Linux Academy

Red Hat Certified Specialist in Security (EX415)

Study Guide

Bob Salmans

Robert.salmans@linuxacademy.com

June 24, 2019

Contents

Section 1: Security Auditing and Automation 1

System Auditing 1

The Linux Audit System 1

Installing the Audit Package 1

Configuring the Audit Service and Options 1

Starting the Audit Service 2

Defining Audit Rules and Controls 2

Defining File System Rules 2

Defining System Call Rules 3

Creating Audit Reports 5

OpenSCAP (Security Content Automation Protocol) 5

Security Compliance with OpenSCAP 5

Using SCAP Workbench 5

Using the OpenSCAP Command Line Utility (`oscap`) 6

Ansible 7

What Is Ansible? 7

YAML 7

Components of Ansible 8

Inventories 8

Modules 8

Playbooks 9

Configuration Files 9

Installing Ansible 10

Creating an `ansible` User to Run Ansible and Connect to Managed Nodes 10

Setting Up an SSH Key Pair 10

Copying an SSH Key to Managed Nodes 10
Setting Up sudo Permissions for the ansible User 10
Running Ad-Hoc (One-Line) Ansible Commands 11
Some Popular Modules and Their Required Arguments 11
Running Playbooks 12
Creating Playbooks (YAML) 12
Section 2: Intrusion Detection 13
AIDE (Advanced Intrusion Detection Environment) 13
What Does AIDE Do? 13
Installing and Configuring AIDE 14
Testing AIDE Functionality 15
Tuning AIDE 15
Section 3: System Hardening 16
Securing SSH 16
Controlling SSH Access to a Host 17
Generating and Distributing SSH Keys 18
Disable the Use of Passwords (Force the Use of SSH Keys) 18
Set the SSH Version 18
Managing sudo 18
Controlling Sudo Access 18
Locking Down sudo 19
Example sudo Scenarios 19
USBGuard 20
What Is USBGuard? 20
How to Install and Configure USBGuard 20
Installation 20

Working with USBGuard 21
Creating Whitelists in USBGuard 21
Rule Options 21
Rule Operators 22
Rule Conditions 22
Negation 22
Rule Language 23
Writing Rules 23
Pluggable Authentication Module (PAM) 24
What Is PAM? 24
Installing PAM 24
Creating a PAM Failed Login (Account Lockout) Policy 24
Creating a PAM Password Complexity Policy 25
Create Rules and Enable Password Complexity Requirements 25
Keep a Password History 26
Controlling `sudo` Access 26
Edit the `/etc/sudoers` File 26
Locking Down `sudo` 27
Section 4: Disk Encryption 27
Storage on Red Hat Linux Hosts 27
Linux Unified Key Setup (LUKS) 27
The Encryption Process 28
Unmount and Close Encrypted Volumes Securely 30
Open and Mount Existing LUKS-Encrypted Volumes 30
Adding a Backup Key to a LUKS-Encrypted Volume 31
Creating a Backup of LUKS Encryption Headers 31

Backing Up a LUKS Encryption Header . . . 31
Restoring a LUKS Encrypted Header . . . 31
Changing a Passphrase on a LUKS-Encrypted Volume . . . 31
Network-Bound Disk Encryption (NBDE) . . . 32
Introduction to NBDE . . . 32
Setting Up NBDE . . . 32
Setting Auto-Decrypt at Boot with `dracut` . . . 33
NBDE Maintenance . . . 33
Section 5: SELinux . . . 34
Introduction to SELinux . . . 34
Modes (/etc/sysconfig/selinux) . . . 34
Policies (/etc/sysconfig/selinux) . . . 34
Enabling SELinux on a Host for the First Time . . . 35
Labeling (Types) . . . 35
Type Enforcement . . . 36
Managing Labels . . . 37
Booleans . . . 38
Working with Booleans . . . 38
Troubleshooting SELinux . . . 38
Additional Troubleshooting Tips . . . 39
Confining Users . . . 40
Using Booleans with SELinux Confined Users . . . 41

Section 1: Security Auditing and Automation

System Auditing

The Linux Audit System

1. The Linux Audit system tracks security events.
2. 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.
3. Auditing is required for security-related certifications such as PCI-DSS, FISMA, and STIG.
4. Use cases for auditing include:
    - Monitoring file access
    - Monitoring system calls (tracking changes to a system)
    - Recording commands run by a user
    - Monitoring network access (IPtables)
5. The Audit system compares system call activity to audit rules to see if the event should be logged.
6. There are three audit utilities:
    - `audisp` — Provides a plug-in mechanism for real-time analytical programs to access audit events.
    - `auditctl` — Interacts with the kernel Audit component to control settings and parameters.
    - `aureport` — Generates reports of audit events.

Installing the Audit Package

```
yum install audit
```

Configuring the Audit Service and Options

1. Edit the config file `/etc/audit/auditd.conf`.
    - **log_file**: Where to store the audit log entries.
    - **max_log_file** (MB): Defines the maximum size of a single audit log file.
    - **num_logs**: Number of log files to keep.
    - **max_log_file_action**: Options are `keep_logs` or `rotate`.
    - **space_left**: Amount of free space left on the disk with audit logs before the `space_left_action` is triggered.
    - **space_left_action**: Action taken when `space_left` amount is realized on the disk containing the audit logs. Options are `email` or `exec`.

- **action_mail_acct**: Email account to send to if using email as the `space_left_action`.

Scenario 1: You want to limit logs to 100 MB of disk space.
Solution: Simply set `max_log_file` to 10 and `num_logs` to 10, which would give you 10 MB * 10 = 100 MB of space. Or you could set `max_log_file` to 20 and `num_logs` to 5, and you'd still be restricting logs to 100 MB of disk space.

Scenario 2: You need to set up low disk space email alerts for your logging volume.
Solution: Use `space_left` to define the alert threshold, then set the `space_left_action` to `email`. Lastly, define the email account to send alerts to with the `action_mail_account` variable.

Starting the Audit Service

1. Start the Audit service:
   - `service auditd start` (Manually start service)
   - `systemctl enable auditd` (Start service at boot time)
   - `service auditd rotate` (Rotate the log files in `/var/log/audit/` by starting a new log file)

Defining Audit Rules and Controls

1. The Audit system uses a set of rules to identify what should be logged.
2. Audit rules can be edited from the command line with the `auditctl` utility (rules are ***not*** persistent across reboots when using the `auditctl` utility).
3. Defining control rules:
   - **-b** — Sets the maximum amount of existing Audit buffers in the kernel
   - **-f** — Sets the action to be taken when a critical error is detected
   - **-e** — Enables and disables the Audit system, or locks its configuration
   - **-r** — Sets the rate of generated messages per second (0 = no rate limit)
   - **-s** — Reports the status of the Audit system
   - **-l** — Lists currently loaded Audit rules
   - **-D** — Deletes all currently loaded Audit rules

Scenario: You want to limit the Audit buffers to 8192.
Solution: `auditctl -b 8192`

Defining File System Rules

```
auditctl -w path_to_file -p permissions -k key_name
```

- `path_to_file` — File or directory to be audited

- `permissions` — Permissions that are logged
  - `r` — Read access to the file or directory
  - `w` — Write access to the file or directory
  - `x` — Execute access to the file or directory
  - `a` — Change in the file's or directory's attributes
- `key_name` — Optional string to help you identify which rule generated a log entry

Example:

1. Log write access and attribute changes to `/etc/passwd`.

   ```
   auditctl -w /etc/passwd -p wa -k passwd_changes
   ```

2. Log write access and attribute changes to SELinux.

   ```
   auditctl -w /etc/selinux/ -p wa -k selinux_changes
   ```

Defining System Call Rules

```
auditctl -a action,filter -S system_call -F field=value -k key_name
```

- **action, filter**: Specify when something is logged
  - **action**: Always or never
  - **filter**: Which kernel rule-matching filter is applied (task, exit, user, exclude)
- **system_call**: Specifies the system call by its name (list located in `/usr/include/asm/unistd_64.h`)
- **field**: Value used to modify the rule to match events based on a specified architecture, group ID, or process ID
- **key_name**: Optional string to help you identify what rule generated a log entry

Example:

- Log every time the `adjtimex` or `settimeofday` system calls are used.

  ```
  auditctl -a always,exit -F arch=b64 -S adjtimex -S settimeofday -k time_change
  ```

Read Rules from a File into `auditctl`

```
auditctl -R /usr/share/doc/audit-version/stig.rules
```

- Allows reading rules into action, but they are not persistent across reboots.

Creating Persistent Rules

- Rules must be entered into `/etc/audit/rules.d/audit.rules`.
- Uses the same rule format as `auditctl` (minus the word “auditctl”).

Example:

1. Delete all previous rules.

   ```
   -D
   ```

2. Set the buffer size.

   ```
   -b 8192
   ```

3. Make the configuration immutable (reboot is required to change audit rules).

   ```
   -e 2
   ```

4. Panic when a failure occurs.

   ```
   -f 2
   ```

5. Generate at most 100 audit messages per second.

   ```
   -r 100
   ```

Preconfigured Rule Sets

- In the `/usr/share/doc/audit-version/` directory, there is a set of preconfigured rule files based on various certification standards.
  - nispom.rules
  - capp.rules
  - lspp.rules
  - stig.rules *
- To put these rules into effect, make a copy of `/etc/audit/rules.d/audit.rules` and copy a rules file to `/etc/audit/rules.d/audit.rules`.

  ```
  cp /etc/audit/rules.d/audit.rules /etc/audit/rules.d/audit.rules_backup
  cp /usr/share/doc/audit-version/rules/stig.rules /etc/audit/rules.d/audit.rules
  ```

Creating Audit Reports

- The `aureport` utility is used to generate summary and columnar reports.
- `aureport` queries all files in `/var/log/audit` to build the report.
- Creating reports:
  - By date: `aureport --start 04/08/2013 00:00:00 --end 04/11/2013 00:00:00`
  - Exec events: `aureport -x`
  - Exec events summary: `aureport -x --summary`
  - Summary of failed events for all users: `aureport -u --failed --summary -i`
  - Failed login attempts per system user: `aureport --login --summary -i`
  - Search for all file access events for user `500` from today: `ausearch --start today --loginuid 500 --raw | aureport -f --summary`

OpenSCAP (Security Content Automation Protocol)

Security Compliance with OpenSCAP

1. OpenSCAP is used to perform automated compliance audits.
   - **Compliance Audit:** Checking whether a host follows the rules of a security policy.
     - Is a firewall installed and enabled?
     - Is FTP disabled?
     - Etc.
2. Components of OpenSCAP on Red Hat:
   - **SCAP Workbench**: Graphical utility used to perform configurations and vulnerability scans on local or remote systems and generate reports.
   - **OpenSCAP**: The `oscap` command line utility is used to perform configurations and vulnerability scans on a local system, validate compliance, and generate reports.
   - **Script Check Enginer (SCE)**: Used to write security content using Bash, Python, or Ruby.
   - **SCAP Security Guide (SSG)**: Provides collections of security policies for Linux systems based on regulatory and government requirements.

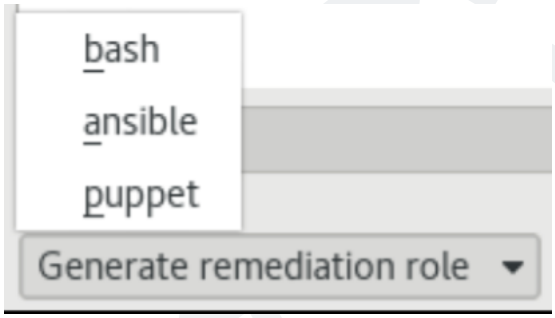
Using SCAP Workbench

1. Install SCAP Workbench.

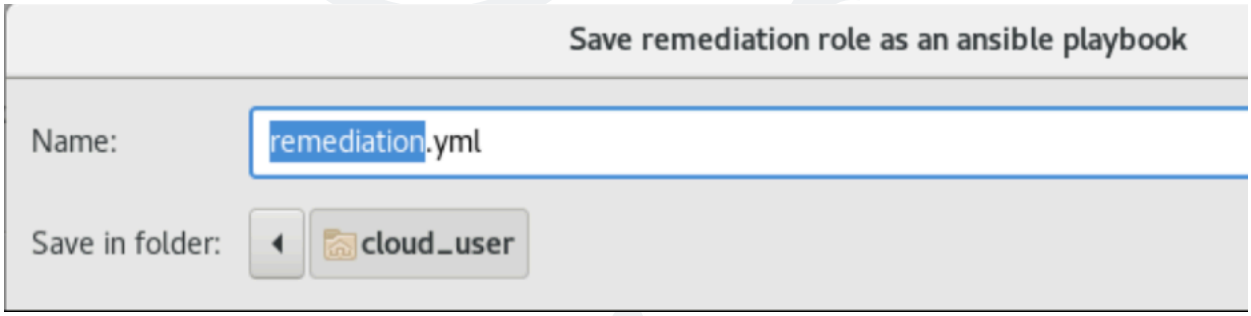
```
yum install -y scap-workbench
```

2. Go to **Applications** > **System Tools** > **SCAP Workbench** to launch the application.

3. Next to *Select content to load*, choose **RHEL 7**, and click the **Load Content** button.
4. Now you can select a profile from the *Profile* dropdown and scan a target.
5. You can also click the **Customize** button next to *Profiles* to make a custom profile.
6. Once you've made a custom profile, click on **File**, then **Save customization only** to save your custom profile.
7. To scan a target, simply select **Local Machine** or **Remote Machine** and then click the **Scan** button at the bottom.
8. Once the scan is complete, you can create a report by clicking on the **Save Results** button and selecting an option.
9. In order to create an Ansible playbook to remediate the findings, click on the **Generate Remediation Role** button, and select **Ansible**.



10. Then provide a name for the remediation role.



Using the OpenSCAP Command Line Utility (`oscap`)

1. `oscap` is installed when you install Scap Workbench.
2. To install `oscap` by itself, run `yum install openscap-scanner`.
3. The SCAP Security Guide (SSG) contains pre-built profiles based on regulatory and government requirements that you can scan with. To download the SSG, run: `yum install scap-security-guide`
4. Oscap command breakdown:
   `oscap [options] module module_operation [module_operation_options]`
   - `module`: Type of SCAP content
   - `module_operation`: A sub-command for the SCAP content

5. Add `--help` to the end of a command for help with a specific sub-command.
6. Scanning a system:
   `oscap [options] module eval [module_operation_options]`
7. Can scan with profiles created with either OVAL or XCCDF (format types used to build SCAP content).
   `oscap oval eval --results scan-results.xml /usr/share/xml/scap/ssg/content/ssg-rhel7-ds.xml`
   - Scan using OVAL-formatted content included in the `ssg-rhel7-ds.xml` file.
   - Save results as `scan-results.xml`.
8. Create a human-readable report:
   `oscap oval generate report scan-results.xml > ssg-scan-report.html`
9. `oscap` can only scan the local host. It cannot scan remote targets.

Ansible

What Is Ansible?

1. Ansible is a framework for automating the management of systems.
2. Run commands and scripts (playbooks) against multiple managed nodes.
   - For example, checking for packages, patching, pushing policies, etc.
3. Generally have a control node and managed nodes.

YAML

1. Human-readable language used by Ansible
2. Highly dependent on formatting (spacing), very sensitive to formatting errors
   Example:

```
- hosts: all
  vars:
  tasks:
  - name: Ensure telnet is removed
    package:
      name: "{{item}}"
      state: absent
    with_items:
      - telnet
  - name: Ensure FTP is removed
    package:
      name: "{{item}}"
      state: absent
    with_items:
      - ftp
```

Components of Ansible

Inventories

1. How Ansible locates and runs against multiple systems (list of hosts).
2. Default location is `/etc/ansible/hosts`.
3. Can define groups of hosts and groups of groups.
4. Variables for connecting to hosts can be stored in hosts file.
5. Don't have to use the default hosts inventory file—can create your own and reference them in an Ansible command with the `-i` flag.

Example Inventory File:

```
[webservers]
web1.ourdomain.com
10.0.0.151
192.168.55.3

[dbservers]
db[1:5].ourdomain.com
172.16.55.48

[webndbservers]
webservers
dbservers
```

Modules

1. Modules are tools to be used on managed nodes.
2. Examples: `ping`, `yum`, `service`, etc.
3. Reference a module with the `-m` flag.
    - **Example:** `ansible labserver1.mydomain.com -m ping`
    - **Example:** `ansible all -m ping`
      (`all` refers to all managed nodes in the default hosts file)
4. Module search: https://docs.ansible.com/ansible/latest/modules/modules_by_category.html
5. Ansible ships with most common modules; can download additional specific modules as needed (AWS, Google, Cisco, etc.).
6. Can write custom modules via Python if you'd like!

Playbooks

1. A playbook is a series of plays.

   Example:

   a. Install an application using the `yum` module.

   b. Make a change to the application's config file.

   c. Restart the service for the application.

2. Playbooks use YAML format.

Sample Playbook:

```
---
- host: webservers
  tasks:
    - name: install apache
      yum:
        name: httpd
        state: present
    - name: esure httpd service starts at boot
      service:
        name: httpd
        enabled: yes
    - name: start httpd service
      service:
        name: httpd
        state: started
    - name copy template to web directory
      template: src=/etc/ansible/index.html dest=/var/www/html
       owner=apache group=apache mode=0644
```

Configuration Files

1. There are several configuration files for Ansible. These are listed below in the order they are checked:

   a. `ANSIBLE_CONFIG` (An environment variable)

   b. `ansible.cfg` (In the current directory)

   c. `ansible.cfg` (Hidden file in the home directory)

   d. `/etc/ansible/ansible.cfg` (Master config)

2. As soon as Ansible finds a file, it stops searching.

Installing Ansible

```
sudo yum install -y ansible
```

Creating an `ansible` User to Run Ansible and Connect to Managed Nodes

1. `sudo useradd ansible` (Must be done on all nodes)

2. `sudo passwd ansible` (Only required on manged nodes, not the manager node)

    - Manager node will use `sudo` to run as `ansible` user

Setting Up an SSH Key Pair

1. On the control node, generate certificates:

    ```
    sudo su - ansible
    ssh-keygen
    ```

2. Press **Enter**.

3. Do ***not*** enter a passphrase! Just press **Enter** twice.

Copying an SSH Key to Managed Nodes

1. `ssh-copy-id host2.domain`

2. Enter the `ansible` user's password.

Setting Up `sudo` Permissions for the `ansible` User

1. The `ansible` account on the control server doesn't have a password and has very limited privileges. We need to allow it to use `sudo` so we can use the account to execute Ansible playbooks and commands.

    ```
    sudo visudo
    ```

2. Under `## Same thing without a password`, add the following line:

    ```
    ansible      ALL=(ALL)       NOPASSWD: ALL
    ```

> **Note**: If you don't want to set up SSH key pairs, you can use the `-k` switch. You'll be prompted for the password for the node you're attempting to run a playbook or command against.

Running Ad-Hoc (One-Line) Ansible Commands

1. Why use ad-hoc commands?
    - Testing module commands to see how they run
    - Using Ansible as a tool to check logs, restart daemon, etc.
    - Vulnerability report comes out and you want to identify which hosts have the vulnerable software installed on them
        - Command for ad-hoc: `ansible`
        - Example: `ansible webservers -b -m yum -a "name=https state=latest" -f 100`
            - `webservers`: Managed nodes in the inventory file
            - `-b`: "Become" — Specifies to become root and run the command
            - `-m`: "Module" — Which Ansible module to use
            - `-a`: "Arguments" — What to pass to the module. Always in double quotes and a spaced list of arguments.
            - `-f`: "Fork" — Number of managed nodes to run the command against at one time. Default value is 5. Will impact performance!

Some Popular Modules and Their Required Arguments

- `ping` — Checks for connectivity. Responds with `pong` on success.

  Example: `ansible lab1.mydomain.com -m ping`

- `setup` — Gathers facts about managed nodes and host info.

  Example: `ansible lab1.mydomain.com -m setup`

- `yum` — Requires two arguments: `name` and `state` of the package.

  Example: `ansible lab1.mydomain.com -b -m yum -a "name=httpd state=latest"`

  Example: `ansible lab1.mydomain.com -b -m yum -a "name=httpd state=absent`

- `service` — Controls daemons and requires two arguments: `name` and `state`.

  Example: `ansible lab1.mydomain.com -b -m service "name=httpd state=restart"`

- `user` — Used to manipulate system users. One argument: `name`.

  Example: `ansible lab1.mydomain.com -b -m user -a "name=Jack"`

  Example: `ansible lab1.mydomain.com -b -m user -a "name=Jack append=yes groups=usbguard"`

- `copy` — Copies files. Two arguments: `src` and `dest`.

  Example:

```
ansible lab1.mydomain.com -b -m copy -a "src=/home/cloud_user/config.cfg
 dest:/etc/app/config.cfg"
```

- `file` — Used for working with files. One argument: `path`.

  Example:

  ```
  ansible lab1.mydomain.com -b -m file -a "path=/home/cloud_user/newfile state=touch"
  ```

  Example:

  ```
  ansible lab1.mydomain.com -b -m file -a "path=/home/cloud_user/newfile mode=0400"
  ```

  Example:

  ```
  ansible lab1.mydomain.com -b -m file -a "path=/home/cloud_user/newfile owner=root"
  ```

- `shell` — Runs shell commands.

  Example: `ansible lab1.mydomain.com -b -m shell -a "shutdown -r now"`

Running Playbooks

- Playbooks are a series of ad-hoc commands run as a single command.
- Command for running a playbook: `ansible-playbook`
- Great for routine tasks, system deployments, etc.
- Basically an Ansible script.
  Example: `ansible-playbook myplaybook.yaml`
- The *limit* flag (`--limit`) allows you to limit the managed nodes a playbook is run against. A playbook may be configured to run against a group of servers, but if you want to run it against a single target, use the `--limit` flag.
  Example: `ansible-playbook myplaybook.yaml --limit testhost.mydomain.com`

Creating Playbooks (YAML)

Example Playbook:

```
---
 - hosts: webservers
   become: yes

   tasks:
   - name: ensure apache is the latest version
     yum: name=httpd state=latest
   - name: write the apache config file
     template: src=/srv/httpd.j2 dest=/etc/httpd.conf

 - hosts: databases
```

```
become: yes

tasks:
- name: ensure postgresql is at the latest version
  yum: name=postgresql state=latest
- name: ensure that postgresql is started
  service: name=postgresql state=started
```

- You can also nest arguments if you'd like (easier to read).

```
tasks:
- name: ensure postgresql is at the latest version
  yum:
    name: postgresql
    state: latest
- name: ensure that postgresql is started
  service:
    name: postgresql
    state: started
    enabled: yes
```

- Three hyphens start every playbook and are ***required***!
- `hosts`: Managed nodes.
- `become`: Change to root to run the modules and arguments.
- `name`: Name of a task.
- Module and arguments.
- Watch your YAML formatting/spacing. If a playbook contains formatting/spacing errors, Ansible won't run it!
- **Retry files:** If a playbook fails on a host, Ansible will create a .retry file called `<playbookname>.retry`. You can run that file, and it will only run against hosts on which the playbook failed.

Section 2: Intrusion Detection

AIDE (Advanced Intrusion Detection Environment)

What Does AIDE Do?

1. AIDE conducts integrity checks of files.
    - When sensitive files are changed
    - Maintains a database of file statuses

- Emails notifications of changes
- Can be used to identify unauthorized changes to help detect system intrusions
- Monitors last time a file was accessed (useful for sensitive files and applications)

2. How to use AIDE:
   - Define directories, files, applications to be monitored.
   - Run an initial scan to create a baseline.
   - Schedule a daily or weekly integrity check (cron job).
   - Check reports and investigate changes.
   - After changes occur, update the database (update your baseline).

Installing and Configuring AIDE

1. Install AIDE.

   ```
   sudo yum install -y aide
   ```

   - Config file: `/etc/aide.conf`

2. Build the baseline database for AIDE.

   ```
   sudo /usr/sbin/aide --init
   ```

   - (Takes 2 minutes to run; pause video.)

3. Create the .db file:

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

4. Copy the .db file to a production location.

   ```
   sudo cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
   ```

5. Run an integrity check on the files.

   ```
   sudo /usr/sbin/aide --check
   ```

   - (Takes almost 2 minutes to run; pause video.)
   - After a `yum update`, you'll see many changed files!

6. Schedule a daily integrity check:
   - Edit `/etc/crontab` and set `MAILTO:root` to send an email to your email address.
   - Schedule this to run in a cron job daily:
     `0 1 * * * /usr/sib/aide --check`

Testing AIDE Functionality

1. Testing AIDE detection functionality:

```
sudo touch /usr/sbin/aide-test
sudo /usr/sbin/aide --check
```

   - Will show two added files because `/sbin/` is a link to `/usr/sbin`.
   - AIDE will continue to alert on the changes until you update the database and rename the updated database.
     - `sudo /usr/sbin/aide --update` (Takes 2 minutes)
     - `sudo cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz`

Tuning AIDE

1. Tuning AIDE with the `/etc/aide.conf` file:

   - Database location
   - Log file location: `/var/log/aide/aide.log`
   - Review ***Default Rules*** — all those letters!
   - Review the ***Watched Parameter*** rule grouping.
   - Review the directories being checked.

   Default Rules:

```
# These are the default rules.
#
#p:     permissions
#i:     inode
#n:     number of links
#u:     user
#g:     group
#s:     size
#b:     block count
#m:     mtime
#a:     atime
#c:     ctime
#S:     check for growing size
#acl:           Access Control Lists
#selinux        SELinux security context
#xattrs:        Extended file attributes
#md5:    md5 checksum
#sha1:   sha1 checksum
#sha256:        sha256 checksum
#sha512:        sha512 checksum
#rmd160: rmd160 checksum
#tiger:  tiger checksum
```

Rule Grouping:

```
FIPSR = p+i+n+u+g+s+m+c+acl+selinux+xattrs+sha256

#R:             p+i+n+u+g+s+m+c+acl+selinux+xattrs+md5
#L:             p+i+n+u+g+acl+selinux+xattrs
#E:             Empty group
#>:             Growing logfile p+u+g+i+n+S+acl+selinux+xattrs

# You can create custom rules like this.
# With MHASH...
# ALLXTRAHASHES = sha1+rmd160+sha256+sha512+whirlpool+tiger+haval+gost+crc32
ALLXTRAHASHES = sha1+rmd160+sha256+sha512+tiger

# Everything but access time (i.e., all changes)
EVERYTHING = R+ALLXTRAHASHES

# Sane, with one good hash.
# NORMAL = sha256
NORMAL = sha256

# For directories, don't bother doing hashes.
DIR = p+i+n+u+g+acl+selinux+xattrs

# Access control only.
PERMS = p+u+g+acl+selinux+xattrs

# Access + inode changes + file type.
STATIC = p+u+g+acl+selinux+xattrs+i+n+b+c+ftype

# Logfiles only check access w/o xattrs.
LOG = p+u+g+n+acl+selinux+ftype

# Content + file type.
CONTENT = sha256+ftype
```

- I like to add `APP-ACCESS = a` (Access time changed)
  - Great for monitoring sensitive application use (e.g., accounting apps).
- Which directories, daemons, files, and applications to monitor.
  - Now it's time to add any other objects you'd like to monitor with AIDE.

Section 3: System Hardening

Securing SSH

Controlling SSH Access to a Host

1. Configured in the `/etc/ssh/sshd_config` file using the following keywords:
    - `DenyUsers`
    - `AllowUsers`
    - `DenyGroups`
    - `AllowGroups`
        - Processed in this order from the top down by the SSH daemon.
        - Once you add a user to `AllowedUsers`, only those users are permitted.
2. Use the `AllowUsers` setting to define access by a user from a specific hostname or IP/subnet.
    - Limit root SSH access to only connect from the 192.168.22.0/24 network:
        - `AllowUsers root@192.168.22.*`
          (Will have to enable root login; by default, it is disabled via the `PermitRootLogin` keyword.)
    - Limit access for user `James` to only connect from host1.office.net:
        - `AllowUsers James@host1.office.net`
    - Combine both of the above into a single line:
        - `AllowUsers root@192.168.22.0/24 James@host1.office.net`
    - Can do the same with groups using `AllowGroups`.
    - Now restart the SSH daemon:
        - `systemctl restart sshd.service`
    - Once you configure `AllowUsers`, only those defined in the list will be permitted to connect via SSH.
3. If you want to permit root SSH login from an IP address or host but not enable root SSH login globally, you'll need to use the `match` keyword.

Example:

```
Match Address 10.15.20.0/24,192.168.55.0/24
      PermitRootLogin yes
```

- The `match` keyword sets a condition. If that condition is met, the next line is run.

Generating and Distributing SSH Keys

1. Ensure you are running as the user for which you want to generate a key. If you need to switch to that user, run the following:
   `sudo - <USERNAME>`
2. Generate the SSH key pair:
   `ssh-keygen`
3. Provide the passphrase.
4. This will generate two files in the user's `~/.ssh/` directory:
   - id_rsa: Private key (keep this safe)
   - id_rsa.pub: Public key (you will share this in the next step)
5. Distribute the key to remote hosts:
   `ssh-copy-id -i user@host.domain`

Disable the Use of Passwords (Force the Use of SSH Keys)

1. To force the use of SSH keys for autentication, disable `PasswordAuthentication` within `sshd_config`:

```
PasswordAuthentication no
```

Set the SSH Version

1. Within the `sshd_config` file, set the version using the `Protocol` keyword:

```
Protocol 2
```

Managing sudo

Controlling Sudo Access

1. `sudo` allows users to run commands with elevated (root) permissions without the need for a root shell.
2. Only users listed in the `/etc/sudoers` file are allowed to use `sudo`.
3. When a user runs a command using `sudo`, they are prompted for their password.
   - Password is stored for a 5-minute period.
4. To edit the `/etc/sudoers` file, use the `visudo` command.
5. To give a user full administrative permissions, add the following line to the `/etc/sudoers` file:
   `james ALL=(ALL) ALL`

- This states that user `james` can use `sudo` from any host and execute any command.

6. `sudo` can be used granularly as well:
   `%users locahost=/usr/sbin/shutdown -h now`
   - This states that any member of the `users` system group can issue the command `shutdown -h now` as long as it's issued from the console.
7. You can also set it so that users don't need to enter a password when using sudo:
   `user_name ALL=(ALL)  NOPASSWD: ALL`
8. Groups can be added to the `sudoers` file as well, using the same format used to add users:
   `group_name ALL=(ALL) ALL`

Locking Down `sudo`

1. By default, all users of the `wheel` group have `sudo` access:
   `%wheel  ALL=(ALL)  ALL`
   This allows people in the `wheel` group to run ***all*** commands and should be changed.
2. By default, `sudo` stores the user password for a period of 5 minutes, which could be exploited.
   - Consider adding the following line to the `sudoers` file to change the default timeout period:
     `Defaults  timestamp_timeout=1` (1 minute)
   - Setting this value to zero requires a password to be entered every time.

Example `sudo` Scenarios

- **Scenario 1:**

  You've been asked to grant `sudo` access to the user `mscott`.

  Solution:

  - **Option 1:** Add the user `mscott` to the `wheel` group.

    ```
    usermod -aG wheel mscott
    ```

  - **Option 2:** Use `visudo` to edit `/etc/sudoers`, and add the following line:

    ```
    mscott ALL=(ALL) ALL
    ```

- **Scenario 2:**

  In order to reduce the risk associated with `sudo`, you've been asked to make `sudo` require a password every time it's used.

  Solution: Run `visudo`, and add the following line to the `/etc/sudoers` file:
  `Defaults  timestamp_timeout=0`

- **Scenario 3:**

  The `helpdesk` group needs to be able to reboot servers using the `sudo` command. We do not want the `helpdesk` group to be able to run any other `sudo` commands.

  Solution: Run `visudo`, and add the following line to the `/etc/sudoers` file:
  `%helpdesk locahost=/usr/sbin/shutdown -r now`

USBGuard

What Is USBGuard?

1. USBGuard is used for whitelisting and blacklisting USB devices based on device attributes.
2. The USBGuard framework provides the following:
    - A daemon component with an inter-process communications (IPC) interface for interaction with policy enforcement
    - A CLI to interact with a USBGuard instance
    - A language for writing USB device authorization policies
    - A C++ API for interacting with the daemon
3. USBGuard terms:
    - `allow` — Communicate with this device
    - `block` — Do not communicate with this device for now
    - `reject` — Ignore this device as if it didn't exist
4. USBGuard help command: `usbguard --help`

How to Install and Configure USBGuard

Installation

1. USBGuard install command: `sudo yum install usbguard`
2. Review the config file `/etc/usbguard/usbguard-daemon.conf`.
    - Rule file path (`/etc/usbguard/rules.conf`)
    - Implicit policy (`block`)
    - IPCAllowedUsers (root)
    - IPCAllowedGroups (blank; set to root)
3. Create a new group and add it to IPCAllowedGroups.

```
groupadd usbguard
usermod -aG usbguard cloud_user
usermod -aG usbguard root
```

4. Verify that this was successful.
   `grep 'usbguard' /etc/group`
5. Update `/etc/usbguard/usbguard-daemon.conf` to show `IPCAllowedGroups=usbguard`.

6. Restart the USBGuard daemon.
   `sudo systemctl restart usbguard.service`
7. Check the status of the USBGuard daemon.
   `sudo systemctl status usbguard`
8. Create the initial rule set.
   `sudo usbguard generate-policy > /etc/usbguard/rules.conf`
   - This creates an allow rule for all devices already connected.
9. Ensure USBGuard starts at boot:
   `sudo systemctl enable usbguard.service`
10. Restart the USBGuard daemon.
    `sudo systemctl restart usbguard.service`
11. Check the status of the USBGuard daemon.
    `sudo systemctl status usbguard`

Working with USBGuard

1. List all devices recognized by USBGuard:
   `sudo usbguard list-devices`
2. Authorize a device within USBGuard:
   `sudo usbguard allow-device 6`
3. Deauthorize a device within USBGuard:
   `sudo usbguard block-device 6`
4. Watch terminal for IPC activity:
   `sudo usbguard watch`

Creating Whitelists in USBGuard

> Note: We're not creating blacklists because whitelists are more secure and are considered best practice!

Rule Options

- `hash "[0-9a-f]{32}"` — Match a hash of the device attributes. (The hash is computed for every device by USBGuard.)
- `name "device-name"` — Match the USB device name attribute.
- `serial "serial-number"` — Match the iSerial USB device attribute.
- `via-port "port-id"` — Match the USB port through which the device is connected.
- `via-port [operator] { "port-id" "port-id" ... }` — Match a set of USB ports.

- `with-interface interface-type` — Match an interface the USB device provides.
- `with-interface [operator] { interface-type interface-type ... }` — Match a set of interface types against the set of interfaces that the USB device provides.

Rule Operators

- `all-of` — Must contain all of the specified values for the rule to match.
- `one-of` — Must contain at least one of the specified values for the rule to match.
- `none-of` — Must not contain any of the specified values for the rule to match.
- `equals` — Must contain exactly the same set of values for the rule to match.
- `equals-ordered` — Must contain exactly the same set of values in the same order for the rule to match.

Rule Conditions

- `localtime(time_range)` — Evaluates to true if the local time is in the specified time range. `time_range` can be written either as HH:MM[:SS] or HH:MM[:SS]-HH:MM[:SS].
- `allowed-matches(query)` — Evaluates to true if an allowed device matches the specified query. The query uses the rule syntax. Conditions in the query are not evaluated.
- `rule-applied` — Evaluates to true if the rule currently being evaluated ever matched a device.
- `rule-applied(past_duration)` — Evaluates to true if the rule currently being evaluated matched a device in the past duration of time specified by the parameter. `past_duration` can be written as HH:MM:SS, HH:MM, or SS.
- `rule-evaluated` — Evaluates to true if the rule currently being evaluated was ever evaluated before.
- `rule-evaluated(past_duration)` — Evaluates to true if the rule currently being evaluated was evaluated in the past duration of time specified by the parameter. `past_duration` can be written as HH:MM:SS, HH:MM, or SS.
- `random` — Evaluates to true/false with a probability of p=0.5.
- `random(p_true)` — Evaluates to true with the specified probability `p_true`.
- `true` — Always evaluates to true.
- `false` — Always evaluates to false.

Negation

- Can negate with the use of an exclamation point (`!`) — see example policy 3 below.

Rule Language

```
rule ::= target device_id device_attributes conditions
target ::= "allow" | "block" | "reject"
device_id ::= "*:*" | vendor_id ":*" | vendor_id ":" product_id
device_attributes ::= device_attributes | attribute
conditions ::= conditions
```

Writing Rules

- Create a `rules.conf` file in your home directory and add rules there.
- Once you are ready to apply the rules, run: `sudo install -m 0600 -o root -g root rules.conf /etc/usbguard/rules.conf`
- Device type (storage, keyboard, etc.) rules are based on the USB classes and program interface IDs found here: https://usb-ids.gowdy.us/read/UC/
- Use `sudo` and write to `rules.conf` in your current directory.
- Once rules are written into your local directory's `rules.conf` file, they will be moved to the `/etc/usbguard/rules.conf` file via a command we'll run below.
- **Do *not*** attempt to write directly to the `/etc/usbguard/rules.conf` file, as that will not work and will cause the USBGuard service to stop functioning.
    - **Example Policy 1:**

      ```
      allow with-interface equals { 08:*:* }
      ```

        - Allows USB mass storage devices (Class 08), denies all other devices via implicit rule in the `/etc/usbguard/usbguard-daemon.conf` file.
    - **Example Policy 2:**

      ```
      allow 1050:0011 name "Yubico Yubikey II" serial "0001234567" via-port "1-2"
       hash "044b5e168d40ee0245478416caf3d998" reject via-port "1-2"
      ```

        - Allows a specific Yubiky on a specific port, rejects all other devices on that port.
    - **Example Policy 3:**

      ```
      allow with-interface one-of { 03:00:01 03:01:01 } if !allowed-matches
       (with-interface one-of { 03:00:01 03:01:01 })
      ```

        - Allows a keyboard-only USB device if there's not already one plugged in.
- Apply policy changes:

  ```
  sudo install -m 0600 -o root -g root rules.conf /etc/usbguard/rules.conf
  sudo systemctl restart usbguard
  ```

- List all rules: `sudo usbguard list-rules`
- Remove a rule: `sudo usbguard remove-rule <ID>`

Pluggable Authentication Module (PAM)

What Is PAM?

PAM is an authentication framework built into Red Hat Linux that is used to provide central authorization and authentication.

As we know, frameworks are like tool boxes, and PAM is no different. Applications can integrate with PAM, creating a PAM-aware application that can use PAM's authentication mechanisms to authenticate users. The tools used to interact with PAM are called modules.

The `/etc/pam.d directory` is where the PAM config files are stored.

Each PAM-aware application has a config file within the `/etc/pam.d` directory. Each config file is named for the service that uses it. For example, the sshd application uses a PAM config file named `sshd`. The application config files indicate which PAM modules are used by the applicaiton.

PAM documentation can be found at `/usr/share/doc/pam-x.x.x` (x.x.x = version number).

Installing PAM

```
yum install pam-devel
```

Creating a PAM Failed Login (Account Lockout) Policy

1. Edit `/etc/pam.d/password-auth` and `/etc/pam.d/system-auth`.
2. Include the following as the second non-commented line in both files:
   ```
   auth        required      pam_faillock.so preauth silent audit deny=3 unlock_time=600
   ```
   - `deny=3` — 3 failed login attempts
   - `unlock_time=600` — Account will auto-unlock in 600 seconds (10 minutes)
3. Include the following as the fourth non-commented line:

   ```
   auth        [default=die] pam_faillock.so authfail audit deny=3 unlock_time=600
   ```

4. Add the following as the first line in the ***Account*** section of both files:

   ```
   account     required      pam_faillock.so
   ```

5. If applying lockout to the root user as well, add the following parameter to the lines created in steps 2 and 3: `even_deny_root`

   Example:

   ```
   auth        required      pam_faillock.so preauth silent audit
   deny=3 even_deny_root unlock_time=600
   ```

6. To exempt specific users from the account lockout policy, add the following to `/etc/pam.d/password-auth` and `/etc/pam.d/system-auth`, above the first line where `pam_faillock` is mentioned:

```
auth [success=1 default=ignore] pam_succeed_if.so user in user1:user2:user3
```

(Replace `user1`, `user2`, and `user3` with the user accounts you want to exempt from account lockout.)

1. To view the number of failed logins by user, run the following command as root:
   `faillock`
2. Failed login attempts are stored in a separate file for each user in the `/var/run/faillock` directory.
3. To unlock a user's account, run the following command as root:
   `faillock --user <USERNAME> --reset`

Creating a PAM Password Complexity Policy

A PAM password complexity policy checks a password's strength against a set of rules in two phases: 1. Checks to see if the password is found in a dictionary file. If not, it moves to step 2. 2. PAM checks rules defined in `/etc/security/pwquality.conf`.

Create Rules and Enable Password Complexity Requirements

1. Add the following line to the password stack in the `/etc/pam.d/passwd` file:
   `password    required    pam_pwquality.so retry=3`
2. Create complexity rules in the `/etc/security/pwquality.conf` file:

```
minlen = 8
minclass = 4
maxsequence = 3
maxrepeat = 3
maxclassrepeat = 3
difok = 5
enforce_for_root = 1
ucredit = -3
lcredit = -3
dcredit = -2
ocredit = -2
badwords = word1, word2, word3
gecoscheck = 1
```

- `minlen` — Minimum password length
- `minclass` — Minimum number of character classes (capital, lowercase, number, special character, ASCII, other)

- `maxrepeat` — Maximum number of repeated characters
- `maxclassrepeat` — Maximum number of repeated characters of the same class
- `difok` — Maximum number of characters in the new password that must not be in the old password
- `enforce_for_root` — Enforce password complexity for root account
- `ucredit` — Required number of uppercase letters (negative value)
- `lcredit` — Required number of lowercase letters (negative value)
- `dcredit` — Required number of digits (negative value)
- `ocredit` — Required number of other characters (negative value)
- `badword` — List of words not permitted in passwords (company name, phone number, address, year, season, sports teams, etc.)
- `gecoscheck` — Check user information (real name, etc.) to ensure it is not used in password

Keep a Password History

1. Insert the following in `/etc/pam.d/system-auth` and `/etc/pam.d/password-auth` after the line `pam_pwquality.so`:

```
password    requisite     pam_pwhistory.so remember=10 use_authtok
```

Controlling `sudo` Access

Edit the `/etc/sudoers` File

1. Use `visudo` to edit the `/etc/sudoers` file.
2. To give a user full root control using `sudo`, add the following line to the `/etc/sudoers` file:
   `<username> ALL=(ALL)  ALL`
   - This states that the user can run `sudo` from any host and execute any command.
3. Rules can be granular as well, such as:
   `%helpdesk localhost=/usr/sbin/shutdown -r now`
   - This would allow any user in the `helpdesk` group to reboot the host.
4. You can do the same for groups: `admin-group ALL=(ALL)  ALL`
   - This gives users in the group `admin-group` full root permissions using `sudo`.
5. Users are prompted for passwords each time they run a `sudo` command. Credentials are kept in cache for 5 minutes.
6. To ***not*** require the use of a password when running `sudo`, use the `NOPASSWD` option:
   `root ALL=(ALL)  NOPASSWD:ALL`

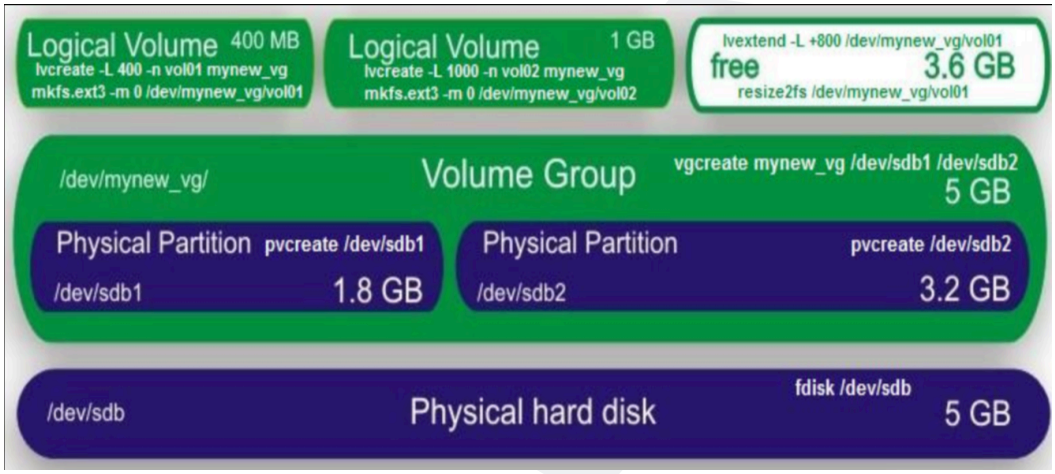
Locking Down `sudo`

1. By default, all members of the `wheel` group have full root access via `sudo`.
    - Is this needed in your environment?
    - If not, comment it out.
      `#wheel  ALL=(all)  ALL`
2. The 5-minute timeout period can be a security risk if someone walks away from a terminal.
    - Consider requiring a password every time `sudo` is used.
    - Set the timeout value in `/etc/sudoers` to 0.
    - `Defaults  timestamp_timeout=0`

Section 4: Disk Encryption

Storage on Red Hat Linux Hosts

1. Physical drives (`/dev/sdb`)
2. Physical partitions (`/dev/sdb1`, `/dev/sdb2`) within a volume group (VG)
3. Volume groups (`/dev/myvolgroup_vg`) mapping to the use of the physical partitions
4. Logical volumes (LVs) (`/dev/myvolgroup_vg-vol01_lv` and `/dev/myvolgroup_vg-vol02_lv`) mounted to space on a volume group
5. Notice the tree structure?

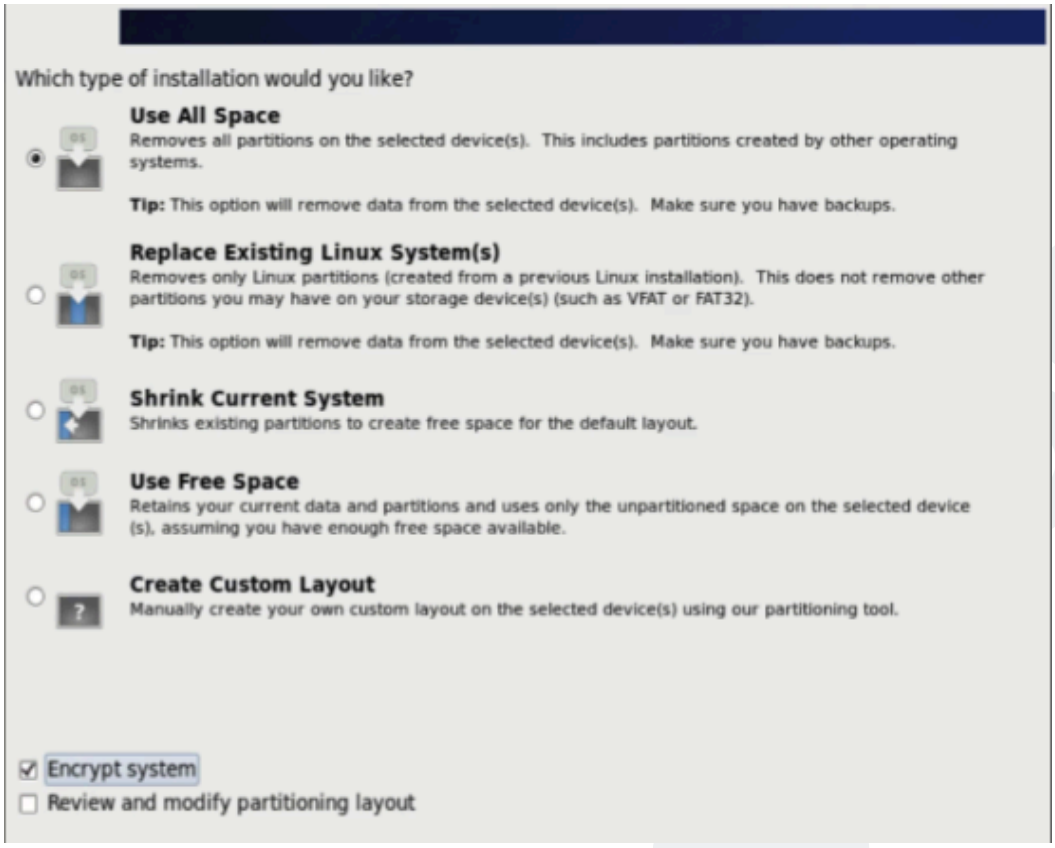


Linux Unified Key Setup (LUKS)

1. What are we trying to do?
    - Protect our data at rest

- Protect against stolen/lost device or drive

2. Data is encrypted while the device is off or the volume is unmounted.
3. If the computer is running, the data is decrypted so the operating system can read it.
4. Encrypt drives at Red Hat install using the **Encrypt System** option (see image). This will require you to enter an encryption passphrase that will have to be entered at boot time to decrypt the drive.



5. You can also encrypt a single partition on a host.
   - The volume must be unmounted at the time of encryption.
   - The volume should also be wiped of data prior to encryption.
6. Encrypting a volume will destroy the data on the volume. Before encrypting, back up the data somewhere else, then encrypt the volume, then restore the data.

The Encryption Process

1. Ensure the LUKS packages are installed.

   ```
   sudo yum install cryptsetup
   ```

2. Identify available space in the targeted volume group (VG).

   ```
   host# vgdisplay
   VG Name                myvolgroup_vg
   ```

```
  System ID
  Format                lvm2
  Metadata areas        2
  Metadata Sequence No  4
  VG access             read/write
  VG status             Resizable
  MAX LV                0
  Cur LV                2
  Open LV               0
  Max PV                0
  Cur PV                2
  Act PV                2
  VG size               4.99 GB
  PE size               4.00 MB
  Total PE              1277
  Alloc PE / size       356 / 1.39GB
  Free PE / size        921 / 3.60 GB   **** THIS LINE ****
  VG UUID               0tssp7-dMvt-H60r-qHMO-Ev4H-YgmI-ZjdZYA
```

3. Create a volume (unless a volume already exists).
   `lvcreate -L 400 -n vol01_lv myvolgroup_vg`
   - `-L`: Size (400 MB)
   - `-n`: Name (vol01_lv)
   - `myvolgroup_vg`: Volume group that the new logical volume will be part of
4. Format the volume with LUKS (encrypt it).
   `cryptsetup luksFormat /dev/mapper/myvolgroup_vg-vol01_lv`
   - You can encrypt using a key file with the `--key-file /path/to/keyfile` option.
   - Encryption options:
     - `--cipher`: Encryption type (AES, Twofish, etc.)
     - `--key-size`: Key size (256, 512, 1024, etc.)
     - `--hash`: Hash type (SHA-256, SHA-512, etc.)
       - Run `cryptsetup --help` to see the default options.
       - Red Hat strongly recommends using the default ciphers.
   - Check for encryption using `blkid`.
5. Open the volume with LUKS (decrypt it so the OS can work with it).
   `cryptsetup luksOpen /dev/mapper/myvolgroup_vg-vol01_lv vol01_lv`
   - This volume will appear under `/dev/mapper/vol01_lv`.
6. Write random data to the new volume using `shred` (protects against usage pattern disclosure).
   `shred -v -n1 /dev/mapper/vol01_lv`

7. Create a mount point for the new volume if one doesn't already exist.
   `mkdir /new_mount_point`
8. Format the new volume so you can store data on it.
   `mkfs.ext4 /dev/mapper/vol01_lv`
9. Mount the volume so it's available for use.
   `mount /dev/mapper/vol01_lv /mount_point`
10. Verify the mapping of the new volume.
    `cryptsetup -v status vol01`
11. Configure the volume to mount at boot time (optional).
12. Add an entry in `/etc/crypttab` for the new volume:
    `home /dev/mapper/myvolgroup_vg-vol01_lv`
13. If `/etc/crypttab` doesn't exist, create it, change the owner to `root:root`, and set the mode to `0744`.
14. Add an entry to `/etc/fstab`:
    `/dev/mapper/myvolgroup_vg-vol01_lv /new_mount_point ext4 0 2`
15. List the partitions.
    `df -h`

Unmount and Close Encrypted Volumes Securely

1. Unmount a volume:
   `umount /mount_point`
2. Close an encrypted volume (put it into an encrypted state):
   `cryptsetup luksClose vol01_lv`

Open and Mount Existing LUKS-Encrypted Volumes

1. Open a LUKS volume (decrypt it):
   `cryptsetup luksOpen /dev/mapper/myvolgroup_vg-vol01_lv vol01_lv`
2. Mount a LUKS partition for use:
   `mount /dev/mapper/vol01_lv /mount_point`
3. Verify the mount:

```
[root@ip-10-0-0-167 cloud_user]# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
xvda                202:0    0    8G  0 disk
└─xvda1             202:1    0    8G  0 part  /
xvdg                202:96   0   20G  0 disk
└─nbde_vg-doctor_lv 253:0    0  100M  0 lvm
  └─doctor_lv       253:1    0   98M  0 crypt /doctor
[root@ip-10-0-0-167 cloud_user]#
```

Adding a Backup Key to a LUKS-Encrypted Volume

> Note: Backup keys are used when the initial key is forgotten or corrupted.

1. Check the key slots for a LUKS-encrypted volume.
   `cryptsetup luksDump /dev/mapper/myvolgroup_vg-vol01_lv`
2. Identify the used key slots.
3. Add a backup key.
   `cryptsetup luksAddKey --key-slot 1 /dev/mapper/myvolgroup_vg-vol01_lv`
4. Enter the original passphrase, then the new passphrase.
5. Check for the new passphrase.
   `cryptsetup luksDump /dev/mapper/myvolgroup_vg-vol01_lv`
6. Look for data in key slot 1.

Creating a Backup of LUKS Encryption Headers

> Note: If the LUKS header is corrupted and no keys are readable, all data will be lost if you don't have a backup of the LUKS header.

Backing Up a LUKS Encryption Header

```
cryptsetup luksHeaderBackup /dev/mapper/myvolgroup_vg-vol01_lv --header-backup-file
 /root/vol01_lv_header.backup
```

- This is the only way to recover encrypted data in the event of header corruption.
- A backup key may be able to recover the data if it's not damaged.

Restoring a LUKS Encrypted Header

```
cryptsetup luksHeaderRestore /dev/mapper/myvolgroup_vg-vol01_lv --header-backup-file
 /root/vol01_lv_header.backup
```

- After a restore, all backup keys are lost and will need to be recreated.

Changing a Passphrase on a LUKS-Encrypted Volume

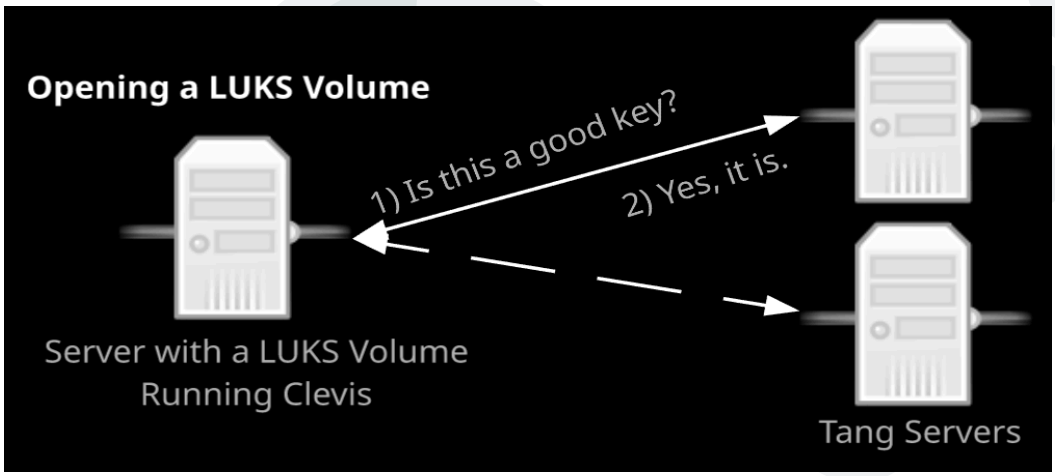
> Note: Removing a LUKS key makes the container permanently inaccessible, so don't do that!

1. Change the passphrase:
   `cryptsetup luksChangeKey /dev/mapper/myvolgroup_vg-vol01_lv`
2. Enter the old passphrase.
3. Enter the new passphrase. (**Note:** You will ***not*** be asked to confirm the new passphrase, so be careful!)

Network-Bound Disk Encryption (NBDE)

Introduction to NBDE

1. Solves the problem of needing to enter a passphrase at system boot to unlock LUKS-encrypted volumes.
2. Consists of a Clevis service running on hosts using LUKS-encrypted volumes and a server (or servers) running a Tang service used to authorize the unlocking of the LUKS volumes.
3. At boot, Clevis reaches out to servers running a Tang service to automate the process of validating keys in order to mount LUKS-encrypted volumes.
4. The actual encryption key is not passed across the network, only a Tang authorization key (see image below).
5. If no Tang servers are available, you can enter the LUKS passphrase to unlock the volumes.



Setting Up NBDE

1. Deploy a Tang server.
   `sudo yum install tang`
2. Enable and start the tangd service:
   `systemctl enable tangd.socket --now`
   - A new set of cryptographic keys is generated at first start and is stored in `/var/db/tang`.
3. It is recommended that you run Tang servers in a high-availability mode via:
   - Multiple Tang servers using DNS round-robin
   - Multiple Tang servers with Clevis having keys to each of them
4. Configure the encryption client (Clevis).
   `sudo yum install clevis clevis-luks clevis-dracut`
5. Bind LUKS to Clevis and assign a key.

```
host# clevis bind luks -d /dev/mapper/myvolgroup_vg-vol01_lv tang
 '{"url":"http://10.0.0.51"}'
The advertisement contains the following signing keys:

hb8QEUPOn24Mmml_gkgABus5zBk

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/mapper/myvolgrou_vg-vol01_lv? [yn]
Enter existing LUKS password: XXXXXXXXX
```

6. This LUKS volume can now be unlocked with either the LUKS passphrase or with the Clevis policy.
7. To verify that the Clevis key was successfully placed in your LUKS header, run the following:

```
host# luksmeta show -d /dev/mapper/myvolgroup_vg-vol01_lv
lv
0   active empty
1   active cb6e8904-81ff-40da-a84a-07ab9ab5715e
2 inactive empty
3 inactive empty
4 inactive empty
5 inactive empty
6 inactive empty
7 inactive empty
```

Setting Auto-Decrypt at Boot with `dracut`

1. Install `clevis-dracut`.
   `sudo yum install clevis-dracut`
2. Enable `dracut` to unlock encrypted partitions using NBFS.
   `dracut -f` (Takes about 30 seconds to run; be patient.)
3. Enable the `clevis-luks-askpass.path` helper.
   `systemctl enable clevis-luks-askpass.path`
4. Reboot and test once the server comes back up.

NBDE Maintenance

1. It is important to rotate Tang keys periodically.

2. To do so, you will use the `jose` utility:

```
DB=/var/db/tang
jose jwk gen -i '{"alg":"ES512"}' -o $DB/new_sig2.jwk
jose jwk gen -i '{"alg":"ECMR"}' -o $DB/new_exc2.jwk
```

3. Next, rename the old keys to have a leading period.

```
mv $DB/TeJGmZfleOdAo9obVK5kxnPCDRo.jwk $DB/.TeJGmZfleOdAo9obVK5kxnPCDRo.jwk
mv $DB/VtaKbbPROZk_axkOzanTuuvAxYw.jwk $DB/.VtaKbbPROZk_axkOzanTuuvAxYw.jwk
```

4. No restart of Tang is required.

Section 5: SELinux

Introduction to SELinux

Modes (/etc/sysconfig/selinux)

- **Enforcing:** Security policy is enforced.
- **Permissive:** Logs warnings, does not enforce (troubleshooting/testing).
  - Logs to `/var/log/audit/audit.log`.
- **Disabled:** No policies are loaded.
- `$SELINUX` variable shows SELinux mode at boot.
- Show current mode: `getenforce`
- Set the SELinux mode:
  - `setenforce [Enforcing (1) -or- Permissive (0)]`
  - `setenforce enforcing` same as `setenforce 1`
  - `setenforce permissive` same as `setenforce 0`
- `sestatus` command gives additional SELinux info.

Policies (/etc/sysconfig/selinux)

- **Targeted:** Targeted processes are protected — most secure
  - Protects all defined processes
  - All unprotected processes are “unconfined”
- **Minimum:** Modified — only selected processes are protected

- **MLS:** Multi-level security protection
  - Advanced setup
  - Outside the scope of this certification
- `$SELINUXTYPE` variable defines SELinux policy at boot

Enabling SELinux on a Host for the First Time

- SELinux is installed by default.
- Set the mode to permissive, or it will most likely hang at boot.
  - Edit `/etc/sysconfig/selinux`, and set the `$SELINUX` variable to permissive.
- Create a file at the root of the file system named `.autorelabel`.
  `touch /.autorelabel`
- Reboot, and watch the relabeling progress bar.
  - Takes about as long as a file system check while it relabels.
- Now set SELinux to enforcing mode.
  `setenforce enforcing`

Labeling (Types)

- Files, ports, directories, and processes are all labeled with an SELinux-defined context.
- Finding contexts for files/directories:
  `ls -Z`
  `system_u:object_r:bin_tl`
  - `system_u` — SELinux user
  - `object_r` — SELinux role
  - `bin_tl` — SELinux type (***very*** **important**)
- Types are how SELinux labels objects.

Example:

Let's take a look at Apache's SELinux type.

```
ls -lZ /usr/sbin/httpd
-rwxr-xr-x.  1  root  root system_u:object_r:httpd_exec_t:s0 485785 Mar 18 00:43 /usr/sbin/httpd
```

- `httpd_exec_t` is the SELinux type (`exec`).

```
ls -dZ /etc/httpd
system_u:object_r:httpd_config_t:s0 /etc/httpd
```

- `httpd_config_t` is the SELinux type (`config`).

```
ls -dZ /var/log/httpd
system_u:object_r:httpd_log_t:s0 /var/log/httpd
```

- `httpd_log_t` is the SELinux type (`log`).

```
ls -dZ /var/www/html
system_u:object_r:httpd_sys_content_t:s0 /var/www/html
```

- `httpd_content_t` is the SELinux type (`content`).

Do you see a pattern here?

Example:

What about the ***ports*** used by Apache?

```
netstat -lZ | grep httpd
tcp6  0  0 [::]:http  [::]:*   LISTEN  7822/httpd  system_u:system_r:httpd_t:s0
```

- We can see `httpd_t` is the type assigned by SELinux.

Example:

How about the httpd process?

```
ps axZ | grep httpd
system_u:system_r:httpd_t:s0     7822 ?        Ss     0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0     7823 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
system_u:system_r:httpd_t:s0     7824 ?        S      0:00 /usr/sbin/httpd -DFOREGROUND
```

- We can see the httpd service is assigned the SELinux type of `httpd_t`.

Type Enforcement

- We can use SELinux to define which types can talk to other types.
- Let's take a look at the type for `/etc/passwd`:

```
ls -lZ /etc/passwdNpow
-rw-r--r--. root root system_u:object_r:passwd_file_t:s0 /etc/passwd
```

- We can see that `passwd_file_t` is the SELinux type for `/etc/passwd`.
- Do we want Apache's `httpd_t` process label talking to the `passwd_file_t` label for `/etc/passwd`?
  - Probably not!
- SELinux simply controls which labels can talk to other labels. Simple!

Managing Labels

- `-Z` is how we view the types (`ls -Z`, `ps -Z`, `netstat -Z`, `id -Z`).
- `semanage` is used to manage SELinux settings for many objects, including:
  - Ports
  - Interfaces
  - Users
  - Logins
  - Nodes
  - Booleans
  - And much, much more
- Use the `chcon` command to change the SELinux context (SELinux user, role, type).
  `chcon -u system_u -r object_r -t httpd_sys_content_t /var/www/html/index.html`
  - Can assign only the type using the following:
    `chcon -t httpd_sys_content_t /var/www/html/index.html`
  - Can also simply reference settings from a known good file to have its content copied over:
    `chcon --reference /var/www/html /var/www/html/index.html`

- Can use `restorecon` to recursively push down contexts through a directory: `restorecon -vR /var/www/html`
  - `-v`: Verbose because we want to see what is being changed
  - `-R`: Makes the command recursive
  - Restores from the settings maintained in `/etc/selinux/targeted/contexts/files/`

Booleans

- Booleans are on/off switches.
  - Can web server access home directories?
  - Can anonymous write to FTP directory?
  - Etc.

Working with Booleans

- List all booleans that exist: `getsebool -a`
- Include policy description: `semanage boolean -l`
- View the booleans that are already set: `getsebool -a | grep ftp`
- Set a boolean value: `setsebool -P ftpd_anon_write on`
  - `-P` makes the change persistent.

Troubleshooting SELinux

- SELinux logs to `/var/log/audit/audit.log`.
- Install `setroubleshoot` and `setroubleshoot-server`, then restart the auditd service.

```
yum install setroubleshoot setroubleshoot-server
service auditd restart
```

- The troubleshoot packages will display alerts anytime SELinux blocks something if you are running a desktop environment.
- `sealert` is a tool used to analyze the audit log and generate a report for identified SELinux issues.

Example:

```
sealert -a /var/log/audit/audit.log
```

Output:

```
host# sealert -a /var/log/audit/audit.log
100% done found 1 alerts in /var/log/audit/audit.log
--------------------------------------------------------------------------------

SELinux is preventing /usr/sbin/httpd from getattr access on the file
/myapps/app3/html/index.html.

*****  Plugin catchall_labels (81.6 confidence) suggests  ********************

If you want to allow httpd to have getattr access on the index.html file
Then you need to change the label on /myapps/app3/html/index.html
Do
# semanage fcontext -a -t FILE_TYPE '/myapps/app3/html/index.html'
where FILE_TYPE is one of the following: sssd_var_lib_t, public_content_t,
anon_inodefs_t, httpd_sys_ra_content_t, httpd_sys_rw_content_t, httpd_sys_rw_content_t,
httpd_w3c_validator_content_t.
Then execute:
restorecon -v '/myapps/app3/html/index.html'

*****  Plugin catchall (17.2 confidence) suggests  ***************************

If you believe that httpd should be allowed getattr access on the index.html file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:  <-- SOLUTION STARTS HERE !!!
# grep httpd /var/log/audit/audit.log | audit2allow -M mypol
# semodule -i mypol.pp
```

Additional Troubleshooting Tips

- If your application is not SELinux-compliant, start by setting SELinux to permissive mode. `setenforce permissive` or `setenforce 0`
- Run the application again, and go through normal use of the application.
- Use `sealert` to create a problem report.
  `sealert -a /var/log/audit/audit.log`
- Look at the results, and it will show you the commands to fix the issues (`allow this access for now by executing:`).
- Now set SELinux back to enforce mode (`setenforce enforcing`), and test your application.
- Also use `journalctl -xe` to search through journal entries for clues.
    - `sealert` will provide messages within journalctl to help as well.

Confining Users

- Each Linux user is mapped to an SELinux user, inheriting restrictions placed on SELinux users.
- The purpose of this is to protect the system from the users.
- To view the mapping of Linux users to SELinux users, use:

```
host# semanage login -l
Login Name           SELinux User         MLS/MCS Range        Service

__default__          unconfined_u         s0-s0:c0.c1023       *
root                 unconfined_u         s0-s0:c0.c1023       *
system_u             system_u             s0-s0:c0.c1023       *
```

- By default, Linux users are mapped to SELinux's `unconfined_u` user.
- Linux users are also mapped to the `__default__` login by default.
- Predefined SELinux users:
  - `sysadm_u`
  - `staff_u`
  - `user_u`
  - `guest_u`
  - `xguest_u`
- Processes run by SELinux users are run under a specific domain.
- The domain naming convention is to switch the `_t` with a `_u`.
  - **Example:** The SElinux user `unconfined_u` runs processes in the `unconfined_t` domain.
- To identify a Linux user's mapped SELinux user, run the `id -Z` command.

```
host# id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

- To assign an SELinux user at the time of Linux account creation, run:

```
useradd -Z user_u useruser`
~]# semanage login -l

Login Name           SELinux User         MLS/MCS Range        Service

__default__          unconfined_u         s0-s0:c0.c1023       *
root                 unconfined_u         s0-s0:c0.c1023       *
system_u             system_u             s0-s0:c0.c1023       *
useruuser            user_u               s0                   *`
```

- To map an existing Linux user to an SELinux user, run:
  `semanage login -a -s user_u existinguser`
  - `-a`: Adds a new record
  - `-s`: Specifies the SELinux user to map a Linux user to
- To remove the mapping between an SELinux user and a Linux user, run:
  `semanage login -d linuxuser`

Using Booleans with SELinux Confined Users

- Change boolean values with the `setsebool` command.
- Add the `-P` switch to make the changes persistent across reboots.
  - Prevent Linux users in the `guest_t` domain from executing applications in their home directories and `/tmp`:
    `setsebool -P guest_exec_content off`
  - Prevent Linux users in the `user_t` domain from executing applications in their home directories and `/tmp`:
    `setbool -P user_exec_content off`
  - Notice a pattern? (`x_exec_content off`)
  - Use the `on` keyword to turn it back on:
    `setbool -P user_exec_content on`

Scenario: We need to change the default port for Apache from port 80 to port 8888 for our developers, who are creating a web application that will run on port 8888.

Solution:

1. `yum install -y httpd elinks`
2. `elinks http://localhost:80`
3. Make sure SELinux isn't using port 8888 for anything else.
   `semanage port -l | grep 8888`
4. `nano /etc/httpd/conf/httpd.conf`
5. `Listen 8888`
6. Restart the httpd service.
   `systemctl restart httpd`
   (It should fail.)
7. Investigate why via `journalctl -xe`.

```
*****  Plugin bind_ports (92.2 confidence) suggests   ************************

If you want to allow /usr/sbin/httpd to bind to network port 8888
Then you need to modify the port type.
Do
# semanage port -a -t PORT_TYPE -p tcp 8888
where PORT_TYPE is one of the following: http_cache_port_t, http_port_t, jboss_manag
```

8. Identify the `PORT_TYPE` for port 80.
   ```
   semanage port -l | grep http
   ```
9. Define the port type in the suggested fix.
   ```
   semanage port -a -t http_port_t -p tcp 8888
   ```
10. Test.
    ```
    elinks http://localhost:8888
    ```
11. Success!