliance of your container or a container image with a specific security
baseline, such as Operating System Protection Profile (OSPP), Payment Card Industry Data Security
Standard (PCI-DSS), and Health Insurance Portability and Accountability Act (HIPAA).

> **NOTE**
>
> The **oscap-podman** command is available from RHEL 8.2. For RHEL 8.1 and 8.0, use the
> workaround described in the Using OpenSCAP for scanning containers in RHEL 8
> Knowledgebase article.

Prerequisites

- The **openscap-utils** and **scap-security-guide** packages are installed.

Procedure

1. Get the ID of a container or a container image, for example:

```
# podman images
REPOSITORY                    TAG     IMAGE ID      CREATED      SIZE
registry.access.redhat.com/ubi8/ubi   latest   096cae65a207   7 weeks ago   239 MB
```

2. Evaluate the compliance of the container image with the HIPAA profile and save scan results into the *report.html* HTML file

```
# oscap-podman 096cae65a207 xccdf eval --report report.html --profile hipaa
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

Replace *096cae65a207* with the ID of your container image and the *hipaa* value with *ospp* or *pci-dss* if you assess security compliance with the OSPP or PCI-DSS baseline. Note that the **oscap-podman** command requires root privileges.

Verification

1. Check the results in a browser of your choice, for example:

```
$ firefox report.html &
```

> **NOTE**
>
> The rules marked as *notapplicable* are rules that do not apply to containerized systems. These rules apply only to bare-metal and virtualized systems.

Additional resources

- **oscap-podman(8)** and **scap-security-guide(8)** man pages.

- **file:///usr/share/doc/scap-security-guide/** directory.

9.12. SUPPORTED VERSIONS OF THE SCAP SECURITY GUIDE IN RHEL

Officially supported versions of the SCAP Security Guide are versions provided in the related minor release of RHEL or in the related batch update of RHEL.

Table 9.2. Supported versions of the SCAP Security Guide in RHEL

Red Hat Enterprise Linux version	SCAP Security Guide version
RHEL 6.6	scap-security-guide-0.1.18-3.el6
RHEL 6.9	scap-security-guide-0.1.28-3.el6
RHEL 6.10	scap-security-guide-0.1.28-4.el6
RHEL 7.2 AUS	scap-security-guide-0.1.25-3.el7
RHEL 7.3 AUS	scap-security-guide-0.1.30-5.el7_3

Red Hat Enterprise Linux version	SCAP Security Guide version
RHEL 7.4 AUS, E4S	scap-security-guide-0.1.33-6.el7_4
RHEL 7.5 (batch update)	scap-security-guide-0.1.36-10.el7_5
RHEL 7.6 EUS	scap-security-guide-0.1.40-13.el7_6
RHEL 7.7 EUS	scap-security-guide-0.1.43-13.el7
RHEL 7.8 (batch update)	scap-security-guide-0.1.46-11.el7
RHEL 7.9 (batch update)	scap-security-guide-0.1.54-7.el7_9
RHEL 8.0 SAP	scap-security-guide-0.1.42-11.el8
RHEL 8.1 EUS	scap-security-guide-0.1.47-8.el8_1
RHEL 8.2 (batch update)	scap-security-guide-0.1.48-10.el8_2
RHEL 8.3	scap-security-guide-0.1.50-16.el8_3
RHEL 8.4	scap-security-guide-0.1.54-5.el8

9.13. SCAP SECURITY GUIDE PROFILES SUPPORTED IN RHEL 8

Use only the SCAP content provided in the particular minor release of RHEL. This is because components that participate in hardening are sometimes updated with new capabilities. SCAP content changes to reflect these updates, but it is not always backward compatible.

In the following tables, you can find the profiles provided in each minor version of RHEL, together with the version of the policy with which the profile aligns.

Table 9.3. SCAP Security Guide profiles supported in RHEL 8.4

Profile name	Profile ID	Policy version
French National Agency for the Security of Information Systems (ANSSI) BP-028 Enhanced Level	**xccdf_org.ssgproject.content_profile_anssi_bp28_enhanced**	1.2
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	**xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary**	1.2

Profile name	Profile ID	Policy version
French National Agency for the Security of Information Systems (ANSSI) BP-028 Minimal Level	**xccdf_org.ssgproject.content_profile_anssi_bp28_minimal**	1.2
CIS Red Hat Enterprise Linux 8 Benchmark	**xccdf_org.ssgproject.content_profile_cis**	1.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
Australian Cyber Security Centre (ACSC) Essential Eight	**xccdf_org.ssgproject.content_profile_e8**	not versioned
Health Insurance Portability and Accountability Act (HIPAA)	**xccdf_org.ssgproject.content_profile_hipaa**	not versioned
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	V1R1

Table 9.4. SCAP Security Guide profiles supported in RHEL 8.3

Profile name	Profile ID	Policy version
CIS Red Hat Enterprise Linux 8 Benchmark	**xccdf_org.ssgproject.content_profile_cis**	1.0.0
Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	**xccdf_org.ssgproject.content_profile_cui**	r1
Australian Cyber Security Centre (ACSC) Essential Eight	**xccdf_org.ssgproject.content_profile_e8**	not versioned
Health Insurance Portability and Accountability Act (HIPAA)	**xccdf_org.ssgproject.content_profile_hipaa**	not versioned

Profile name	Profile ID	Policy version
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
[DRAFT] The Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	draft

Table 9.5. SCAP Security Guide profiles supported in RHEL 8.2

Profile name	Profile ID	Policy version
Australian Cyber Security Centre (ACSC) Essential Eight	**xccdf_org.ssgproject.content_profile_e8**	not versioned
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1
[DRAFT] DISA STIG for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_stig**	draft

Table 9.6. SCAP Security Guide profiles supported in RHEL 8.1

Profile name	Profile ID	Policy version
Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1

Table 9.7. SCAP Security Guide profiles supported in RHEL 8.0

Profile name	Profile ID	Policy version
OSPP - Protection Profile for General Purpose Operating Systems	**xccdf_org.ssgproject.content_profile_ospp**	draft

Profile name	Profile ID	Policy version
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 8	**xccdf_org.ssgproject.content_profile_pci-dss**	3.2.1

Additional Resources

- For information about profiles in RHEL 7, see SCAP Security Guide profiles supported in RHEL 7

9.14. RELATED INFORMATION

- The OpenSCAP project page – The home page of the OpenSCAP project provides detailed information about the **oscap** utility and other components and projects related to SCAP.

- The SCAP Workbench project page – The home page of the SCAP Workbench project provides detailed information about the **scap-workbench** application.

- The SCAP Security Guide (SSG) project page – The home page of the SSG project that provides the latest security content for Red Hat Enterprise Linux.

- Red Hat Security Demos: Creating Customized Security Policy Content to Automate Security Compliance – A hands-on lab to get initial experience in automating security compliance using the tools that are included in Red Hat Enterprise Linux to comply with both industry standard security policies and custom security policies. If you want training or access to these lab exercises for your team, contact your Red Hat account team for additional details.

- Red Hat Security Demos: Defend Yourself with RHEL Security Technologies – A hands-on lab to learn how to implement security at all levels of your RHEL system, using the key security technologies available to you in Red Hat Enterprise Linux, including OpenSCAP. If you want training or access to these lab exercises for your team, contact your Red Hat account team for additional details.

- National Institute of Standards and Technology (NIST) SCAP page – This page represents a vast collection of SCAP-related materials, including SCAP publications, specifications, and the SCAP Validation Program.

- National Vulnerability Database (NVD) – This page represents the largest repository of SCAP content and other SCAP standards-based vulnerability management data.

- Red Hat OVAL content repository – This is a repository containing OVAL definitions for vulnerabilities of Red Hat Enterprise Linux systems. This is the recommended source of vulnerability content.

- MITRE CVE – This is a database of publicly known security vulnerabilities provided by the MITRE corporation. For RHEL, using OVAL CVE content provided by Red Hat is recommended.

- MITRE OVAL – This is an OVAL-related project provided by the MITRE corporation. Among other OVAL-related information, these pages contain the OVAL language and a repository of OVAL content with thousands of OVAL definitions. Note that for scanning RHEL, using OVAL CVE content provided by Red Hat is recommended.

- Managing security compliance in Red Hat Satellite – This set of guides describes, among other topics, how to maintain system security on multiple systems by using OpenSCAP.

CHAPTER 10. CHECKING INTEGRITY WITH AIDE

Advanced Intrusion Detection Environment (**AIDE**) is a utility that creates a database of files on the system, and then uses that database to ensure file integrity and detect system intrusions.

10.1. INSTALLING AIDE

The following steps are necessary to install **AIDE** and to initiate its database.

Prerequisites

- The **AppStream** repository is enabled.

Procedure

1. To install the *aide* package:

   ```
   # yum install aide
   ```

2. To generate an initial database:

   ```
   # aide --init
   ```

   > **NOTE**
   >
   > In the default configuration, the **aide --init** command checks just a set of directories and files defined in the **/etc/aide.conf** file. To include additional directories or files in the **AIDE** database, and to change their watched parameters, edit **/etc/aide.conf** accordingly.

3. To start using the database, remove the **.new** substring from the initial database file name:

   ```
   # mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
   ```

4. To change the location of the **AIDE** database, edit the **/etc/aide.conf** file and modify the **DBDIR** value. For additional security, store the database, configuration, and the **/usr/sbin/aide** binary file in a secure location such as a read-only media.

10.2. PERFORMING INTEGRITY CHECKS WITH AIDE

Prerequisites

- **AIDE** is properly installed and its database is initialized. See Installing AIDE

Procedure

1. To initiate a manual check:

   ```
   # aide --check
   Start timestamp: 2018-07-11 12:41:20 +0200 (AIDE 0.16)
   AIDE found differences between database and filesystem!!
   ```

> ...
> [trimmed for clarity]

2. At a minimum, configure the system to run **AIDE** weekly. Optimally, run **AIDE** daily. For example, to schedule a daily execution of **AIDE** at *04:05* a.m. using the **cron** command, add the following line to the **/etc/crontab** file:

> 05 4 * * * root /usr/sbin/aide --check

10.3. UPDATING AN AIDE DATABASE

After verifying the changes of your system such as, package updates or configuration files adjustments, Red Hat recommends updating your baseline **AIDE** database.

Prerequisites

- **AIDE** is properly installed and its database is initialized. See Installing AIDE

Procedure

1. Update your baseline **AIDE** database:

> # aide --update

    The **aide --update** command creates the **/var/lib/aide/aide.db.new.gz** database file.

2. To start using the updated database for integrity checks, remove the **.new** substring from the file name.

10.4. RELATED INFORMATION

- **aide(1)** man page

CHAPTER 11. ENCRYPTING BLOCK DEVICES USING LUKS

Disk encryption protects the data on a block device by encrypting it. To access the device's decrypted contents, a user must provide a passphrase or key as authentication. This is particularly important when it comes to mobile computers and removable media: it helps to protect the device's contents even if it has been physically removed from the system. The LUKS format is a default implementation of block device encryption in RHEL.

11.1. LUKS DISK ENCRYPTION

The Linux Unified Key Setup-on-disk-format (LUKS) enables you to encrypt block devices and it provides a set of tools that simplifies managing the encrypted devices. LUKS allows multiple user keys to decrypt a master key, which is used for the bulk encryption of the partition.

RHEL utilizes LUKS to perform block device encryption. By default, the option to encrypt the block device is unchecked during the installation. If you select the option to encrypt your disk, the system prompts you for a passphrase every time you boot the computer. This passphrase "unlocks" the bulk encryption key that decrypts your partition. If you choose to modify the default partition table, you can choose which partitions you want to encrypt. This is set in the partition table settings.

What LUKS does

- LUKS encrypts entire block devices and is therefore well-suited for protecting contents of mobile devices such as removable storage media or laptop disk drives.

- The underlying contents of the encrypted block device are arbitrary, which makes it useful for encrypting swap devices. This can also be useful with certain databases that use specially formatted block devices for data storage.

- LUKS uses the existing device mapper kernel subsystem.

- LUKS provides passphrase strengthening, which protects against dictionary attacks.

- LUKS devices contain multiple key slots, allowing users to add backup keys or passphrases.

What LUKS does *not* do

- Disk-encryption solutions like LUKS protect the data only when your system is off. Once the system is on and LUKS has decrypted the disk, the files on that disk are available to anyone who would normally have access to them.

- LUKS is not well-suited for scenarios that require many users to have distinct access keys to the same device. The LUKS1 format provides eight key slots, LUKS2 up to 32 key slots.

- LUKS is not well-suited for applications requiring file-level encryption.

Ciphers

The default cipher used for LUKS is **aes-xts-plain64**. The default key size for LUKS is 512 bits. The default key size for LUKS with **Anaconda** (XTS mode) is 512 bits. Ciphers that are available are:

- AES - Advanced Encryption Standard

- Twofish (a 128-bit block cipher)

- Serpent

Additional resources

- [LUKS Project Home Page](#)

- [LUKS On-Disk Format Specification](#)

- [FIPS PUB 197](#)

11.2. LUKS VERSIONS IN RHEL 8

In RHEL 8, the default format for LUKS encryption is LUKS2. The legacy LUKS1 format remains fully supported and it is provided as a format compatible with earlier RHEL releases.

The LUKS2 format is designed to enable future updates of various parts without a need to modify binary structures. LUKS2 internally uses JSON text format for metadata, provides redundancy of metadata, detects metadata corruption and allows automatic repairs from a metadata copy.

> **IMPORTANT**
>
> Do not use LUKS2 in systems that must be compatible with legacy systems that support only LUKS1. Note that RHEL 7 supports the LUKS2 format since version 7.6.

> ⚠️ **WARNING**
>
> LUKS2 and LUKS1 use different commands to encrypt the disk. Using the wrong command for a LUKS version might cause data loss.
>
> | LUKS version | Encryption command |
> |---|---|
> | LUKS2 | **cryptsetup reencrypt** |
> | LUKS1 | **cryptsetup-reencrypt** |

Online re-encryption

The LUKS2 format supports re-encrypting encrypted devices while the devices are in use. For example, you do not have to unmount the file system on the device to perform the following tasks:

- Change the volume key

- Change the encryption algorithm

When encrypting a non-encrypted device, you must still unmount the file system. You can remount the file system after a short initialization of the encryption.

The LUKS1 format does not support online re-encryption.

Conversion

The LUKS2 format is inspired by LUKS1. In certain situations, you can convert LUKS1 to LUKS2. The conversion is not possible specifically in the following scenarios:

- A LUKS1 device is marked as being used by a Policy-Based Decryption (PBD - Clevis) solution. The **cryptsetup** tool refuses to convert the device when some **luksmeta** metadata are detected.

- A device is active. The device must be in the inactive state before any conversion is possible.

11.3. OPTIONS FOR DATA PROTECTION DURING LUKS2 RE-ENCRYPTION

LUKS2 provides several options that prioritize performance or data protection during the re-encryption process:

checksum

This is the default mode. It balances data protection and performance.
This mode stores individual checksums of the sectors in the re-encryption area, so the recovery process can detect which sectors LUKS2 already re-encrypted. The mode requires that the block device sector write is atomic.

journal

That is the safest mode but also the slowest. This mode journals the re-encryption area in the binary area, so LUKS2 writes the data twice.

none

This mode prioritizes performance and provides no data protection. It protects the data only against safe process termination, such as the **SIGTERM** signal or the user pressing **Ctrl**+**C**. Any unexpected system crash or application crash might result in data corruption.

You can select the mode using the **--resilience** option of **cryptsetup**.

If a LUKS2 re-encryption process terminates unexpectedly by force, LUKS2 can perform the recovery in one of the following ways:

- Automatically, during the next LUKS2 device open action. This action is triggered either by the **cryptsetup open** command or by attaching the device with **systemd-cryptsetup**.

- Manually, by using the **cryptsetup repair** command on the LUKS2 device.

11.4. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2

This procedure encrypts existing data on a not yet encrypted device using the LUKS2 format. A new LUKS header is stored in the head of the device.

Prerequisites

- The block device contains a file system.

- You have backed up your data.

> **WARNING**
>
> You might lose your data during the encryption process: due to a hardware, kernel, or human failure. Ensure that you have a reliable backup before you start encrypting the data.

Procedure

1. Unmount all file systems on the device that you plan to encrypt. For example:

   ```
   # umount /dev/sdb1
   ```

2. Make free space for storing a LUKS header. Choose one of the following options that suits your scenario:

   - In the case of encrypting a logical volume, you can extend the logical volume without resizing the file system. For example:

     ```
     # lvextend -L+32M vg00/lv00
     ```

   - Extend the partition using partition management tools, such as **parted**.

   - Shrink the file system on the device. You can use the **resize2fs** utility for the ext2, ext3, or ext4 file systems. Note that you cannot shrink the XFS file system.

3. Initialize the encryption. For example:

   ```
   # cryptsetup reencrypt \ --encrypt \ --init-only \ --reduce-device-size 32M \ /dev/sdb1 sdb1_encrypted
   ```

   The command asks you for a passphrase and starts the encryption process.

4. Mount the device:

   ```
   # mount /dev/mapper/sdb1_encrypted /mnt/sdb1_encrypted
   ```

5. Start the online encryption:

   ```
   # cryptsetup reencrypt --resume-only /dev/sdb1
   ```

Additional resources

- **cryptsetup(8)**, **lvextend(8)**, **resize2fs(8)**, and **parted(8)** man pages

11.5. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2 WITH A DETACHED HEADER

This procedure encrypts existing data on a block device without creating free space for storing a LUKS header. The header is stored in a detached location, which also serves as an additional layer of security. The procedure uses the LUKS2 encryption format.

Prerequisites

- The block device contains a file system.

- You have backed up your data.

> **WARNING**
>
> You might lose your data during the encryption process: due to a hardware, kernel, or human failure. Ensure that you have a reliable backup before you start encrypting the data.

Procedure

1. Unmount all file systems on the device. For example:

   ```
   # umount /dev/sdb1
   ```

2. Initialize the encryption:

   ```
   # cryptsetup reencrypt \ --encrypt \ --init-only \ --header /path/to/header \ /dev/sdb1 sdb1_encrypted
   ```

   Replace */path/to/header* with a path to the file with a detached LUKS header. The detached LUKS header has to be accessible so that the encrypted device can be unlocked later.

   The command asks you for a passphrase and starts the encryption process.

3. Mount the device:

   ```
   # mount /dev/mapper/sdb1_encrypted /mnt/sdb1_encrypted
   ```

4. Start the online encryption:

   ```
   # cryptsetup reencrypt --resume-only --header /path/to/header /dev/sdb1
   ```

Additional resources

- **cryptsetup(8)** man page

11.6. ENCRYPTING A BLANK BLOCK DEVICE USING LUKS2

This procedure provides information about encrypting a blank block device using the LUKS2 format.

Prerequisites

- A blank block device.

Procedure

1. Setup a partition as an encrypted LUKS partition:

   ```
   # cryptsetup luksFormat /dev/sdb1
   ```

2. Open an encrypted LUKS partition:

   ```
   # cryptsetup open /dev/sdb1 sdb1_encrypted
   ```

   This unlocks the partition and maps it to a new device using the device mapper. This alerts kernel that **device** is an encrypted device and should be addressed through LUKS using the **/dev/mapper/device_mapped_name** so as not to overwrite the encrypted data.

3. To write encrypted data to the partition, it must be accessed through the device mapped name. To do this, you must create a file system. For example:

   ```
   # mkfs -t ext4 /dev/mapper/sdb1_encrypted
   ```

4. Mount the device:

   ```
   # mount /dev/mapper/sdb1_encrypted mount-point
   ```

Additional resources

- **cryptsetup(8)** man page

11.7. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE ROLE

You can use the **storage** role to create and configure a volume encrypted with LUKS by running an Ansible playbook.

Prerequisites

- You have Red Hat Ansible Engine installed on the system from which you want to run the playbook.

  > **NOTE**
  >
  > You do not have to have Red Hat Ansible Automation Platform installed on the systems on which you want to create the volume.

- You have the **rhel-system-roles** package installed on the Ansible controller.

- You have an inventory file detailing the systems on which you want to deploy a LUKS encrypted volume using the storage System Role.

Procedure

1. Create a new ***playbook.yml*** file with the following content:

```
- hosts: all
  vars:
    storage_volumes:
      - name: barefs
        type: disk
        disks:
         - sdb
        fs_type: xfs
        fs_label: label-name
        mount_point: /mnt/data
        encryption: true
        encryption_password: your-password
  roles:
   - rhel-system-roles.storage
```

2. Optional: Verify playbook syntax:

```
# ansible-playbook --syntax-check playbook.yml
```

3. Run the playbook on your inventory file:

```
# ansible-playbook -i inventory.file /path/to/file/playbook.yml
```

Additional resources

- Encrypting block devices using LUKS

- **/usr/share/ansible/roles/rhel-system-roles.storage/README.md** file

CHAPTER 12. CONFIGURING AUTOMATED UNLOCKING OF ENCRYPTED VOLUMES USING POLICY-BASED DECRYPTION

The Policy-Based Decryption (PBD) is a collection of technologies that enable unlocking encrypted root and secondary volumes of hard drives on physical and virtual machines. PBD uses a variety of unlocking methods, such as user passwords, a Trusted Platform Module (TPM) device, a PKCS #11 device connected to a system, for example, a smart card, or a special network server.

PBD allows combining different unlocking methods into a policy, which makes it possible to unlock the same volume in different ways. The current implementation of the PBD in Red Hat Enterprise Linux consists of the Clevis framework and plug-ins called *pins*. Each pin provides a separate unlocking capability. Currently, the following pins are available:

- **tang** - allows volumes to be unlocked using a network server

- **tpm2** - allows volumes to be unlocked using a TPM2 policy

The Network Bound Disc Encryption (NBDE) is a subcategory of PBD that allows binding encrypted volumes to a special network server. The current implementation of the NBDE includes a Clevis pin for the Tang server and the Tang server itself.

12.1. NETWORK-BOUND DISK ENCRYPTION

In Red Hat Enterprise Linux, NBDE is implemented through the following components and technologies:

Figure 12.1. NBDE scheme when using a LUKS1-encrypted volume. The luksmeta package is not used for LUKS2 volumes.
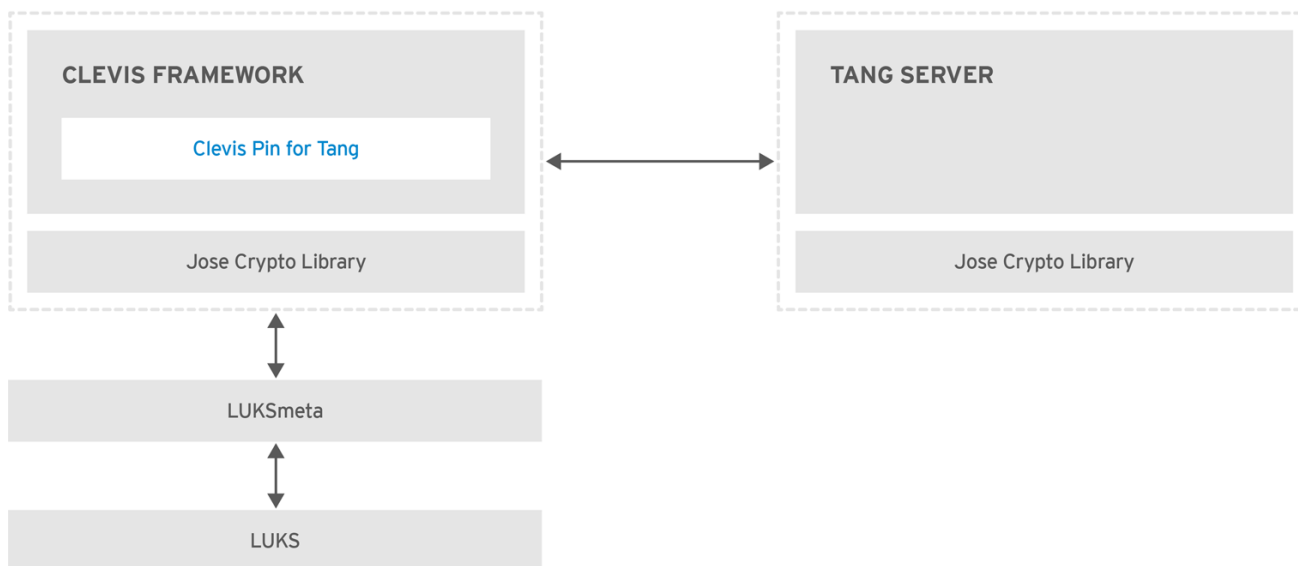
RHEL_453350_0717

Tang is a server for binding data to network presence. It makes a system containing your data available when the system is bound to a certain secure network. Tang is stateless and does not require TLS or authentication. Unlike escrow-based solutions, where the server stores all encryption keys and has knowledge of every key ever used, Tang never interacts with any client keys, so it never gains any identifying information from the client.

Clevis is a pluggable framework for automated decryption. In NBDE, Clevis provides automated unlocking of LUKS volumes. The **clevis** package provides the client side of the feature.

A *Clevis pin* is a plug-in into the Clevis framework. One of such pins is a plug-in that implements interactions with the NBDE server — Tang.

Clevis and Tang are generic client and server components that provide network-bound encryption. In Red Hat Enterprise Linux, they are used in conjunction with LUKS to encrypt and decrypt root and non-root storage volumes to accomplish Network-Bound Disk Encryption.

Both client- and server-side components use the *José* library to perform encryption and decryption operations.

When you begin provisioning NBDE, the Clevis pin for Tang server gets a list of the Tang server's advertised asymmetric keys. Alternatively, since the keys are asymmetric, a list of Tang's public keys can be distributed out of band so that clients can operate without access to the Tang server. This mode is called *offline provisioning*.

The Clevis pin for Tang uses one of the public keys to generate a unique, cryptographically-strong encryption key. Once the data is encrypted using this key, the key is discarded. The Clevis client should store the state produced by this provisioning operation in a convenient location. This process of encrypting data is the *provisioning step*.

The LUKS version 2 (LUKS2) is the default disk-encryption format in Red Hat Enterprise Linux 8, hence, the provisioning state for NBDE is stored as a token in a LUKS2 header. The leveraging of provisioning state for NBDE by the **luksmeta** package is used only for volumes encrypted with LUKS1.

The Clevis pin for Tang supports both LUKS1 and LUKS2 without specification need. Clevis can encrypt plain-text files but you have to use the **cryptsetup** tool for encrypting block devices. See the Encrypting block devices using LUKS for more information.

When the client is ready to access its data, it loads the metadata produced in the provisioning step and it responds to recover the encryption key. This process is the *recovery step*.

In NBDE, Clevis binds a LUKS volume using a pin so that it can be automatically unlocked. After successful completion of the binding process, the disk can be unlocked using the provided Dracut unlocker.

> **NOTE**
>
> If the **kdump** kernel crash dumping mechanism is set to save the content of the system memory to a LUKS-encrypted device, you are prompted for entering a password during the second kernel boot.

12.2. INSTALLING AN ENCRYPTION CLIENT - CLEVIS

Use this procedure to deploy and start using the Clevis pluggable framework on your system.

Procedure

1. To install Clevis and its pins on a system with an encrypted volume:

   ```
   # yum install clevis
   ```

2. To decrypt data, use a **clevis decrypt** command and provide a cipher text in the JSON Web Encryption (JWE) format, for example:

   ```
   $ clevis decrypt < secret.jwe
   ```

Additional resources

- **clevis(1)** man page

- Built-in CLI help after entering the **clevis** command without any argument:

```
$ clevis
Usage: clevis COMMAND [OPTIONS]

clevis decrypt            Decrypts using the policy defined at encryption time
clevis encrypt sss        Encrypts using a Shamir's Secret Sharing policy
clevis encrypt tang       Encrypts using a Tang binding server policy
clevis encrypt tpm2       Encrypts using a TPM2.0 chip binding policy
clevis luks bind          Binds a LUKS device using the specified policy
clevis luks list          Lists pins bound to a LUKSv1 or LUKSv2 device
clevis luks pass          Returns the LUKS passphrase used for binding a particular slot.
clevis luks regen         Regenerate LUKS metadata
clevis luks report        Report any key rotation on the server side
clevis luks unbind        Unbinds a pin bound to a LUKS volume
clevis luks unlock        Unlocks a LUKS volume
```

12.3. DEPLOYING A TANG SERVER WITH SELINUX IN ENFORCING MODE

Use this procedure to deploy a Tang server running on a custom port as a confined service in SELinux enforcing mode.

Prerequisites

- The **policycoreutils-python-utils** package and its dependencies are installed.

Procedure

1. To install the **tang** package and its dependencies, enter the following command as **root**:

   ```
   # yum install tang
   ```

2. Pick an unoccupied port, for example, *7500/tcp*, and allow the **tangd** service to bind to that port:

   ```
   # semanage port -a -t tangd_port_t -p tcp 7500
   ```

   Note that a port can be used only by one service at a time, and thus an attempt to use an already occupied port implies the **ValueError: Port already defined** error message.

3. Open the port in the firewall:

   ```
   # firewall-cmd --add-port=7500/tcp
   # firewall-cmd --runtime-to-permanent
   ```

4. Enable the **tangd** service:

   ```
   # systemctl enable tangd.socket
   ```

5. Create an override file:

```
# systemctl edit tangd.socket
```

6. In the following editor screen, which opens an empty **override.conf** file located in the **/etc/systemd/system/tangd.socket.d/** directory, change the default port for the Tang server from 80 to the previously picked number by adding the following lines:

```
[Socket]
ListenStream=
ListenStream=7500
```

Save the file and exit the editor.

7. Reload the changed configuration:

```
# systemctl daemon-reload
```

8. Check that your configuration is working:

```
# systemctl show tangd.socket -p Listen
Listen=[::]:7500 (Stream)
```

9. Start the **tangd** service:

```
# systemctl start tangd.socket
```

Because **tangd** uses the **systemd** socket activation mechanism, the server starts as soon as the first connection comes in. A new set of cryptographic keys is automatically generated at the first start. To perform cryptographic operations such as manual key generation, use the **jose** utility.

Additional resources

- **tang(8)**, **semanage(8)**, **firewall-cmd(1)**, **jose(1)**, **systemd.unit(5)**, and **systemd.socket(5)** man pages

12.4. ROTATING TANG SERVER KEYS AND UPDATING BINDINGS ON CLIENTS

Use the following steps to rotate your Tang server keys and update existing bindings on clients. The precise interval at which you should rotate them depends on your application, key sizes, and institutional policy.

Alternatively, you can rotate Tang keys by using the **nbde_server** RHEL system role. See Using the nbde_server system role for setting up multiple Tang servers for more information.

Prerequisites

- A Tang server is running.

- The **clevis** and **clevis-luks** packages are installed on your clients.

- Note that **clevis luks list**, **clevis luks report**, and **clevis luks regen** have been introduced in RHEL 8.2.

Procedure

1. Rename all keys in the **/var/db/tang** key database directory to have a leading `.` to hide them from advertisement. Note that the file names in the following example differs from unique file names in the key database directory of your Tang server:

```
# cd /var/db/tang
# ls -l
-rw-r--r--. 1 root root 349 Feb  7 14:55 UV6dqXSwe1bRKG3KbJmdiR020hY.jwk
-rw-r--r--. 1 root root 354 Feb  7 14:55 y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk
# mv UV6dqXSwe1bRKG3KbJmdiR020hY.jwk .UV6dqXSwe1bRKG3KbJmdiR020hY.jwk
# mv y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk .y9hxLTQSiSB5jSEGWnjhY8fDTJU.jwk
```

2. Check that you renamed and therefore hid all keys from the Tang server advertisement:

```
# ls -l
total 0
```

3. Generate new keys using the **/usr/libexec/tangd-keygen** command in **/var/db/tang** on the Tang server:

```
# /usr/libexec/tangd-keygen /var/db/tang
# ls /var/db/tang
3ZWS6-cDrCG61UPJS2BMmPU4I54.jwk zyLuX6hijUy_PSeUEFDi7hi38.jwk
```

4. Check that your Tang server advertises the signing key from the new key pair, for example:

```
# tang-show-keys 7500
3ZWS6-cDrCG61UPJS2BMmPU4I54
```

5. On your NBDE clients, use the **clevis luks report** command to check if the keys advertised by the Tang server remains the same. You can identify slots with the relevant binding using the **clevis luks list** command, for example:

```
# clevis luks list -d /dev/sda2
1: tang '{"url":"http://tang.srv"}'
# clevis luks report -d /dev/sda2 -s 1
...
Report detected that some keys were rotated.
Do you want to regenerate luks metadata with "clevis luks regen -d /dev/sda2 -s 1"? [ynYN]
```

6. To regenerate LUKS metadata for the new keys either press **y** to the prompt of the previous command, or use the **clevis luks regen** command:

```
# clevis luks regen -d /dev/sda2 -s 1
```

7. When you are sure that all old clients use the new keys, you can remove the old keys from the Tang server, for example:

```
# cd /var/db/tang
# rm .*.jwk
```

> **WARNING**
>
> Removing the old keys while clients are still using them can result in data loss. If you accidentally remove such keys, use the **clevis luks regen** command on the clients, and provide your LUKS password manually.

Additional resources

- **tang-show-keys(1)**, **clevis-luks-list(1)**, **clevis-luks-report(1)**, and **clevis-luks-regen(1)** man pages

12.5. CONFIGURING AUTOMATED UNLOCKING USING A TANG KEY IN THE WEB CONSOLE

Configure automated unlocking of a LUKS-encrypted storage device using a key provided by a Tang server.

Prerequisites

- The RHEL 8 web console has been installed.
  For details, see Installing the web console.

- The **cockpit-storaged** package is installed on your system.

- The **cockpit.socket** service is running at port 9090.

- The **clevis**, **tang**, and **clevis-dracut** packages are installed.

- A Tang server is running.

Procedure

1. Open the RHEL web console by entering the following address in a web browser:

   https://*localhost*:9090

   Replace the *localhost* part by the remote server's host name or IP address when you connect to a remote system.

2. Provide your credentials and click **Storage**. Select an encrypted device and click **Encryption** in the **Content** part:

3. Click **+** in the **Keys** section to add a Tang key:
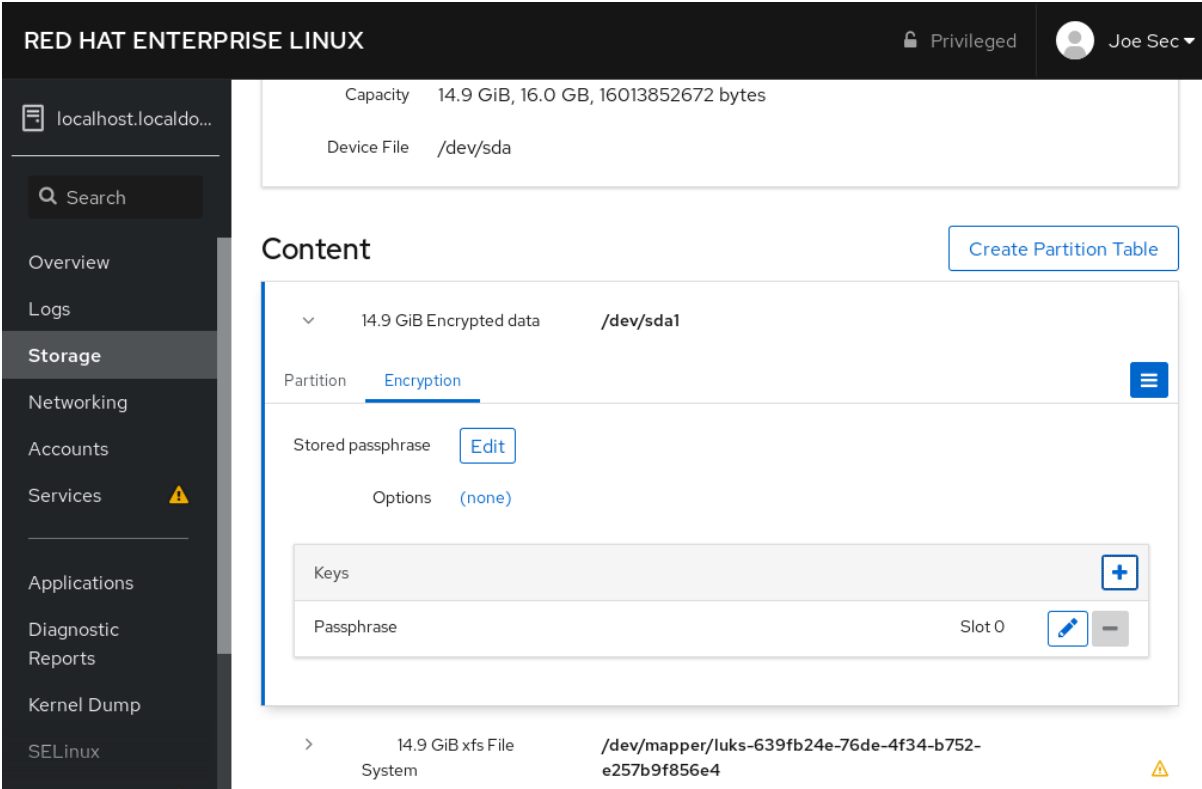
4. Provide the address of your Tang server and a password that unlocks the LUKS-encrypted device. Click **Add** to confirm:

5. The following dialog window provides a command to verify that the key hash matches. RHEL 8.2 introduced the **tang-show-keys** script, and you can obtain the key hash using the following command on the Tang server running on the port *7500*:

```
# tang-show-keys 7500
3ZWS6-cDrCG61UPJS2BMmPU4I54
```

On RHEL 8.1 and earlier, obtain the key hash using the following command:

```
# curl -s localhost:7500/adv | jose fmt -j- -g payload -y -o- | jose jwk use -i- -r -u verify -o- |
jose jwk thp -i-
3ZWS6-cDrCG61UPJS2BMmPU4I54
```

6. Click **Trust key** when the key hashes in the web console and in the output of previously listed commands are the same:

Verify key

Make sure the key hash from the Tang server matches:

3ZWS6-cDrCG61UPJS2BMmPU4I54

Manually check with SSH: `ssh localhost tang-show-keys 7500`

If tang-show-keys is not available, run the following:

```
ssh localhost "curl -s localhost:7500/adv |
  jose fmt -j- -g payload -y -o- |
  jose jwk use -i- -r -u verify -o- |
  jose jwk thp -i-"
```

Cancel   Trust key

7. To enable the early boot system to process the disk binding, click **Terminal** at the bottom of the left navigation bar and enter the following commands:

```
# yum install clevis-dracut
# grubby --update-kernel=ALL --args="rd.neednet=1"
# dracut -fv --regenerate-all
```

Verification

1. Check that the newly added Tang key is now listed in the **Keys** section with the **Keyserver** type:

| | 14.9 GiB Encrypted data | /dev/sda1 |

Partition   Encryption   ☰

Stored passphrase   Edit

Options   (none)

Keys		+
Passphrase	Slot 0	✏ −
Keyserver localhost:7500	Slot 1	✏ −

2. Verify that the bindings are available for the early boot, for example:

```
# lsinitrd | grep clevis
clevis
clevis-pin-sss
clevis-pin-tang
clevis-pin-tpm2
-rwxr-xr-x   1 root     root        1600 Feb 11 16:30 usr/bin/clevis
-rwxr-xr-x   1 root     root        1654 Feb 11 16:30 usr/bin/clevis-decrypt
...
-rwxr-xr-x   2 root     root          45 Feb 11 16:30 usr/lib/dracut/hooks/initqueue/settled/60-
clevis-hook.sh
-rwxr-xr-x   1 root     root        2257 Feb 11 16:30 usr/libexec/clevis-luks-askpass
```

Additional resources

- [Getting started using the RHEL web console](#)

12.6. BASIC NBDE AND TPM2 ENCRYPTION-CLIENT OPERATIONS

The Clevis framework can encrypt plain-text files and decrypt both ciphertexts in the JSON Web Encryption (JWE) format and LUKS-encrypted block devices. Clevis clients can use either Tang network servers or Trusted Platform Module 2.0 (TPM 2.0) chips for cryptographic operations.

The following commands demonstrate the basic functionality provided by Clevis on examples containing plain-text files. You can also use them for troubleshooting your NBDE or Clevis+TPM deployments.

Encryption client bound to a Tang server

- To check that a Clevis encryption client binds to a Tang server, use the **clevis encrypt tang** sub-command:

```
$ clevis encrypt tang '{"url":"http://tang.srv:port"}' < input-plain.txt > secret.jwe
The advertisement contains the following signing keys:

_OsIk0T-E2l6qjfdDiwVmidoZjA

Do you wish to trust these keys? [ynYN] y
```

Change the *http://tang.srv:port* URL in the previous example to match the URL of the server where **tang** is installed. The *secret.jwe* output file contains your encrypted cipher text in the JWE format. This cipher text is read from the *input-plain.txt* input file.

Alternatively, if your configuration requires a non-interactive communication with a Tang server without SSH access, you can download an advertisement and save it to a file:

```
$ curl -sfg http://tang.srv:port/adv -o adv.jws
```

Use the advertisement in the *adv.jws* file for any following tasks, such as encryption of files or messages:

```
$ echo 'hello' | clevis encrypt tang '{"url":"http://tang.srv:port","adv":"adv.jws"}'
```

- To decrypt data, use the **clevis decrypt** command and provide the cipher text (JWE):

  $ clevis decrypt < *secret.jwe* > *output-plain.txt*

Encryption client using TPM 2.0

- To encrypt using a TPM 2.0 chip, use the **clevis encrypt tpm2** sub-command with the only argument in form of the JSON configuration object:

  $ clevis encrypt tpm2 '{}' < *input-plain.txt* > *secret.jwe*

  To choose a different hierarchy, hash, and key algorithms, specify configuration properties, for example:

  $ clevis encrypt tpm2 '{"hash":"sha1","key":"rsa"}' < *input-plain.txt* > *secret.jwe*

- To decrypt the data, provide the ciphertext in the JSON Web Encryption (JWE) format:

  $ clevis decrypt < *secret.jwe* > *output-plain.txt*

The pin also supports sealing data to a Platform Configuration Registers (PCR) state. That way, the data can only be unsealed if the PCR hashes values match the policy used when sealing.

For example, to seal the data to the PCR with index 0 and 7 for the SHA-1 bank:

$ clevis encrypt tpm2 '{"pcr_bank":"sha1","pcr_ids":"0,7"}' < *input-plain.txt* > *secret.jwe*

> **WARNING**
>
> Hashes in PCRs can be rewritten, and you no longer can unlock your encrypted volume. For this reason, add a strong passphrase that enable you to unlock the encrypted volume manually even when a value in a PCR changes.
>
> If the system cannot automatically unlock your encrypted volume after an upgrade of the **shim-x64** package, follow the steps in the Clevis TPM2 no longer decrypts LUKS devices after a restart KCS article.

Additional resources

- **clevis-encrypt-tang(1)**, **clevis-luks-unlockers(7)**, **clevis(1)**, and **clevis-encrypt-tpm2(1)** man pages

- **clevis**, **clevis decrypt**, and **clevis encrypt tang** commands without any arguments show the built-in CLI help, for example:

  ```
  $ clevis encrypt tang
  Usage: clevis encrypt tang CONFIG < PLAINTEXT > JWE
  ...
  ```

12.7. CONFIGURING MANUAL ENROLLMENT OF LUKS-ENCRYPTED VOLUMES

Use the following steps to configure unlocking of LUKS-encrypted volumes with NBDE.

Prerequisites

- A Tang server is running and available.

Procedure

1. To automatically unlock an existing LUKS-encrypted volume, install the **clevis-luks** subpackage:

   ```
   # yum install clevis-luks
   ```

2. Identify the LUKS-encrypted volume for PBD. In the following example, the block device is referred as */dev/sda2*:

   ```
   # lsblk
   NAME                            MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
   sda                             8:0    0    12G  0 disk
   ├─sda1                          8:1    0     1G  0 part  /boot
   └─sda2                          8:2    0    11G  0 part
     └─luks-40e20552-2ade-4954-9d56-565aa7994fb6 253:0   0   11G  0 crypt
       ├─rhel-root                 253:0   0   9.8G  0 lvm   /
       └─rhel-swap                 253:1   0   1.2G  0 lvm   [SWAP]
   ```

3. Bind the volume to a Tang server using the **clevis luks bind** command:

   ```
   # clevis luks bind -d /dev/sda2 tang '{"url":"http://tang.srv"}'
   The advertisement contains the following signing keys:

   _OsIk0T-E2l6qjfdDiwVmidoZjA

   Do you wish to trust these keys? [ynYN] y
   You are about to initialize a LUKS device for metadata storage.
   Attempting to initialize it may result in data loss if data was
   already written into the LUKS header gap in a different format.
   A backup is advised before initialization is performed.

   Do you wish to initialize /dev/sda2? [yn] y
   Enter existing LUKS password:
   ```

   This command performs four steps:

   a. Creates a new key with the same entropy as the LUKS master key.

   b. Encrypts the new key with Clevis.

   c. Stores the Clevis JWE object in the LUKS2 header token or uses LUKSMeta if the non-default LUKS1 header is used.

   d. Enables the new key for use with LUKS.

> **NOTE**
>
> The binding procedure assumes that there is at least one free LUKS password slot. The **clevis luks bind** command takes one of the slots.

4. The volume can now be unlocked with your existing password as well as with the Clevis policy.

5. To enable the early boot system to process the disk binding, use the **dracut** tool on an already installed system:

   ```
   # yum install clevis-dracut
   ```

   In Red Hat Enterprise Linux 8, Clevis produces a generic **initrd** (initial ramdisk) without host-specific configuration options and does not automatically add parameters such as **rd.neednet=1** to the kernel command line. If your configuration relies on a Tang pin that requires network during early boot, use the **--hostonly-cmdline** argument and **dracut** adds **rd.neednet=1** when it detects a Tang binding:

   ```
   # dracut -fv --regenerate-all --hostonly-cmdline
   ```

   Alternatively, create a .conf file in the **/etc/dracut.conf.d/**, and add the **hostonly_cmdline=yes** option to the file, for example:

   ```
   # echo "hostonly_cmdline=yes" > /etc/dracut.conf.d/clevis.conf
   ```

   > **NOTE**
   >
   > You can also ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the system where Clevis is installed:
   >
   > ```
   > # grubby --update-kernel=ALL --args="rd.neednet=1"
   > ```

   Then you can use **dracut** without **--hostonly-cmdline**:

   ```
   # dracut -fv --regenerate-all
   ```

Verification

1. To verify that the Clevis JWE object is successfully placed in a LUKS header, use the **clevis luks list** command:

   ```
   # clevis luks list -d /dev/sda2
   1: tang '{"url":"http://tang.srv:port"}'
   ```

IMPORTANT

To use NBDE for clients with static IP configuration (without DHCP), pass your network configuration to the **dracut** tool manually, for example:

```
# dracut -fv --regenerate-all --kernel-cmdline
"ip=192.0.2.10::192.0.2.1:255.255.255.0::ens3:none"
```

Alternatively, create a .conf file in the **/etc/dracut.conf.d/** directory with the static network information. For example:

```
# cat /etc/dracut.conf.d/static_ip.conf
kernel_cmdline="ip=192.0.2.10::192.0.2.1:255.255.255.0::ens3:none"
```

Regenerate the initial RAM disk image:

```
# dracut -fv --regenerate-all
```

Additional resources

- **clevis-luks-bind(1)** and **dracut.cmdline(7)** man pages.

- RHEL Network boot options

12.8. CONFIGURING MANUAL ENROLLMENT OF LUKS-ENCRYPTED VOLUMES USING A TPM 2.0 POLICY

Use the following steps to configure unlocking of LUKS-encrypted volumes by using a Trusted Platform Module 2.0 (TPM 2.0) policy.

Prerequisites

- An accessible TPM 2.0-compatible device.

- A system with the 64-bit Intel or 64-bit AMD architecture.

Procedure

1. To automatically unlock an existing LUKS-encrypted volume, install the **clevis-luks** subpackage:

   ```
   # yum install clevis-luks
   ```

2. Identify the LUKS-encrypted volume for PBD. In the following example, the block device is referred as *dev/sda2*:

   ```
   # lsblk
   NAME                           MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
   sda                            8:0    0    12G 0 disk
   ├─sda1                          8:1    0     1G  0 part /boot
   └─sda2                          8:2    0    11G  0 part
     └─luks-40e20552-2ade-4954-9d56-565aa7994fb6 253:0   0    11G  0 crypt
       ├─rhel-root                 253:0   0   9.8G  0 lvm  /
       └─rhel-swap                 253:1   0   1.2G  0 lvm  [SWAP]
   ```

–

3. Bind the volume to a TPM 2.0 device using the **clevis luks bind** command, for example:

```
# clevis luks bind -d /dev/sda2 tpm2 '{"hash":"sha1","key":"rsa"}'
...
Do you wish to initialize /dev/sda2? [yn] y
Enter existing LUKS password:
```

This command performs four steps:

a. Creates a new key with the same entropy as the LUKS master key.

b. Encrypts the new key with Clevis.

c. Stores the Clevis JWE object in the LUKS2 header token or uses LUKSMeta if the non-default LUKS1 header is used.

d. Enables the new key for use with LUKS.

> **NOTE**
>
> The binding procedure assumes that there is at least one free LUKS password slot. The **clevis luks bind** command takes one of the slots.

Alternatively, if you want to seal data to specific Platform Configuration Registers (PCR) states, add the **pcr_bank** and **pcr_ids** values to the **clevis luks bind** command, for example:

```
# clevis luks bind -d /dev/sda2 tpm2
'{"hash":"sha1","key":"rsa","pcr_bank":"sha1","pcr_ids":"0,1"}'
```

> **WARNING**
>
> Because the data can only be unsealed if PCR hashes values match the policy used when sealing and the hashes can be rewritten, add a strong passphrase that enable you to unlock the encrypted volume manually when a value in a PCR changes.
>
> If the system cannot automatically unlock your encrypted volume after an upgrade of the **shim-x64** package, follow the steps in the Clevis TPM2 no longer decrypts LUKS devices after a restart KCS article.

4. The volume can now be unlocked with your existing password as well as with the Clevis policy.

5. To enable the early boot system to process the disk binding, use the **dracut** tool on an already installed system:
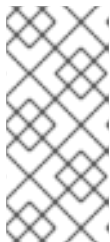
```
# yum install clevis-dracut
# dracut -fv --regenerate-all
```

Verification

1. To verify that the Clevis JWE object is successfully placed in a LUKS header, use the **clevis luks list** command:

   ```
   # clevis luks list -d /dev/sda2
   1: tpm2 '{"hash":"sha1","key":"rsa"}'
   ```

Additional resources

- **clevis-luks-bind(1)**, **clevis-encrypt-tpm2(1)**, and **dracut.cmdline(7)** man pages

12.9. REMOVING A CLEVIS PIN FROM A LUKS-ENCRYPTED VOLUME MANUALLY

Use the following procedure for manual removing the metadata created by the **clevis luks bind** command and also for wiping a key slot that contains passphrase added by Clevis.

IMPORTANT

The recommended way to remove a Clevis pin from a LUKS-encrypted volume is through the **clevis luks unbind** command. The removal procedure using **clevis luks unbind** consists of only one step and works for both LUKS1 and LUKS2 volumes. The following example command removes the metadata created by the binding step and wipe the key slot *1* on the */dev/sda2* device:

```
# clevis luks unbind -d /dev/sda2 -s 1
```

Prerequisites

- A LUKS-encrypted volume with a Clevis binding.

Procedure

1. Check which LUKS version the volume, for example */dev/sda2*, is encrypted by and identify a slot and a token that is bound to Clevis:

   ```
   # cryptsetup luksDump /dev/sda2
   LUKS header information
   Version:        2
   ...
   Keyslots:
     0: luks2
   ...
   1: luks2
       Key:        512 bits
       Priority:   normal
       Cipher:     aes-xts-plain64
   ...
       Tokens:
        0: clevis
            Keyslot:  1
   ...
   ```

In the previous example, the Clevis token is identified by *0* and the associated key slot is *1*.

2. In case of LUKS2 encryption, remove the token:

   ```
   # cryptsetup token remove --token-id 0 /dev/sda2
   ```

3. If your device is encrypted by LUKS1, which is indicated by the **Version: 1** string in the output of the **cryptsetup luksDump** command, perform this additional step with the **luksmeta wipe** command:

   ```
   # luksmeta wipe -d /dev/sda2 -s 1
   ```

4. Wipe the key slot containing the Clevis passphrase:

   ```
   # cryptsetup luksKillSlot /dev/sda2 1
   ```

Additional resources

- **clevis-luks-unbind(1)**, **cryptsetup(8)**, and **luksmeta(8)** man pages

12.10. CONFIGURING AUTOMATED ENROLLMENT OF LUKS-ENCRYPTED VOLUMES USING KICKSTART

Follow the steps in this procedure to configure an automated installation process that uses Clevis for enrollment of LUKS-encrypted volumes.

Procedure

1. Instruct Kickstart to partition the disk such that LUKS encryption has enabled for all mount points, other than **/boot**, with a temporary password. The password is temporary for this step of the enrollment process.

   ```
   part /boot --fstype="xfs" --ondisk=vda --size=256
   part / --fstype="xfs" --ondisk=vda --grow --encrypted --passphrase=temppass
   ```

   Note that OSPP-complaint systems require a more complex configuration, for example:

   ```
   part /boot --fstype="xfs" --ondisk=vda --size=256
   part / --fstype="xfs" --ondisk=vda --size=2048 --encrypted --passphrase=temppass
   part /var --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
   part /tmp --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
   part /home --fstype="xfs" --ondisk=vda --size=2048 --grow --encrypted --passphrase=temppass
   part /var/log --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
   part /var/log/audit --fstype="xfs" --ondisk=vda --size=1024 --encrypted --passphrase=temppass
   ```

2. Install the related Clevis packages by listing them in the **%packages** section:

   ```
   %packages
   clevis-dracut
   %end
   ```

3. Optionally, to ensure that you can unlock the encrypted volume manually when required, add a strong passphrase before you remove the temporary passphrase. See the How to add a passphrase, key, or keyfile to an existing LUKS device article for more information.

4. Call **clevis luks bind** to perform binding in the **%post** section. Afterward, remove the temporary password:

```
%post
curl -sfg http://tang.srv/adv -o adv.jws
clevis luks bind -f -k- -d /dev/vda2 \
tang '{"url":"http://tang.srv","adv":"adv.jws"}' <<< "temppass"
cryptsetup luksRemoveKey /dev/vda2 <<< "temppass"
%end
```

In the previous example, note that we download the advertisement from the Tang server as part of our binding configuration, enabling binding to be completely non-interactive.

> **WARNING**
>
> The **cryptsetup luksRemoveKey** command prevents any further administration of a LUKS2 device on which you apply it. You can recover a removed master key using the **dmsetup** command only for LUKS1 devices.

You can use an analogous procedure when using a TPM 2.0 policy instead of a Tang server.

Additional resources

- **clevis(1)**, **clevis-luks-bind(1)**, **cryptsetup(8)**, and **dmsetup(8)** man pages

- Installing Red Hat Enterprise Linux 8 using Kickstart

12.11. CONFIGURING AUTOMATED UNLOCKING OF A LUKS-ENCRYPTED REMOVABLE STORAGE DEVICE

Use this procedure to set up an automated unlocking process of a LUKS-encrypted USB storage device.

Procedure

1. To automatically unlock a LUKS-encrypted removable storage device, such as a USB drive, install the **clevis-udisks2** package:

```
# yum install clevis-udisks2
```

2. Reboot the system, and then perform the binding step using the **clevis luks bind** command as described in Configuring manual enrollment of LUKS-encrypted volumes, for example:

```
# clevis luks bind -d /dev/sdb1 tang '{"url":"http://tang.srv"}'
```

3. The LUKS-encrypted removable device can be now unlocked automatically in your GNOME desktop session. The device bound to a Clevis policy can be also unlocked by the **clevis luks unlock** command:

```
# clevis luks unlock -d /dev/sdb1
```

You can use an analogous procedure when using a TPM 2.0 policy instead of a Tang server.

Additional resources

- **clevis-luks-unlockers(7)** man page

12.12. DEPLOYING HIGH-AVAILABILITY NBDE SYSTEMS

Tang provides two methods for building a high-availability deployment:

Client redundancy (recommended)

Clients should be configured with the ability to bind to multiple Tang servers. In this setup, each Tang server has its own keys and clients can decrypt by contacting a subset of these servers. Clevis already supports this workflow through its **sss** plug-in. Red Hat recommends this method for a high-availability deployment.

Key sharing

For redundancy purposes, more than one instance of Tang can be deployed. To set up a second or any subsequent instance, install the **tang** packages and copy the key directory to the new host using **rsync** over **SSH**. Note that Red Hat does not recommend this method because sharing keys increases the risk of key compromise and requires additional automation infrastructure.

12.12.1. High-available NBDE using Shamir's Secret Sharing

Shamir's Secret Sharing (SSS) is a cryptographic scheme that divides a secret into several unique parts. To reconstruct the secret, a number of parts is required. The number is called threshold and SSS is also referred to as a thresholding scheme.

Clevis provides an implementation of SSS. It creates a key and divides it into a number of pieces. Each piece is encrypted using another pin including even SSS recursively. Additionally, you define the threshold **t**. If an NBDE deployment decrypts at least **t** pieces, then it recovers the encryption key and the decryption process succeeds. When Clevis detects a smaller number of parts than specified in the threshold, it prints an error message.

12.12.1.1. Example 1: Redundancy with two Tang servers

The following command decrypts a LUKS-encrypted device when at least one of two Tang servers is available:

```
# clevis luks bind -d /dev/sda1 sss '{"t":1,"pins":{"tang":[{"url":"http://tang1.srv"},
{"url":"http://tang2.srv"}]}}'
```

The previous command used the following configuration scheme:

```
{
   "t":1,
   "pins":{
      "tang":[
```

```
                {
                    "url":"http://tang1.srv"
                },
                {
                    "url":"http://tang2.srv"
                }
            ]
        }
    }
```

In this configuration, the SSS threshold **t** is set to **1** and the **clevis luks bind** command successfully reconstructs the secret if at least one from two listed **tang** servers is available.

12.12.1.2. Example 2: Shared secret on a Tang server and a TPM device

The following command successfully decrypts a LUKS-encrypted device when both the **tang** server and the **tpm2** device are available:

```
# clevis luks bind -d /dev/sda1 sss '{"t":2,"pins":{"tang":[{"url":"http://tang1.srv"}], "tpm2":
{"pcr_ids":"0,7"}}}'
```

The configuration scheme with the SSS threshold 't' set to '2' is now:

```
{
    "t":2,
    "pins":{
        "tang":[
            {
                "url":"http://tang1.srv"
            }
        ],
        "tpm2":{
            "pcr_ids":"0,7"
        }
    }
}
```

Additional resources

- **tang(8)** (section **High Availability**), **clevis(1)** (section **Shamir's Secret Sharing**), and **clevis-encrypt-sss(1)** man pages

12.13. DEPLOYMENT OF VIRTUAL MACHINES IN A NBDE NETWORK

The **clevis luks bind** command does not change the LUKS master key. This implies that if you create a LUKS-encrypted image for use in a virtual machine or cloud environment, all the instances that run this image will share a master key. This is extremely insecure and should be avoided at all times.

This is not a limitation of Clevis but a design principle of LUKS. If you wish to have encrypted root volumes in a cloud, you need to make sure that you perform the installation process (usually using Kickstart) for each instance of Red Hat Enterprise Linux in a cloud as well. The images cannot be shared without also sharing a LUKS master key.

If you intend to deploy automated unlocking in a virtualized environment, Red Hat strongly recommends

that you use systems such as lorax or virt-install together with a Kickstart file (see Configuring automated enrollment of LUKS-encrypted volumes using Kickstart) or another automated provisioning tool to ensure that each encrypted VM has a unique master key.

> **NOTE**
>
> Automated unlocking with a TPM 2.0 policy is not supported in a virtual machine.

Additional resources

- **clevis-luks-bind(1)** man page

12.14. BUILDING AUTOMATICALLY-ENROLLABLE VM IMAGES FOR CLOUD ENVIRONMENTS USING NBDE

Deploying automatically-enrollable encrypted images in a cloud environment can provide a unique set of challenges. Like other virtualization environments, it is recommended to reduce the number of instances started from a single image to avoid sharing the LUKS master key.

Therefore, the best practice is to create customized images that are not shared in any public repository and that provide a base for the deployment of a limited amount of instances. The exact number of instances to create should be defined by deployment's security policies and based on the risk tolerance associated with the LUKS master key attack vector.

To build LUKS-enabled automated deployments, systems such as Lorax or virt-install together with a Kickstart file should be used to ensure master key uniqueness during the image building process.

Cloud environments enable two Tang server deployment options which we consider here. First, the Tang server can be deployed within the cloud environment itself. Second, the Tang server can be deployed outside of the cloud on independent infrastructure with a VPN link between the two infrastructures.

Deploying Tang natively in the cloud does allow for easy deployment. However, given that it shares infrastructure with the data persistence layer of ciphertext of other systems, it may be possible for both the Tang server's private key and the Clevis metadata to be stored on the same physical disk. Access to this physical disk permits a full compromise of the ciphertext data.

> **IMPORTANT**
>
> For this reason, Red Hat strongly recommends maintaining a physical separation between the location where the data is stored and the system where Tang is running. This separation between the cloud and the Tang server ensures that the Tang server's private key cannot be accidentally combined with the Clevis metadata. It also provides local control of the Tang server if the cloud infrastructure is at risk.

12.15. DEPLOYING TANG AS A CONTAINER

The **rhel8-tang** container image provides Tang-server decryption capabilities for Clevis clients that run either in OpenShift Container Platform (OCP) clusters or in separate virtual machines.

Prerequisites

- The **podman** package and its dependencies are installed on the system.

- You have logged in on the **registry.redhat.io** container catalog using the **podman login registry.redhat.io** command. See Red Hat Container Registry Authentication for more information.

- The Clevis client is installed on systems containing LUKS-encrypted volumes that you want to automatically unlock by using a Tang server.

Procedure

1. Pull the **rhel8-tang** container image from the **registry.redhat.io** registry:

   ```
   # podman pull registry.redhat.io/rhel8/rhel8-tang
   ```

2. Run the container, specify its port, and specify the path to the Tang keys. The previous example runs the **rhel8-tang** container, specifies the port *7500*, and indicates a path to the Tang keys of the **/var/db/tang** directory:

   ```
   # podman run -d -p 7500:_7500_ -v tang-keys:/var/db/tang --name tang
   registry.redhat.io/rhel8/rhel8-tang
   ```

   Note that Tang uses port 80 by default but this may collide with other services such as the Apache HTTP server.

3. [Optional] For increased security, rotate the Tang keys periodically. You can use the **tangd-rotate-keys** script, for example:

   ```
   # podman run --rm -v tang-keys:/var/db/tang registry.redhat.io/rhel8/rhel8-tang tangd-rotate-
   keys -v -d /var/db/tang
   Rotated key 'rZAMKAseaXBe0rcKXL1hCCIq-DY.jwk' -> .'rZAMKAseaXBe0rcKXL1hCCIq-
   DY.jwk'
   Rotated key 'x1AIpc6WmnCU-CabD8_4q18vDuw.jwk' -> .'x1AIpc6WmnCU-
   CabD8_4q18vDuw.jwk'
   Created new key GrMMX_WfdqomIU_4RyjpcdlXb0E.jwk
   Created new key _dTTfn17sZZqVAp80u3ygFDHtjk.jwk
   Keys rotated successfully.
   ```

Verification

- On a system that contains LUKS-encrypted volumes for automated unlocking by the presence of the Tang server, check that the Clevis client can encrypt and decrypt a plain-text message using Tang:

   ```
   # echo test | clevis encrypt tang '{"url":"http://localhost:_7500_"}' | clevis decrypt
   The advertisement contains the following signing keys:

   x1AIpc6WmnCU-CabD8_4q18vDuw

   Do you wish to trust these keys? [ynYN] y
   test
   ```

   The previous example command shows the **test** string at the end of its output when a Tang server is available on the *localhost* URL and communicates through port *7500*.

Additional resources

- **podman(1)**, **clevis(1)**, and **tang(8)** man pages

12.16. INTRODUCTION TO THE CLEVIS AND TANG SYSTEM ROLES

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems.

RHEL 8.3 introduced Ansible roles for automated deployments of Policy-Based Decryption (PBD) solutions using Clevis and Tang. The **rhel-system-roles** package contains these system roles, the related examples, and also the reference documentation.

The **nbde_client** system role enables you to deploy multiple Clevis clients in an automated way. Note that the **nbde_client** role supports only Tang bindings, and you cannot use it for TPM2 bindings at the moment.

The **nbde_client** role requires volumes that are already encrypted using LUKS. This role supports to bind a LUKS-encrypted volume to one or more Network-Bound (NBDE) servers - Tang servers. You can either preserve the existing volume encryption with a passphrase or remove it. After removing the passphrase, you can unlock the volume only using NBDE. This is useful when a volume is initially encrypted using a temporary key or password that you should remove after the system you provision the system.

If you provide both a passphrase and a key file, the role uses what you have provided first. If it does not find any of these valid, it attempts to retrieve a passphrase from an existing binding.

PBD defines a binding as a mapping of a device to a slot. This means that you can have multiple bindings for the same device. The default slot is slot 1.

The **nbde_client** role provides also the **state** variable. Use the **present** value for either creating a new binding or updating an existing one. Contrary to a **clevis luks bind** command, you can use **state: present** also for overwriting an existing binding in its device slot. The **absent** value removes a specified binding.

Using the **nbde_server** role, you can deploy and manage a Tang server as part of an automated disk encryption solution. This role supports the following features:

- Rotating Tang keys

- Deploying and backing up Tang keys

Additional resources

- For a detailed reference on Network-Bound Disk Encryption (NBDE) role variables, install the **rhel-system-roles** package, and see the **README.md** and **README.html** files in the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/doc/rhel-system-roles/nbde_server/** directories.

- For example system-roles playbooks, install the **rhel-system-roles** package, and see the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directories.

- For more information on RHEL System Roles, see Introduction to RHEL System Roles

12.17. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS

Follow the steps to prepare and apply an Ansible playbook containing your Tang-server settings.

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system. See the [How do I download and install Red Hat Ansible Engine](#) article for more information.

Procedure

1. Enable the RHEL Ansible repository, for example:

   # subscription-manager repos --enable ansible-2-for-rhel-8-x86_64-rpms

2. Install Ansible Engine:

   # yum install ansible

3. Install RHEL system roles:

   # yum install rhel-system-roles

4. Prepare your playbook containing settings for Tang servers. You can either start from the scratch, or use one of the example playbooks from the **/usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/** directory.

   # cp /usr/share/ansible/roles/rhel-system-roles.nbde_server/examples/simple_deploy.yml *./my-tang-playbook.yml*

5. Edit the playbook in a text editor of your choice, for example:

   # vi *my-tang-playbook.yml*

6. Add the required parameters. The following example playbook ensures deploying of your Tang server and a key rotation:

   ```
   ---
   - hosts: all

     vars:
       nbde_server_rotate_keys: yes

     roles:
       - linux-system-roles.nbde_server
   ```

7. Apply the finished playbook:

   # ansible-playbook -i *host1*,*host2*,*host3 my-tang-playbook.yml*
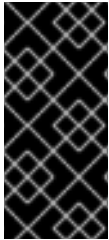
IMPORTANT

To ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the systems where Clevis is installed:

```
# grubby --update-kernel=ALL --args="rd.neednet=1"
```

Additional resources

- For more information, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/nbde_server/** and **usr/share/ansible/roles/rhel-system-roles.nbde_server/** directories.

12.18. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS

Follow the steps to prepare and apply an Ansible playbook containing your Clevis-client settings.

NOTE

The **nbde_client** system role supports only Tang bindings. This means that you cannot use it for TPM2 bindings at the moment.

Prerequisites

- Your Red Hat Ansible Engine subscription is attached to the system. See the How do I download and install Red Hat Ansible Engine article for more information.

- Your volumes are already encrypted by LUKS.

Procedure

1. Enable the RHEL Ansible repository, for example:

   ```
   # subscription-manager repos --enable ansible-2-for-rhel-8-x86_64-rpms
   ```

2. Install Ansible Engine:

   ```
   # yum install ansible
   ```

3. Install RHEL system roles:

   ```
   # yum install rhel-system-roles
   ```

4. Prepare your playbook containing settings for Clevis clients. You can either start from the scratch, or use one of the example playbooks from the **/usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/** directory.

   ```
   # cp /usr/share/ansible/roles/rhel-system-roles.nbde_client/examples/high_availability.yml
   ./my-clevis-playbook.yml
   ```

5. Edit the playbook in a text editor of your choice, for example:

```
# vi my-clevis-playbook.yml
```

6. Add the required parameters. The following example playbook configures Clevis clients for automated unlocking of two LUKS-encrypted volumes by when at least one of two Tang servers is available:

```
---
- hosts: all

  vars:
    nbde_client_bindings:
      - device: /dev/rhel/root
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com
      - device: /dev/rhel/swap
        encryption_key_src: /etc/luks/keyfile
        servers:
          - http://server1.example.com
          - http://server2.example.com

  roles:
    - linux-system-roles.nbde_client
```

7. Apply the finished playbook:

```
# ansible-playbook -i host1,host2,host3 my-clevis-playbook.yml
```

> **IMPORTANT**
>
> To ensure that networking for a Tang pin is available during early boot by using the **grubby** tool on the system where Clevis is installed:
>
> ```
> # grubby --update-kernel=ALL --args="rd.neednet=1"
> ```

Additional resources

- For details about the parameters and additional information about the **nbde_client** role, install the **rhel-system-roles** package, and see the **/usr/share/doc/rhel-system-roles/nbde_client/** and **/usr/share/ansible/roles/rhel-system-roles.nbde_client/** directories.

12.19. ADDITIONAL RESOURCES

- **tang(8)**, **clevis(1)**, **jose(1)**, and **clevis-luks-unlockers(7)** man pages

- [How to set up Network-Bound Disk Encryption with multiple LUKS devices (Clevis + Tang unlocking)](#) Knowledgebase article

CHAPTER 13. AUDITING THE SYSTEM

Audit does not provide additional security to your system; rather, it can be used to discover violations of security policies used on your system. These violations can further be prevented by additional security measures such as SELinux.

13.1. LINUX AUDIT

The Linux Audit system provides a way to track security-relevant information on your system. Based on pre-configured rules, Audit generates log entries to record as much information about the events that are happening on your system as possible. This information is crucial for mission-critical environments to determine the violator of the security policy and the actions they performed.

The following list summarizes some of the information that Audit is capable of recording in its log files:

- Date and time, type, and outcome of an event.

- Sensitivity labels of subjects and objects.

- Association of an event with the identity of the user who triggered the event.

- All modifications to Audit configuration and attempts to access Audit log files.

- All uses of authentication mechanisms, such as SSH, Kerberos, and others.

- Changes to any trusted database, such as **/etc/passwd**.

- Attempts to import or export information into or from the system.

- Include or exclude events based on user identity, subject and object labels, and other attributes.

The use of the Audit system is also a requirement for a number of security-related certifications. Audit is designed to meet or exceed the requirements of the following certifications or compliance guides:

- Controlled Access Protection Profile (CAPP)

- Labeled Security Protection Profile (LSPP)

- Rule Set Base Access Control (RSBAC)

- National Industrial Security Program Operating Manual (NISPOM)

- Federal Information Security Management Act (FISMA)

- Payment Card Industry — Data Security Standard (PCI-DSS)

- Security Technical Implementation Guides (STIG)

Audit has also been:

- Evaluated by National Information Assurance Partnership (NIAP) and Best Security Industries (BSI).

- Certified to LSPP/CAPP/RSBAC/EAL4+ on Red Hat Enterprise Linux 5.

- Certified to Operating System Protection Profile / Evaluation Assurance Level 4+ (OSPP/EAL4+) on Red Hat Enterprise Linux 6.

Use Cases

Watching file access

Audit can track whether a file or a directory has been accessed, modified, executed, or the file's attributes have been changed. This is useful, for example, to detect access to important files and have an Audit trail available in case one of these files is corrupted.

Monitoring system calls

Audit can be configured to generate a log entry every time a particular system call is used. This can be used, for example, to track changes to the system time by monitoring the **settimeofday**, **clock_adjtime**, and other time-related system calls.

Recording commands run by a user

Audit can track whether a file has been executed, so rules can be defined to record every execution of a particular command. For example, a rule can be defined for every executable in the **/bin** directory. The resulting log entries can then be searched by user ID to generate an audit trail of executed commands per user.

Recording execution of system pathnames

Aside from watching file access which translates a path to an inode at rule invocation, Audit can now watch the execution of a path even if it does not exist at rule invocation, or if the file is replaced after rule invocation. This allows rules to continue to work after upgrading a program executable or before it is even installed.

Recording security events

The **pam_faillock** authentication module is capable of recording failed login attempts. Audit can be set up to record failed login attempts as well and provides additional information about the user who attempted to log in.

Searching for events

Audit provides the **ausearch** utility, which can be used to filter the log entries and provide a complete audit trail based on several conditions.

Running summary reports

The **aureport** utility can be used to generate, among other things, daily reports of recorded events. A system administrator can then analyze these reports and investigate suspicious activity further.

Monitoring network access

The **iptables** and **ebtables** utilities can be configured to trigger Audit events, allowing system administrators to monitor network access.

> **NOTE**
>
> System performance may be affected depending on the amount of information that is collected by Audit.

13.2. AUDIT SYSTEM ARCHITECTURE

The Audit system consists of two main parts: the user-space applications and utilities, and the kernel-side system call processing. The kernel component receives system calls from user-space applications and filters them through one of the following filters: **user**, **task**, **fstype**, or **exit**.

Once a system call passes the **exclude** filter, it is sent through one of the aforementioned filters, which, based on the Audit rule configuration, sends it to the Audit daemon for further processing.

The user-space Audit daemon collects the information from the kernel and creates entries in a log file. Other Audit user-space utilities interact with the Audit daemon, the kernel Audit component, or the Audit log files:

- **auditctl** – the Audit control utility interacts with the kernel Audit component to manage rules and to control many settings and parameters of the event generation process.

- The remaining Audit utilities take the contents of the Audit log files as input and generate output based on user's requirements. For example, the **aureport** utility generates a report of all recorded events.

In RHEL 8, the Audit dispatcher daemon (**audisp**) functionality is integrated in the Audit daemon (**auditd**). Configuration files of plugins for the interaction of real-time analytical programs with Audit events are located in the **/etc/audit/plugins.d/** directory by default.

13.3. CONFIGURING AUDITD FOR A SECURE ENVIRONMENT

The default **auditd** configuration should be suitable for most environments. However, if your environment has to meet strict security policies, the following settings are suggested for the Audit daemon configuration in the **/etc/audit/auditd.conf** file:

log_file

The directory that holds the Audit log files (usually **/var/log/audit/**) should reside on a separate mount point. This prevents other processes from consuming space in this directory and provides accurate detection of the remaining space for the Audit daemon.

max_log_file

Specifies the maximum size of a single Audit log file, must be set to make full use of the available space on the partition that holds the Audit log files.

max_log_file_action

Decides what action is taken once the limit set in **max_log_file** is reached, should be set to **keep_logs** to prevent Audit log files from being overwritten.

space_left

Specifies the amount of free space left on the disk for which an action that is set in the **space_left_action** parameter is triggered. Must be set to a number that gives the administrator enough time to respond and free up disk space. The **space_left** value depends on the rate at which the Audit log files are generated.

space_left_action

It is recommended to set the **space_left_action** parameter to **email** or **exec** with an appropriate notification method.

admin_space_left

Specifies the absolute minimum amount of free space for which an action that is set in the **admin_space_left_action** parameter is triggered, must be set to a value that leaves enough space to log actions performed by the administrator.

admin_space_left_action

Should be set to **single** to put the system into single-user mode and allow the administrator to free up some disk space.

disk_full_action

Specifies an action that is triggered when no free space is available on the partition that holds the Audit log files, must be set to **halt** or **single**. This ensures that the system is either shut down or operating in single-user mode when Audit can no longer log events.

disk_error_action

Specifies an action that is triggered in case an error is detected on the partition that holds the Audit log files, must be set to **syslog**, **single**, or **halt**, depending on your local security policies regarding the handling of hardware malfunctions.

flush

Should be set to **incremental_async**. It works in combination with the **freq** parameter, which determines how many records can be sent to the disk before forcing a hard synchronization with the hard drive. The **freq** parameter should be set to **100**. These parameters assure that Audit event data is synchronized with the log files on the disk while keeping good performance for bursts of activity.

The remaining configuration options should be set according to your local security policy.

13.4. STARTING AND CONTROLLING AUDITD

Once **auditd** is configured, start the service to collect Audit information and store it in the log files. Use the following command as the root user to start **auditd**:

```
# service auditd start
```

To configure **auditd** to start at boot time:

```
# systemctl enable auditd
```

A number of other actions can be performed on **auditd** using the **service auditd *action*** command, where *action* can be one of the following:

stop

Stops **auditd**.

restart

Restarts **auditd**.

reload or **force-reload**

Reloads the configuration of **auditd** from the **/etc/audit/auditd.conf** file.

rotate

Rotates the log files in the **/var/log/audit/** directory.

resume

Resumes logging of Audit events after it has been previously suspended, for example, when there is not enough free space on the disk partition that holds the Audit log files.

condrestart or **try-restart**

Restarts **auditd** only if it is already running.

status

Displays the running status of **auditd**.

> **NOTE**
>
> The **service** command is the only way to correctly interact with the **auditd** daemon. You need to use the **service** command so that the **auid** value is properly recorded. You can use the **systemctl** command only for two actions: **enable** and **status**.

13.5. UNDERSTANDING AUDIT LOG FILES

By default, the Audit system stores log entries in the **/var/log/audit/audit.log** file; if log rotation is enabled, rotated **audit.log** files are stored in the same directory.

Add the following Audit rule to log every attempt to read or modify the **/etc/ssh/sshd_config** file:

```
# auditctl -w /etc/ssh/sshd_config -p warx -k sshd_config
```

If the **auditd** daemon is running, for example, using the following command creates a new event in the Audit log file:

```
$ cat /etc/ssh/sshd_config
```

This event in the **audit.log** file looks as follows:

```
type=SYSCALL msg=audit(1364481363.243:24287): arch=c000003e syscall=2 success=no exit=-13
a0=7fffd19c5592 a1=0 a2=7fffd19c4b50 a3=a items=1 ppid=2686 pid=3538 auid=1000 uid=1000
gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=pts0 ses=1
comm="cat" exe="/bin/cat" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
key="sshd_config"
type=CWD msg=audit(1364481363.243:24287):  cwd="/home/shadowman"
type=PATH msg=audit(1364481363.243:24287): item=0 name="/etc/ssh/sshd_config" inode=409248
dev=fd:00 mode=0100600 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:etc_t:s0
nametype=NORMAL cap_fp=none cap_fi=none cap_fe=0 cap_fver=0
type=PROCTITLE msg=audit(1364481363.243:24287) :
proctitle=636174002F6574632F7373682F737368645F636F6E666967
```

The above event consists of four records, which share the same time stamp and serial number. Records always start with the **type=** keyword. Each record consists of several *name=value* pairs separated by a white space or a comma. A detailed analysis of the above event follows:

First Record

type=SYSCALL

The **type** field contains the type of the record. In this example, the **SYSCALL** value specifies that this record was triggered by a system call to the kernel.

msg=audit(1364481363.243:24287):

The **msg** field records:

- a time stamp and a unique ID of the record in the form **audit(*time_stamp*:*ID*)**. Multiple records can share the same time stamp and ID if they were generated as part of the same Audit event. The time stamp is using the Unix time format – seconds since 00:00:00 UTC on 1 January 1970.

- various event-specific *name=value* pairs provided by the kernel or user-space applications.

arch=c000003e

The **arch** field contains information about the CPU architecture of the system. The value, **c000003e**, is encoded in hexadecimal notation. When searching Audit records with the **ausearch** command, use the **-i** or **--interpret** option to automatically convert hexadecimal values into their human-readable equivalents. The **c000003e** value is interpreted as **x86_64**.

syscall=2

The **syscall** field records the type of the system call that was sent to the kernel. The value, **2**, can be matched with its human-readable equivalent in the **/usr/include/asm/unistd_64.h** file. In this case, **2** is the **open** system call. Note that the **ausyscall** utility allows you to convert system call numbers to their human-readable equivalents. Use the **ausyscall --dump** command to display a listing of all system calls along with their numbers. For more information, see the **ausyscall**(8) man page.

success=no

The **success** field records whether the system call recorded in that particular event succeeded or failed. In this case, the call did not succeed.

exit=-13

The **exit** field contains a value that specifies the exit code returned by the system call. This value varies for a different system call. You can interpret the value to its human-readable equivalent with the following command:

```
# ausearch --interpret --exit -13
```

Note that the previous example assumes that your Audit log contains an event that failed with exit code **-13**.

a0=7fffd19c5592, a1=0, a2=7fffd19c5592, a3=a

The **a0** to **a3** fields record the first four arguments, encoded in hexadecimal notation, of the system call in this event. These arguments depend on the system call that is used; they can be interpreted by the **ausearch** utility.

items=1

The **items** field contains the number of PATH auxiliary records that follow the syscall record.

ppid=2686

The **ppid** field records the Parent Process ID (PPID). In this case, **2686** was the PPID of the parent process such as **bash**.

pid=3538

The **pid** field records the Process ID (PID). In this case, **3538** was the PID of the **cat** process.

auid=1000

The **auid** field records the Audit user ID, that is the loginuid. This ID is assigned to a user upon login and is inherited by every process even when the user's identity changes, for example, by switching user accounts with the **su - john** command.

uid=1000

The **uid** field records the user ID of the user who started the analyzed process. The user ID can be interpreted into user names with the following command: **ausearch -i --uid** *UID*.

gid=1000

The **gid** field records the group ID of the user who started the analyzed process.

euid=1000

The **euid** field records the effective user ID of the user who started the analyzed process.

suid=1000

The **suid** field records the set user ID of the user who started the analyzed process.

fsuid=1000

The **fsuid** field records the file system user ID of the user who started the analyzed process.

egid=1000

The **egid** field records the effective group ID of the user who started the analyzed process.

sgid=1000

The **sgid** field records the set group ID of the user who started the analyzed process.

fsgid=1000

The **fsgid** field records the file system group ID of the user who started the analyzed process.

tty=pts0

The **tty** field records the terminal from which the analyzed process was invoked.

ses=1

The **ses** field records the session ID of the session from which the analyzed process was invoked.

comm="cat"

The **comm** field records the command-line name of the command that was used to invoke the analyzed process. In this case, the **cat** command was used to trigger this Audit event.

exe="/bin/cat"

The **exe** field records the path to the executable that was used to invoke the analyzed process.

subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023

The **subj** field records the SELinux context with which the analyzed process was labeled at the time of execution.

key="sshd_config"

The **key** field records the administrator-defined string associated with the rule that generated this event in the Audit log.

Second Record

type=CWD

In the second record, the **type** field value is **CWD** — current working directory. This type is used to record the working directory from which the process that invoked the system call specified in the first record was executed.
The purpose of this record is to record the current process's location in case a relative path winds up being captured in the associated PATH record. This way the absolute path can be reconstructed.

msg=audit(1364481363.243:24287)

The **msg** field holds the same time stamp and ID value as the value in the first record. The time stamp is using the Unix time format - seconds since 00:00:00 UTC on 1 January 1970.

cwd="/home/*user_name*"

The **cwd** field contains the path to the directory in which the system call was invoked.

Third Record

type=PATH

In the third record, the **type** field value is **PATH**. An Audit event contains a **PATH**-type record for every path that is passed to the system call as an argument. In this Audit event, only one path (**/etc/ssh/sshd_config**) was used as an argument.

msg=audit(1364481363.243:24287):

The **msg** field holds the same time stamp and ID value as the value in the first and second record.

item=0

The **item** field indicates which item, of the total number of items referenced in the **SYSCALL** type record, the current record is. This number is zero-based; a value of **0** means it is the first item.

name="/etc/ssh/sshd_config"

The **name** field records the path of the file or directory that was passed to the system call as an argument. In this case, it was the **/etc/ssh/sshd_config** file.

inode=409248

The **inode** field contains the inode number associated with the file or directory recorded in this event. The following command displays the file or directory that is associated with the **409248** inode number:

```
# find / -inum 409248 -print
/etc/ssh/sshd_config
```

dev=fd:00

The **dev** field specifies the minor and major ID of the device that contains the file or directory recorded in this event. In this case, the value represents the **/dev/fd/0** device.

mode=0100600

The **mode** field records the file or directory permissions, encoded in numerical notation as returned by the **stat** command in the **st_mode** field. See the **stat(2)** man page for more information. In this case, **0100600** can be interpreted as **-rw-------**, meaning that only the root user has read and write permissions to the **/etc/ssh/sshd_config** file.

ouid=0

The **ouid** field records the object owner's user ID.

ogid=0

The **ogid** field records the object owner's group ID.

rdev=00:00

The **rdev** field contains a recorded device identifier for special files only. In this case, it is not used as the recorded file is a regular file.

obj=system_u:object_r:etc_t:s0

The **obj** field records the SELinux context with which the recorded file or directory was labeled at the time of execution.

nametype=NORMAL

The **nametype** field records the intent of each path record's operation in the context of a given syscall.

cap_fp=none

The **cap_fp** field records data related to the setting of a permitted file system-based capability of the file or directory object.

cap_fi=none

The **cap_fi** field records data related to the setting of an inherited file system-based capability of the file or directory object.

cap_fe=0

The **cap_fe** field records the setting of the effective bit of the file system-based capability of the file or directory object.

cap_fver=0

The **cap_fver** field records the version of the file system-based capability of the file or directory object.

Fourth Record

type=PROCTITLE

The **type** field contains the type of the record. In this example, the **PROCTITLE** value specifies that this record gives the full command-line that triggered this Audit event, triggered by a system call to the kernel.

proctitle=636174002F6574632F7373682F737368645F636F6E666967

The **proctitle** field records the full command-line of the command that was used to invoke the analyzed process. The field is encoded in hexadecimal notation to not allow the user to influence the Audit log parser. The text decodes to the command that triggered this Audit event. When searching Audit records with the **ausearch** command, use the **-i** or **--interpret** option to automatically convert hexadecimal values into their human-readable equivalents. The **636174002F6574632F7373682F737368645F636F6E666967** value is interpreted as **cat /etc/ssh/sshd_config**.

13.6. USING AUDITCTL FOR DEFINING AND EXECUTING AUDIT RULES

The Audit system operates on a set of rules that define what is captured in the log files. Audit rules can be set either on the command line using the **auditctl** utility or in the **/etc/audit/rules.d/** directory.

The **auditctl** command enables you to control the basic functionality of the Audit system and to define rules that decide which Audit events are logged.

File-system rules examples

1. To define a rule that logs all write access to, and every attribute change of, the **/etc/passwd** file:

   ```
   # auditctl -w /etc/passwd -p wa -k passwd_changes
   ```

2. To define a rule that logs all write access to, and every attribute change of, all the files in the **/etc/selinux/** directory:

   ```
   # auditctl -w /etc/selinux/ -p wa -k selinux_changes
   ```

System-call rules examples

1. To define a rule that creates a log entry every time the **adjtimex** or **settimeofday** system calls are used by a program, and the system uses the 64-bit architecture:

   ```
   # auditctl -a always,exit -F arch=b64 -S adjtimex -S settimeofday -k time_change
   ```

2. To define a rule that creates a log entry every time a file is deleted or renamed by a system user whose ID is 1000 or larger:

   ```
   # auditctl -a always,exit -S unlink -S unlinkat -S rename -S renameat -F auid>=1000 -F auid!=4294967295 -k delete
   ```

   Note that the **-F auid!=4294967295** option is used to exclude users whose login UID is not set.

Executable-file rules

To define a rule that logs all execution of the **/bin/id** program, execute the following command:

```
# auditctl -a always,exit -F exe=/bin/id -F arch=b64 -S execve -k execution_bin_id
```

Additional resources

- **audictl(8)** man page.

13.7. DEFINING PERSISTENT AUDIT RULES

To define Audit rules that are persistent across reboots, you must either directly include them in the **/etc/audit/rules.d/audit.rules** file or use the **augenrules** program that reads rules located in the **/etc/audit/rules.d/** directory.

Note that the **/etc/audit/audit.rules** file is generated whenever the **auditd** service starts. Files in **/etc/audit/rules.d/** use the same **auditctl** command-line syntax to specify the rules. Empty lines and text following a hash sign (#) are ignored.

Furthermore, you can use the **auditctl** command to read rules from a specified file using the **-R** option, for example:

```
# auditctl -R /usr/share/audit/sample-rules/30-stig.rules
```

13.8. USING PRE-CONFIGURED RULES FILES

In the **/usr/share/audit/sample-rules** directory, the **audit** package provides a set of pre-configured rules files according to various certification standards:

30-nispom.rules

Audit rule configuration that meets the requirements specified in the Information System Security chapter of the National Industrial Security Program Operating Manual.

30-ospp-v42*.rules

Audit rule configuration that meets the requirements defined in the OSPP (Protection Profile for General Purpose Operating Systems) profile version 4.2.

30-pci-dss-v31.rules

Audit rule configuration that meets the requirements set by Payment Card Industry Data Security Standard (PCI DSS) v3.1.

30-stig.rules

Audit rule configuration that meets the requirements set by Security Technical Implementation Guides (STIG).

To use these configuration files, copy them to the **/etc/audit/rules.d/** directory and use the **augenrules --load** command, for example:

```
# cd /usr/share/audit/sample-rules/
# cp 10-base-config.rules 30-stig.rules 31-privileged.rules 99-finalize.rules /etc/audit/rules.d/
# augenrules --load
```

You can order Audit rules using a numbering scheme. See the **/usr/share/audit/sample-rules/README-rules** file for more information.

Additional resources

- **audit.rules(7)** man page.

13.9. USING AUGENRULES TO DEFINE PERSISTENT RULES

The **augenrules** script reads rules located in the **/etc/audit/rules.d/** directory and compiles them into an **audit.rules** file. This script processes all files that end with **.rules** in a specific order based on their natural sort order. The files in this directory are organized into groups with the following meanings:

- 10 – Kernel and auditctl configuration

- 20 – Rules that could match general rules but you want a different match

- 30 – Main rules

- 40 – Optional rules

- 50 – Server-specific rules

- 70 – System local rules

- 90 – Finalize (immutable)

The rules are not meant to be used all at once. They are pieces of a policy that should be thought out and individual files copied to **/etc/audit/rules.d/**. For example, to set a system up in the STIG configuration, copy rules **10-base-config**, **30-stig**, **31-privileged**, and **99-finalize**.

Once you have the rules in the **/etc/audit/rules.d/** directory, load them by running the **augenrules** script with the **--load** directive:

```
# augenrules --load
/sbin/augenrules: No change
No rules
enabled 1
failure 1
pid 742
rate_limit 0
...
```

Additional resources

- **audit.rules(8)** and **augenrules(8)** man pages.

13.10. DISABLING AUGENRULES

Use the following steps to disable the **augenrules** utility. This switches Audit to use rules defined in the **/etc/audit/audit.rules** file.

Procedure

1. Copy the **/usr/lib/systemd/system/auditd.service** file to the **/etc/systemd/system/** directory:

   ```
   # cp -f /usr/lib/systemd/system/auditd.service /etc/systemd/system/
   ```

2. Edit the **/etc/systemd/system/auditd.service** file in a text editor of your choice, for example:

```
# vi /etc/systemd/system/auditd.service
```

3. Comment out the line containing **augenrules**, and uncomment the line containing the **auditctl -R** command:

```
#ExecStartPost=-/sbin/augenrules --load
ExecStartPost=-/sbin/auditctl -R /etc/audit/audit.rules
```

4. Reload the **systemd** daemon to fetch changes in the **auditd.service** file:

```
# systemctl daemon-reload
```

5. Restart the **auditd** service:

```
# service auditd restart
```

Additional resources

- **augenrules(8)** and **audit.rules(8)** man pages.

- Auditd service restart overrides changes made to /etc/audit/audit.rules .

13.11. RELATED INFORMATION

- The RHEL Audit System Reference Knowledgebase article.

- The Auditd execution options in a container Knowledgebase article.

- The Linux Audit Documentation Project page .

- The **audit** package provides documentation in the **/usr/share/doc/audit/** directory.

- **auditd(8)**, **auditctl(8)**, **ausearch(8)**, **audit.rules(7)**, **audispd.conf(5)**, **audispd(8)**, **auditd.conf(5)**, **ausearch-expression(5)**, **aulast(8)**, **aulastlog(8)**, **aureport(8)**, **ausyscall(8)**, **autrace(8)**, and **auvirt(8)** man pages.

CHAPTER 14. BLOCKING AND ALLOWING APPLICATIONS USING FAPOLICYD

Setting and enforcing a policy that either allows or denies application execution based on a rule set efficiently prevents the execution of unknown and potentially malicious software.

14.1. INTRODUCTION TO FAPOLICYD

The **fapolicyd** software framework controls the execution of applications based on a user-defined policy. This is one of the most efficient ways to prevent running untrusted and possibly malicious applications on the system.

The **fapolicyd** framework provides the following components:

- **fapolicyd** service

- **fapolicyd** command-line utilities

- **fapolicyd** RPM plugin

- **fapolicyd** rule language

The administrator can define the **allow** and **deny** execution rules for any application with the possibility of auditing based on a path, hash, MIME type, or trust.

The **fapolicyd** framework introduces the concept of trust. An application is trusted when it is properly installed by the system package manager, and therefore it is registered in the system RPM database. The **fapolicyd** daemon uses the RPM database as a list of trusted binaries and scripts. The **fapolicyd** RPM plugin registers any system update that is handled by either the YUM package manager or the RPM Package Manager. The plugin notifies the **fapolicyd** daemon about changes in this database. Other ways of adding applications require the creation of custom rules and restarting the **fapolicyd** service.

The **fapolicyd** service configuration is located in the **/etc/fapolicyd/** directory with the following structure:

- The **fapolicyd.rules** file contains **allow** and **deny** execution rules.

- The **fapolicyd.conf** file contains daemon's configuration options. This file is useful primarily for performance-tuning purposes.

You can use one of the ways for **fapolicyd** integrity checking:

- file-size checking

- comparing SHA-256 hashes

- Integrity Measurement Architecture (IMA) subsystem

By default, **fapolicyd** does no integrity checking. Integrity checking based on the file size is fast, but an attacker can replace the content of the file and preserve its byte size. Computing and checking SHA-256 checksums is more secure, but it affects the performance of the system. The **integrity = ima** option in **fapolicyd.conf** requires support for files extended attributes (also known as *xattr*) on all file systems containing executable files.

Additional resources

- **fapolicyd(8)**, **fapolicyd.rules(5)**, and **fapolicyd.conf(5)** man pages.

- The Enhancing security with the kernel integrity subsystem chapter in the RHEL 8 Managing, monitoring, and updating the kernel document.

14.2. DEPLOYING FAPOLICYD

To deploy the **fapolicyd** framework in RHEL:

Procedure

1. Install the **fapolicyd** package:

   ```
   # yum install fapolicyd
   ```

2. Enable and start the **fapolicyd** service:

   ```
   # systemctl enable --now fapolicyd
   ```

Verification

1. Verify that the **fapolicyd** service is running correctly:

   ```
   # systemctl status fapolicyd
   ● fapolicyd.service - File Access Policy Daemon
     Loaded: loaded (/usr/lib/systemd/system/fapolicyd.service; enabled; vendor p>
     Active: active (running) since Tue 2019-10-15 18:02:35 CEST; 55s ago
    Process: 8818 ExecStart=/usr/sbin/fapolicyd (code=exited, status=0/SUCCESS)
   Main PID: 8819 (fapolicyd)
      Tasks: 4 (limit: 11500)
     Memory: 78.2M
     CGroup: /system.slice/fapolicyd.service
             └─8819 /usr/sbin/fapolicyd

   Oct 15 18:02:35 localhost.localdomain systemd[1]: Starting File Access Policy D>
   Oct 15 18:02:35 localhost.localdomain fapolicyd[8819]: Initialization of the da>
   Oct 15 18:02:35 localhost.localdomain fapolicyd[8819]: Reading RPMDB into memory
   Oct 15 18:02:35 localhost.localdomain systemd[1]: Started File Access Policy Da>
   Oct 15 18:02:36 localhost.localdomain fapolicyd[8819]: Creating database
   ```

2. Log in as a user without root privileges, and check that **fapolicyd** is working, for example:

   ```
   $ cp /bin/ls /tmp
   $ /tmp/ls
   bash: /tmp/ls: Operation not permitted
   ```

14.3. MARKING FILES AS TRUSTED USING AN ADDITIONAL SOURCE OF TRUST

You can use this procedure for using an additional source of trust for **fapolicyd**. Before RHEL 8.3, **fapolicyd** trusted only files contained in the RPM database. The **fapolicyd** framework now supports also

use of the **/etc/fapolicyd/fapolicyd.trust** plain-text file as a source of trust. You can either modify **fapolicyd.trust** directly with a text editor or through **fapolicyd** CLI commands.
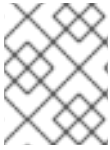
> **NOTE**
>
> Prefer marking files as trusted using **fapolicyd.trust** instead of writing custom **fapolicyd** rules.

Prerequisites

- The **fapolicyd** framework is deployed on your system.

Procedure

1. Copy your custom binary to the required directory, for example:

   ```
   $ cp /bin/ls /tmp
   $ /tmp/ls
   bash: /tmp/ls: Operation not permitted
   ```

2. Mark your custom binary as trusted:

   ```
   # fapolicyd-cli --file add /tmp/ls
   ```

   Note that previous command add the corresponding line to **/etc/fapolicyd/fapolicyd.trust**.

3. Update the **fapolicyd** database:

   ```
   # fapolicyd-cli --update
   ```

4. Restart **fapolicyd**:

   ```
   # systemctl restart fapolicyd
   ```

Verification

1. Check that your custom binary can be now executed, for example:

   ```
   $ /tmp/ls
   ls
   ```

Additional resources

- **fapolicyd.trust(5)** man page.

14.4. ADDING CUSTOM ALLOW AND DENY RULES FOR FAPOLICYD

The default set of rules in the **fapolicyd** package does not affect system functions. For custom scenarios, such as storing binaries and scripts in a non-standard directory or adding applications without the **yum** or **rpm** installers, you must modify existing or add new rules. The following steps demonstrate adding a new rule to allow a custom binary.

Prerequisites

Prerequisites

- The **fapolicyd** framework is deployed on your system.

Procedure

1. Copy your custom binary to the required directory, for example:

   ```
   $ cp /bin/ls /tmp
   $ /tmp/ls
   bash: /tmp/ls: Operation not permitted
   ```

2. Stop the **fapolicyd** service:

   ```
   # systemctl stop fapolicyd
   ```

3. Use debug mode to identify a corresponding rule. Because the output of the **fapolicyd --debug** command is verbose and you can stop it only by pressing **Ctrl**+**C** or killing the corresponding process, redirect the error output to a file:

   ```
   # fapolicyd --debug 2> fapolicy.output &
   [1] 51341
   ```

   Alternatively, you can run **fapolicyd** debug mode in another terminal.

4. Repeat the command that was not permitted:

   ```
   $ /tmp/ls
   bash: /tmp/ls: Operation not permitted
   ```

5. Stop debug mode by resuming it in the foreground and pressing **Ctrl**+**C**:

   ```
   # fg
   fapolicyd --debug
   ^Cshutting down...
   Inter-thread max queue depth 1
   Allowed accesses: 2
   Denied accesses: 1
   [...]
   ```

   Alternatively, kill the process of **fapolicyd** debug mode:

   ```
   # kill 51341
   ```

6. Find a rule that denies the execution of your application:

   ```
   # cat fapolicy.output
   [...]
   rule:9 dec=deny_audit perm=execute auid=1000 pid=51362 exe=/usr/bin/bash : file=/tmp/ls
   ftype=application/x-executable
   [...]
   ```

7. Add a new **allow** rule *before* the rule that denied the execution of your custom binary in the **/etc/fapolicyd/fapolicyd.rules** file. The output of the previous command indicated that the rule is the rule number *9* in this example:

```
allow perm=execute exe=/usr/bin/bash trust=1 : path=/tmp/ls ftype=application/x-executable trust=0
```

Alternatively, you can allow executions of all binaries in the **/tmp** directory by adding the following rule in the **/etc/fapolicyd/fapolicyd.rules** file:

```
allow perm=execute exe=/usr/bin/bash trust=1 : dir=/tmp/ all trust=0
```

8. To prevent changes in the content of your custom binary, define the required rule using an SHA-256 checksum:

```
$ sha256sum /tmp/ls
780b75c90b2d41ea41679fcb358c892b1251b68d1927c80fbc0d9d148b25e836  ls
```

Change the rule to the following definition:

```
allow perm=execute exe=/usr/bin/bash trust=1 : sha256hash=780b75c90b2d41ea41679fcb358c892b1251b68d1927c80fbc0d9d148b25e836
```

9. Start the **fapolicyd** service:

```
# systemctl start fapolicyd
```

Verification

1. Check that your custom binary can be now executed, for example:

```
$ /tmp/ls
ls
```

Additional resources

- **fapolicyd.trust(5)** man page.

14.5. ENABLING FAPOLICYD INTEGRITY CHECKS

By default, **fapolicyd** does not perform integrity checking. You can configure **fapolicyd** to perform integrity checks by comparing either file sizes or SHA-256 hashes. You can also set integrity checks by using the Integrity Measurement Architecture (IMA) subsystem.

Prerequisites

- The **fapolicyd** framework is deployed on your system.

Procedure

1. Open the **/etc/fapolicyd/fapolicyd.conf** file in a text editor of your choice, for example:

> # vi /etc/fapolicyd/fapolicyd.conf

2. Change the value of the **integrity** option from **none** to **sha256**, save the file, and exit the editor:

> integrity = sha256

3. Restart the **fapolicyd** service:

> # systemctl restart fapolicyd

Verification

1. Back up the file used for the verification:

> # cp /bin/more /bin/more.bak

2. Change the content of the **/bin/more** binary:

> # cat /bin/less > /bin/more

3. Use the changed binary as a regular user:

> # su *example.user*
> $ /bin/more /etc/redhat-release
> bash: /bin/more: Operation not permitted

4. Revert the changes:

> # mv -f /bin/more.bak /bin/more

14.6. TROUBLESHOOTING PROBLEMS RELATED TO FAPOLICYD

The following section provides tips for basic troubleshooting of the **fapolicyd** application framework and guidance for adding applications using the **rpm** command.

Installing applications using **rpm**

- If you install an application using the **rpm** command, you have to perform a manual refresh of the **fapolicyd** RPM database:

    1. Install your *application*:

    > # rpm -i *application*.rpm

    2. Refresh the database:

    > # fapolicyd-cli --update

    If you skip this step, the system can freeze and must be restarted.

Service status

- If **fapolicyd** does not work correctly, check the service status:

  ```
  # systemctl status fapolicyd
  ```

Debug mode

- Debug mode provides detailed information about matched rules, database status, and more. To switch **fapolicyd** to debug mode:

  1. Stop the **fapolicyd** service:

     ```
     # systemctl stop fapolicyd
     ```

  2. Use debug mode to identify a corresponding rule:

     ```
     # fapolicyd --debug
     ```

     Because the output of the **fapolicyd --debug** command is verbose, you can redirect the error output to a file:

     ```
     # fapolicyd --debug 2> fapolicy.output
     ```

Removing the **fapolicyd** database

- To solve problems related to the **fapolicyd** database, try to remove the database file:

  ```
  # systemctl stop fapolicyd
  # fapolicyd-cli --delete-db
  ```

> **WARNING**
>
> Do not remove the **/var/lib/fapolicyd/** directory. The **fapolicyd** framework automatically restores only the database file in this directory.

Dumping the **fapolicyd** database

- The **fapolicyd** contains entries from all enabled trust sources. You can check the entries after dumping the database:

  ```
  # fapolicyd-cli --dump-db
  ```

Application pipe

- In rare cases, removing the **fapolicyd** pipe file can solve a lockup:

  ```
  # rm -f /var/run/fapolicyd/fapolicyd.fifo
  ```

Additional resources

- **fapolicyd-cli(1)** man page.

14.7. ADDITIONAL RESOURCES

- **fapolicyd**-related man pages listed by using the **man -k fapolicyd** command.

- The FOSDEM 2020 fapolicyd presentation.

CHAPTER 15. PROTECTING SYSTEMS AGAINST INTRUSIVE USB DEVICES

USB devices can be loaded with spyware, malware, or Trojans, which can steal your data or damage your system. As a Red Hat Enterprise Linux administrator, you can prevent such USB attacks with **USBGuard**.
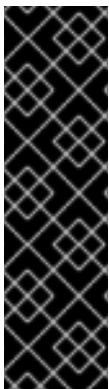
15.1. USBGUARD

With the USBGuard software framework, you can protect your systems against intrusive USB devices by using basic lists of permitted and forbidden devices based on the USB device authorization feature in the kernel.

The USBGuard framework provides the following components:

- The system service component with an inter-process communication (IPC) interface for dynamic interaction and policy enforcement

- The command-line interface to interact with a running **usbguard** system service

- The rule language for writing USB device authorization policies

- The C++ API for interacting with the system service component implemented in a shared library

The **usbguard** system service configuration file (**/etc/usbguard/usbguard-daemon.conf**) includes the options to authorize the users and groups to use the IPC interface.

> **IMPORTANT**
>
> The system service provides the USBGuard public IPC interface. In Red Hat Enterprise Linux, the access to this interface is limited to the root user only by default.
>
> Consider setting either the **IPCAccessControlFiles** option (recommended) or the **IPCAllowedUsers** and **IPCAllowedGroups** options to limit access to the IPC interface.
>
> Ensure that you do not leave the Access Control List (ACL) unconfigured as this exposes the IPC interface to all local users and allows them to manipulate the authorization state of USB devices and modify the USBGuard policy.

15.2. INSTALLING USBGUARD

Use this procedure to install and initiate the **USBGuard** framework.

Procedure

1. Install the **usbguard** package:

   ```
   # yum install usbguard
   ```

2. Create an initial rule set:

   ```
   # usbguard generate-policy > /etc/usbguard/rules.conf
   ```

3. Start the **usbguard** daemon and ensure that it starts automatically on boot:

```
# systemctl enable --now usbguard
```

Verification

1. Verify that the **usbguard** service is running:

```
# systemctl status usbguard
● usbguard.service - USBGuard daemon
  Loaded: loaded (/usr/lib/systemd/system/usbguard.service; enabled; vendor preset:
disabled)
  Active: active (running) since Thu 2019-11-07 09:44:07 CET; 3min 16s ago
    Docs: man:usbguard-daemon(8)
 Main PID: 6122 (usbguard-daemon)
   Tasks: 3 (limit: 11493)
  Memory: 1.2M
  CGroup: /system.slice/usbguard.service
          └─6122 /usr/sbin/usbguard-daemon -f -s -c /etc/usbguard/usbguard-daemon.conf

Nov 07 09:44:06 localhost.localdomain systemd[1]: Starting USBGuard daemon...
Nov 07 09:44:07 localhost.localdomain systemd[1]: Started USBGuard daemon.
```

2. List USB devices recognized by **USBGuard**:

```
# usbguard list-devices
4: allow id 1d6b:0002 serial "0000:02:00.0" name "xHCI Host Controller" hash...
```

Additional resources

- **usbguard(1)** and **usbguard-daemon.conf(5)** man pages.

15.3. BLOCKING AND AUTHORIZING A USB DEVICE USING CLI

This procedure outlines how to authorize and block a USB device using the **usbguard** command.

Prerequisites

- The **usbguard** service is installed and running.

Procedure

1. List USB devices recognized by **USBGuard**:

```
# usbguard list-devices
1: allow id 1d6b:0002 serial "0000:00:06.7" name "EHCI Host Controller" hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" parent-hash
"4PHGcaDKWtPjKDwYpIRG722cB9SlGz9l9Iea93+Gt9c=" via-port "usb1" with-interface
09:00:00

...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
"CrXgiaWIf2bZAU+5WkzOE7y0rdSO82XMzubn7HDb95Q=" parent-hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" via-port "1-3" with-interface
08:06:50
```

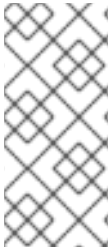2. Authorize the device *6* to interact with the system:

> # usbguard allow-device *6*

3. Deauthorize and remove the device *6*:

> # usbguard reject-device *6*

4. Deauthorize and retain the device *6*:

> # usbguard block-device *6*

> **NOTE**
>
> **USBGuard** uses the *block* and *reject* terms with the following meanings:
>
> - *block*: do not interact with this device for now.
>
> - *reject*: ignore this device as if it does not exist.

Additional resources

- **usbguard(1)** man page.

- Built-in help listed by using the **usbguard --help** command.

15.4. PERMANENTLY BLOCKING AND AUTHORIZING A USB DEVICE

You can permanently block and authorize a USB device using the **-p** option. This adds a device-specific rule to the current policy.

Prerequisites

- The **usbguard** service is installed and running.

Procedure

1. Configure SELinux to allow the **usbguard** daemon to write rules.

   a. Display the **semanage** Booleans relevant to **usbguard**.

   > # semanage boolean -l | grep usbguard
   > usbguard_daemon_write_conf   (off , off) Allow usbguard to daemon write conf
   > usbguard_daemon_write_rules   (on   ,   on) Allow usbguard to daemon write rules

   b. Optional: If the **usbguard_daemon_write_rules** Boolean is turned off, turn it on.

   > # semanage boolean -m --on usbguard_daemon_write_rules

2. List USB devices recognized by **USBGuard**:

   > # usbguard list-devices
   > 1: allow id 1d6b:0002 serial "0000:00:06.7" name "EHCI Host Controller" hash

"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" parent-hash
"4PHGcaDKWtPjKDwYpIRG722cB9SlGz9l9Iea93+Gt9c=" via-port "usb1" with-interface
09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
"CrXgiaWIf2bZAU+5WkzOE7y0rdSO82XMzubn7HDb95Q=" parent-hash
"JDOb0BiktYs2ct3mSQKopnOOV2h9MGYADwhT+oUtF2s=" via-port "1-3" with-interface
08:06:50

3. Permanently authorize the device *6* to interact with the system:

```
# usbguard allow-device 6 -p
```

4. Permanently deauthorize and remove the device *6*:

```
# usbguard reject-device 6 -p
```

5. Permanently deauthorize and retain the device *6*:

```
# usbguard block-device 6 -p
```

> **NOTE**
>
> **USBGuard** uses the terms *block* and *reject* with the following meanings:
>
> - *block*: do not interact with this device for now.
>
> - *reject*: ignore this device as if it does not exist.

Verification

1. Check that **USBGuard** rules include the changes you made.

```
# usbguard list-rules
```

Additional resources

- **usbguard(1)** man page.

- Built-in help listed by using the **usbguard --help** command.

15.5. CREATING A CUSTOM POLICY FOR USB DEVICES

The following procedure contains steps for creating a rule set for USB devices that reflects the requirements of your scenario.

Prerequisites

- The **usbguard** service is installed and running.

- The **/etc/usbguard/rules.conf** file contains an initial rule set generated by the **usbguard generate-policy** command.

Procedure

1. Create a policy which authorizes the currently connected USB devices, and store the generated rules to the **rules.conf** file:

   > # usbguard generate-policy --no-hashes > *./rules.conf*

   The **--no-hashes** option does not generate hash attributes for devices. Avoid hash attributes in your configuration settings because they might not be persistent.

2. Edit the **rules.conf** file with a text editor of your choice, for example:

   > # vi *./rules.conf*

3. Add, remove, or edit the rules as required. For example, the following rule allows only devices with a single mass storage interface to interact with the system:

   > allow with-interface equals { 08:*:* }

   See the **usbguard-rules.conf(5)** man page for a detailed rule-language description and more examples.

4. Install the updated policy:

   > # install -m 0600 -o root -g root *rules.conf* /etc/usbguard/rules.conf

5. Restart the **usbguard** daemon to apply your changes:

   > # systemctl restart usbguard

Verification

1. Check that your custom rules are in the active policy, for example:

   > # usbguard list-rules
   > ...
   > 4: allow with-interface 08:*:*
   > ...

Additional resources

- **usbguard-rules.conf(5)** man page.

15.6. CREATING A STRUCTURED CUSTOM POLICY FOR USB DEVICES

You can organize your custom **USBGuard** policy in several **.conf** files within the **/etc/usbguard/rules.d/** directory. The **usbguard-daemon** then combines the main **rules.conf** file with the **.conf** files within the directory in alphabetical order.

Prerequisites

- The **usbguard** service is installed and running.

Procedure

1. Create a policy which authorizes the currently connected USB devices, and store the generated rules to a new **.conf** file, for example, *policy.conf*.

   > # usbguard generate-policy --no-hashes > ./*policy.conf*

   The **--no-hashes** option does not generate hash attributes for devices. Avoid hash attributes in your configuration settings because they might not be persistent.

2. Display the *policy.conf* file with a text editor of your choice, for example:

   > # vi ./*policy.conf*
   > ...
   > allow id 04f2:0833 serial "" name "USB Keyboard" via-port "7-2" with-interface { 03:01:01 03:00:00 } with-connect-type "unknown"
   > ...

3. Move selected lines into a separate **.conf** file.

   > **NOTE**
   >
   > The two digits at the beginning of the file name specify the order in which the daemon reads the configuration files.

   For example, copy the rules for your keyboards into a new **.conf** file.

   > # grep "*USB Keyboard*" ./*policy.conf* > ./*10keyboards.conf*

4. Install the new policy to the **/etc/usbguard/rules.d/** directory.

   > # install -m 0600 -o root -g root *10keyboards.conf* /etc/usbguard/rules.d/*10keyboards.conf*

5. Move the rest of the lines to a main **rules.conf** file.

   > # grep -v "*USB Keyboard*" ./policy.conf > ./rules.conf

6. Install the remaining rules.

   > # install -m 0600 -o root -g root *rules.conf* /etc/usbguard/rules.conf

7. Restart the **usbguard** daemon to apply your changes.

   > # systemctl restart usbguard

Verification

1. Display all active **USBGuard** rules.

   > # usbguard list-rules
   > ...
   > 15: allow id 04f2:0833 serial "" name "USB Keyboard" hash

"kxM/iddRe/WSCocgiuQlVs6Dn0VEza7KiHoDeTz0fyg=" parent-hash
"2i6ZBJfTI5BakXF7Gba84/Cp1gslnNc1DM6vWQpie3s=" via-port "7-2" with-interface {
03:01:01 03:00:00 } with-connect-type "unknown"
...

2. Display the contents of the **rules.conf** file and all the **.conf** files in the **/etc/usbguard/rules.d/** directory.

   # cat /etc/usbguard/rules.conf /etc/usbguard/rules.d/*.conf

3. Verify that the active rules contain all the rules from the files and are in the correct order.

Additional resources

- **usbguard-rules.conf(5)** man page.

15.7. AUTHORIZING USERS AND GROUPS TO USE THE USBGUARD IPC INTERFACE

Use this procedure to authorize a specific user or a group to use the USBGuard public IPC interface. By default, only the root user can use this interface.

Prerequisites

- The **usbguard** service is installed and running.

- The **/etc/usbguard/rules.conf** file contains an initial rule set generated by the **usbguard generate-policy** command.

Procedure

1. Edit the **/etc/usbguard/usbguard-daemon.conf** file with a text editor of your choice:

   # vi /etc/usbguard/usbguard-daemon.conf

2. For example, add a line with a rule that allows all users in the **wheel** group to use the IPC interface, and save the file:

   IPCAllowGroups=wheel

3. You can add users or groups also with the **usbguard** command. For example, the following command enables the *joesec* user to have full access to the **Devices** and **Exceptions** sections. Furthermore, *joesec* can list the current policy and listen to policy signals.

   # usbguard add-user *joesec* --devices ALL --policy list,listen --exceptions ALL

   To remove the granted permissions for the *joesec* user, use the **usbguard remove-user joesec** command.

4. Restart the **usbguard** daemon to apply your changes:

   # systemctl restart usbguard

Additional resources

- **usbguard(1)** and **usbguard-rules.conf(5)** man pages.

15.8. LOGGING USBGUARD AUTHORIZATION EVENTS TO THE LINUX AUDIT LOG

Use the following steps to integrate logging of USBguard authorization events to the standard Linux Audit log. By default, the **usbguard** daemon logs events to the **/var/log/usbguard/usbguard-audit.log** file.

Prerequisites

- The **usbguard** service is installed and running.

- The **auditd** service is running.

Procedure

1. Edit the **usbguard-daemon.conf** file with a text editor of your choice:

   ```
   # vi /etc/usbguard/usbguard-daemon.conf
   ```

2. Change the **AuditBackend** option from **FileAudit** to **LinuxAudit**:

   ```
   AuditBackend=LinuxAudit
   ```

3. Restart the **usbguard** daemon to apply the configuration change:

   ```
   # systemctl restart usbguard
   ```

Verification

1. Query the **audit** daemon log for a USB authorization event, for example:

   ```
   # ausearch -ts recent -m USER_DEVICE
   ```

Additional resources

- **usbguard-daemon.conf(5)** man page.

15.9. ADDITIONAL RESOURCES

- **usbguard(1)**, **usbguard-rules.conf(5)**, **usbguard-daemon(8)**, and **usbguard-daemon.conf(5)** man pages.

- USBGuard Homepage.