



Red Hat Enterprise Linux 9

Security hardening

Enhancing security of Red Hat Enterprise Linux 9 systems

Red Hat Enterprise Linux 9 Security hardening

Enhancing security of Red Hat Enterprise Linux 9 systems

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Learn the processes and practices for securing Red Hat Enterprise Linux servers and workstations against local and remote intrusion, exploitation, and malicious activity. By using these approaches and tools, you can create a more secure computing environment for the data center, workplace, and home.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	6
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	7
CHAPTER 1. SECURING RHEL DURING INSTALLATION	8
1.1. BIOS AND UEFI SECURITY	8
1.1.1. BIOS passwords	8
1.1.2. Non-BIOS-based systems security	8
1.2. DISK PARTITIONING	8
1.3. RESTRICTING NETWORK CONNECTIVITY DURING THE INSTALLATION PROCESS	9
1.4. INSTALLING THE MINIMUM AMOUNT OF PACKAGES REQUIRED	9
1.5. POST-INSTALLATION PROCEDURES	9
CHAPTER 2. INSTALLING THE SYSTEM IN FIPS MODE	11
2.1. FEDERAL INFORMATION PROCESSING STANDARD (FIPS)	11
2.2. INSTALLING THE SYSTEM WITH FIPS MODE ENABLED	11
2.3. ADDITIONAL RESOURCES	12
CHAPTER 3. USING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES	13
3.1. SYSTEM-WIDE CRYPTOGRAPHIC POLICIES	13
Tool for managing crypto policies	14
Strong crypto defaults by removing insecure cipher suites and protocols	14
Algorithms disabled in all policy levels	14
Algorithms enabled in the crypto-policies levels	15
3.2. SWITCHING THE SYSTEM-WIDE CRYPTOGRAPHIC POLICY TO MODE COMPATIBLE WITH EARLIER RELEASES	16
3.3. SETTING UP SYSTEM-WIDE CRYPTOGRAPHIC POLICIES IN THE WEB CONSOLE	16
3.4. SWITCHING THE SYSTEM TO FIPS MODE	17
3.5. ENABLING FIPS MODE IN A CONTAINER	18
3.6. LIST OF RHEL APPLICATIONS USING CRYPTOGRAPHY THAT IS NOT COMPLIANT WITH FIPS 140-3	19
3.7. EXCLUDING AN APPLICATION FROM FOLLOWING SYSTEM-WIDE CRYPTO POLICIES	20
3.7.1. Examples of opting out of system-wide crypto policies	21
3.8. CUSTOMIZING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES WITH SUBPOLICIES	21
3.9. RE-ENABLING SHA-1	23
3.10. CREATING AND SETTING A CUSTOM SYSTEM-WIDE CRYPTOGRAPHIC POLICY	24
CHAPTER 4. SETTING A CUSTOM CRYPTOGRAPHIC POLICY ACROSS SYSTEMS	26
4.1. CRYPTO_POLICIES SYSTEM ROLE VARIABLES AND FACTS	26
4.2. SETTING A CUSTOM CRYPTOGRAPHIC POLICY USING THE CRYPTO_POLICIES SYSTEM ROLE	26
4.3. ADDITIONAL RESOURCES	28
CHAPTER 5. CONFIGURING APPLICATIONS TO USE CRYPTOGRAPHIC HARDWARE THROUGH PKCS #11 ..	29
5.1. CRYPTOGRAPHIC HARDWARE SUPPORT THROUGH PKCS #11	29
5.2. USING SSH KEYS STORED ON A SMART CARD	29
5.3. CONFIGURING APPLICATIONS TO AUTHENTICATE USING CERTIFICATES FROM SMART CARDS	31
5.4. USING HSMS PROTECTING PRIVATE KEYS IN APACHE	31
5.5. USING HSMS PROTECTING PRIVATE KEYS IN NGINX	32
5.6. ADDITIONAL RESOURCES	32
CHAPTER 6. CONTROLLING ACCESS TO SMART CARDS USING POLKIT	33
6.1. SMART-CARD ACCESS CONTROL THROUGH POLKIT	33
6.2. TROUBLESHOOTING PROBLEMS RELATED TO PC/SC AND POLKIT	33

6.3. DISPLAYING MORE DETAILED INFORMATION ABOUT POLKIT AUTHORIZATION TO PC/SC	35
6.4. ADDITIONAL RESOURCES	36
CHAPTER 7. SCANNING THE SYSTEM FOR CONFIGURATION COMPLIANCE AND VULNERABILITIES	37
7.1. CONFIGURATION COMPLIANCE TOOLS IN RHEL	37
7.2. VULNERABILITY SCANNING	38
7.2.1. Red Hat Security Advisories OVAL feed	38
7.2.2. Scanning the system for vulnerabilities	39
7.2.3. Scanning remote systems for vulnerabilities	39
7.3. CONFIGURATION COMPLIANCE SCANNING	40
7.3.1. Configuration compliance in RHEL	40
7.3.2. Possible results of an OpenSCAP scan	41
7.3.3. Viewing profiles for configuration compliance	42
7.3.4. Assessing configuration compliance with a specific baseline	43
7.4. REMEDIATING THE SYSTEM TO ALIGN WITH A SPECIFIC BASELINE	43
7.5. REMEDIATING THE SYSTEM TO ALIGN WITH A SPECIFIC BASELINE USING AN SSG ANSIBLE PLAYBOOK	44
7.6. CREATING A REMEDIATION ANSIBLE PLAYBOOK TO ALIGN THE SYSTEM WITH A SPECIFIC BASELINE	46
7.7. CREATING A REMEDIATION BASH SCRIPT FOR A LATER APPLICATION	46
7.8. SCANNING THE SYSTEM WITH A CUSTOMIZED PROFILE USING SCAP WORKBENCH	47
7.8.1. Using SCAP Workbench to scan and remediate the system	47
7.8.2. Customizing a security profile with SCAP Workbench	49
7.8.3. Additional resources	51
7.9. DEPLOYING SYSTEMS THAT ARE COMPLIANT WITH A SECURITY PROFILE IMMEDIATELY AFTER AN INSTALLATION	51
7.9.1. Profiles not compatible with Server with GUI	51
7.9.2. Deploying baseline-compliant RHEL systems using the graphical installation	52
7.9.3. Deploying baseline-compliant RHEL systems using Kickstart	53
7.10. SCANNING CONTAINER AND CONTAINER IMAGES FOR VULNERABILITIES	54
7.11. ASSESSING SECURITY COMPLIANCE OF A CONTAINER OR A CONTAINER IMAGE WITH A SPECIFIC BASELINE	55
7.12. SCAP SECURITY GUIDE PROFILES SUPPORTED IN RHEL 9	56
7.13. ADDITIONAL RESOURCES	59
CHAPTER 8. ENSURING SYSTEM INTEGRITY WITH KEYLIME	60
8.1. HOW KEYLIME WORKS	60
8.2. CONFIGURING KEYLIME VERIFIER AND REGISTRAR	61
8.3. DEPLOYING KEYLIME FOR MEASURED BOOT ATTESTATION	66
8.4. DEPLOYING KEYLIME FOR RUNTIME MONITORING	70
CHAPTER 9. CHECKING INTEGRITY WITH AIDE	76
9.1. INSTALLING AIDE	76
9.2. PERFORMING INTEGRITY CHECKS WITH AIDE	76
9.3. UPDATING AN AIDE DATABASE	77
9.4. FILE-INTEGRITY TOOLS: AIDE AND IMA	77
9.5. ADDITIONAL RESOURCES	78
CHAPTER 10. ENCRYPTING BLOCK DEVICES USING LUKS	79
10.1. LUKS DISK ENCRYPTION	79
10.2. LUKS VERSIONS IN RHEL	80
10.3. OPTIONS FOR DATA PROTECTION DURING LUKS2 RE-ENCRYPTION	81
10.4. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2	81
10.5. ENCRYPTING EXISTING DATA ON A BLOCK DEVICE USING LUKS2 WITH A DETACHED HEADER	83

10.6. ENCRYPTING A BLANK BLOCK DEVICE USING LUKS2	84
10.7. CREATING A LUKS ENCRYPTED VOLUME USING THE STORAGE RHEL SYSTEM ROLE	85
CHAPTER 11. CONFIGURING AUTOMATED UNLOCKING OF ENCRYPTED VOLUMES USING POLICY-BASED DECRYPTION	87
11.1. NETWORK-BOUND DISK ENCRYPTION	87
11.2. INSTALLING AN ENCRYPTION CLIENT - CLEVIS	88
11.3. DEPLOYING A TANG SERVER WITH SELINUX IN ENFORCING MODE	89
11.4. ROTATING TANG SERVER KEYS AND UPDATING BINDINGS ON CLIENTS	90
11.5. CONFIGURING AUTOMATED UNLOCKING USING A TANG KEY IN THE WEB CONSOLE	92
11.6. BASIC NBDE AND TPM2 ENCRYPTION-CLIENT OPERATIONS	95
11.7. CONFIGURING MANUAL ENROLLMENT OF LUKS-ENCRYPTED VOLUMES	97
11.8. CONFIGURING MANUAL ENROLLMENT OF LUKS-ENCRYPTED VOLUMES USING A TPM 2.0 POLICY	99
11.9. REMOVING A CLEVIS PIN FROM A LUKS-ENCRYPTED VOLUME MANUALLY	101
11.10. CONFIGURING AUTOMATED ENROLLMENT OF LUKS-ENCRYPTED VOLUMES USING KICKSTART	102
11.11. CONFIGURING AUTOMATED UNLOCKING OF A LUKS-ENCRYPTED REMOVABLE STORAGE DEVICE	103
11.12. DEPLOYING HIGH-AVAILABILITY NBDE SYSTEMS	104
11.12.1. High-available NBDE using Shamir's Secret Sharing	104
11.12.1.1. Example 1: Redundancy with two Tang servers	105
11.12.1.2. Example 2: Shared secret on a Tang server and a TPM device	105
11.13. DEPLOYMENT OF VIRTUAL MACHINES IN A NBDE NETWORK	106
11.14. BUILDING AUTOMATICALLY-ENROLLABLE VM IMAGES FOR CLOUD ENVIRONMENTS USING NBDE	106
11.15. DEPLOYING TANG AS A CONTAINER	107
11.16. INTRODUCTION TO THE NBDE_CLIENT AND NBDE_SERVER SYSTEM ROLES (CLEVIS AND TANG)	108
11.17. USING THE NBDE_SERVER SYSTEM ROLE FOR SETTING UP MULTIPLE TANG SERVERS	109
11.18. USING THE NBDE_CLIENT SYSTEM ROLE FOR SETTING UP MULTIPLE CLEVIS CLIENTS	110
CHAPTER 12. AUDITING THE SYSTEM	112
12.1. LINUX AUDIT	112
12.2. AUDIT SYSTEM ARCHITECTURE	113
12.3. CONFIGURING AUDITD FOR A SECURE ENVIRONMENT	114
12.4. STARTING AND CONTROLLING AUDITD	115
12.5. UNDERSTANDING AUDIT LOG FILES	116
12.6. USING AUDITCTL FOR DEFINING AND EXECUTING AUDIT RULES	120
12.7. DEFINING PERSISTENT AUDIT RULES	121
12.8. USING PRE-CONFIGURED RULES FILES	121
12.9. USING AUGENRULES TO DEFINE PERSISTENT RULES	122
12.10. DISABLING AUGENRULES	122
12.11. SETTING UP AUDIT TO MONITOR SOFTWARE UPDATES	123
12.12. MONITORING USER LOGIN TIMES WITH AUDIT	125
12.13. ADDITIONAL RESOURCES	126
CHAPTER 13. BLOCKING AND ALLOWING APPLICATIONS USING FAPOLICYD	127
13.1. INTRODUCTION TO FAPOLICYD	127
13.2. DEPLOYING FAPOLICYD	128
13.3. MARKING FILES AS TRUSTED USING AN ADDITIONAL SOURCE OF TRUST	129
13.4. ADDING CUSTOM ALLOW AND DENY RULES FOR FAPOLICYD	130
13.5. ENABLING FAPOLICYD INTEGRITY CHECKS	133
13.6. TROUBLESHOOTING PROBLEMS RELATED TO FAPOLICYD	134
13.7. ADDITIONAL RESOURCES	136

CHAPTER 14. PROTECTING SYSTEMS AGAINST INTRUSIVE USB DEVICES	137
14.1. USBGUARD	137
14.2. INSTALLING USBGUARD	137
14.3. BLOCKING AND AUTHORIZING A USB DEVICE USING CLI	138
14.4. PERMANENTLY BLOCKING AND AUTHORIZING A USB DEVICE	139
14.5. CREATING A CUSTOM POLICY FOR USB DEVICES	140
14.6. CREATING A STRUCTURED CUSTOM POLICY FOR USB DEVICES	141
14.7. AUTHORIZING USERS AND GROUPS TO USE THE USBGUARD IPC INTERFACE	143
14.8. LOGGING USBGUARD AUTHORIZATION EVENTS TO THE LINUX AUDIT LOG	144
14.9. ADDITIONAL RESOURCES	144
CHAPTER 15. CONFIGURING A REMOTE LOGGING SOLUTION	145
15.1. THE RSYSLOG LOGGING SERVICE	145
15.2. INSTALLING RSYSLOG DOCUMENTATION	145
15.3. CONFIGURING A SERVER FOR REMOTE LOGGING OVER TCP	146
15.4. CONFIGURING REMOTE LOGGING TO A SERVER OVER TCP	148
15.5. CONFIGURING TLS-ENCRYPTED REMOTE LOGGING	149
15.6. CONFIGURING A SERVER FOR RECEIVING REMOTE LOGGING INFORMATION OVER UDP	152
15.7. CONFIGURING REMOTE LOGGING TO A SERVER OVER UDP	154
15.8. LOAD BALANCING HELPER IN RSYSLOG	156
15.9. CONFIGURING RELIABLE REMOTE LOGGING	156
15.10. SUPPORTED RSYSLOG MODULES	158
15.11. CONFIGURING THE NETCONSOLE SERVICE TO LOG KERNEL MESSAGES TO A REMOTE HOST	158
15.12. ADDITIONAL RESOURCES	159
CHAPTER 16. USING THE LOGGING SYSTEM ROLE	160
16.1. THE LOGGING SYSTEM ROLE	160
16.2. LOGGING SYSTEM ROLE PARAMETERS	160
16.3. APPLYING A LOCAL LOGGING SYSTEM ROLE	161
16.4. FILTERING LOGS IN A LOCAL LOGGING SYSTEM ROLE	163
16.5. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE	165
16.6. USING THE LOGGING SYSTEM ROLE WITH TLS	168
16.6.1. Configuring client logging with TLS	168
16.6.2. Configuring server logging with TLS	170
16.7. USING THE LOGGING SYSTEM ROLES WITH RELP	171
16.7.1. Configuring client logging with RELP	172
16.7.2. Configuring server logging with RELP	174
16.8. ADDITIONAL RESOURCES	175

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 feedback on our documentation. Let us know how we can improve it.

Submitting comments on specific passages

1. View the documentation in the **Multi-page HTML** format and ensure that you see the **Feedback** button in the upper right corner after the page fully loads.
2. Use your cursor to highlight the part of the text that you want to comment on.
3. Click the **Add Feedback** button that appears near the highlighted text.
4. Add your feedback and click **Submit**.

Submitting feedback through Bugzilla (account required)

1. Log in to the [Bugzilla](#) website.
2. Select the correct version from the **Version** menu.
3. Enter a descriptive title in the **Summary** field.
4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
5. Click **Submit Bug**.

CHAPTER 1. 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.

1.1. 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.

1.1.1. 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.

1.1.2. 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.

1.2. 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 9 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 9 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.

1.3. RESTRICTING NETWORK CONNECTIVITY DURING THE INSTALLATION PROCESS

When installing Red Hat Enterprise Linux 9, 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 9 from a network.

1.4. 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.

1.5. POST-INSTALLATION PROCEDURES

The following steps are the security-related procedures that should be performed immediately after installation of Red Hat Enterprise Linux 9.

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

```
# dnf 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] Because 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 2. INSTALLING THE SYSTEM IN FIPS MODE

To enable the cryptographic module self-checks mandated by the Federal Information Processing Standard (FIPS) 140-3, you have to operate RHEL 9 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.



NOTE

The cryptographic modules of RHEL 9 are not yet certified for the FIPS 140-3 requirements.

2.1. FEDERAL INFORMATION PROCESSING STANDARD (FIPS)

The Federal Information Processing Standard (FIPS) Publication 140-3 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 140-3 standard ensures that cryptographic tools implement their algorithms correctly. One of the mechanisms for that is runtime self-checks. See the full FIPS 140-3 standard at [FIPS PUB 140-3](#) for further details and other specifications of the FIPS standard.

To learn about compliance requirements, see the [Red Hat Government Standards](#) page.

2.2. INSTALLING THE SYSTEM WITH FIPS MODE ENABLED

To enable the cryptographic module self-checks mandated by the Federal Information Processing Standard (FIPS) Publication 140-3, enable FIPS mode during the system installation.



IMPORTANT

Red Hat recommends installing RHEL 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

2.3. ADDITIONAL RESOURCES

- [Switching the system to FIPS mode](#)
- [Enabling FIPS mode in a container](#)

CHAPTER 3. USING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES

The system-wide cryptographic 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.

3.1. SYSTEM-WIDE CRYPTOGRAPHIC POLICIES

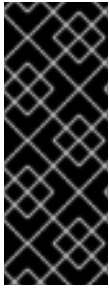
When a system-wide policy is set up, applications in RHEL follow it and refuse to use algorithms and protocols that do not meet the policy, unless you explicitly request the application to do so. That is, the policy applies to the default behavior of applications when running with the system-provided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

RHEL 9 contains the following predefined policies:

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 6 and earlier; it is less secure due to an increased attack surface. SHA-1 is allowed to be used as TLS hash, signature, and algorithm. CBC-mode ciphers are allowed to be used with SSH. Applications using GnuTLS allow certificates signed with SHA-1. 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.
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 DNSSec or as an HMAC. SHA2-224 and SHA3-224 hashes are disabled. 128-bit ciphers are disabled. CBC-mode ciphers are disabled except in Kerberos. 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 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 Red Hat Enterprise Linux.

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, you should opt-out from cryptographic-policies for components that interact with that system or re-enable specific algorithms using custom policies.



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 9. 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)

Algorithms disabled in all policy levels

The following algorithms are disabled in **LEGACY**, **DEFAULT**, **FUTURE** and **FIPS** cryptographic policies included in RHEL 9. They can be enabled only by applying a custom cryptographic policy or by an explicit configuration of individual applications, but the resulting configuration would not be considered supported.

- TLS older than version 1.2 (since RHEL 9, was < 1.0 in RHEL 8)
- DTLS older than version 1.2 (since RHEL 9, was < 1.0 in RHEL 8)
- DH with parameters < 2048 bits (since RHEL 9, was < 1024 bits in RHEL 8)

- RSA with key size < 2048 bits (since RHEL 9, was < 1024 bits in RHEL 8)
- DSA (since RHEL 9, was < 1024 bits in RHEL 8)
- 3DES (since RHEL 9)
- RC4 (since RHEL 9)
- FFDHE-1024 (since RHEL 9)
- DHE-DSS (since RHEL 9)
- Camellia (since RHEL 9)
- ARIA
- IKEv1 (since RHEL 8)

Algorithms enabled in the crypto-policies levels

The following table shows the comparison of all four crypto-policies levels with regard to select algorithms.

	LEGACY	DEFAULT	FIPS	FUTURE
IKEv1	no	no	no	no
3DES	no	no	no	no
RC4	no	no	no	no
DH	min. 2048-bit	min. 2048-bit	min. 2048-bit	min. 3072-bit
RSA	min. 2048-bit	min. 2048-bit	min. 2048-bit	min. 3072-bit
DSA	no	no	no	no
TLS v1.1 and older	no	no	no	no
TLS v1.2 and newer	yes	yes	yes	yes
SHA-1 in digital signatures and certificates	yes	no	no	no
CBC mode ciphers	yes	no ^[a]	no ^[b]	no ^[c]
Symmetric ciphers with keys < 256 bits	yes	yes	yes	no

LEGACY	DEFAULT	FIPS	FUTURE
[a] CBC ciphers are disabled for SSH			
[b] CBC ciphers are disabled for all protocols except Kerberos			
[c] CBC ciphers are disabled for all protocols except Kerberos			

Additional resources

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

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

The default system-wide cryptographic policy in Red Hat Enterprise Linux 9 does not allow communication using older, insecure protocols. For environments that require to be compatible with Red Hat Enterprise Linux 6 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.
- For defining custom cryptographic policies, see the **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.

3.3. SETTING UP SYSTEM-WIDE CRYPTOGRAPHIC POLICIES IN THE WEB CONSOLE

You can choose from predefined system-wide cryptographic policy levels and switch between them

directly in the Red Hat Enterprise Linux web console interface. If you set a custom policy on your system, the web console displays the policy in the **Overview** page as well as the **Change crypto policy** dialog window.

Prerequisites

- The RHEL 9 web console has been installed. For details, see [Installing and enabling the web console](#).
- You have administrator privileges.

Procedure

1. Log in to the RHEL web console. For more information, see [Logging in to the web console](#).
2. In the **Configuration** card of the **Overview** page, click your current policy value next to **Crypto policy**.
3. In the **Change crypto policy** dialog window, click on the policy level that you want to start using.
4. Click the **Apply and reboot** button.

Verification

- Log back in and check that the **Crypto policy** value corresponds to the one you selected.

3.4. 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 140-3. 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 9 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.



NOTE

The cryptographic modules of RHEL 9 are not yet certified for the FIPS 140-3 requirements.

Procedure

1. To switch the system to FIPS mode:

```
# fips-mode-setup --enable
Kernel initramdisks are being regenerated. This might take some time.
Setting system policy to FIPS
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
```

```
to fully take place.  
FIPS mode will be enabled.  
Please reboot the system for the setting to take effect.
```

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
- [Installing the system in FIPS mode](#)
- [Security Requirements for Cryptographic Modules](#) on the National Institute of Standards and Technology (NIST) web site.

3.5. ENABLING FIPS MODE IN A CONTAINER

On systems with FIPS mode enabled, the **podman** utility automatically configures containers to FIPS mode. On systems not in FIPS mode, you can switch a container to FIPS mode by using a single command later.



NOTE

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.



NOTE

The cryptographic modules of RHEL 9 are not yet certified for the FIPS 140-3 requirements.

Prerequisites

- The host system must be in FIPS mode.

Procedure

- Use the following command in a container that you want to switch to FIPS mode:

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

Additional resources

- [Switching the system to FIPS mode](#) .
- [Installing the system in FIPS mode](#)

3.6. LIST OF RHEL APPLICATIONS USING CRYPTOGRAPHY THAT IS NOT COMPLIANT WITH FIPS 140-3

Red Hat recommends utilizing libraries from the core crypto components set, as they are guaranteed to pass all relevant crypto certifications, such as FIPS 140-3, and also follow the RHEL system-wide crypto policies.

See the [RHEL core crypto components](#) article for an overview of the core cryptographic 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 cryptographic certifications apply to them.

Table 3.1. List of RHEL 8 applications using cryptography that is not compliant with FIPS 140-3

Application	Details
Bacula	Implements the CRAM-MD5 authentication protocol.
Cyrus SASL	Uses the SCRAM-SHA-1 authentication method.
Dovecot	Uses SCRAM-SHA-1.
Emacs	Uses SCRAM-SHA-1.
FreeRADIUS	Uses MD5 and SHA-1 for authentication protocols.
Ghostscript	Custom cryptography implementation (MD5, RC4, SHA-2, AES) to encrypt and decrypt documents.
GRUB2	Supports legacy firmware protocols requiring SHA-1 and includes the libgcrypt library.
ipxe	Implements TLS stack.
Kerberos	Preserves support for SHA-1 (interoperability with Windows).
lasso	The lasso_wsse_username_token_derive_key() key derivation function (KDF) uses SHA-1.
MariaDB, MariaDB Connector	The mysql_native_password authentication plugin uses SHA-1.
MySQL	mysql_native_password uses SHA-1.
OpenIPMI	The RAKP-HMAC-MD5 authentication method is not approved for FIPS usage and does not work in FIPS mode.

Application	Details
Ovmf (UEFI firmware), Edk2, shim	Full crypto stack (an embedded copy of the OpenSSL library).
perl-CPAN	Digest MD5 authentication.
perl-Digest-HMAC, perl-Digest-SHA	Uses HMAC, HMAC-SHA1, HMAC-MD5, SHA-1, SHA-224, and so on.
perl-Mail-DKIM	The Signer class uses the RSA-SHA1 algorithm by default.
PKCS #12 file processing (OpenSSL, GnuTLS, NSS, Firefox, Java)	All uses of PKCS #12 are not FIPS-compliant, because the Key Derivation Function (KDF) used for calculating the whole-file HMAC is not FIPS-approved. As such, PKCS #12 files are considered to be plain text for the purposes of FIPS compliance. For key-transport purposes, wrap PKCS #12 (.p12) files using a FIPS-approved encryption scheme.
Poppler	Can save PDFs with signatures, passwords, and encryption based on non-allowed algorithms if they are present in the original PDF (for example MD5, RC4, and SHA-1).
PostgreSQL	KDF uses SHA-1.
QAT Engine	Mixed hardware and software implementation of cryptographic primitives (RSA, EC, DH, AES, ...)
Ruby	Provides insecure MD5 and SHA-1 library functions.
Samba	Preserves support for RC4 and DES (interoperability with Windows).
Syslinux	BIOS passwords use SHA-1.
Unbound	DNS specification requires that DNSSEC resolvers use a SHA-1-based algorithm in DNSKEY records for validation.
Valgrind	AES, SHA hashes. ^[a]
[a] Re-implements in software hardware-offload operations, such as AES-NI or SHA-1 and SHA-2 on ARM.	

3.7. 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.

3.7.1. 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 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

3.8. CUSTOMIZING SYSTEM-WIDE CRYPTOGRAPHIC POLICIES WITH SUBPOLICIES

Use this procedure to adjust the set of enabled cryptographic algorithms or protocols.

You can either apply custom subpolicies on top of an existing system-wide cryptographic policy or define such a policy from scratch.

The concept of scoped policies allows enabling different sets of algorithms for different back ends. You can limit each configuration directive to specific protocols, libraries, or services.

Furthermore, directives can use asterisks for specifying multiple values using wildcards.

The **/etc/crypto-policies/state/CURRENT.pol** file lists all settings in the currently applied system-wide cryptographic policy after wildcard expansion. To make your cryptographic policy more strict, consider using values listed in the **/usr/share/crypto-policies/policies/FUTURE.pol** file.

You can find example subpolicies in the **/usr/share/crypto-policies/policies/modules/** directory. The subpolicy files in this directory contain also descriptions in lines that are commented out.

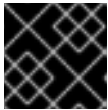
Procedure

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

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

2. Create subpolicies for your adjustments, for example:

```
# touch MYCRYPTO-1.pmod
# touch SCOPES-AND-WILDCARDS.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 MYCRYPTO-1.pmod
```

```
min_rsa_size = 3072
hash = SHA2-384 SHA2-512 SHA3-384 SHA3-512
```

```
# vi SCOPES-AND-WILDCARDS.pmod
```

```
# Disable the AES-128 cipher, all modes
cipher = -AES-128-*
```

```
# Disable CHACHA20-POLY1305 for the TLS protocol (OpenSSL, GnuTLS, NSS, and
OpenJDK)
cipher@TLS = -CHACHA20-POLY1305
```

```
# Allow using the FFDHE-1024 group with the SSH protocol (libssh and OpenSSH)
group@SSH = FFDHE-1024+
```

```
# Disable all CBC mode ciphers for the SSH protocol (libssh and OpenSSH)
```

```
cipher@SSH = -* -CBC
```

```
# Allow the AES-256-CBC cipher in applications using libssh
cipher@libssh = AES-256-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:MYCRYPTO-1:SCOPES-AND-WILDCARDS
```

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

```
# reboot
```

Verification

- Check that the `/etc/crypto-policies/state/CURRENT.pol` file contains your changes, for example:

```
$ cat /etc/crypto-policies/state/CURRENT.pol | grep rsa_size
min_rsa_size = 3072
```

Additional resources

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

3.9. RE-ENABLING SHA-1

The use of the SHA-1 algorithm for creating and verifying signatures is restricted in the **DEFAULT** cryptographic policy. If your scenario requires the use of SHA-1 for verifying existing or third-party cryptographic signatures, you can enable it by applying the **SHA1** subpolicy, which RHEL 9 provides by default. Note that it weakens the security of the system.

Prerequisites

- The system uses the **DEFAULT** system-wide cryptographic policy.

Procedure

1. Apply the **SHA1** subpolicy to the **DEFAULT** cryptographic policy:

```
# update-crypto-policies --set DEFAULT:SHA1
Setting system policy to DEFAULT:SHA1
Note: System-wide crypto policies are applied on application start-up.
It is recommended to restart the system for the change of policies
to fully take place.
```

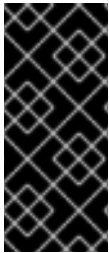
2. Restart the system:

```
# reboot
```

Verification

- Display the current cryptographic policy:

```
# update-crypto-policies --show  
DEFAULT:SHA1
```



IMPORTANT

Switching to the **LEGACY** cryptographic policy by using the **update-crypto-policies --set LEGACY** command also enables SHA-1 for signatures. However, the **LEGACY** cryptographic policy makes your system much more vulnerable by also enabling other weak cryptographic algorithms. Use this workaround only for scenarios that require the enablement of other legacy cryptographic algorithms than SHA-1 signatures.

Additional resources

- [SSH from RHEL 9 to RHEL 6 systems does not work](#) KCS article
- [Packages signed with SHA-1 cannot be installed or upgraded](#) KCS article

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

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

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](#) Red Hat blog article

CHAPTER 4. SETTING A CUSTOM CRYPTOGRAPHIC POLICY ACROSS SYSTEMS

As an administrator, you can use the **crypto_policies** RHEL System Role to quickly and consistently configure custom cryptographic policies across many different systems using the Ansible Core package.

4.1. 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 the System Role applies to the managed nodes. For details about the different crypto policies, 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 policies available on the system.

crypto_policies_available_subpolicies

Lists all available subpolicies available on the system.

Additional resources

- [Creating and setting a custom system-wide cryptographic policy](#).

4.2. 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 Core configures other systems.

On the control node:

- The **ansible-core** and **rhel-system-roles** packages are installed.

IMPORTANT

RHEL 8.0–8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the [How to download and install Red Hat Ansible Engine](#) Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#) Knowledgebase article.

- 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: rhel-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 cryptographic 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: rhel-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.

4.3. 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 5. 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 RHEL, 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.

5.1. 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.

RHEL 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 the system. 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

- [The PKCS #11 URI Scheme](#)
- [Controlling access to smart cards](#)

5.2. 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

5.3. CONFIGURING APPLICATIONS TO AUTHENTICATE USING CERTIFICATES FROM SMART CARDS

Authentication using smart cards in applications may increase security and simplify automation.

- 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.



NOTE

Because a PIN is a security measure that controls access to keys stored on a smart card and the configuration file contains the PIN in the plain-text form, consider additional protection to prevent an attacker from reading the PIN. For example, you can use the **pin-source** attribute and provide a **file:** URI for reading the PIN from a file. See [RFC 7512: PKCS #11 URI Scheme Query Attribute Semantics](#) for more information. Note that using a command path as a value of the **pin-source** attribute is not supported.

- 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.

5.4. 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 use 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 the **/usr/share/httpd/manual/mod/mod_ssl.html** file.

5.5. 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.

5.6. ADDITIONAL RESOURCES

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

CHAPTER 6. CONTROLLING ACCESS TO SMART CARDS USING POLKIT

To cover possible threats that cannot be prevented by mechanisms built into smart cards, such as PINs, PIN pads, and biometrics, and for more fine-grained control, RHEL uses the **polkit** framework for controlling access control to smart cards.

System administrators can configure **polkit** to fit specific scenarios, such as smart-card access for non-privileged or non-local users or services.

6.1. SMART-CARD ACCESS CONTROL THROUGH POLKIT

The Personal Computer/Smart Card (PC/SC) protocol specifies a standard for integrating smart cards and their readers into computing systems. In RHEL, the **pcsc-lite** package provides middleware to access smart cards that use the PC/SC API. A part of this package, the **pcscd** (PC/SC Smart Card) daemon, ensures that the system can access a smart card using the PC/SC protocol.

Because access-control mechanisms built into smart cards, such as PINs, PIN pads, and biometrics, do not cover all possible threats, RHEL uses the **polkit** framework for more robust access control. The **polkit** authorization manager can grant access to privileged operations. In addition to granting access to disks, you can use **polkit** also to specify policies for securing smart cards. For example, you can define which users can perform which operations with a smart card.

After installing the **pcsc-lite** package and starting the **pcscd** daemon, the system enforces policies defined in the **/usr/share/polkit-1/actions/** directory. The default system-wide policy is in the **/usr/share/polkit-1/actions/org.debian.pcsc-lite.policy** file. Polkit policy files use the XML format and the syntax is described in the **polkit(8)** man page.

The **polkitd** service monitors the **/etc/polkit-1/rules.d/** and **/usr/share/polkit-1/rules.d/** directories for any changes in rule files stored in these directories. The files contain authorization rules in JavaScript format. System administrators can add custom rule files in both directories, and **polkitd** reads them in lexical order based on their file name. If two files have the same names, then the file in **/etc/polkit-1/rules.d/** is read first.

Additional resources

- **polkit(8)**, **polkitd(8)**, and **pcscd(8)** man pages.

6.2. TROUBLESHOOTING PROBLEMS RELATED TO PC/SC AND POLKIT

Polkit policies that are automatically enforced after you install the **pcsc-lite** package and start the **pcscd** daemon may ask for authentication in the user's session even if the user does not directly interact with a smart card. In GNOME, you can see the following error message:

Authentication is required to access the PC/SC daemon

Note that the system can install the **pcsc-lite** package as a dependency when you install other packages related to smart cards such as **opensc**.

If your scenario does not require any interaction with smart cards and you want to prevent displaying authorization requests for the PC/SC daemon, you can remove the **pcsc-lite** package. Keeping the minimum of necessary packages is a good security practice anyway.

If you use smart cards, start troubleshooting by checking the rules in the system-provided policy file at **/usr/share/polkit-1/actions/org.debian.pcsc-lite.policy**. You can add your custom rule files to the policy in the **/etc/polkit-1/rules.d/** directory, for example, **03-allow-pcscd.rules**. Note that the rule files use the JavaScript syntax, the policy file is in the XML format.

To understand what authorization requests the system displays, check the Journal log, for example:

```
$ journalctl -b | grep pcsc
...
Process 3087 (user: 1001) is NOT authorized for action: access_pcsc
...
```

The previous log entry means that the user is not authorized to perform an action by the policy. You can solve this denial by adding a corresponding rule to **/etc/polkit-1/rules.d/**.

You can search also for log entries related to the **polkitd** unit, for example:

```
$ journalctl -u polkit
...
polkitd[NNN]: Error compiling script /etc/polkit-1/rules.d/00-debug-pcscd.rules
...
polkitd[NNN]: Operator of unix-session:c2 FAILED to authenticate to gain authorization for action
org.debian.pcsc-lite.access_pcsc for unix-process:4800:14441 [/usr/libexec/gsd-smartcard] (owned
by unix-user:group)
...
```

In the previous output, the first entry means that the rule file contains some syntax error. The second entry means that the user failed to gain the access to **pcscd**.

You can also list all applications that use the PC/SC protocol by a short script. Create an executable file, for example, **pcsc-apps.sh**, and insert the following code:

```
#!/bin/bash

cd /proc

for p in [0-9]*
do
  if grep libpcsc-lite.so.1.0.0 $p/maps &> /dev/null
  then
    echo -n "process: "
    cat $p/cmdline
    echo " ($p)"
  fi
done
```

Run the script as **root**:

```
# ./pcsc-apps.sh
process: /usr/libexec/gsd-smartcard (3048)
enable-sync --auto-ssl-client-auth --enable-crashpad (4828)
...
```

Additional resources

- **journalctl**, **polkit(8)**, **polkitd(8)**, and **pcscd(8)** man pages.

6.3. DISPLAYING MORE DETAILED INFORMATION ABOUT POLKIT AUTHORIZATION TO PC/SC

In the default configuration, the **polkit** authorization framework sends only limited information to the Journal log. You can extend **polkit** log entries related to the PC/SC protocol by adding new rules.

Prerequisites

- You have installed the **pcsc-lite** package on your system.
- The **pcscd** daemon is running.

Procedure

1. Create a new file in the **/etc/polkit-1/rules.d/** directory:

```
# touch /etc/polkit-1/rules.d/00-test.rules
```

2. Edit the file in an editor of your choice, for example:

```
# vi /etc/polkit-1/rules.d/00-test.rules
```

3. Insert the following lines:

```
polkit.addRule(function(action, subject) {
  if (action.id == "org.debian.pcsc-lite.access_pcsc" ||
      action.id == "org.debian.pcsc-lite.access_card") {
    polkit.log("action=" + action);
    polkit.log("subject=" + subject);
  }
});
```

Save the file, and exit the editor.

4. Restart the **pcscd** and **polkit** services:

```
# systemctl restart pcscd.service pcscd.socket polkit.service
```

Verification

1. Make an authorization request for **pcscd**. For example, open the Firefox web browser or use the **pkcs11-tool -L** command provided by the **opensc** package.
2. Display the extended log entries, for example:

```
# journalctl -u polkit --since "1 hour ago"
polkitd[1224]: <no filename>:4: action=[Action id='org.debian.pcsc-lite.access_pcsc']
polkitd[1224]: <no filename>:5: subject=[Subject pid=2020481 user='user'
groups=user,wheel,mock,wireshark seat=null session=null local=true active=true]
```

Additional resources

- **polkit(8)** and **polkitd(8)** man pages.

6.4. ADDITIONAL RESOURCES

- [Controlling access to smart cards](#) Red Hat Blog article.

CHAPTER 7. SCANNING THE SYSTEM FOR CONFIGURATION COMPLIANCE AND VULNERABILITIES

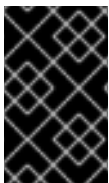
A compliance audit is a process of determining whether a given object follows all the rules specified in a compliance policy. The compliance policy is defined by security professionals who specify the required settings, often in the form of a checklist, that a computing environment should use.

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.

7.1. 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.



IMPORTANT

You can experience memory-consumption problems while using **OpenSCAP**, which can cause stopping the program prematurely and prevent generating any result files. See the [OpenSCAP memory-consumption problems](#) Knowledgebase article for details.

- **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](#) .

7.2. VULNERABILITY SCANNING

7.2.1. 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.



NOTE

The [Red Hat Insights for Red Hat Enterprise Linux compliance service](#) helps IT security and compliance administrators to assess, monitor, and report on the security policy compliance of Red Hat Enterprise Linux systems. You can also create and manage your SCAP security policies entirely within the compliance service UI.

Additional resources

- [Red Hat and OVAL compatibility](#)
- [Red Hat and CVE compatibility](#)
- [Notifications and Advisories](#) in the [Product Security Overview](#)
- [Security Data Metrics](#)

7.2.2. 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 **openscap-scanner** and **bzip2** packages are installed.

Procedure

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

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -  
-decompress > rhel-9.oval.xml
```

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

```
# oscap oval eval --report vulnerability.html rhel-9.oval.xml
```

Verification

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

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

Additional resources

- **oscap(8)** man page
- [Red Hat OVAL definitions](#)
- [OpenSCAP memory consumption problems](#)

7.2.3. 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 **openscap-utils** and **bzip2** packages are installed on the system you use for scanning.

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

Procedure

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

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -  
-decompress > rhel-9.oval.xml
```

2. 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-9.oval.xml
```

Additional resources

- **oscap-ssh(8)**
- [Red Hat OVAL definitions](#)
- [OpenSCAP memory consumption problems](#)

7.3. CONFIGURATION COMPLIANCE SCANNING

7.3.1. Configuration compliance in RHEL

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 align your systems with the Operating System Protection Profile (OSPP), and if you are a payment processor, you might have to align your systems 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.

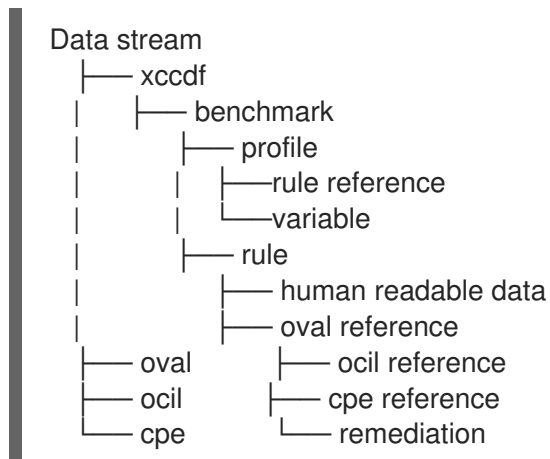


IMPORTANT

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 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



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](#).

7.3.2. 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 7.1. 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 .

Result	Explanation
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 .

7.3.3. 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** subcommand.

Prerequisites

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

Procedure

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

```
$ ls /usr/share/xml/scap/ssg/content/
ssg-rhel9-ds.xml
```

- Display detailed information about a selected data stream using the **oscap info** subcommand. 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-rhel9-ds.xml
Profiles:
...
Title: Australian Cyber Security Centre (ACSC) Essential Eight
Id: xccdf_org.ssgproject.content_profile_e8
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 9
Id: xccdf_org.ssgproject.content_profile_pci-dss
...
```

- 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-rhel9-ds.xml
...
Profile
Title: [RHEL9 DRAFT] Health Insurance Portability and Accountability Act (HIPAA)
Id: xccdf_org.ssgproject.content_profile_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. The Security Rule requires appropriate administrative,
```

physical and technical safeguards to ensure the confidentiality, integrity, and security of electronic protected health information. This profile configures Red Hat Enterprise Linux 9 to the HIPAA Security Rule identified for securing of electronic protected health information. Use of this profile in no way guarantees or makes claims against legal compliance against the HIPAA Security Rule(s).

Additional resources

- **scap-security-guide(8)** man page
- [OpenSCAP memory consumption problems](#)

7.3.4. 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:

```
$ oscap xccdf eval --report report.html --profile hipaa /usr/share/xml/scap/ssg/content/ssg-rhel9-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-rhel9-ds.xml
```

Additional resources

- **scap-security-guide(8)** man page
- **SCAP Security Guide** documentation in the */usr/share/doc/scap-security-guide/* directory
- */usr/share/doc/scap-security-guide/guides/ssg-rhel9-guide-index.html* – [Guide to the Secure Configuration of Red Hat Enterprise Linux 9] installed with the **scap-security-guide-doc** package
- [OpenSCAP memory consumption problems](#)

7.4. REMEDIATING THE SYSTEM TO ALIGN WITH A SPECIFIC BASELINE

Use this procedure to remediate the RHEL 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 system.

Procedure

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

```
# oscap xccdf eval --profile hipaa --remediate /usr/share/xml/scap/ssg/content/ssg-rhel9-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-rhel9-ds.xml
```

Additional resources

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

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

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

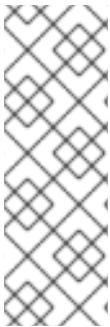


WARNING

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.
- The **ansible-core** package is installed. See the [Ansible Installation Guide](#) for more information.



NOTE

In RHEL 8.6 and later versions, Ansible Engine is replaced by the **ansible-core** package, which contains only built-in modules. Note that many Ansible remediations use modules from the community and Portable Operating System Interface (POSIX) collections, which are not included in the built-in modules. In this case, you can use Bash remediations as a substitute to Ansible remediations. The Red Hat Connector in RHEL 9 includes the necessary Ansible modules to enable the remediation playbooks to function with Ansible Core.

Procedure

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

```
# ansible-playbook -i localhost, -c local /usr/share/scap-security-guide/ansible/rhel9-  
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-rhel9-ds.xml
```

Additional resources

- **scap-security-guide(8)** and **oscap(8)** man pages
- [Ansible Documentation](#)

7.6. 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.



NOTE

In RHEL 9, Ansible Engine is replaced by the **ansible-core** package, which contains only built-in modules. Note that many Ansible remediations use modules from the community and Portable Operating System Interface (POSIX) collections, which are not included in the built-in modules. In this case, you can use Bash remediations as a substitute for Ansible remediations. The Red Hat Connector in RHEL 9.0 includes the necessary Ansible modules to enable the remediation playbooks to function with Ansible Core.

Prerequisites

- The **scap-security-guide** package is installed.

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-rhel9-ds.xml
```

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

```
# oscap xccdf generate fix --fix-type ansible --profile hipaa --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

- 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](#)

7.7. 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 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-rhel9-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

- 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

7.8. 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.

7.8.1. 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.

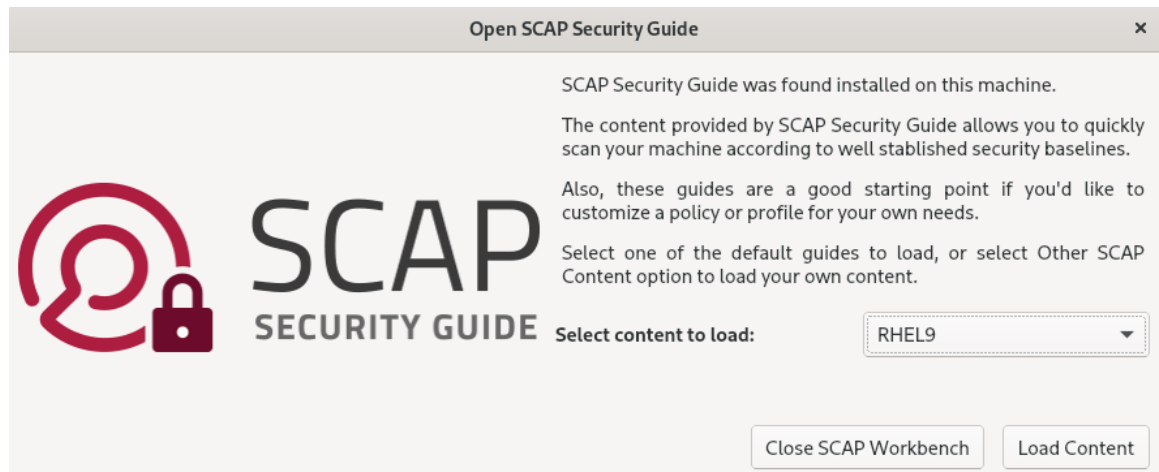
Procedure

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.



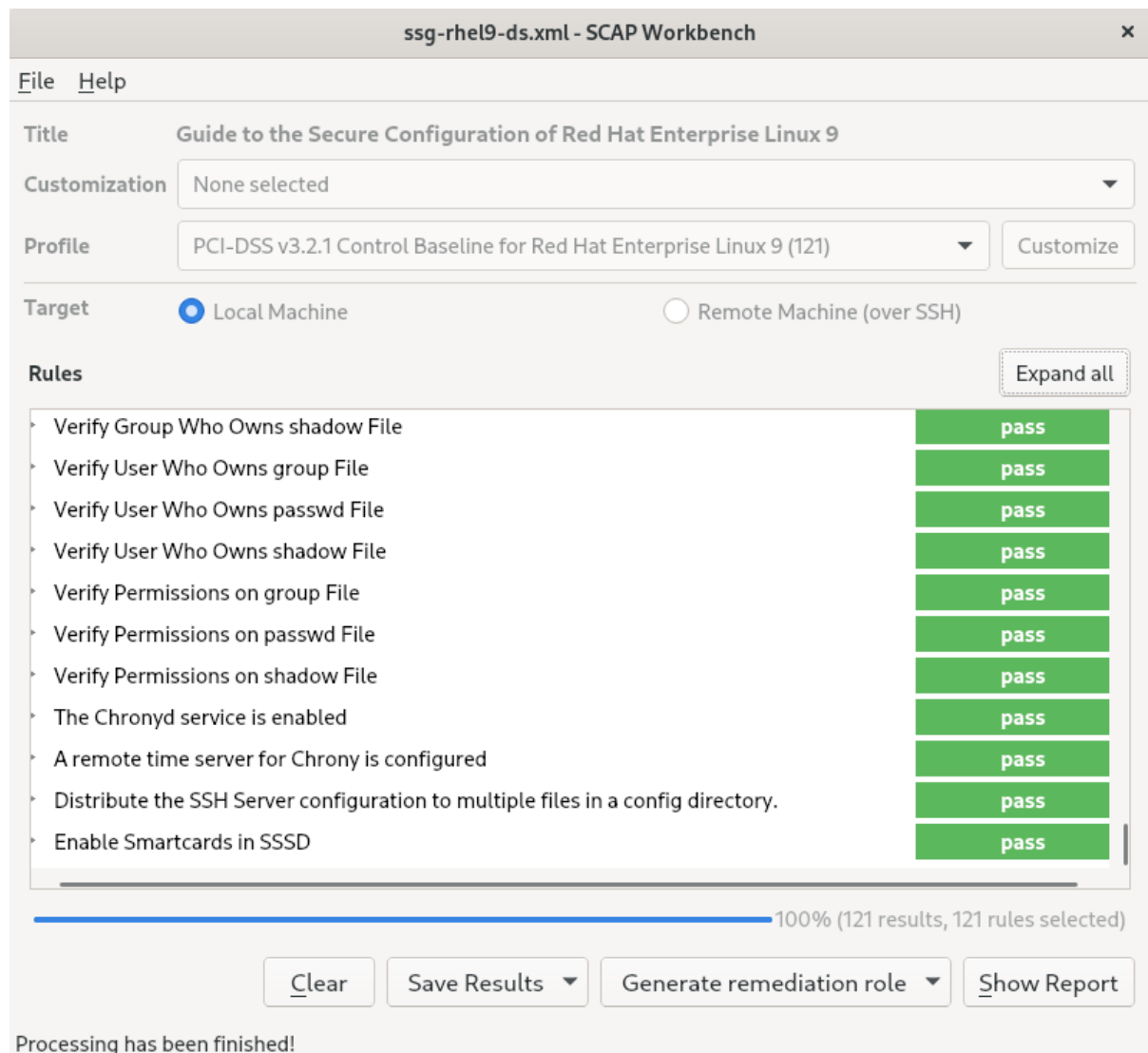
3. You can allow automatic correction of the system configuration by selecting the **Remediate** check box. With this option enabled, **SCAP Workbench** attempts to change the system configuration in accordance with the security rules applied by the policy. This process should fix the related checks 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.



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

7.8.2. 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.

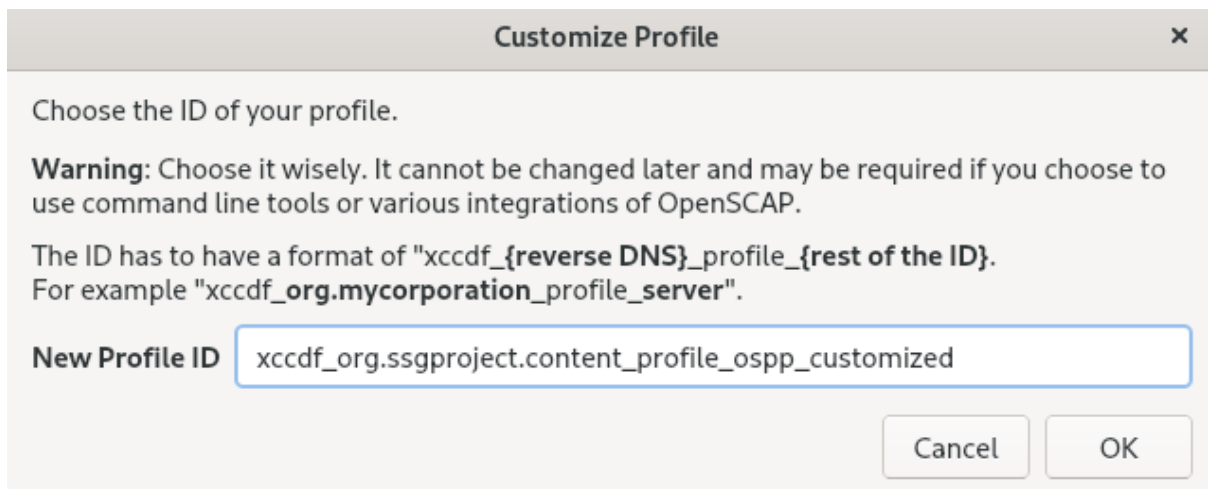
The following procedure demonstrates the use of **SCAP Workbench** for customizing (tailoring) a profile. You can also save the tailored profile for use with the **oscap** command-line utility.

Prerequisites

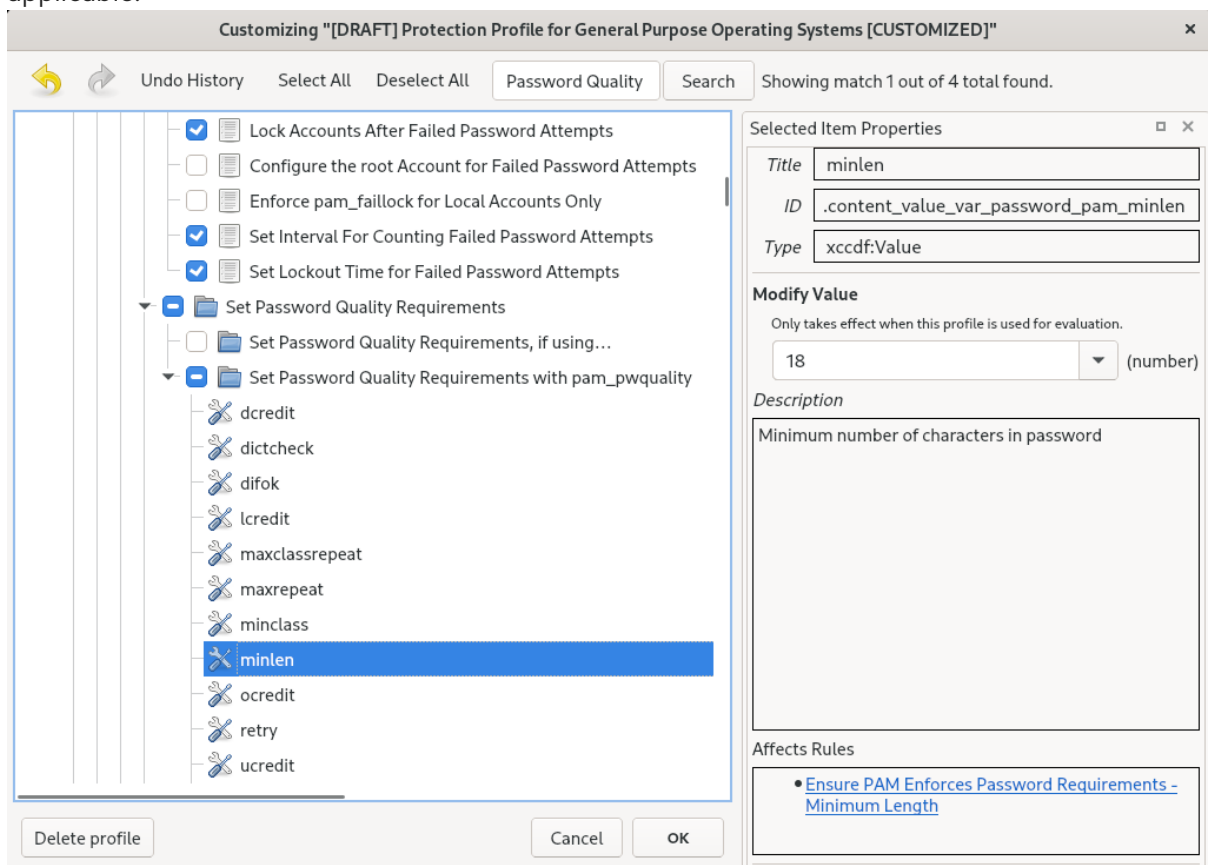
- The **scap-workbench** package is installed on your system.

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.



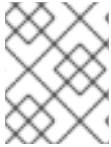
3. Find a rule to modify using either the tree structure with rules organized into logical groups or the **Search** field.
4. Include or exclude rules using check boxes in the tree structure, or modify values in rules where applicable.



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 **oscaped** command-line utility.

7.8.3. Additional resources

- **scap-workbench(8)** man page
- `/usr/share/doc/scap-workbench/user_manual.html` file provided by the **scap-workbench** package
- [Deploy customized SCAP policies with Satellite 6.x](#) KCS article

7.9. 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.

7.9.1. Profiles not compatible with Server with GUI

Certain security profiles provided as part of the **SCAP Security Guide** are not compatible with the extended package set included in the **Server with GUI** base environment. Therefore, do not select **Server with GUI** when installing systems compliant with one of the following profiles:

Table 7.2. Profiles not compatible with Server with GUI

Profile name	Profile ID	Justification	Notes
[DRAFT] CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	Packages xorg-x11-server-Xorg , xorg-x11-server-common , xorg-x11-server-utils , and xorg-x11-server-Xwayland are part of the Server with GUI package set, but the policy requires their removal.	

Profile name	Profile ID	Justification	Notes
[DRAFT] CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	Packages xorg-x11-server-Xorg , xorg-x11-server-common , xorg-x11-server-utils , and xorg-x11-server-Xwayland are part of the Server with GUI package set, but the policy requires their removal.	
[DRAFT] DISA STIG for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	Packages xorg-x11-server-Xorg , xorg-x11-server-common , xorg-x11-server-utils , and xorg-x11-server-Xwayland are part of the Server with GUI package set, but the policy requires their removal.	To install a RHEL system as a Server with GUI aligned with DISA STIG, you can use the DISA STIG with GUI profile BZ#1648162

7.9.2. 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).



WARNING

Certain security profiles provided as part of the **SCAP Security Guide** are not compatible with the extended package set included in the **Server with GUI** base environment. For additional details, see [Profiles not compatible with a GUI server](#).

Prerequisites

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

- 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-rhel9-ds.xml
```

Additional resources

- [Configuring manual partitioning](#)

7.9.3. 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 9 system.

Procedure

1. Open the **/usr/share/scap-security-guide/kickstart/ssg-rhel9-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.
3. Start a Kickstart installation as described in [Performing an automated installation using Kickstart](#).



IMPORTANT

Passwords in Kickstart files are not 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-rhel9-ds.xml
```

Additional resources

- [OSCAP Anaconda Add-on](#)

7.10. SCANNING CONTAINER AND CONTAINER IMAGES FOR VULNERABILITIES

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

Prerequisites

- The **openscap-utils** and **bzip2** packages are installed.

Procedure

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

```
# wget -O - https://www.redhat.com/security/data/oval/v2/RHEL9/rhel-9.oval.xml.bz2 | bzip2 -
-decompress > rhel-9.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/ubi9/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-9.oval.xml
```

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

Verification

- 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.

7.11. 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).

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/ubi9/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-rhel9-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

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

7.12. SCAP SECURITY GUIDE PROFILES SUPPORTED IN RHEL 9

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 RHEL 9, together with the version of the policy with which the profile aligns.

Table 7.3. SCAP Security Guide profiles supported in RHEL 9.1

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 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	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
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 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.1.0 and RHEL 9.1.1:DRAFT ^[a] RHEL 9.1.2 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.1.0 and RHEL 9.1.1:DRAFT ^[a] RHEL 9.1.2 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.1.0 and RHEL 9.1.1:DRAFT ^[a] RHEL 9.1.2 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.1.0 and RHEL 9.1.1:DRAFT ^[a] RHEL 9.1.2 and later:1.0.0

Profile name	Profile ID	Policy version
[DRAFT] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
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
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	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 9	xccdf_org.ssgproject.content_profile_pci-dss	3.2.1
[DRAFT] DISA STIG for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	DRAFT ^[b]
[DRAFT] DISA STIG with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	DRAFT ^[b]
<p>[a] CIS has not yet published an official benchmark for RHEL 9</p> <p>[b] DISA has not yet published an official benchmark for RHEL 9</p>		

Table 7.4. SCAP Security Guide profiles supported in RHEL 9.0

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 High Level	xccdf_org.ssgproject.content_profile_anssi_bp28_high	1.2

Profile name	Profile ID	Policy version
French National Agency for the Security of Information Systems (ANSSI) BP-028 Intermediary Level	xccdf_org.ssgproject.content_profile_anssi_bp28_intermediary	1.2
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 9 Benchmark for Level 2 - Server	xccdf_org.ssgproject.content_profile_cis	RHEL 9.0.0 to RHEL 9.0.6:DRAFT [a] RHEL 9.0.7 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Server	xccdf_org.ssgproject.content_profile_cis_server_l1	RHEL 9.0.0 to RHEL 9.0.6:DRAFT [a] RHEL 9.0.7 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 1 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l1	RHEL 9.0.0 to RHEL 9.0.6:DRAFT [a] RHEL 9.0.7 and later:1.0.0
CIS Red Hat Enterprise Linux 9 Benchmark for Level 2 - Workstation	xccdf_org.ssgproject.content_profile_cis_workstation_l2	RHEL 9.0.0 to RHEL 9.0.6:DRAFT [a] RHEL 9.0.7 and later:1.0.0
[DRAFT] Unclassified Information in Non-federal Information Systems and Organizations (NIST 800-171)	xccdf_org.ssgproject.content_profile_cui	r2
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
Australian Cyber Security Centre (ACSC) ISM Official	xccdf_org.ssgproject.content_profile_ism_o	not versioned
Protection Profile for General Purpose Operating Systems	xccdf_org.ssgproject.content_profile_ospp	RHEL 9.0.0 to RHEL 9.0.2:DRAFT RHEL 9.0.3 and later:4.2.1
PCI-DSS v3.2.1 Control Baseline for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_pci-dss	3.2.1
[DRAFT] DISA STIG for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig	DRAFT [b]

Profile name	Profile ID	Policy version
[DRAFT] DISA STIG with GUI for Red Hat Enterprise Linux 9	xccdf_org.ssgproject.content_profile_stig_gui	DRAFT ^[b]

7.13. ADDITIONAL RESOURCES

- [Supported versions of the SCAP Security Guide in RHEL](#)
- [The OpenSCAP project page](#) provides detailed information about the **oscap** utility and other components and projects related to SCAP.
- [The SCAP Workbench project page](#) provides detailed information about the **scap-workbench** application.
- [The SCAP Security Guide \(SSG\) project page](#) provides the latest security content for Red Hat Enterprise Linux.
- [Using OpenSCAP for security compliance and vulnerability scanning](#) - A hands-on lab on running tools based on the Security Content Automation Protocol (SCAP) standard for compliance and vulnerability scanning in RHEL.
- [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 RHEL 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 RHEL, 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](#) has a vast collection of SCAP-related materials, including SCAP publications, specifications, and the SCAP Validation Program.
- [National Vulnerability Database \(NVD\)](#) has the largest repository of SCAP content and other SCAP standards-based vulnerability management data.
- [Red Hat OVAL content repository](#) contains OVAL definitions for vulnerabilities of RHEL 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 8. ENSURING SYSTEM INTEGRITY WITH KEYLIME

With Keylime, you can verify the state of systems at boot and continuously monitor the integrity of remote systems. You can also send encrypted files to the monitored systems, and specify automated actions triggered whenever a monitored system fails the integrity test.

8.1. HOW KEYLIME WORKS

Keylime's concept of trust is based on the Trusted Platform Module (TPM) technology. A TPM is a hardware, firmware, or virtual component with integrated cryptographic keys. By polling TPM quotes and comparing the hashes of objects, Keylime provides initial and runtime monitoring of remote systems.

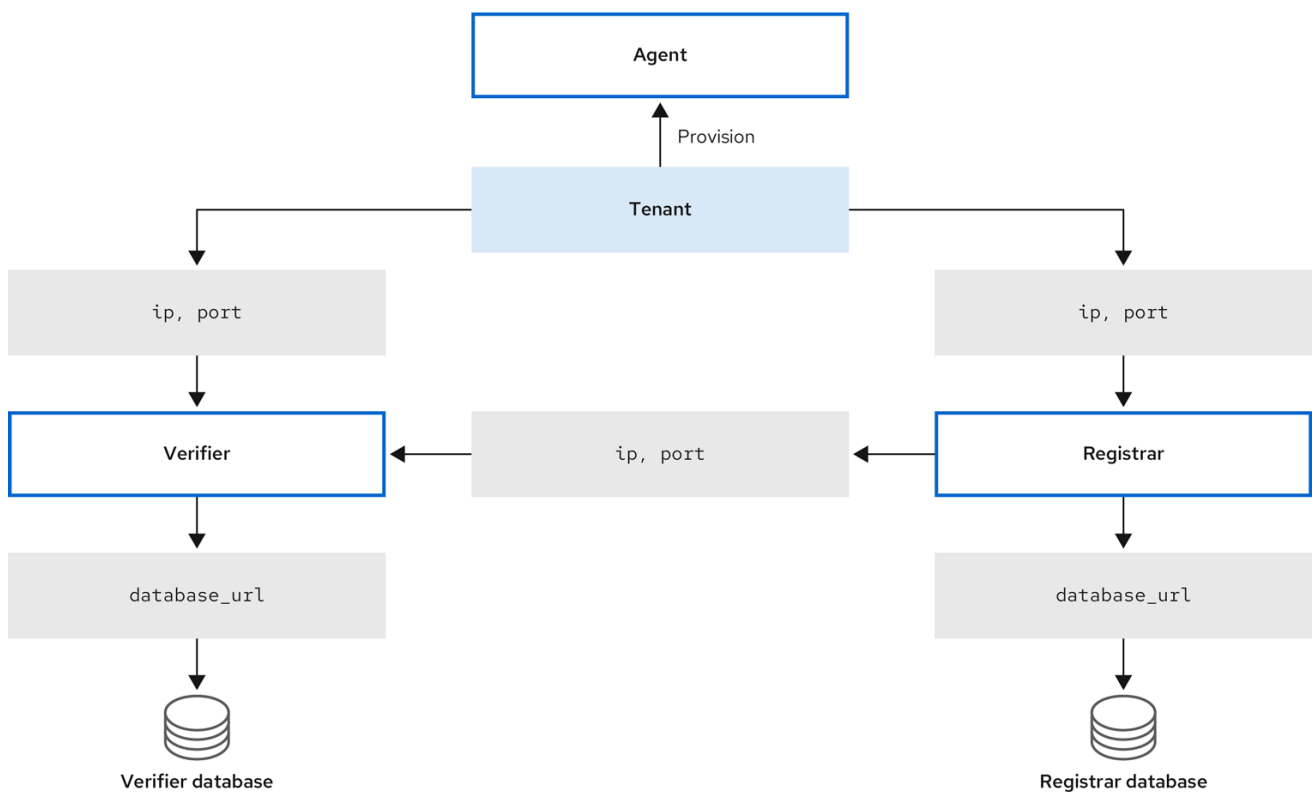
Keylime consists of three main components:

- The *verifier* initially and continuously verifies the integrity of the systems that run the agent.
- The *registrar* contains a database of all agents and it hosts the public keys of the TPM vendors.
- The *agent* is the component deployed to remote systems measured by the verifier.

In addition, Keylime uses the **keylime_tenant** utility for many functions, including provisioning the agents on the target systems.

You can configure the agent so that Keylime performs *measured boot* on the monitored systems, or *runtime integrity monitoring* of the monitored systems, or both.

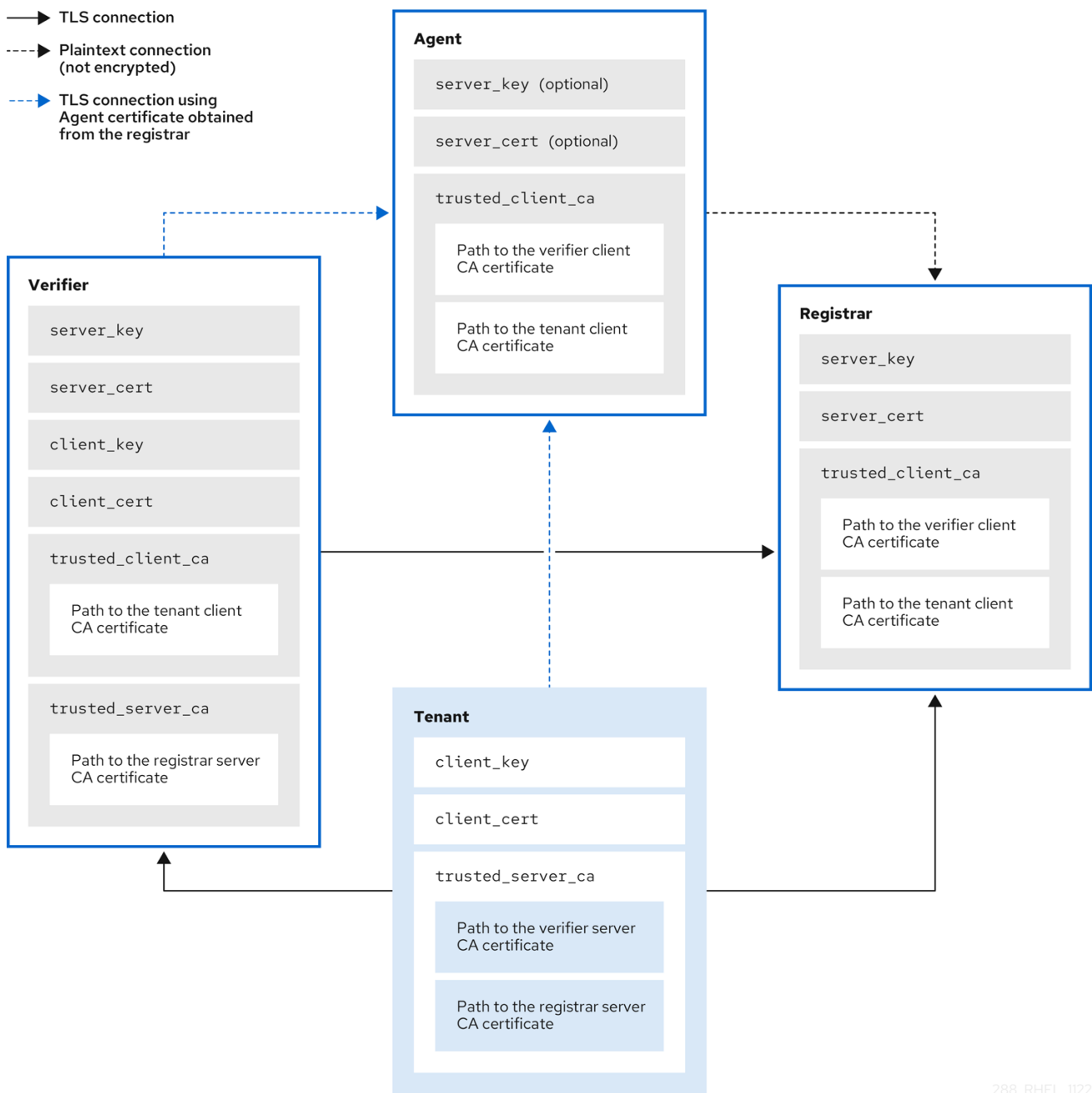
Figure 8.1. Connections between Keylime components through configurations



288_RHEL_1122

Keylime ensures the integrity of the monitored systems in a chain of trust by using keys and certificates exchanged between the components and the tenant. For a secure foundation of this chain, use a certificate authority (CA) you can trust.

Figure 8.2. Connections between Keylime components certificates and keys

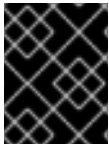


NOTE

If the agent receives no key and certificate, it generates a key and a self-signed certificate with no involvement from the CA.

8.2. CONFIGURING KEYLIME VERIFIER AND REGISTRAR

The verifier and registrar are the two components necessary for monitoring the agents. You can install them on a single system or on two separate systems, depending on your requirements. Running the verifier and registrar on separate systems provides a better performance.



IMPORTANT

To maintain the chain of trust, keep the systems that run the verifier and registrar secure and under your control.

- The *verifier* is the most important component in Keylime. It does initial and periodic checks of system integrity and supports bootstrapping a cryptographic key securely with the agent. The verifier uses mutual Transport Layer Security (TLS) for its control interface.
- The *registrar* is an HTTPS service that accepts trusted platform module (TPM) public keys. It then presents an interface to obtain these public keys for checking quotes.



NOTE

To keep the configuration files organized within the drop-in directories, use file names with a two-digit number prefix, for example **/etc/keylime/verifier.conf.d/00-registrar-ip.conf**. The configuration processing reads the files inside the drop-in directory in lexicographic order and sets each option to the last value it read.

Prerequisites

- Administrator rights to the system or systems on which you want to install Keylime components
- Network connection between the systems
- Access to two databases, where Keylime saves data from the registrar and from the verifier
You can use any of the following databases:
 - SQLite (default)
 - PostgreSQL
 - MySQL / MariaDB
- Valid keys and certificates from your certificate authority.

Procedure

1. Install the required Keylime components. You can install the verifier and registrar on a single system or two separate systems, depending on your requirements:

- To install all components of Keylime:

```
# dnf install keylime
```

- To install only the Keylime verifier:

```
# dnf install keylime-verifier
```

- To install only the Keylime registrar:

```
# dnf install keylime-registrar
```

2. When installing the verifier and registrar on separate systems, define the IP address and port of

the registrar in the verifier configuration. On the system where the verifier is installed, create a new **.conf** file in the **/etc/keylime/verifier.conf.d/** directory, for example, **/etc/keylime/verifier.conf.d/00-registrar-ip.conf**, with the following content:

```
[verifier]

registrar_ip = <registrar_IP_address>
```

- Replace **<registrar_IP_address>** with the registrar's IP address.
 - Optionally, you can also change the registrar's port from the default value **8891** by using the **registrar_port = <registrar_port>** option.
3. Optional: Configure the verifier's database for the list of agents. The default configuration uses an SQLite database in the verifiers's **/var/lib/keylime/cv_data.sqlite** directory. You can define a different database by creating a new **.conf** file in the **/etc/keylime/verifier.conf.d/** directory, for example, **/etc/keylime/verifier.conf.d/00-db-ip.conf**, with the following content:

```
[verifier]
# Database URL Configuration
database_url = <database_url>
```

Replace **<database_url>** with the URL of the database.

4. Add certificates and keys to the verifier. The verifier needs the following keys and certificates:

- **server_key**
 - **server_cert**
 - **client_key**
 - **client_cert**
 - **trusted_client_ca**
 - Path to the tenant client CA certificate
 - **trusted_server_ca**
 - Path to the registrar server CA certificate
- a. You can use the default configuration and load the keys and certificates to the **/var/lib/keylime/cv_ca** directory.
- b. Alternatively, you can define the location of the keys and certificates in the configuration. Create a new **.conf** file in the **/etc/keylime/verifier.conf.d/** directory, for example, **/etc/keylime/verifier.conf.d/00-keys-and-certs.conf**, with the following content:

```
[verifier]
tls_dir = default
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
client_key = </path/to/client_key>
```

```
client_key_password = <passphrase2>
client_cert = </path/to/client_cert>
trusted_server_ca = ['</path/to/ca/cert3>', '</path/to/ca/cert4>']
```

**NOTE**

Use absolute paths to define key and certificate locations. Alternatively, you can define a directory in the **tls_dir** option and use paths relative to that directory.

5. Optional: Configure the registrar's database for the list of agents. The default configuration uses an SQLite database in the registrar's **/var/lib/keylime/reg_data.sqlite** directory. You can create a new **.conf** file in the **/etc/keylime/registrar.conf.d/** directory, for example, **/etc/keylime/registrar.conf.d/00-db-ip.conf**, with the following content:

```
[registrar]
# Database URL Configuration
database_url = <database_url>
```

Replace **<database_url>** with the URL of the database.

6. Add certificates and keys to the registrar. The registrar requires the following keys and certificates:

- **server_key**
 - **server_cert**
 - **trusted_client_ca**
 - Path to the verifier client CA certificate
 - Path to the tenant client CA certificate
- a. You can use the default configuration and load the keys and certificates to the **/var/lib/keylime/reg_ca** directory.
 - b. Alternatively, you can define the location of the keys and certificates in the configuration. Create a new **.conf** file in the **/etc/keylime/registrar.conf.d/** directory, for example, **/etc/keylime/registrar.conf.d/00-keys-and-certs.conf**, with the following content:

```
[registrar]
tls_dir = default
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
```

**NOTE**

Use absolute paths to define key and certificate locations. Alternatively, you can define a directory in the **tls_dir** option and use paths relative to that directory.

7. Start the verifier service:



NOTE

Start the verifier before starting the registrar, so that the configuration files can load in the correct order. If you need to stop Keylime, stop the services in the reverse order.

```
$ systemctl start keylime_verifier
```

8. Start the registrar service:

```
$ systemctl start keylime_registrar
```

Verification

1. Check the status of the Keylime services:

```
$ systemctl status keylime_verifier
● keylime_verifier.service - The Keylime verifier
   Loaded: loaded (/usr/lib/systemd/system/keylime_verifier.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-11-09 10:10:08 EST; 1min 45s ago
   ...
$ systemctl status keylime_registrar
● keylime_registrar.service - The Keylime registrar service
   Loaded: loaded (/usr/lib/systemd/system/keylime_registrar.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2022-11-09 10:10:17 EST; 1min 42s ago
   ...
```

2. Check the status of the verifier:

```
$ keylime_tenant -c cvstatus
Reading configuration from ['/etc/keylime/logging.conf']
2022-10-14 12:56:08.155 - keylime.tpm - INFO - TPM2-TOOLS Version: 5.2
Reading configuration from ['/etc/keylime/tenant.conf']
2022-10-14 12:56:08.157 - keylime.tenant - INFO - Setting up client TLS...
2022-10-14 12:56:08.158 - keylime.tenant - INFO - Using default client_cert option for tenant
2022-10-14 12:56:08.158 - keylime.tenant - INFO - Using default client_key option for tenant
2022-10-14 12:56:08.178 - keylime.tenant - INFO - TLS is enabled.
2022-10-14 12:56:08.178 - keylime.tenant - WARNING - Using default UUID d432fbb3-d2f1-4a97-9ef7-75bd81c00000
2022-10-14 12:56:08.221 - keylime.tenant - INFO - Verifier at 127.0.0.1 with Port 8881 does not have agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000.
```

If correctly set up, and if no agent is configured, the verifier does not recognize agent UUID.

3. Check the status of the registrar:

```
$ keylime_tenant -c regstatus
Reading configuration from ['/etc/keylime/logging.conf']
2022-10-14 12:56:02.114 - keylime.tpm - INFO - TPM2-TOOLS Version: 5.2
```

```

Reading configuration from [/etc/keylime/tenant.conf]
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Setting up client TLS...
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Using default client_cert option for tenant
2022-10-14 12:56:02.116 - keylime.tenant - INFO - Using default client_key option for tenant
2022-10-14 12:56:02.137 - keylime.tenant - INFO - TLS is enabled.
2022-10-14 12:56:02.137 - keylime.tenant - WARNING - Using default UUID d432fbb3-d2f1-4a97-9ef7-75bd81c00000
2022-10-14 12:56:02.171 - keylime.registrar_client - CRITICAL - Error: could not get agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 data from Registrar Server: 404
2022-10-14 12:56:02.172 - keylime.registrar_client - CRITICAL - Response code 404: agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 not found
2022-10-14 12:56:02.172 - keylime.tenant - INFO - Agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 does not exist on the registrar. Please register the agent with the registrar.
2022-10-14 12:56:02.172 - keylime.tenant - INFO - {"code": 404, "status": "Agent d432fbb3-d2f1-4a97-9ef7-75bd81c00000 does not exist on registrar 127.0.0.1 port 8891.", "results": {}}
```

If correctly set up, and if no agent is configured, the registrar does not recognize agent UUID.

Next steps

After the Keylime verifier and registrar are configured and running, you can deploy the Keylime agent on the monitored systems to perform one or both of the following functions:

- [Section 8.3, “Deploying Keylime for measured boot attestation”](#)
- [Section 8.4, “Deploying Keylime for runtime monitoring”](#)

8.3. DEPLOYING KEYLIME FOR MEASURED BOOT ATTESTATION



WARNING

Currently, Keylime might fail attestation of systems that access multiple files measured by the Integrity Measurement Architecture (IMA) in quick succession. For more information, see [RHBZ#2138167](#).

For measured boot attestation, the Keylime agent must be running on the monitored systems. You can remotely provision the Keylime agent by using the **keylime_tenant** utility.

When provisioning the agent, you can also define a file that Keylime sends to the monitored system. Keylime encrypts the file sent to the agent, and decrypts it only if agent’s system complies with the TPM policy and the IMA allowlist.

To ignore changes of specific files or within specific directories, configure a Keylime excludes list.

By default, the Keylime agent stores all its data in the **/var/lib/keylime/** directory of the monitored system.



NOTE

To keep the configuration files organized within the drop-in directories, use file names with a two-digit number prefix, for example **/etc/keylime/agent.conf.d/00-registrar-ip.conf**. The configuration processing reads the files inside the drop-in directory in lexicographic order and sets each option to the last value it read.

Prerequisites

- Network access to the Keylime verifier and registrar. For more information, see [Section 8.2, “Configuring Keylime verifier and registrar”](#).
- Administrator rights to the system or systems on which you want to install Keylime components
- TPM chip on the system
 - You can verify that your system has a TPM by entering the **tpm2_pcrread** command. If the output from this command displays several hashes, you have a TPM.
- Integrity measurement architecture (IMA) enabled on the agent system. For more information, see [Enabling integrity measurement architecture and extended verification module](#).

Procedure

1. Install the Keylime agent on the system you want to monitor:

```
# dnf install keylime-agent
```

This command installs the **keylime-agent-rust** package.

2. Define the registrar’s IP address and port in the configuration files. Create a new **.conf** file in the **/etc/keylime/agent.conf.d/** directory, for example, **/etc/keylime/agent.conf.d/00-registrar-ip.conf**, with the following content:

```
[agent]

registrar_ip = "<registrar_IP_address>"
```

- Replace **<registrar_IP_address>** with the registrar’s IP address.
- Optionally, you can also change the registrar’s port from the default value **8890** by using the **registrar_port = <registrar_port>** option.



NOTE

Because the Keylime agent configuration uses the TOML format which is different from the INI format used for the configuration of the other components, the IP address must be in quotation marks.

3. Optional: Load existing keys and certificates for the agent. If the agent receives no **server_key** and **server_cert**, it generates its own key and a self-signed certificate. The agent accepts the following keys and certificates:
 - **server_key** (optional)

- **server_cert** (optional)
- **trusted_client_ca**
 - Path to the verifier client CA certificate
 - Path to the tenant client CA certificate

Define the location of the keys and certificates in the configuration. Create a new **.conf** file in the **/etc/keylime/agent.conf.d/** directory, for example, **/etc/keylime/agent.conf.d/00-keys-and-certs.conf**, with the following content:

```
[agent]
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
enc_keyname = </path/to/derived_cti_key>
```



NOTE

Use absolute paths to define key and certificate locations. The Keylime agent does not accept relative paths.

4. Generate a policy from the measured boot log of the current state of the system:

CAUTION

In some scenarios, the Keylime measured boot policy generation script might cause a segmentation fault and core dump. For more information, see [RHBZ#2140670](#).

```
$ /usr/share/keylime/scripts/create_mb_refstate
/sys/kernel/security/tpm0/binary_bios_measurements
<./measured_boot_reference_state.json>
```

Replace **<./measured_boot_reference_state.json>** with the path where the script saves the generated policy.



IMPORTANT

The policy generated with the **create_mb_refstate** script is based on the current state of the system and is very strict. Any modifications of the system including kernel updates and system updates will change the boot process and the system will fail the attestation.

5. Optional: To define a list of files and directories excluded from Keylime measurements, create a file named, for example, **<excludelist>**, and enter files and directories to exclude. The exclude list accepts Python regular expressions. For more information, see [Regular expression operations at docs.python.org](#). For example, to exclude all files in the **/tmp/** directory:

```
/tmp/.*
```


- Provision a new agent by using the **keylime_tenant** utility. You can provision the agent from any system connected to the network and provided with the correct keys and certificates:

```
$ keylime_tenant -c add -t <agent.ip> -v <verifier.ip> -u <agent-uuid> --mb_refstate
<./measured_boot_reference_state.json> --exclude <excludelist> -f <filetosend>
```

- Replace **<agent.ip>** with the agent's IP address.
- Replace **<verifier.ip>** with the verifier's IP address.
- Replace **<agent-uuid>** with the agent's UUID.
- Replace **<./measured_boot_reference_state.json>** with the path to the measured boot policy.
- Replace **<excludelist>** with the path to the exclude list file. The **--exclude** option is optional; provisioning the agent works even without delivering a file.
- Replace **<filetosend>** with the path to the file to be delivered to the agent. The **-f** option is optional; provisioning the agent works even without delivering a file.



NOTE

Keylime encrypts the file sent to the agent, and decrypts it only if agent's system complies with the TPM policy and the IMA allowlist. By default, Keylime decompresses **.zip** files. You can also specify a script that runs after the file is decrypted by using the **payload_script = "autorun.sh"** option.

As an example, with the following command, **keylime_tenant** provisions a new Keylime agent at **127.0.0.1** with UUID **d432fbb3-d2f1-4a97-9ef7-75bd81c00000** and connects it to a verifier at **127.0.0.2**. It also encrypts a file named **payload1** and sends it to the agent. Keylime decrypts the file only if agent's system complies with the TPM policy and the IMA allowlist:

```
$ keylime_tenant -c add -t 127.0.0.1 -v 127.0.0.2 -u d432fbb3-d2f1-4a97-9ef7-75bd81c00000
-f payload1
```

- Start the **keylime_agent** service:

```
$ systemctl start keylime_agent
```



NOTE

You can stop Keylime from monitoring a node by using the following command from any system in the network with the correct keys and certificates:

```
$ keylime_tenant -c delete -t <agent.ip> -u <agent.uuid>
```

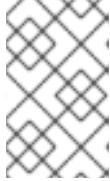
Verification

- List the UUIDs of all agents registered with the registrar:

```
$ keylime_tenant -c reglist -r <registrar_IP_address> -rp <registrar_port>
2022-10-07 13:52:54.388 - keylime.tenant - INFO - From registrar 127.0.0.1 port 8891
```

```
retrieved {"code": 200, "status": "Success", "results": {"uuids": ["d432fbb3-d2f1-4a97-9ef7-75bd81c00000"]}}
```

- Replace **<registrar_IP_address>** with the registrar's IP address.
- Optionally, you can also change the registrar's port from the default value **8891** by using the **registrar_port = <registrar_port>** option.



NOTE

If you need to list agents repeatedly, you can define the registrar's IP address and port in a configuration snippet in the **/etc/keylime/tenant.conf.d/** directory. Then, you can enter the command without the **-r** and **-rp** options.

2. Check the status of the verifier:

```
$ keylime_tenant -c cvstatus
Reading configuration from [/etc/keylime/logging.conf]
...
{"d432fbb3-d2f1-4a97-9ef7-75bd81c00000": {"operational_state": "Get Quote", "v":
"rMUdRQtojtoufQGLS5mur9yrH7ZDhivxAVihhMILTc=", "ip": "127.0.0.1", "port": 9002,
"tpm_policy": "{\"mask\": \"0x0\"}", "meta_data": "{\"cert_serial\": 2, \"subject\":
\"OU=53,O=MITLL,L=Lexington,ST=MA,CN=d432fbb3-d2f1-4a97-9ef7-
75bd81c00000,C=US\"}", "allowlist_len": 6, "mb_refstate_len": 0, "accept_tpm_hash_algs":
["sha512", "sha384", "sha256", "sha1"], "accept_tpm_encryption_algs": ["ecc", "rsa"],
"accept_tpm_signing_algs": ["ecschnorr", "rsassa"], "hash_alg": "sha256", "enc_alg": "rsa",
"sign_alg": "rsassa", "verifier_id": "default", "verifier_ip": "127.0.0.1", "verifier_port": 8881,
"severity_level": null, "last_event_id": null, "attestation_count": 7, "last_received_quote":
1665753341}}
```

If the verifier and agent are correctly configured, the output displays the correct agent UUID followed by **{"operational_state": "Get Quote"}**.

3. Check the status of the registrar.

```
$ keylime_tenant -c regstatus
Reading configuration from [/etc/keylime/logging.conf]
...
==\n-----END CERTIFICATE-----\n", "ip": "127.0.0.1", "port": 9002, "regcount": 1,
"operational_state": "Registered"}}
```

If the registrar and agent are correctly configured, the output displays the agent's IP address, and port, followed by **"operational_state": "Registered"**.

Additional resources

- For additional advanced options for the **keylime_tenant** utility, enter the **keylime_tenant -h** command.

8.4. DEPLOYING KEYLIME FOR RUNTIME MONITORING



WARNING

Currently, Keylime might fail attestation of systems that access multiple files measured by the Integrity Measurement Architecture (IMA) in quick succession. For more information, see [RHBZ#2138167](#).

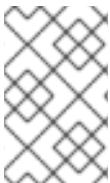
To verify that the initial state of monitored systems is correct, the Keylime agent must be running on the monitored systems and you must provide an allowlist to Keylime. When you provision a new agent, Keylime checks that the files on the system correspond to the state you defined in the allowlist and continuously monitors the files.



IMPORTANT

To ensure maximum security, create the allowlist on a fully encrypted, air-gapped computer permanently isolated from the Internet. To prevent tampering when transferring the allowlist to other systems, disable all network cards and sign the allowlist hash.

You can generate an allowlist from **initramfs** by using a Keylime script, but you can also generate hashes for your applications files or admin scripts that will run on the remotely monitored system, and manually enter them into the allowlist file.



NOTE

Because Keylime runtime monitoring uses Integrity measurement architecture (IMA) to measure large numbers of files, it might have a significant impact on the performance of your system.

When provisioning the agent, you can also define a file that Keylime sends to the monitored system. Keylime encrypts the file sent to the agent, and decrypts it only if agent's system complies with the TPM policy and the IMA allowlist.

To ignore changes of specific files or within specific directories, configure a Keylime excludes list.

By default, the Keylime agent stores all its data in the **/var/lib/keylime/** directory of the monitored system.



NOTE

To keep the configuration files organized within the drop-in directories, use file names with a two-digit number prefix, for example **/etc/keylime/agent.conf.d/00-registrar-ip.conf**. The configuration processing reads the files inside the drop-in directory in lexicographic order and sets each option to the last value it read.

Prerequisites

- Network access to the Keylime verifier and registrar. For more information, see [Section 8.2, "Configuring Keylime verifier and registrar"](#).

- Administrator rights to the system or systems on which you want to install Keylime components
- TPM chip on the system
 - You can verify that your system has a TPM by entering the **tpm2_pcrread** command. If the output from this command displays several hashes, you have a TPM.
- Integrity measurement architecture (IMA) enabled on the agent system. For more information, see [Enabling integrity measurement architecture and extended verification module](#) .

Procedure

1. Install the Keylime agent on the system to be monitored:

```
# dnf install keylime-agent
```

This command installs the **keylime-agent-rust** package.

2. Create a new IMA policy by entering the following content into the **/etc/ima/ima-policy** file:

```
# PROC_SUPER_MAGIC
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC
dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC
dont_measure fsmagic=0x01021994
# RAMFS_MAGIC
dont_measure fsmagic=0x858458f6
# SECURITYFS_MAGIC
dont_measure fsmagic=0x73636673
# SELINUX_MAGIC
dont_measure fsmagic=0xf97cff8c
# CGROUP_SUPER_MAGIC
dont_measure fsmagic=0x27e0eb
# OVERLAYFS_MAGIC
dont_measure fsmagic=0x794c7630
# Don't measure log, audit or tmp files
dont_measure obj_type=var_log_t
dont_measure obj_type=auditd_log_t
dont_measure obj_type=tmp_t
# MEASUREMENTS
measure func=BPRM_CHECK
measure func=FILE_MMAP mask=MAY_EXEC
measure func=MODULE_CHECK uid=0
```

3. Replace the default IMA policy with the new IMA policy:

```
# cat /etc/ima/ima-policy > /sys/kernel/security/ima/policy
```

4. Reboot the system to apply the new IMA policy.
5. Generate an allowlist from the current state of the system:

```
# /usr/share/keylime/scripts/create_allowlist.sh -o <allowlist.txt> -h sha256sum
```

Replace **<allowlist.txt>** with the file name of the allowlist.



IMPORTANT

Use the SHA-256 hash function. SHA-1 is not secure and has been deprecated in RHEL 9. For additional information, see [SHA-1 deprecation in Red Hat Enterprise Linux 9](#).

- Define the registrar's IP address and port in the configuration files. Create a new **.conf** file in the **/etc/keylime/agent.conf.d/** directory, for example, **/etc/keylime/agent.conf.d/00-registrar-ip.conf**, with the following content:

```
[agent]

registrar_ip = "<registrar_IP_address>"
```

- Replace **<registrar_IP_address>** with the registrar's IP address.
- Optionally, you can also change the registrar's port from the default value **8890** by using the **registrar_port = <registrar_port>** option.



NOTE

Because the Keylime agent configuration uses the TOML format which is different from the INI format used for configuration of the other components, the IP address must be in quotation marks.

- Optional: Load existing keys and certificates for the agent. If the agent receives no **server_key** and **server_cert**, it generates its own key and a self-signed certificate. The agent accepts the following keys and certificates:

- server_key** (optional)
- server_cert** (optional)
- trusted_client_ca**
 - Path to the verifier client CA certificate
 - Path to the tenant client CA certificate

Define the location of the keys and certificates in the configuration. Create a new **.conf** file in the **/etc/keylime/agent.conf.d/** directory, for example, **/etc/keylime/agent.conf.d/00-keys-and-certs.conf**, with the following content:

```
[agent]
server_key = </path/to/server_key>
server_key_password = <passphrase1>
server_cert = </path/to/server_cert>
trusted_client_ca = ['</path/to/ca/cert1>', '</path/to/ca/cert2>']
enc_keyname = </path/to/derived_cti_key>
```

**NOTE**

Use absolute paths to define key and certificate locations. The Keylime agent does not accept relative paths.

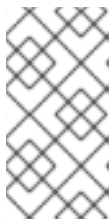
8. Optional: You can define a list of files or directories excluded from Keylime measurements by creating a file named, for example, **<excludelist>**, and entering files and directories to exclude. The exclude list accepts Python regular expressions. For more information, see [Regular expression operations at docs.python.org](https://docs.python.org/3/library/re.html). For example, to exclude all files in the **/tmp/** directory, enter:

```
/tmp/.*
```

9. Provision a new agent by using the **keylime_tenant** utility. You can provision the agent from any system connected to the network and provided with the correct keys and certificates:

```
$ keylime_tenant -c add -t <agent.ip> -v <verifier.ip> -u <agent.uuid> --allowlist  
_<allowlist.txt> --exclude <excludelist> -f <filetosend>
```

- Replace **<agent.ip>** with the agent's IP address.
- Replace **<verifier.ip>** with the verifier's IP address.
- Replace **<agent.uuid>** with the agent's UUID.
- Replace **<allowlist.txt>** with the path to the allowlist file.
- Replace **<excludelist>** with the path to the exclude list file. The **--exclude** option is optional; provisioning the agent works even without delivering a file.
- Replace **<filetosend>** with the path to the file to be delivered to the agent. The **-f** option is optional; provisioning the agent works even without delivering a file.

**NOTE**

Keylime encrypts the file sent to the agent, and decrypts it only if agent's system complies with the TPM policy and the IMA allowlist. By default, Keylime decompresses **.zip** files. You can also specify a script that runs after the file is decrypted by using the **payload_script = "autorun.sh"** option.

As an example, with the following command, **keylime_tenant** provisions a new Keylime agent at **127.0.0.1** with UUID **d432fbb3-d2f1-4a97-9ef7-75bd81c00000** and connects it to a verifier at **127.0.0.2** and loads an allowlist named **allowlist1.txt**. It also encrypts a file named **payload1** and sends it to the agent. Keylime decrypts the file only if the TPM policy configured in **/etc/keylime/verifier.conf** is satisfied:

```
$ keylime_tenant -c add -t 127.0.0.1 -v 127.0.0.2 -u d432fbb3-d2f1-4a97-9ef7-75bd81c00000  
-f payload1 --allowlist allowlist1.txt
```

10. Start the **keylime_agent** service:

```
$ systemctl start keylime_agent
```



NOTE

You can stop Keylime from monitoring a node by using the following command from any system in the network with the correct keys and certificates:

```
$ keylime_tenant -c delete -t <agent.ip> -u <agent.uuid>
```

Verification

1. Create a new file named, for example, **bad-script.sh**, and insert the following content that performs an action not permitted by the allowlist:

```
#!/bin/sh

echo -e "Hello Evil!"
```

2. Make the script executable:

```
# chmod +x bad-script.sh
```

3. Attempt to run the script. If the allowlist is configured correctly, Keylime attestation of this agent fails and shows a similar output:

```
# ./bad-script.sh
keylime.tpm - INFO - Checking IMA measurement list...
keylime.ima - WARNING - File not found in allowlist: /root/bad-script.sh
keylime.ima - ERROR - IMA ERRORS: template-hash 0 fnf 1 hash 0 good 781
keylime.cloudverifier - WARNING - agent D432FBB3-D2F1-4A97-9EF7-75BD81C00000
failed, stopping polling
```

Additional resources

- For more information about IMA, see [Enhancing security with the kernel integrity subsystem](#).

CHAPTER 9. 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.

9.1. 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:

```
# dnf 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.

9.2. 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]
```

- 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
```

9.3. 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

- Update your baseline **AIDE** database:

```
# aide --update
```

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

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

9.4. FILE-INTEGRITY TOOLS: AIDE AND IMA

Red Hat Enterprise Linux provides several tools for checking and preserving the integrity of files and directories on your system. The following table helps you decide which tool better fits your scenario.

Table 9.1. Comparison between AIDE and IMA

Question	Advanced Intrusion Detection Environment (AIDE)	Integrity Measurement Architecture (IMA)
What	AIDE is a utility that creates a database of files and directories on the system. This database serves for checking file integrity and detect intrusion detection.	IMA detects if a file is altered by checking file measurement (hash values) compared to previously stored extended attributes.
How	AIDE uses rules to compare the integrity state of the files and directories.	IMA uses file hash values to detect the intrusion.

Question	Advanced Intrusion Detection Environment (AIDE)	Integrity Measurement Architecture (IMA)
Why	Detection - AIDE detects if a file is modified by verifying the rules.	Detection and Prevention - IMA detects and prevents an attack by replacing the extended attribute of a file.
Usage	AIDE detects a threat when the file or directory is modified.	IMA detects a threat when someone tries to alter the entire file.
Extension	AIDE checks the integrity of files and directories on the local system.	IMA ensures security on the local and remote systems.

9.5. ADDITIONAL RESOURCES

- **aide(1)** man page
- [Kernel integrity subsystem](#)

CHAPTER 10. 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.

10.1. 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 uses 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](#)

10.2. LUKS VERSIONS IN RHEL

In RHEL, 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.

10.3. 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.

10.4. 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. Add an entry for a persistent mapping to **/etc/crypttab**

- a. Find the **luksUUID**:

```
# cryptsetup luksUUID /dev/mapper/sdb1_encrypted
```

This displays the **luksUUID** of the selected device.

- b. Open the **/etc/crypttab** file in a text editor of your choice and add a device in this file:

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

```
/dev/mapper/sdb1_encrypted luks_uuid none
```

- c. Refresh initramfs with **dracut**:

```
$ dracut -f --regenerate-all
```

6. Add an entry for a persistent mounting to the **/etc/fstab** file:

- a. Find the **FS UUID** of the active LUKS block device

```
$ blkid -p /dev/mapper/sdb1_encrypted
```

- b. Open the **/etc/fstab** file in a text editor of your choice and add a device in this file, for example:

```
$ vi /etc/fstab
fs__uuid /home auto rw,user,auto 0 0
```

7. Start the online encryption:

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

Additional resources

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

10.5. 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

10.6. 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

10.7. CREATING A LUKS ENCRYPTED VOLUME USING THE `STORAGE` RHEL SYSTEM ROLE

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

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 Core configures other systems.
On the control node:
 - The **ansible-core** and **rhel-system-roles** packages are installed.

IMPORTANT

RHEL 8.0–8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the [How to download and install Red Hat Ansible Engine](#) Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#) Knowledgebase article.

- An inventory file which lists the managed nodes.

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

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

CHAPTER 11. CONFIGURING AUTOMATED UNLOCKING OF ENCRYPTED VOLUMES USING POLICY-BASED DECRYPTION

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 RHEL 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 unlocking volumes using a network server
- **tpm2** - allows unlocking volumes using a TPM2 policy
- **sss** - allows deploying high-availability systems using the Shamir's Secret Sharing (SSS) cryptographic scheme

11.1. NETWORK-BOUND DISK ENCRYPTION

The Network Bound Disc Encryption (NBDE) is a subcategory of Policy-Based Decryption (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.

In RHEL, NBDE is implemented through the following components and technologies:

Figure 11.1. 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 RHEL, 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 RHEL, 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.

Additional resources

- [NBDE \(Network-Bound Disk Encryption\) Technology](#) Knowledgebase article
- **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

11.2. 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:

```
# dnf 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 edit  Edit a binding from a clevis-bound slot in a LUKS device
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 clevis binding
clevis luks report Report tang keys' rotations
clevis luks unbind Unbinds a pin bound to a LUKS volume
clevis luks unlock Unlocks a LUKS volume
```

11.3. 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 **polycoreutils-python-utils** package and its dependencies are installed.
- The **firewalld** service is running.

Procedure

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

```
# dnf 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 restart 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

11.4. 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.

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

11.5. 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 9 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.

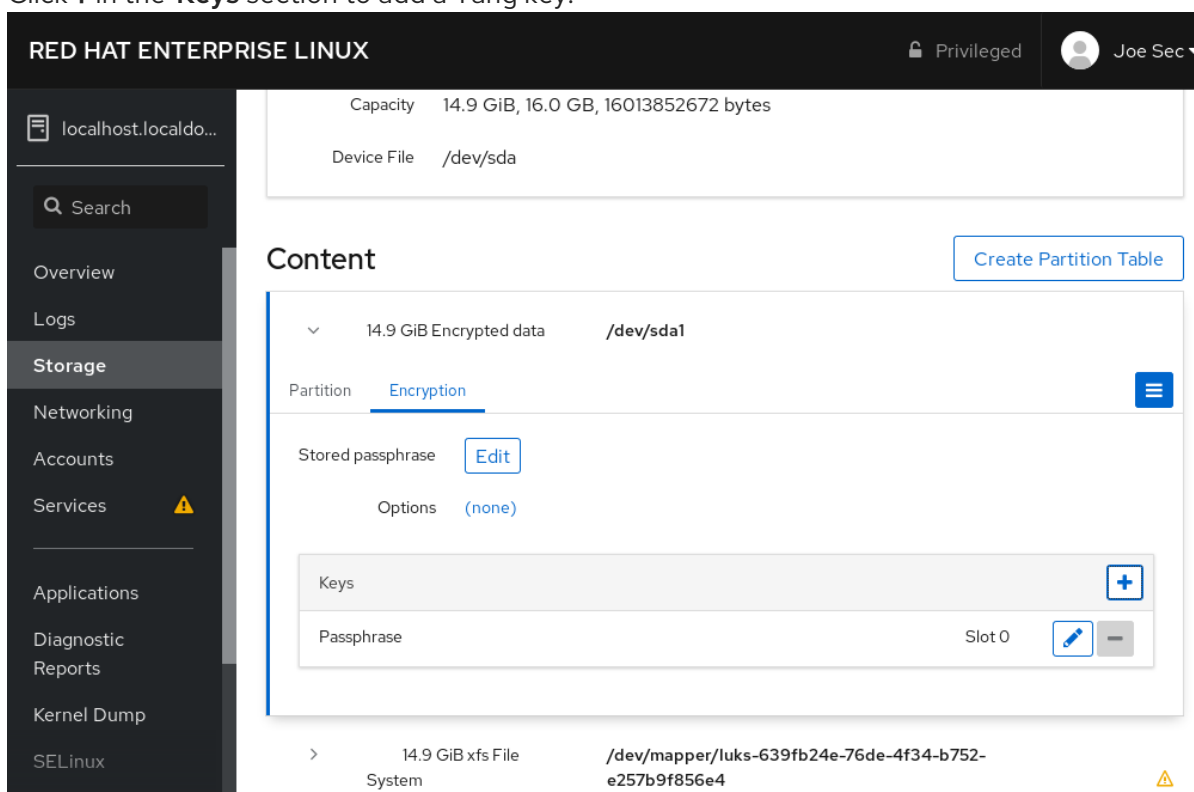
Procedure

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.

- Provide your credentials and click **Storage**. Click **>** to expand details of the encrypted device you want to unlock using the Tang server, and click **Encryption**.
- Click **+** in the **Keys** section to add a Tang key:



- Provide the address of your Tang server and a password that unlocks the LUKS-encrypted device. Click **Add** to confirm:

Add Key

Key source ☐ Passphrase ☒ Tang keyserver

Keyserver address

Disk passphrase

Saving a new passphrase requires unlocking the disk. Please provide a current disk passphrase.

The following dialog window provides a command to verify that the key hash matches.

- In a terminal on the Tang server, use the **tang-show-keys** command to display the key hash for comparison. In this example, the Tang server is running on the port 7500:

```
# tang-show-keys 7500
fM-EwYeiTxS66X3s1UAYwsGKGnxnpIl8ig0KOQmr9CM
```

- 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:

```
# dnf 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		<div>+</div>
Passphrase	Slot 0	<div><div></div><div>-</div></div>
Keyserver localhost:7500	Slot 1	<div><div></div><div>-</div></div>

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](#)

11.6. 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":"sha256","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-256 bank:

```
$ clevis encrypt tpm2 '{"pcr_bank":"sha256","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
...
```

11.7. 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:

```
# dnf 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:

```
_Oslk0T-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.

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

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

```
# dnf install clevis-dracut
```

In RHEL, 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](#)

11.8. 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:

```
# dnf 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":"sha256","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":"sha256","key":"rsa","pcr_bank":"sha256","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:

```
# dnf 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":"sha256","key":"rsa"}
```

Additional resources

- **clevis-luks-bind(1)**, **clevis-encrypt-tpm2(1)**, and **dracut.cmdline(7)** man pages

11.9. 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

11.10. 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 the 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-compliant 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
clevis-luks
```

```
clevis-systemd
%end
```

- 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.
- Call **clevis luks bind** to perform binding in the **%post** section. Afterward, remove the temporary password:

```
%post
clevis luks bind -y -k - -d /dev/vda2 \
tang '{"url":"http://tang.srv"}' <<< "temppass"
cryptsetup luksRemoveKey /dev/vda2 <<< "temppass"
dracut -fv --regenerate-all
%end
```

If your configuration relies on a Tang pin that requires network during early boot or you use NBDE clients with static IP configurations, you have to modify the **dracut** command as described in [Configuring manual enrollment of LUKS-encrypted volumes](#).

Note that the **-y** option for the **clevis luks bind** command is available from RHEL 8.3. In RHEL 8.2 and older, replace **-y** by **-f** in the **clevis luks bind** command and download the advertisement from the Tang server:

```
%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"
dracut -fv --regenerate-all
%end
```



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 9 using Kickstart](#)

11.11. 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:

```
# dnf 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

11.12. 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.

11.12.1. 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.

11.12.1.1. 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.

11.12.1.2. 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

11.13. 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 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 your scenario requires having encrypted root volumes in a cloud, perform the installation process (usually using Kickstart) for each instance of Red Hat Enterprise Linux in the cloud as well. The images cannot be shared without also sharing a LUKS master key.

To deploy automated unlocking in a virtualized environment, 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.

Additional resources

- **clevis-luks-bind(1)** man page

11.14. BUILDING AUTOMATICALLY-ENROLLABLE VM IMAGES FOR CLOUD ENVIRONMENTS USING NBDE

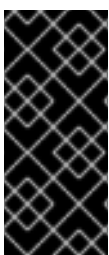
Deploying automatically-enrollable encrypted images in a cloud environment can provide a unique set of challenges. Like other virtualization environments, it is recommended to reduce the number of instances started from a single image to avoid sharing the LUKS master key.

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.

11.15. DEPLOYING TANG AS A CONTAINER

The **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 **tang** container image from the **registry.redhat.io** registry:

```
# podman pull registry.redhat.io/rhel9/tang
```

2. Run the container, specify its port, and specify the path to the Tang keys. The previous example runs the **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/rhel9/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/rhel9/tang tangd-rotate-keys -
v -d /var/db/tang
Rotated key 'rZAMKAseaXBe0rcKXL1hCClq-DY.jwk' -> '.rZAMKAseaXBe0rcKXL1hCClq-
DY.jwk'
Rotated key 'x1Alpc6WmnCU-CabD8_4q18vDuw.jwk' -> '.x1Alpc6WmnCU-
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:
```

```
x1Alpc6WmnCU-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

11.16. INTRODUCTION TO THE **NBDE_CLIENT** AND **NBDE_SERVER** SYSTEM ROLES (CLEVIS AND TANG)

RHEL System Roles is a collection of Ansible roles and modules that provide a consistent configuration interface to remotely manage multiple RHEL systems.

You can use 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 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_client** System 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](#)

11.17. 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

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **`nbde_server`** System Role.
- Access and permissions to a control node, which is a system from which Red Hat Ansible Core configures other systems.
On the control node:
 - The **`ansible-core`** and **`rhel-system-roles`** packages are installed.



IMPORTANT

RHEL 8.0–8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **`ansible`**, **`ansible-playbook`**, connectors such as **`docker`** and **`podman`**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the [How to download and install Red Hat Ansible Engine](#) Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **`ansible-core`** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#) Knowledgebase article.

- An inventory file which lists the managed nodes.

Procedure

1. 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
```

2. Edit the playbook in a text editor of your choice, for example:

```
# vi my-tang-playbook.yml
```

3. Add the required parameters. The following example playbook ensures deploying of your Tang server and a key rotation:

```
---
- hosts: all

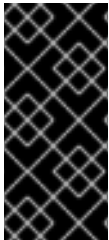
  vars:
    nbde_server_rotate_keys: yes

  roles:
    - rhel-system-roles.nbde_server
```

4. Apply the finished playbook:

```
# ansible-playbook -i inventory-file my-tang-playbook.yml
```

Where: * **inventory-file** is the inventory file. * **logging-playbook.yml** is the playbook you use.



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.

11.18. 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

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **nbde_client** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
- The Ansible Core package is installed on the control machine.
- The **rhel-system-roles** package is installed on the system from which you want to run the playbook.

Procedure

1. 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
```

2. Edit the playbook in a text editor of your choice, for example:

```
# vi my-clevis-playbook.yml
```

3. 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:
  - rhel-system-roles.nbde_client
```

4. 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 System 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.

CHAPTER 12. 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.

12.1. 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 **nftables**, **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.

12.2. 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 9, 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.

12.3. CONFIGURING AUDITD FOR A SECURE ENVIRONMENT

The default **auditd** configuration should be suitable for most environments. However, if your environment must 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. The **max_log_file** parameter specifies the maximum file size in megabytes. The value given must be numeric.

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. If the value of **space_left** is specified as a whole number, it is interpreted as an absolute size in megabytes (MiB). If the value is specified as a number between 1 and 99 followed by a percentage sign (for example, 5%), the Audit daemon calculates the absolute size in megabytes based on the size of the file system containing **log_file**.

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. The numeric value for this parameter should be lower than the number for **space_left**. You can also append a percent sign (for example, 1%) to the number to have the audit daemon calculate the number based on the disk partition size.

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.

12.4. STARTING AND CONTROLLING AUDITD

After **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
```

You can temporarily disable **auditd** with the **# auditctl -e 0** command and re-enable it with **# auditctl -e 1**.

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 **audit** value is properly recorded. You can use the **systemctl** command only for two actions: **enable** and **status**.

12.5. 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 audit=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.

audit=1000

The **audit** 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.

oid=0

The **oid** 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**.

12.6. 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

- **auditctl(8)** man page.

12.7. 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
```

12.8. 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.

12.9. 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.

12.10. 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](#) .

12.11. SETTING UP AUDIT TO MONITOR SOFTWARE UPDATES

You can use the pre-configured rule **44-installers.rules** to configure Audit to monitor the following utilities that install software:

- **dnf** [2]
- **yum**
- **pip**
- **npm**
- **cpan**
- **gem**
- **luarocks**

To monitor the **rpm** utility, install the **rpm-plugin-audit** package. Audit will then generate **SOFTWARE_UPDATE** events when it installs or updates a package. You can list these events by entering **ausearch -m SOFTWARE_UPDATE** on the command line.



NOTE

Pre-configured rule files cannot be used on systems with the **ppc64le** and **aarch64** architectures.

Prerequisites

- **auditd** is configured in accordance with the settings provided in [Configuring auditd for a secure environment](#).

Procedure

1. Copy the pre-configured rule file **44-installers.rules** from the **/usr/share/audit/sample-rules/** directory to the **/etc/audit/rules.d/** directory:

```
# cp /usr/share/audit/sample-rules/44-installers.rules /etc/audit/rules.d/
```

2. Load the audit rules:

```
# augenrules --load
```

Verification

1. List the loaded rules:

```
# auditctl -l
-p x-w /usr/bin/dnf-3 -k software-installer
-p x-w /usr/bin/yum -k software-installer
-p x-w /usr/bin/pip -k software-installer
-p x-w /usr/bin/npm -k software-installer
-p x-w /usr/bin/cpan -k software-installer
-p x-w /usr/bin/gem -k software-installer
-p x-w /usr/bin/luarocks -k software-installer
```

2. Perform an installation, for example:

```
# dnf reinstall -y vim-enhanced
```

3. Search the Audit log for recent installation events, for example:

```
# ausearch -ts recent -k software-installer
time->Thu Dec 16 10:33:46 2021
type=PROCTITLE msg=audit(1639668826.074:298):
proctitle=2F7573722F6C6962657865632F706C61746666F726D2D707974686F6E002F75737
22F62696E2F646E66007265696E7374616C6C002D790076696D2D656E68616E636564
type=PATH msg=audit(1639668826.074:298): item=2 name="/lib64/ld-linux-x86-64.so.2"
inode=10092 dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
obj=system_u:object_r:ld_so_t:s0 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0
```



```
cap_fver=0 cap_frootid=0
type=PATH msg=audit(1639668826.074:298): item=1 name="/usr/libexec/platform-python"
inode=4618433 dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00
obj=system_u:object_r:bin_t:s0 nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0
cap_fver=0 cap_frootid=0
type=PATH msg=audit(1639668826.074:298): item=0 name="/usr/bin/dnf" inode=6886099
dev=fd:01 mode=0100755 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:rpm_exec_t:s0
nametype=NORMAL cap_fp=0 cap_fi=0 cap_fe=0 cap_fver=0 cap_frootid=0
type=CWD msg=audit(1639668826.074:298): cwd="/root"
type=EXECVE msg=audit(1639668826.074:298): argc=5 a0="/usr/libexec/platform-python"
a1="/usr/bin/dnf" a2="reinstall" a3="-y" a4="vim-enhanced"
type=SYSCALL msg=audit(1639668826.074:298): arch=c000003e syscall=59 success=yes
exit=0 a0=55c437f22b20 a1=55c437f2c9d0 a2=55c437f2aeb0 a3=8 items=3 ppid=5256
pid=5375 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0 ses=3
comm="dnf" exe="/usr/libexec/platform-python3.6"
subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="software-installer"
```

12.12. MONITORING USER LOGIN TIMES WITH AUDIT

To monitor which users logged in at specific times, you do not need to configure Audit in any special way. You can use the **ausearch** or **aureport** tools, which provide different ways of presenting the same information.

Prerequisites

- **auditd** is configured in accordance with the settings provided in [Configuring auditd for a secure environment](#).

Procedure

To display user log in times, use any one of the following commands:

- Search the audit log for the **USER_LOGIN** message type:

```
# ausearch -m USER_LOGIN -ts '12/02/2020' '18:00:00' -sv no
time->Mon Nov 22 07:33:22 2021
type=USER_LOGIN msg=audit(1637584402.416:92): pid=1939 uid=0 auid=4294967295
ses=4294967295 subj=system_u:system_r:sshd_t:s0-s0:c0.c1023 msg='op=login acct="
(unknown)" exe="/usr/sbin/sshd" hostname=? addr=10.37.128.108 terminal=ssh res=failed'
```

- You can specify the date and time with the **-ts** option. If you do not use this option, **ausearch** provides results from today, and if you omit time, **ausearch** provides results from midnight.
- You can use the **-sv yes** option to filter out successful login attempts and **-sv no** for unsuccessful login attempts.
- Pipe the raw output of the **ausearch** command into the **aualast** utility, which displays the output in a format similar to the output of the **last** command. For example:

```
# ausearch --raw | aualast --stdin
root  ssh      10.37.128.108  Mon Nov 22 07:33 - 07:33 (00:00)
root  ssh      10.37.128.108  Mon Nov 22 07:33 - 07:33 (00:00)
root  ssh      10.22.16.106   Mon Nov 22 07:40 - 07:40 (00:00)
reboot system boot 4.18.0-348.6.el8 Mon Nov 22 07:33
```

- Display the list of login events by using the **aureport** command with the **--login -i** options.

```
# aureport --login -i

Login Report
=====
# date time auid host term exe success event
=====
1. 11/16/2021 13:11:30 root 10.40.192.190 ssh /usr/sbin/sshd yes 6920
2. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6925
3. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6930
4. 11/16/2021 13:11:31 root 10.40.192.190 ssh /usr/sbin/sshd yes 6935
5. 11/16/2021 13:11:33 root 10.40.192.190 ssh /usr/sbin/sshd yes 6940
6. 11/16/2021 13:11:33 root 10.40.192.190 /dev/pts/0 /usr/sbin/sshd yes 6945
```

Additional resources

- The **ausearch(8)** man page.
- The **aulast(8)** man page.
- The **aureport(8)** man page.

12.13. ADDITIONAL RESOURCES

- 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.

[2] Because **dnf** is a symlink in RHEL, the path in the **dnf** Audit rule must include the target of the symlink. To receive correct Audit events, modify the **44-installers.rules** file by changing the **path=/usr/bin/dnf** path to **/usr/bin/dnf-3**.

CHAPTER 13. 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.

13.1. 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
- **fagenrules** script

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 DNF 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 **/etc/fapolicyd/fapolicyd.trust** file contains a list of trusted files. You can also use multiple trust files in the **/etc/fapolicyd/trust.d/** directory.
- The **/etc/fapolicyd/rules.d/** directory for files containing **allow** and **deny** execution rules. The **fagenrules** script merges these component rules files to the **/etc/fapolicyd/compiled.rules** file.
- The **fapolicyd.conf** file contains the daemon's configuration options. This file is useful primarily for performance-tuning purposes.

Rules in **/etc/fapolicyd/rules.d/** are organized in several files, each representing a different policy goal. The numbers at the beginning of the corresponding file names determine the order in **/etc/fapolicyd/compiled.rules**:

- 10 - language rules
- 20 - Dracut-related Rules
- 21 - rules for updaters

- 30 - patterns
- 40 - ELF rules
- 41 - shared objects rules
- 42 - trusted ELF rules
- 70 - trusted language rules
- 72 - shell rules
- 90 - deny execute rules
- 95 - allow open rules

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)**, **fapolicyd.conf(5)**, **fapolicyd.trust(13)**, **fagenrules(8)**, and **fapolicyd-cli(1)** man pages.
- The [Enhancing security with the kernel integrity subsystem](#) chapter in the [Managing, monitoring, and updating the kernel](#) document.
- The documentation installed with the **fapolicyd** package in the **/usr/share/doc/fapolicyd/** directory and the **/usr/share/fapolicyd/sample-rules/README-rules** file.

13.2. DEPLOYING FAPOLICYD

To deploy the **fapolicyd** framework in RHEL:

Procedure

1. Install the **fapolicyd** package:

```
# dnf 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
```

13.3. MARKING FILES AS TRUSTED USING AN ADDITIONAL SOURCE OF TRUST

The **fapolicyd** framework trusts files contained in the RPM database. You can mark additional files as trusted by adding the corresponding entries to the **/etc/fapolicyd/fapolicyd.trust** plain-text file or the **/etc/fapolicyd/trust.d/** directory, which supports separating a list of trusted files into more files. You can modify **fapolicyd.trust** or the files in **/etc/fapolicyd/trust.d** either directly using a text editor or through **fapolicyd-cli** commands.



NOTE

Marking files as trusted using **fapolicyd.trust** or **trust.d/** is better than writing custom **fapolicyd** rules due to performance reasons.

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, and store the corresponding entry to the **myapp** file in **/etc/fapolicyd/trust.d/**:

```
# fapolicyd-cli --file add /tmp/ls --trust-file myapp
```

- If you skip the **--trust-file** option, then the previous command adds the corresponding line to **/etc/fapolicyd/fapolicyd.trust**.
- To mark all existing files in a directory as trusted, provide the directory path as an argument of the **--file** option, for example: **fapolicyd-cli --file add /tmp/my_bin_dir/ --trust-file myapp**.

3. Update the **fapolicyd** database:

```
# fapolicyd-cli --update
```

NOTE

Changing the content of a trusted file or directory changes their checksum, and therefore **fapolicyd** no longer considers them trusted.

To make the new content trusted again, refresh the file trust database by using the **fapolicyd-cli --file update** command. If you do not provide any argument, the entire database refreshes. Alternatively, you can specify a path to a specific file or directory. Then, update the database by using **fapolicyd-cli --update**.

Verification

1. Check that your custom binary can be now executed, for example:

```
$ /tmp/ls
ls
```

Additional resources

- **fapolicyd.trust(13)** man page.

13.4. 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 **dnf** or **rpm** installers, you must either mark additional files as trusted or add new custom rules.

For basic scenarios, prefer [Marking files as trusted using an additional source of trust](#) . In more advanced scenarios such as allowing to execute a custom binary only for specific user and group identifiers, add new custom rules to the **/etc/fapolicyd/rules.d/** directory.

The following steps demonstrate adding a new rule to allow a custom binary.

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. In this case, you can limit the output only to access denials by using the **--debug-deny** option instead of **--debug**:

```
# fapolicyd --debug-deny 2> fapolicy.output &
[1] 51341
```

Alternatively, you can run **fapolicyd** debug mode in another terminal.

4. Repeat the command that **fapolicyd** denied:

```
$ /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 2> fapolicy.output
^C
...
```

Alternatively, kill the process of **fapolicyd** debug mode:

```
# kill 51341
```

6. Find a rule that denies the execution of your application:

```
# cat fapolicy.output | grep 'deny_audit'
...
rule=13 dec=deny_audit perm=execute auid=0 pid=6855 exe=/usr/bin/bash : path=/tmp/ls
ftype=application/x-executable trust=0
```

7. Locate the file that contains a rule that prevented the execution of your custom binary. In this case, the **deny_audit perm=execute** rule belongs to the **90-deny-execute.rules** file:

```
# ls /etc/fapolicyd/rules.d/
10-languages.rules 40-bad-elf.rules 72-shell.rules
20-dracut.rules 41-shared-obj.rules 90-deny-execute.rules
21-updaters.rules 42-trusted-elf.rules 95-allow-open.rules
30-patterns.rules 70-trusted-lang.rules
```

```
# cat /etc/fapolicyd/rules.d/90-deny-execute.rules
# Deny execution for anything untrusted

deny_audit perm=execute all : all
```

8. Add a new **allow** rule to the file that lexically *precedes* the rule file that contains the rule that denied the execution of your custom binary in the **/etc/fapolicyd/rules.d/** directory:

```
# touch /etc/fapolicyd/rules.d/80-myapps.rules
# vi /etc/fapolicyd/rules.d/80-myapps.rules
```

Insert the following rule to the **80-myapps.rules** file:

```
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 to the rule file in **/etc/fapolicyd/rules.d/**:

```
allow perm=execute exe=/usr/bin/bash trust=1 : dir=/tmp/ trust=0
```

9. 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
```

10. Check that the list of compiled differs from the rule set in **/etc/fapolicyd/rules.d/**, and update the list, which is stored in the **/etc/fapolicyd/compiled.rules** file:

```
# fagenrules --check
/usr/sbin/fagenrules: Rules have changed and should be updated
# fagenrules --load
```

11. Check that your custom rule is in the list of **fapolicyd** rules before the rule that prevented the execution:

```
# fapolicyd-cli --list
...
13. allow perm=execute exe=/usr/bin/bash trust=1 : path=/tmp/ls ftype=application/x-
executable trust=0
14. deny_audit perm=execute all : all
...
```

12. 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.rules(5)** and **fapolicyd-cli(1)** man pages.
- The documentation installed with the **fapolicyd** package in the **/usr/share/fapolicyd/sample-rules/README-rules** file.

13.5. 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
```

13.6. 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
```

fapolicyd-cli checks and listings

- The **--check-config**, **--check-watch_fs**, and **--check-trustdb** options help you find syntax errors, not-yet-watched file systems, and file mismatches, for example:

```
# fapolicyd-cli --check-config
Daemon config is OK

# fapolicyd-cli --check-trustdb
/etc/selinux/targeted/contexts/files/file_contexts mismatches: size sha256
/etc/selinux/targeted/policy/policy.31 mismatches: size sha256
```

- Use the **--list** option to check the current list of rules and their order:

```
# fapolicyd-cli --list
...
9. allow perm=execute all : trust=1
10. allow perm=open all : ftype=%languages trust=1
```

```
11. deny_audit perm=any all : ftype=%languages
12. allow perm=any all : ftype=text/x-shellscript
13. deny_audit perm=execute all : all
...
```

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
```

Alternatively, to limit the output only to entries when **fapolicyd** denies access, use the **--debug-deny** option:

```
# fapolicyd --debug-deny
```

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.

13.7. ADDITIONAL RESOURCES

- **fapolicyd**-related man pages listed by using the **man -k fapolicyd** command.
- The [FOSDEM 2020 fapolicyd](#) presentation.

CHAPTER 14. 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**.

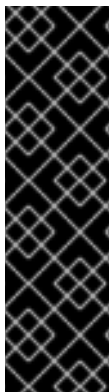
14.1. USBGUARD

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 **IPCAuthorizedUsers** and **IPCAuthorizedGroups** 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.

14.2. INSTALLING USBGUARD

Use this procedure to install and initiate the **USBGuard** framework.

Procedure

1. Install the **usbguard** package:

```
# dnf 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.

14.3. 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
   "4PHGcaDKWtPjKDwYpIRG722cB9SIGz9l9lea93+Gt9c=" via-port "usb1" with-interface
   09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
   "CrXgiaWlf2bZAU+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.

14.4. 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
"4PHGcaDKWtPjKDwYpIRG722cB9SIgz9l9lea93+Gt9c=" via-port "usb1" with-interface
09:00:00
...
6: block id 1b1c:1ab1 serial "000024937962" name "Voyager" hash
"CrXgiaWlf2bZAU+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.

14.5. 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.

14.6. 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/WSCocgiuQIVs6Dn0VEza7KiHoDeTz0fyg=" 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.

14.7. 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:

```
IPCAAllowGroups=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 and modify the current policy:

```
# usbguard add-user joesec --devices ALL --policy modify,list --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.

14.8. 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.

14.9. ADDITIONAL RESOURCES

- **usbguard(1)**, **usbguard-rules.conf(5)**, **usbguard-daemon(8)**, and **usbguard-daemon.conf(5)** man pages.
- [USBGuard Homepage](#).

CHAPTER 15. CONFIGURING A REMOTE LOGGING SOLUTION

To ensure that logs from various machines in your environment are recorded centrally on a logging server, you can configure the **Rsyslog** application to record logs that fit specific criteria from the client system to the server.

15.1. THE RSYSLOG LOGGING SERVICE

The Rsyslog application, in combination with the **systemd-journald** service, provides local and remote logging support in Red Hat Enterprise Linux. The **rsyslogd** daemon continuously reads **syslog** messages received by the **systemd-journald** service from the Journal. **rsyslogd** then filters and processes these **syslog** events and records them to **rsyslog** log files or forwards them to other services according to its configuration.

The **rsyslogd** daemon also provides extended filtering, encryption protected relaying of messages, input and output modules, and support for transportation using the TCP and UDP protocols.

In **/etc/rsyslog.conf**, which is the main configuration file for **rsyslog**, you can specify the rules according to which **rsyslogd** handles the messages. Generally, you can classify messages by their source and topic (facility) and urgency (priority), and then assign an action that should be performed when a message fits these criteria.

In **/etc/rsyslog.conf**, you can also see a list of log files maintained by **rsyslogd**. Most log files are located in the **/var/log/** directory. Some applications, such as **httpd** and **samba**, store their log files in a subdirectory within **/var/log/**.

Additional resources

- The **rsyslogd(8)** and **rsyslog.conf(5)** man pages.
- Documentation installed with the **rsyslog-doc** package in the **/usr/share/doc/rsyslog/html/index.html** file.

15.2. INSTALLING RSYSLOG DOCUMENTATION

The Rsyslog application has extensive online documentation that is available at <https://www.rsyslog.com/doc/>, but you can also install the **rsyslog-doc** documentation package locally.

Prerequisites

- You have activated the **AppStream** repository on your system.
- You are authorized to install new packages using **sudo**.

Procedure

- Install the **rsyslog-doc** package:

```
# dnf install rsyslog-doc
```

Verification

- Open the **/usr/share/doc/rsyslog/html/index.html** file in a browser of your choice, for example:

■

```
$ firefox /usr/share/doc/rsyslog/html/index.html &
```

15.3. CONFIGURING A SERVER FOR REMOTE LOGGING OVER TCP

The Rsyslog application enables you to both run a logging server and configure individual systems to send their log files to the logging server. To use remote logging through TCP, configure both the server and the client. The server collects and analyzes the logs sent by one or more client systems.

With the Rsyslog application, you can maintain a centralized logging system where log messages are forwarded to a server over the network. To avoid message loss when the server is not available, you can configure an action queue for the forwarding action. This way, messages that failed to be sent are stored locally until the server is reachable again. Note that such queues cannot be configured for connections using the UDP protocol.

The **omfwd** plug-in provides forwarding over UDP or TCP. The default protocol is UDP. Because the plug-in is built in, it does not have to be loaded.

By default, **rsyslog** uses TCP on port **514**.

Prerequisites

- Rsyslog is installed on the server system.
- You are logged in as **root** on the server.
- The **policyscoreutils-python-utils** package is installed for the optional step using the **semanage** command.
- The **firewalld** service is running.

Procedure

1. Optional: To use a different port for **rsyslog** traffic, add the **syslogd_port_t** SELinux type to port. For example, enable port **30514**:

```
# semanage port -a -t syslogd_port_t -p tcp 30514
```

2. Optional: To use a different port for **rsyslog** traffic, configure **firewalld** to allow incoming **rsyslog** traffic on that port. For example, allow TCP traffic on port **30514**:

```
# firewall-cmd --zone=<zone-name> --permanent --add-port=30514/tcp
success
# firewall-cmd --reload
```

3. Create a new file in the **/etc/rsyslog.d/** directory named, for example, **remotelog.conf**, and insert the following content:

```
# Define templates before the rules that use them
# Per-Host templates for remote systems
template(name="TmplAuthpriv" type="list") {
    constant(value="/var/log/remote/auth/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
}
```

```

    constant(value=".log")
  }

  template(name="TplMsg" type="list") {
    constant(value="/var/log/remote/msg/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
  }

  # Provides TCP syslog reception
  module(load="imtcp")

  # Adding this ruleset to process remote messages
  ruleset(name="remote1"){
    authpriv.* action(type="omfile" DynaFile="TplAuthpriv")
    *.info;mail.none;authpriv.none;cron.none
    action(type="omfile" DynaFile="TplMsg")
  }

  input(type="imtcp" port="30514" ruleset="remote1")

```

4. Save the changes to the **/etc/rsyslog.d/remotelog.conf** file.

5. Test the syntax of the **/etc/rsyslog.conf** file:

```

# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run...
rsyslogd: End of config validation run. Bye.

```

6. Make sure the **rsyslog** service is running and enabled on the logging server:

```
# systemctl status rsyslog
```

7. Restart the **rsyslog** service.

```
# systemctl restart rsyslog
```

8. Optional: If **rsyslog** is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

Your log server is now configured to receive and store log files from the other systems in your environment.

Additional resources

- **rsyslogd(8)**, **rsyslog.conf(5)**, **semanage(8)**, and **firewall-cmd(1)** man pages.
- Documentation installed with the **rsyslog-doc** package in the **/usr/share/doc/rsyslog/html/index.html** file.

15.4. CONFIGURING REMOTE LOGGING TO A SERVER OVER TCP

Follow this procedure to configure a system for forwarding log messages to a server over the TCP protocol. The **omfwd** plug-in provides forwarding over UDP or TCP. The default protocol is UDP. Because the plug-in is built in, you do not have to load it.

Prerequisites

- The **rsyslog** package is installed on the client systems that should report to the server.
- You have configured the server for remote logging.
- The specified port is permitted in SELinux and open in firewall.
- The system contains the **polycoreutils-python-utils** package, which provides the **semanage** command for adding a non-standard port to the SELinux configuration.

Procedure

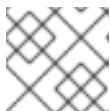
1. Create a new file in the **/etc/rsyslog.d/** directory named, for example, **10-remotelog.conf**, and insert the following content:

```
*.* action(type="omfwd"
        queue.type="linkedlist"
        queue.filename="example_fwd"
        action.resumeRetryCount="-1"
        queue.saveOnShutdown="on"
        target="example.com" port="30514" protocol="tcp"
    )
```

Where:

- **queue.type="linkedlist"** enables a LinkedList in-memory queue,
- **queue.filename** defines a disk storage. The backup files are created with the **example_fwd** prefix in the working directory specified by the preceding global **workDirectory** directive,
- the **action.resumeRetryCount -1** setting prevents **rsyslog** from dropping messages when retrying to connect if server is not responding,
- enabled **queue.saveOnShutdown="on"** saves in-memory data if **rsyslog** shuts down,
- the last line forwards all received messages to the logging server, port specification is optional.

With this configuration, **rsyslog** sends messages to the server but keeps messages in memory if the remote server is not reachable. A file on disk is created only if **rsyslog** runs out of the configured memory queue space or needs to shut down, which benefits the system performance.



NOTE

Rsyslog processes configuration files **/etc/rsyslog.d/** in the lexical order.

2. Restart the **rsyslog** service.


```
# systemctl restart rsyslog
```

Verification

To verify that the client system sends messages to the server, follow these steps:

1. On the client system, send a test message:

```
# logger test
```

2. On the server system, view the `/var/log/messages` log, for example:

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

Where *hostname* is the host name of the client system. Note that the log contains the user name of the user that entered the **logger** command, in this case **root**.

Additional resources

- **rsyslogd(8)** and **rsyslog.conf(5)** man pages.
- Documentation installed with the **rsyslog-doc** package in the `/usr/share/doc/rsyslog/html/index.html` file.

15.5. CONFIGURING TLS-ENCRYPTED REMOTE LOGGING

By default, Rsyslog sends remote-logging communication in the plain text format. If your scenario requires to secure this communication channel, you can encrypt it using TLS.

To use encrypted transport through TLS, configure both the server and the client. The server collects and analyzes the logs sent by one or more client systems.

You can use either the **ossli** network stream driver (OpenSSL) or the **gtls** stream driver (GnuTLS).



NOTE

If you have a separate system with higher security, for example, a system that is not connected to any network or has stricter authorizations, use the separate system as the certifying authority (CA).

Prerequisites

- You have **root** access to both the client and server systems.
- The **rsyslog** and **rsyslog-openssl** packages are installed on the server and the client systems.
- If you use the **gtls** network stream driver, install the **rsyslog-gnutls** package instead of **rsyslog-openssl**.
- If you generate certificates using the **certtool** command, install the **gnutls-utils** package.
- On your logging server, the following certificates are in the `/etc/pki/ca-trust/source/anchors/` directory and your system configuration is updated by using the **update-ca-trust** command:

- **ca-cert.pem** - a CA certificate that can verify keys and certificates on logging servers and clients.
- **server-cert.pem** - a public key of the logging server.
- **server-key.pem** - a private key of the logging server.
- On your logging clients, the following certificates are in the **/etc/pki/ca-trust/source/anchors/** directory and your system configuration is updated by using **update-ca-trust**:
 - **ca-cert.pem** - a CA certificate that can verify keys and certificates on logging servers and clients.
 - **client-cert.pem** - a public key of a client.
 - **client-key.pem** - a private key of a client.

Procedure

1. Configure the server for receiving encrypted logs from your client systems:
 - a. Create a new file in the **/etc/rsyslog.d/** directory named, for example, **securelogser.conf**.
 - b. To encrypt the communication, the configuration file must contain paths to certificate files on your server, a selected authentication method, and a stream driver that supports TLS encryption. Add the following lines to the **/etc/rsyslog.d/securelogser.conf** file:

```
# Set certificate files
global(
    DefaultNetstreamDriverCAFile="/etc/pki/ca-trust/source/anchors/ca-cert.pem"
    DefaultNetstreamDriverCertFile="/etc/pki/ca-trust/source/anchors/server-cert.pem"
    DefaultNetstreamDriverKeyFile="/etc/pki/ca-trust/source/anchors/server-key.pem"
)

# TCP listener
module(
    load="imtcp"
    PermittedPeer=["client1.example.com", "client2.example.com"]
    StreamDriver.AuthMode="x509/name"
    StreamDriver.Mode="1"
    StreamDriver.Name="openssl"
)

# Start up listener at port 514
input(
    type="imtcp"
    port="514"
)
```



NOTE

If you prefer the GnuTLS driver, use the **StreamDriver.Name="gtls"** configuration option. See the documentation installed with the **rsyslog-doc** package for more information about less strict authentication modes than **x509/name**.

- c. Save the changes to the **/etc/rsyslog.d/securelogser.conf** file.
- d. Verify the syntax of the **/etc/rsyslog.conf** file and any files in the **/etc/rsyslog.d/** directory:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run (level 1)...
rsyslogd: End of config validation run. Bye.
```

- e. Make sure the **rsyslog** service is running and enabled on the logging server:

```
# systemctl status rsyslog
```

- f. Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

- g. Optional: If Rsyslog is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

2. Configure clients for sending encrypted logs to the server:

- a. On a client system, create a new file in the **/etc/rsyslog.d/** directory named, for example, **securelogcli.conf**.
- b. Add the following lines to the **/etc/rsyslog.d/securelogcli.conf** file:

```
# Set certificate files
global(
    DefaultNetstreamDriverCAFile="/etc/pki/ca-trust/source/anchors/ca-cert.pem"
    DefaultNetstreamDriverCertFile="/etc/pki/ca-trust/source/anchors/client-cert.pem"
    DefaultNetstreamDriverKeyFile="/etc/pki/ca-trust/source/anchors/client-key.pem"
)

# Set up the action for all messages
*. * action(
    type="omfwd"
    StreamDriver="ossf"
    StreamDriverMode="1"
    StreamDriverPermittedPeers="server.example.com"
    StreamDriverAuthMode="x509/name"
    target="server.example.com" port="514" protocol="tcp"
)
```



NOTE

If you prefer the GnuTLS driver, use the **StreamDriver.Name="gtls"** configuration option.

- c. Save the changes to the **/etc/rsyslog.d/securelogser.conf** file.

- d. Verify the syntax of the `/etc/rsyslog.conf` file and other files in the `/etc/rsyslog.d/` directory:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run (level 1)...
rsyslogd: End of config validation run. Bye.
```

- e. Make sure the **rsyslog** service is running and enabled on the logging server:

```
# systemctl status rsyslog
```

- f. Restart the **rsyslog** service:

```
# systemctl restart rsyslog
```

- g. Optional: If Rsyslog is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

Verification

To verify that the client system sends messages to the server, follow these steps:

1. On the client system, send a test message:

```
# logger test
```

2. On the server system, view the `/var/log/messages` log, for example:

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

Where ***hostname*** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- The **certtool(1)**, **openssl(1)**, **update-ca-trust(8)**, **rsyslogd(8)**, and **rsyslog.conf(5)** man pages.
- Documentation installed with the **rsyslog-doc** package at `/usr/share/doc/rsyslog/html/index.html`.
- The [Using the Logging System Role](#) article.

15.6. CONFIGURING A SERVER FOR RECEIVING REMOTE LOGGING INFORMATION OVER UDP

The **Rsyslog** application enables you to configure a system to receive logging information from remote systems. To use remote logging through UDP, configure both the server and the client. The receiving server collects and analyzes the logs sent by one or more client systems. By default, **rsyslog** uses UDP on port **514** to receive log information from remote systems.

Follow this procedure to configure a server for collecting and analyzing logs sent by one or more client systems over the UDP protocol.

Prerequisites

- Rsyslog is installed on the server system.
- You are logged in as **root** on the server.
- The **polycoreutils-python-utils** package is installed for the optional step using the **semanage** command.
- The **firewalld** service is running.

Procedure

1. Optional: To use a different port for **rsyslog** traffic than the default port **514**:
 - a. Add the **syslogd_port_t** SELinux type to the SELinux policy configuration, replacing **portno** with the port number you want **rsyslog** to use:

```
# semanage port -a -t syslogd_port_t -p udp portno
```

- b. Configure **firewalld** to allow incoming **rsyslog** traffic, replacing **portno** with the port number and **zone** with the zone you want **rsyslog** to use:

```
# firewall-cmd --zone=zone --permanent --add-port=portno/udp
success
# firewall-cmd --reload
```

- c. Reload the firewall rules:

```
# firewall-cmd --reload
```

2. Create a new **.conf** file in the **/etc/rsyslog.d/** directory, for example, **remotelogserv.conf**, and insert the following content:

```
# Define templates before the rules that use them
# Per-Host templates for remote systems
template(name="TmplAuthpriv" type="list") {
    constant(value="/var/log/remote/auth/")
    property(name="hostname")
    constant(value="")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

template(name="TmplMsg" type="list") {
    constant(value="/var/log/remote/msg/")
    property(name="hostname")
    constant(value="")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}
```

```
# Provides UDP syslog reception
module(load="imudp")

# This ruleset processes remote messages
ruleset(name="remote1"){
    authpriv.* action(type="omfile" DynaFile="TmplAuthpriv")
    *.info;mail.none;authpriv.none;cron.none
    action(type="omfile" DynaFile="TmplMsg")
}

input(type="imudp" port="514" ruleset="remote1")
```

Where **514** is the port number **rsyslog** uses by default. You can specify a different port instead.

3. Verify the syntax of the **/etc/rsyslog.conf** file and all **.conf** files in the **/etc/rsyslog.d/** directory:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-2.el8, config validation run...
```

4. Restart the **rsyslog** service.

```
# systemctl restart rsyslog
```

5. Optional: If **rsyslog** is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

Additional resources

- **rsyslogd(8)**, **rsyslog.conf(5)**, **semanage(8)**, and **firewall-cmd(1)** man pages.
- Documentation installed with the **rsyslog-doc** package in the **/usr/share/doc/rsyslog/html/index.html** file.

15.7. CONFIGURING REMOTE LOGGING TO A SERVER OVER UDP

Follow this procedure to configure a system for forwarding log messages to a server over the UDP protocol. The **omfwd** plug-in provides forwarding over UDP or TCP. The default protocol is UDP. Because the plug-in is built in, you do not have to load it.

Prerequisites

- The **rsyslog** package is installed on the client systems that should report to the server.
- You have configured the server for remote logging as described in [Configuring a server for receiving remote logging information over UDP](#).

Procedure

1. Create a new **.conf** file in the **/etc/rsyslog.d/** directory, for example, **10-remotelogcli.conf**, and insert the following content:

```
*.* action(type="omfwd"
    queue.type="linkedlist")
```

```

queue.filename="example_fwd"
action.resumeRetryCount="-1"
queue.saveOnShutdown="on"
target="example.com" port="portno" protocol="udp"
)

```

Where:

- **queue.type="linkedlist"** enables a LinkedList in-memory queue.
- **queue.filename** defines a disk storage. The backup files are created with the **example_fwd** prefix in the working directory specified by the preceding global **workDirectory** directive.
- The **action.resumeRetryCount -1** setting prevents **rsyslog** from dropping messages when retrying to connect if the server is not responding.
- **enabled queue.saveOnShutdown="on"** saves in-memory data if **rsyslog** shuts down.
- **portno** is the port number you want **rsyslog** to use. The default value is **514**.
- The last line forwards all received messages to the logging server, port specification is optional.
With this configuration, **rsyslog** sends messages to the server but keeps messages in memory if the remote server is not reachable. A file on disk is created only if **rsyslog** runs out of the configured memory queue space or needs to shut down, which benefits the system performance.



NOTE

Rsyslog processes configuration files **/etc/rsyslog.d/** in the lexical order.

2. Restart the **rsyslog** service.

```
# systemctl restart rsyslog
```

3. Optional: If **rsyslog** is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

Verification

To verify that the client system sends messages to the server, follow these steps:

1. On the client system, send a test message:

```
# logger test
```

2. On the server system, view the **/var/log/remote/msg/hostname/root.log** log, for example:

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

Where **hostname** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- **rsyslogd(8)** and **rsyslog.conf(5)** man pages.
- Documentation installed with the **rsyslog-doc** package at `/usr/share/doc/rsyslog/html/index.html`.

15.8. LOAD BALANCING HELPER IN RSYSLOG

The **RebindInterval** setting specifies an interval at which the current connection is broken and is re-established. This setting applies to TCP, UDP, and RELP traffic. The load balancers perceive it as a new connection and forward the messages to another physical target system.

The **RebindInterval** setting proves to be helpful in scenarios when a target system has changed its IP address. The Rsyslog application caches the IP address when the connection establishes, therefore, the messages are sent to the same server. If the IP address changes, the UDP packets will be lost until the Rsyslog service restarts. Re-establishing the connection will ensure the IP to be resolved by DNS again.

```
action(type="omfwd" protocol="tcp" RebindInterval="250" target="example.com" port="514" ...)
```

```
action(type="omfwd" protocol="udp" RebindInterval="250" target="example.com" port="514" ...)
```

```
action(type="omrelp" RebindInterval="250" target="example.com" port="6514" ...)
```

15.9. CONFIGURING RELIABLE REMOTE LOGGING

With the Reliable Event Logging Protocol (RELP), you can send and receive **syslog** messages over TCP with a much reduced risk of message loss. RELP provides reliable delivery of event messages, which makes it useful in environments where message loss is not acceptable. To use RELP, configure the **imrelp** input module, which runs on the server and receives the logs, and the **omrelp** output module, which runs on the client and sends logs to the logging server.

Prerequisites

- You have installed the **rsyslog**, **librelp**, and **rsyslog-relp** packages on the server and the client systems.
- The specified port is permitted in SELinux and open in the firewall.

Procedure

1. Configure the client system for reliable remote logging:
 - a. On the client system, create a new **.conf** file in the `/etc/rsyslog.d/` directory named, for example, **relpclient.conf**, and insert the following content:

```
module(load="omrelp")
*. * action(type="omrelp" target="_target_IP_" port="_target_port_")
```

Where:

- **target_IP** is the IP address of the logging server.
- **target_port** is the port of the logging server.

b. Save the changes to the `/etc/rsyslog.d/relpclient.conf` file.

c. Restart the **rsyslog** service.

```
# systemctl restart rsyslog
```

d. Optional: If **rsyslog** is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

2. Configure the server system for reliable remote logging:

a. On the server system, create a new **.conf** file in the `/etc/rsyslog.d/` directory named, for example, **relpserv.conf**, and insert the following content:

```
ruleset(name="relp"){
    *.* action(type="omfile" file="_log_path_")
}

module(load="imrelp")
input(type="imrelp" port="_target_port_" ruleset="relp")
```

Where:

- **log_path** specifies the path for storing messages.
- **target_port** is the port of the logging server. Use the same value as in the client configuration file.

b. Save the changes to the `/etc/rsyslog.d/relpserv.conf` file.

c. Restart the **rsyslog** service.

```
# systemctl restart rsyslog
```

d. Optional: If **rsyslog** is not enabled, ensure the **rsyslog** service starts automatically after reboot:

```
# systemctl enable rsyslog
```

Verification

To verify that the client system sends messages to the server, follow these steps:

1. On the client system, send a test message:

```
# logger test
```

2. On the server system, view the log at the specified **log_path**, for example:

```
# cat /var/log/remote/msg/hostname/root.log
Feb 25 03:53:17 hostname root[6064]: test
```

Where **hostname** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- **rsyslogd(8)** and **rsyslog.conf(5)** man pages.
- Documentation installed with the **rsyslog-doc** package in the **/usr/share/doc/rsyslog/html/index.html** file.

15.10. SUPPORTED RSYSLOG MODULES

To expand the functionality of the Rsyslog application, you can use specific modules. Modules provide additional inputs (Input Modules), outputs (Output Modules), and other functionalities. A module can also provide additional configuration directives that become available after you load the module.

You can list the input and output modules installed on your system by entering the following command:

```
# ls /usr/lib64/rsyslog/{i,o}m*
```

You can view the list of all available **rsyslog** modules in the **/usr/share/doc/rsyslog/html/configuration/modules/idx_output.html** file after you install the **rsyslog-doc** package.

15.11. CONFIGURING THE NETCONSOLE SERVICE TO LOG KERNEL MESSAGES TO A REMOTE HOST

When logging to disk or using a serial console is not possible, you can use the **netconsole** kernel module and the same-named service to log kernel messages over a network to a remote **rsyslog** service.

Prerequisites

- A system log service, such as **rsyslog** is installed on the remote host.
- The remote system log service is configured to receive incoming log entries from this host.

Procedure

1. Install the **netconsole-service** package:

```
# dnf install netconsole-service
```

2. Edit the **/etc/sysconfig/netconsole** file and set the **SYSLOGADDR** parameter to the IP address of the remote host:

```
# SYSLOGADDR=192.0.2.1
```

3. Enable and start the **netconsole** service:

```
# systemctl enable --now netconsole
```

Verification steps

- Display the **/var/log/messages** file on the remote system log server.

Additional resources

- [Configuring a remote logging solution](#)

15.12. ADDITIONAL RESOURCES

- Documentation installed with the **rsyslog-doc** package in the **/usr/share/doc/rsyslog/html/index.html** file
- **rsyslog.conf(5)** and **rsyslogd(8)** man pages
- [Configuring system logging without journald or with minimized journald usage](#) Knowledgebase article
- [Negative effects of the RHEL default logging setup on performance and their mitigations](#) Knowledgebase article
- The [Using the Logging System Role](#) chapter

CHAPTER 16. USING THE `logging` SYSTEM ROLE

As a system administrator, you can use the **logging** System Role to configure a RHEL host as a logging server to collect logs from many client systems.

16.1. THE `logging` SYSTEM ROLE

With the **logging** System Role, you can deploy logging configurations on local and remote hosts.

To apply a **logging** System Role on one or more systems, you define the logging configuration in a *playbook*. A *playbook* is a list of one or more plays. Playbooks are human-readable, and they are written in the YAML format. For more information about playbooks, see [Working with playbooks](#) in Ansible documentation.

The set of systems that you want to configure according to the *playbook* is defined in an *inventory file*. For more information on creating and using inventories, see [How to build your inventory](#) in Ansible documentation.

Logging solutions provide multiple ways of reading logs and multiple logging outputs.

For example, a logging system can receive the following inputs:

- local files,
- **systemd/journal**,
- another logging system over the network.

In addition, a logging system can have the following outputs:

- logs stored in the local files in the **/var/log** directory,
- logs sent to Elasticsearch,
- logs forwarded to another logging system.

With the **logging** System Role, you can combine the inputs and outputs to fit your scenario. For example, you can configure a logging solution that stores inputs from **journal** in a local file, whereas inputs read from files are both forwarded to another logging system and stored in the local log files.

16.2. `logging` SYSTEM ROLE PARAMETERS

In a **logging** System Role *playbook*, you define the inputs in the **logging_inputs** parameter, outputs in the **logging_outputs** parameter, and the relationships between the inputs and outputs in the **logging_flows** parameter. The **logging** System Role processes these variables with additional options to configure the logging system. You can also enable encryption.



NOTE

Currently, the only available logging system in the **logging** System Role is **Rsyslog**.

- **logging_inputs**: List of inputs for the logging solution.

- **name**: Unique name of the input. Used in the **logging_flows**: inputs list and a part of the generated **config** file name.
- **type**: Type of the input element. The type specifies a task type which corresponds to a directory name in **roles/rsyslog/{tasks,vars}/inputs/**.
 - **basics**: Inputs configuring inputs from **systemd** journal or **unix** socket.
 - **kernel_message**: Load **imklog** if set to **true**. Default to **false**.
 - **use_imuxsock**: Use **imuxsock** instead of **imjournal**. Default to **false**.
 - **ratelimit_burst**: Maximum number of messages that can be emitted within **ratelimit_interval**. Default to **20000** if **use_imuxsock** is false. Default to **200** if **use_imuxsock** is true.
 - **ratelimit_interval**: Interval to evaluate **ratelimit_burst**. Default to 600 seconds if **use_imuxsock** is false. Default to 0 if **use_imuxsock** is true. 0 indicates rate limiting is turned off.
 - **persist_state_interval**: Journal state is persisted every **value** messages. Default to **10**. Effective only when **use_imuxsock** is false.
 - **files**: Inputs configuring inputs from local files.
 - **remote**: Inputs configuring inputs from the other logging system over network.
- **state**: State of the configuration file. **present** or **absent**. Default to **present**.
- **logging_outputs**: List of outputs for the logging solution.
 - **files**: Outputs configuring outputs to local files.
 - **forwards**: Outputs configuring outputs to another logging system.
 - **remote_files**: Outputs configuring outputs from another logging system to local files.
- **logging_flows**: List of flows that define relationships between **logging_inputs** and **logging_outputs**. The **logging_flows** variable has the following keys:
 - **name**: Unique name of the flow
 - **inputs**: List of **logging_inputs** name values
 - **outputs**: List of **logging_outputs** name values.

Additional resources

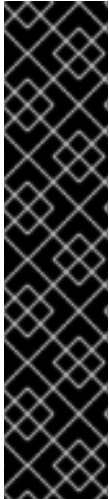
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**

16.3. APPLYING A LOCAL LOGGING SYSTEM ROLE

Prepare and apply an Ansible playbook to configure a logging solution on a set of separate machines. Each machine records logs locally.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
On the control node:
 - The **ansible-core** and **rhel-system-roles** packages are installed.



IMPORTANT

RHEL 8.0–8.5 provided access to a separate Ansible repository that contains Ansible Engine 2.9 for automation based on Ansible. Ansible Engine contains command-line utilities such as **ansible**, **ansible-playbook**, connectors such as **docker** and **podman**, and many plugins and modules. For information on how to obtain and install Ansible Engine, see the [How to download and install Red Hat Ansible Engine](#) Knowledgebase article.

RHEL 8.6 and 9.0 have introduced Ansible Core (provided as the **ansible-core** package), which contains the Ansible command-line utilities, commands, and a small set of built-in Ansible plugins. RHEL provides this package through the AppStream repository, and it has a limited scope of support. For more information, see the [Scope of support for the Ansible Core package included in the RHEL 9 and RHEL 8.6 and later AppStream repositories](#) Knowledgebase article.

- An inventory file which lists the managed nodes.



NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

1. Create a playbook that defines the required role:
 - a. Create a new YAML file and open it in a text editor, for example:

```
# vi logging-playbook.yml
```

- b. Insert the following content:

```
---
- name: Deploying basics input and implicit files output
  hosts: all
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: system_input
        type: basics
    logging_outputs:
      - name: files_output
        type: files
    logging_flows:
```

```
- name: flow1
  inputs: [system_input]
  outputs: [files_output]
```

2. Run the playbook on a specific inventory:

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

Where:

- **inventory-file** is the inventory file.
- **logging-playbook.yml** is the playbook you use.

Verification

1. Test the syntax of the **/etc/rsyslog.conf** file:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run...
rsyslogd: End of config validation run. Bye.
```

2. Verify that the system sends messages to the log:

- a. Send a test message:

```
# logger test
```

- b. View the **/var/log/messages** log, for example:

```
# cat /var/log/messages
Aug  5 13:48:31 hostname root[6778]: test
```

Where `hostname` is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

16.4. FILTERING LOGS IN A LOCAL LOGGING SYSTEM ROLE

You can deploy a logging solution which filters the logs based on the **rsyslog** property-based filter.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.

On the control node:

- Red Hat Ansible Core is installed
- The **rhel-system-roles** package is installed
- An inventory file which lists the managed nodes.



NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

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

```
---
- name: Deploying files input and configured files output
  hosts: all
  roles:
    - linux-system-roles.logging
  vars:
    logging_inputs:
      - name: files_input
        type: basics
    logging_outputs:
      - name: files_output0
        type: files
        property: msg
        property_op: contains
        property_value: error
        path: /var/log/errors.log
      - name: files_output1
        type: files
        property: msg
        property_op: "!contains"
        property_value: error
        path: /var/log/others.log
    logging_flows:
      - name: flow0
        inputs: [files_input]
        outputs: [files_output0, files_output1]
```

Using this configuration, all messages that contain the **error** string are logged in **/var/log/errors.log**, and all other messages are logged in **/var/log/others.log**.

You can replace the **error** property value with the string by which you want to filter.

You can modify the variables according to your preferences.

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
```

Verification

1. Test the syntax of the **/etc/rsyslog.conf** file:


```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run...
rsyslogd: End of config validation run. Bye.
```

2. Verify that the system sends messages that contain the **error** string to the log:

- a. Send a test message:

```
# logger error
```

- b. View the **/var/log/errors.log** log, for example:

```
# cat /var/log/errors.log
Aug  5 13:48:31 hostname root[6778]: error
```

Where **hostname** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**

16.5. APPLYING A REMOTE LOGGING SOLUTION USING THE LOGGING SYSTEM ROLE

Follow these steps to prepare and apply a Red Hat Ansible Core playbook to configure a remote logging solution. In this playbook, one or more clients take logs from **systemd-journal** and forward them to a remote server. The server receives remote input from **remote_rsyslog** and **remote_files** and outputs the logs to local files in directories named by remote host names.

Prerequisites

- Access and permissions to one or more *managed nodes*, which are systems you want to configure with the **logging** System Role.
- Access and permissions to a *control node*, which is a system from which Red Hat Ansible Core configures other systems.
On the control node:
 - The **ansible-core** and **rhel-system-roles** packages are installed.
 - An inventory file which lists the managed nodes.



NOTE

You do not have to have the **rsyslog** package installed, because the System Role installs **rsyslog** when deployed.

Procedure

1. Create a playbook that defines the required role:

- a. Create a new YAML file and open it in a text editor, for example:

```
# vi logging-playbook.yml
```

- b. Insert the following content into the file:

```
---
- name: Deploying remote input and remote_files output
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: remote_udp_input
        type: remote
        udp_ports: [ 601 ]
      - name: remote_tcp_input
        type: remote
        tcp_ports: [ 601 ]
    logging_outputs:
      - name: remote_files_output
        type: remote_files
    logging_flows:
      - name: flow_0
        inputs: [remote_udp_input, remote_tcp_input]
        outputs: [remote_files_output]

- name: Deploying basics input and forwards output
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
    logging_outputs:
      - name: forward_output0
        type: forwards
        severity: info
        target: _host1.example.com_
        udp_port: 601
      - name: forward_output1
        type: forwards
        facility: mail
        target: _host1.example.com_
        tcp_port: 601
    logging_flows:
      - name: flows0
        inputs: [basic_input]
        outputs: [forward_output0, forward_output1]

[basic_input]
[forward_output0, forward_output1]
```

Where **host1.example.com** is the logging server.

**NOTE**

You can modify the parameters in the playbook to fit your needs.

**WARNING**

The logging solution works only with the ports defined in the SELinux policy of the server or client system and open in the firewall. The default SELinux policy includes ports 601, 514, 6514, 10514, and 20514. To use a different port, [modify the SELinux policy on the client and server systems](#). Configuring the firewall through System Roles is not yet supported.

2. Create an inventory file that lists your servers and clients:

a. Create a new file and open it in a text editor, for example:

```
# vi inventory.ini
```

b. Insert the following content into the inventory file:

```
[servers]
server ansible_host=host1.example.com
[clients]
client ansible_host=host2.example.com
```

Where:

- **host1.example.com** is the logging server.
- **host2.example.com** is the logging client.

3. Run the playbook on your inventory.

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

Where:

- **inventory.ini** is the inventory file.
- **logging-playbook.yml** is the playbook you created.

Verification

1. On both the client and the server system, test the syntax of the **/etc/rsyslog.conf** file:

```
# rsyslogd -N 1
rsyslogd: version 8.1911.0-6.el8, config validation run (level 1), master config
/etc/rsyslog.conf
rsyslogd: End of config validation run. Bye.
```

2. Verify that the client system sends messages to the server:

a. On the client system, send a test message:

```
# logger test
```

b. On the server system, view the `/var/log/messages` log, for example:

```
# cat /var/log/messages
Aug  5 13:48:31 host2.example.com root[6778]: test
```

Where **host2.example.com** is the host name of the client system. Note that the log contains the user name of the user that entered the logger command, in this case **root**.

Additional resources

- [Preparing a control node and managed nodes to use RHEL System Roles](#)
- Documentation installed with the **rhel-system-roles** package in `/usr/share/ansible/roles/rhel-system-roles.logging/README.html`
- [RHEL System Roles](#) KB article

16.6. USING THE LOGGING SYSTEM ROLE WITH TLS

Transport Layer Security (TLS) is a cryptographic protocol designed to securely communicate over the computer network.

As an administrator, you can use the **logging** RHEL System Role to configure secure transfer of logs using Red Hat Ansible Automation Platform.

16.6.1. Configuring client logging with TLS

You can use the **logging** System Role to configure logging in RHEL systems that are logged on a local machine and can transfer logs to the remote logging system with TLS by running an Ansible playbook.

This procedure configures TLS on all hosts in the clients group in the Ansible inventory. The TLS protocol encrypts the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure TLS.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

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

```
---
- name: Deploying files input and forwards output with certs
  hosts: clients
```

```

roles:
  - rhel-system-roles.logging
vars:
  logging_pki_files:
    - ca_cert_src: /local/path/to/ca_cert.pem
      cert_src: /local/path/to/cert.pem
      private_key_src: /local/path/to/key.pem
  logging_inputs:
    - name: input_name
      type: files
      input_log_path: /var/log/containers/*.log
  logging_outputs:
    - name: output_name
      type: forwards
      target: your_target_host
      tcp_port: 514
      tls: true
      pki_authmode: x509/name
      permitted_server: 'server.example.com'
  logging_flows:
    - name: flow_name
      inputs: [input_name]
      outputs: [output_name]

```

The playbook uses the following parameters:

logging_pki_files

Using this parameter you can configure TLS and has to pass **ca_cert_src**, **cert_src**, and **private_key_src** parameters.

ca_cert

Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.

cert

Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.

private_key

Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.

ca_cert_src

Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.

cert_src

Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.

private_key_src

Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.

tls

Using this parameter ensures secure transfer of logs over the network. If you do not want a secure wrapper, you can set **tls: true**.

2. Verify playbook syntax:

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

3. Run the playbook on your inventory file:

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

16.6.2. Configuring server logging with TLS

You can use the **logging** System Role to configure logging in RHEL systems as a server and can receive logs from the remote logging system with TLS by running an Ansible playbook.

This procedure configures TLS on all hosts in the server group in the Ansible inventory.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure TLS.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

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

```
---
- name: Deploying remote input and remote_files output with certs
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_pki_files:
      - ca_cert_src: /local/path/to/ca_cert.pem
        cert_src: /local/path/to/cert.pem
        private_key_src: /local/path/to/key.pem
    logging_inputs:
      - name: input_name
        type: remote
        tcp_ports: 514
        tls: true
        permitted_clients: ['clients.example.com']
    logging_outputs:
      - name: output_name
        type: remote_files
        remote_log_path: /var/log/remote/%FROMHOST%/%PROGRAMNAME:::secpath-
replace%.log
        async_writing: true
        client_count: 20
        io_buffer_size: 8192
    logging_flows:
```

```
- name: flow_name
  inputs: [input_name]
  outputs: [output_name]
```

The playbook uses the following parameters:

logging_pki_files

Using this parameter you can configure TLS and has to pass **ca_cert_src**, **cert_src**, and **private_key_src** parameters.

ca_cert

Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.

cert

Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.

private_key

Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.

ca_cert_src

Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.

cert_src

Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.

private_key_src

Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.

tls

Using this parameter ensures secure transfer of logs over the network. If you do not want a secure wrapper, you can set **tls: true**.

2. Verify playbook syntax:

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

3. Run the playbook on your inventory file:

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

16.7. USING THE LOGGING SYSTEM ROLES WITH RELP

Reliable Event Logging Protocol (RELP) is a networking protocol for data and message logging over the TCP network. It ensures reliable delivery of event messages and you can use it in environments that do not tolerate any message loss.

The RELP sender transfers log entries in form of commands and the receiver acknowledges them once they are processed. To ensure consistency, RELP stores the transaction number to each transferred command for any kind of message recovery.

You can consider a remote logging system in between the RELP Client and RELP Server. The RELP Client transfers the logs to the remote logging system and the RELP Server receives all the logs sent by the remote logging system.

Administrators can use the **logging** System Role to configure the logging system to reliably send and receive log entries.

16.7.1. Configuring client logging with RELP

You can use the **logging** System Role to configure logging in RHEL systems that are logged on a local machine and can transfer logs to the remote logging system with RELP by running an Ansible playbook.

This procedure configures RELP on all hosts in the **clients** group in the Ansible inventory. The RELP configuration uses Transport Layer Security (TLS) to encrypt the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure RELP.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

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

```
---
- name: Deploying basic input and relp output
  hosts: clients
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: basic_input
        type: basics
    logging_outputs:
      - name: relp_client
        type: relp
        target: _logging.server.com_
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/client-cert.pem_
        private_key: _/etc/pki/tls/private/client-key.pem_
        pki_authmode: name
        permitted_servers:
          - '*.server.example.com'
    logging_flows:
      - name: _example_flow_
        inputs: [basic_input]
        outputs: [relp_client]
```


The playbooks uses following settings:

- **target:** This is a required parameter that specifies the host name where the remote logging system is running.
- **port:** Port number the remote logging system is listening.
- **tls:** Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the **tls** variable to **false**. By default **tls** parameter is set to true while working with RELP and requires key/certificates and triplets **{ca_cert, cert, private_key}** and/or **{ca_cert_src, cert_src, private_key_src}**.
 - If **{ca_cert_src, cert_src, private_key_src}** triplet is set, the default locations **/etc/pki/tls/certs** and **/etc/pki/tls/private** are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet
 - If **{ca_cert, cert, private_key}** triplet is set, files are expected to be on the default path before the logging configuration.
 - If both the triplets are set, files are transferred from local path from control node to specific path of the managed node.
- **ca_cert:** Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.
- **cert:** Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.
- **private_key:** Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.
- **ca_cert_src:** Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.
- **cert_src:** Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.
- **private_key_src:** Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.
- **pki_authmode:** Accepts the authentication mode as **name** or **fingerprint**.
- **permitted_servers:** List of servers that will be allowed by the logging client to connect and send logs over TLS.
- **inputs:** List of logging input dictionary.
- **outputs:** List of logging output dictionary.

2. Optional: Verify playbook syntax.

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

3. Run the playbook:

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

16.7.2. Configuring server logging with RELP

You can use the **logging** System Role to configure logging in RHEL systems as a server and can receive logs from the remote logging system with RELP by running an Ansible playbook.

This procedure configures RELP on all hosts in the **server** group in the Ansible inventory. The RELP configuration uses TLS to encrypt the message transmission for secure transfer of logs over the network.

Prerequisites

- You have permissions to run playbooks on managed nodes on which you want to configure RELP.
- The managed nodes are listed in the inventory file on the control node.
- The **ansible** and **rhel-system-roles** packages are installed on the control node.

Procedure

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

```
---
- name: Deploying remote input and remote_files output
  hosts: server
  roles:
    - rhel-system-roles.logging
  vars:
    logging_inputs:
      - name: relp_server
        type: relp
        port: 20514
        tls: true
        ca_cert: _/etc/pki/tls/certs/ca.pem_
        cert: _/etc/pki/tls/certs/server-cert.pem_
        private_key: _/etc/pki/tls/private/server-key.pem_
        pki_authmode: name
        permitted_clients:
          - '_*example.client.com_'
    logging_outputs:
      - name: _remote_files_output_
        type: _remote_files_
    logging_flows:
      - name: _example_flow_
        inputs: _relp_server_
        outputs: _remote_files_output_
```

The playbooks uses following settings:

- **port**: Port number the remote logging system is listening.
- **tls**: Ensures secure transfer of logs over the network. If you do not want a secure wrapper you can set the **tls** variable to **false**. By default **tls** parameter is set to true while working with RELP and requires key/certificates and triplets {**ca_cert**, **cert**, **private_key**} and/or {**ca_cert_src**, **cert_src**, **private_key_src**}.

- If **{ca_cert_src, cert_src, private_key_src}** triplet is set, the default locations **/etc/pki/tls/certs** and **/etc/pki/tls/private** are used as the destination on the managed node to transfer files from control node. In this case, the file names are identical to the original ones in the triplet
- If **{ca_cert, cert, private_key}** triplet is set, files are expected to be on the default path before the logging configuration.
- If both the triplets are set, files are transferred from local path from control node to specific path of the managed node.
- **ca_cert**: Represents the path to CA certificate. Default path is **/etc/pki/tls/certs/ca.pem** and the file name is set by the user.
- **cert**: Represents the path to cert. Default path is **/etc/pki/tls/certs/server-cert.pem** and the file name is set by the user.
- **private_key**: Represents the path to private key. Default path is **/etc/pki/tls/private/server-key.pem** and the file name is set by the user.
- **ca_cert_src**: Represents local CA cert file path which is copied to the target host. If **ca_cert** is specified, it is copied to the location.
- **cert_src**: Represents the local cert file path which is copied to the target host. If **cert** is specified, it is copied to the location.
- **private_key_src**: Represents the local key file path which is copied to the target host. If **private_key** is specified, it is copied to the location.
- **pki_authmode**: Accepts the authentication mode as **name** or **fingerprint**.
- **permitted_clients**: List of clients that will be allowed by the logging server to connect and send logs over TLS.
- **inputs**: List of logging input dictionary.
- **outputs**: List of logging output dictionary.

2. Optional: Verify playbook syntax.

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

3. Run the playbook:

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

16.8. ADDITIONAL RESOURCES

- [Preparing a control node and managed nodes to use RHEL System Roles](#)
- Documentation installed with the **rhel-system-roles** package in **/usr/share/ansible/roles/rhel-system-roles.logging/README.html**.
- [RHEL System Roles](#)
- **ansible-playbook(1)** man page.

