ProLUG Course Books

Produced by Scott Champine and ProLUG

Scott Champine

Table of contents

1. Home

1.1 The Professional Linux Users Group (ProLUG)

ProLUG

Welcome to The Professional Linux Users Group, hopefully this stuff is useful to ya

ProLUG.org

ProLUG Big Book of Labs

Labs from @Het_Tanis himself

Het's Big Book of Labs

Join The ProLUG Discord

Come join a group of dedicated Systems enthusiasts on Discord!

Join us on Discord!

Beginners

If you are new and just want to get oriented

→ Start Here

Course Books

ProLUG welcomes all prospective contributors, if you'd like to help out check in at the repo!

Course Books Repo

1.1.1 In the Beginning

Founded approximately 15 years ago, the Professional Linux User Group (ProLUG) began as a vision of Het Tanis, known by his community alias 'Scott Champine.' Het identified the need for an informal yet structured space where Linux professionals could share knowledge, collaborate, and grow together. What started as local in-person meetups quickly gained traction, thanks to the increasing demand for open-source collaboration and the widespread adoption of Linux in both enterprises and personal projects.

1.1.2 Why ProLUG Started

ProLUG was born out of the recognition that Linux professionals often face challenges that are best solved through peer collaboration and hands-on experience. The community's founding principles were rooted in creating an environment where newcomers could learn from experienced professionals, and seasoned users could gain exposure to advanced topics and emerging technologies. Its core mission was simple yet impactful: to provide continuous growth opportunities in Linux system administration, automation, and cloud technologies.

Some of the key motivations behind ProLUG's formation include:

- **Peer Support:** Helping members solve technical challenges through discussion and advice from experts.
- **Knowledge Sharing:** Encouraging open sharing of tips, tricks, configurations, and scripts related to Linux and open-source tools.
- **Hands-on Learning:** Providing access to practical labs, exercises, and real-world scenarios for hands-on training.
- **Community Mentorship:** Offering a space for members to mentor and be mentored by others in different stages of their careers.
- **Certification Prep:** Assisting members in preparing for recognized industry certifications.

1.1.3 The Expansion into an Online Community

While initially focused on local in-person meetings, ProLUG embraced online platforms to extend its reach globally. The switch to a virtual model enabled:

- **Global Networking:** Professionals and enthusiasts from around the world could now connect, learn, and collaborate without geographical limitations.
- **24/7 Discussion:** Via platforms like Discord, members could share insights, discuss Linux problems, and exchange ideas anytime, anywhere.
- **Greater Diversity:** The online expansion diversified the member base, incorporating individuals from various industries and technical backgrounds, creating a rich environment for problem-solving.

1.1.4 Interactive Labs and Training Programs

One of ProLUG's most successful expansions has been its focus on interactive, hands-on labs. To bridge the gap between theory and practice, Het Tanis launched a series of labs on platforms like Killercoda, covering a variety of topics including:

- **Linux Essentials and System Administration**
- **Ansible Automation**
- **Kubernetes and Container Orchestration**
- **Security and Network Hardening**

With over 50 interactive labs available and more being continuously developed, members benefit from practical scenarios that simulate real-world challenges. The labs cater to beginners, intermediates, and experts, ensuring everyone has something to gain.

1.1.5 Certification and Career Development

In 2024, ProLUG launched its first structured certification course: **Enterprise Linux Administration**. This program was designed to provide a comprehensive curriculum covering topics such as:

- Advanced Linux system configuration
- Enterprise networking and services
- Security management
- Scripting and automation

The first cohort of graduates successfully completed the program in January 2025, marking a major milestone in ProLUG's commitment to professional development. Many graduates have reported success stories, such as landing new jobs, securing promotions, or gaining confidence in their Linux expertise.

1.1.6 What is a User Group?

A **user group** is a community of individuals who come together to share common interests, typically in a specific area of technology, such as Linux. These groups can be local or online and serve as platforms for:

- **Collaboration:** Members work together to troubleshoot, build projects, and share experiences.
- **Networking:** Opportunities to connect with professionals, mentors, and employers within the field.
- **Learning:** Workshops, presentations, and discussions that cover new and emerging technologies.
- **Career Growth:** Access to resources, training programs, and job opportunities.

ProLUG is a prime example of how a user group can grow beyond its initial purpose, evolving into a vibrant global community with practical learning opportunities and real-world outcomes.

1.1.7 Success Stories

Being part of ProLUG has proven highly beneficial for many members, with success stories ranging from career advancements to personal growth:

- **Job Opportunities:** Members have found jobs in system administration, DevOps, and cloud engineering roles through networking within ProLUG.
- **Certifications:** Many members have successfully obtained Linux-related certifications, including RHCSA, RHCE, and LFCS, using ProLUG's resources and mentorship programs.
- **Skill Development:** Through interactive labs and group discussions, members have honed skills in automation (Ansible), scripting (Bash, Python), containerization (Docker, Kubernetes), and more.
- **Mentorship Relationships:** Senior professionals have mentored newcomers, creating a cycle of continuous learning and knowledge sharing.

1.1.8 Current Milestones

- **3,000+ Members:** ProLUG's global community continues to grow rapidly, attracting Linux enthusiasts and professionals from various backgrounds.
- **50+ Interactive Labs:** Covering diverse topics, from basic Linux administration to advanced enterprise systems management.
- **Ongoing Training Programs:** Continuous updates to certification preparation courses, interactive workshops, and guided lab exercises.

ProLUG's commitment to fostering a collaborative environment has made it a go-to community for anyone interested in Linux. Whether you're a beginner looking to learn the basics or an experienced professional aiming to advance your career, ProLUG offers a pathway to success.

1.2 Getting Started

Welcome to the Professional Linux Users Group! You're likely here because you want to take a structured approach to learning Linux. Our community is passionate about sharing knowledge. We come together voluntarily to create these web-books as companion materials for our interactive courses.

We call them interactive because participation with your peers during the course time frame is required to earn a certificate. That said, you are also free to follow along with the materials at your own pace. Inside each book, you'll find everything you need: labs, worksheets, and recorded lecture videos.

A Bit About Linux

Linux, a **UNIX-like operating system**, is built around the **kernel**—the core component that acts as a bridge between hardware and software. The kernel manages resources such as memory, processes, and devices, while we as users interact with it indirectly through a **shell**, entering commands that the system interprets and executes.

On top of the Linux kernel sit many different **distributions (distros)**. A distribution is a curated package that typically includes:

- The Linux kernel
- A collection of software tools and utilities (typically called the Operating System)
- Package management systems
- Optionally, a desktop environment for graphical interaction

These distributions vary in design and purpose -- some focus on ease of use, others on performance, stability, or security. Whether you choose Ubuntu, Fedora, Arch, or another distro, the underlying kernel remains the same.

Once you learn Linux fundamentals, your skills are portable across nearly all distributions.

1.2.1 1. Learning Basic Linux Skills

The best way to understand Linux is through hands-on practice. This is sometimes called Time on Tools (TOT)—the time you spend directly working with the commands, files, and systems yourself. Reading guides and watching tutorials are helpful, but it's the act of typing commands, troubleshooting mistakes, and seeing the results that makes the knowledge stick.

Killercoda

Killercoda is a website that hosts interactive labs developed by many different creators. Scott Champine (Het Tanis) has created a number of labs including the basics of Linux. By creating an account on `Killercoda` you will be able to gain hands on experience with the `Linux Command Line Terminal` through your web browser on any computer free of charge.

→ Linux Labs by Het

Joining the Discord

Our Discord server holds regular scheduled events where one can actively or passively participate. It is also a place to ask questions or get involved in projects.

→ Link to Discord

"Beginners Start Here" Playlist

→ Het_Tanis' Full "Beginners Start Here" Playlist

1.2.2 2. Setting Up a Local Linux Install

Killercoda is a great way for newcomers to dip their toes into Linux. However, because Killercoda creates **ephemeral virtual environments**, everything is temporary — you can't keep your progress or return to the same setup later. The next logical step is to set up something more **permanent**.

There are several ways you can run Linux at home:

- **Virtual Machine (VM)**: Run Linux inside Windows or macOS using software like VirtualBox or VMware.
- **Windows Subsystem for Linux (WSL)**: A convenient way to run Linux alongside Windows without a full VM.
- **HomeLab with a Type 1 Hypervisor (e.g., Proxmox)**: Run multiple virtual machines or containers on one dedicated server.
- **Direct Install**: Put Linux on an old desktop, laptop, or single-board computer (like a Raspberry Pi).
- **Virtual Private Server (VPS)**: Rent a remote Linux server from a hosting provider.

Each approach has pros and cons depending on your goals:

- Some setups include a **desktop environment** (a Graphical User Interface/GUI), while others are **headless** (no graphical interface, only command-line).
- A dedicated hypervisor can host **multiple test environments**, while a direct install gives you a **single dedicated machine**.

Take some time to research which option best fits your needs, budget, and comfort level.

Some resources to get you started with a local Linux installation:

- Ubuntu Server Install (no GUI): https://ubuntu.com/tutorials/install-ubuntu-server#1-overview
- Ubuntu Desktop Install (with a GUI): https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
- Rocky Linux Install (with a GUI): https://krython.com/post/installing-rocky-linux-9-complete-installation-guide

1.2.3 3. Participation in a Course

The ProLUG Certified Enterprise Linux Administration Course is meant to teach you the fundamentals of Linux. It is a long and involved course at 16 weeks with roughly 160 hours of contact time. Each unit has worksheets and labs to complete, along with discussion posts to participate in.

To see when the course is actively running, check the events in Discord.

The course book is available for anyone to complete at any time. However, participation during an active course run is the only way to receive the certification.

→ Linux Admin Course Book

1.3 Qualifying for the Certification

Each course run through the Professional Linux Users Group (ProLUG) allows you to earn a certification upon completion.

Certificates are awarded to those who complete the course within the timeframe that it is being run through the ProLUG Discord.

- To see when courses are running, join the ProLUG Discord server and check the Events section.

If you aim to earn the certification for completing this course, you must follow the guidelines set forth in this document.

There are four main components to earning the certification.

1. Worksheet Completion
2. Discussion Questions
3. Lab Completion
4. Final Project

1.3.1 Worksheet Completion

Each unit has a corresponding worksheet.
On this worksheet are discussion questions, terms/definitions, **optional** "digging deeper" sections, and reflection questions.

These worksheets must be filled out and kept until the end of the course.
Upon reaching the end, they are to be submitted to the instructor (Scott Champine).

1.3.2 Worksheet Submission Format

The format in which you submit these worksheets is up to you.

Some students prefer to keep them in a GitHub repository, others prefer to just keep them as files on their machines and submit via email.

1.3.3 Discussion Questions

Each unit's worksheet contains multiple discussion questions.
Each discussion question has its own thread in the ProLUG Discord server, in the `#course-discussion-posts` forum channel.

To qualify for certification, you must:

- Post your answer to each discussion question in the correct thread.
- Respond to another student's answer in the same thread.

The goal of this is not to create busywork, but to spark discussions and see things from other points of view.

1.3.4 Lab Completion

Each unit has a lab that is to be completed.

The labs, like the worksheets, should be also completed and saved until the end of the course.

These labs should be submitted to the instructor along with the worksheets in the same format of your choice.

1.3.5 Final Project

Each ProLUG course has students complete a capstone project.

This is a requirement for earning a ProLUG course certification.

The project must meet the standards set forth in the Final Project Outline (or otherwise be approved by the instructor, Scott Champine).

1.4 Contributing to ProLUG MkDocs

1.4.1 Contributing to the ProLUG Linux Course Books

The Professional Linux Users Group (ProLUG) provides a set of requirements and guidelines to contribute to this project. Below are steps to ensure contributors are adhering to those guidelines and fostering a productive version control environment.

> **Using AI for Contributions**
>
> We **do not** welcome AI-generated contributions on this project. See our AI/LLM Contribtion Policy for more details.

How to be a Successful Contributor

To be an effective contributor, understanding Git, whether through the command line or an external tool, will be an important part of contributing. To this effect it is important that any individual who contributes to this project have a working understanding of committing, merging, and other fundamental Git workflows.

For clarity, this project utilizes GitHub for remote repositories and CI/CD testing pipeline workflows. Git and GitHub are two separate entities where GitHub provides the storage and hosting services, and Git provides the version control.

Prospective contributors are directed to several resources should they feel their competency with Git or GitHub falls short:

Git documentation and GitHub video tutorials:

- ByteByteGo's Git Explained in 4 Minutes (4m)
- Fireship's How to use Git and Github (12m)
- freeCodeCamp's Git and GitHub Crash Course (1hr)

AI/LLM Contribution Policy

The use of AI, LLMs in particular, can take the burden off of the contributor when it comes to generating content.

However, prospective contributors should be aware that AI-generated contributions are **not welcome** on this project.

Acceptable uses of AI/LLMs for this project include:

- Getting general ideas
- Spell checking
- Flow checking

Using AI to **aid** in the writing process or to check your work is acceptable.
For example, if English is not your first language and you'd like to check your work for syntax or spelling errors, AI may be used for this purpose.

If you do employ the assistance of AI for your contribution, we ask that you explain how you used AI, and to what extent, in your pull request description.

Any pull requests that are found to have been generated by AI may be rejected.

Determining if the contribution is AI-generated will be at the discretion of reviewers.

> **Note**
>
> Anybody can participate in code review, it is not limited to maintainers. We encourage everyone to review each other's pull requests. To review a pull request, simply go to the pull request's "Files Changed" tab and leave a review.

If any reviewer believes that the contribution may have been AI-generated, they may post a change request with a comment stating that they believe it came from AI.
If the reviewer is also a maintainer, they may reject the pull request.

Learning Markdown

This project uses Markdown, which is a markup language used to create formatted text using a plaintext editor. Many developers and engineers have switched to using Markdown to write technical documentation.

It's easy to learn, and any prospective contributor will be expected to understand basic markdown syntax.

Below are some resources to get you started with Markdown:

- MkDocs Markdown Overview
- Commonmark Markdown Cheatsheet
- Dillinger, a Web-based Markdown Editor and Previewer
- HowToGeek: Brief Overview of Markdown

Signing your Git Commits with SSH

Contributors who elect to contribute through the command line will need to verify their identities before their commits can be accepted. **This step is not required if contributors will be submitting changes via GitHub.com itself** since users will have verified their identities with GitHub's own verification process.

To reiterate, individuals contributing via command line will need to sign their commits through SSH. Signing GitHub commits helps ProLUG validate incoming commits from trusted contributors that reside outside the GitHub ecosystem. It can be quite trivial to impersonate users on GitHub and it is in the best interest of the project and contributors to observe this security practice.

It should also be noted that GitHub supplies tools like GitHub CLI that abstract away the process of signing and verifying commits from the command line. GitHub provides a `gh auth login` function to facilitate the procedure which contributors can employ without the necessary changes suggested below.

To Sign your Git Commits with SSH:

Generate an SSH key pair if you don't have one:

```
ssh-keygen -t ed25519
```

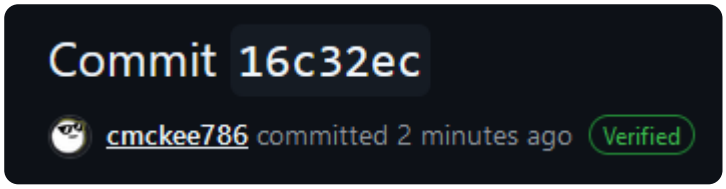
Add SSH public key ('.pub' suffix) to GitHub as "Signing Key".

* GitHub.com -> Profile -> Settings -> GPG and SSH Keys -> Add SSH Key -> Drop down -> Signing Key

Below is a bash script that will attempt to configure signing Git commits on a localhost:

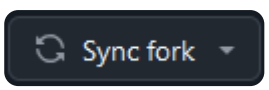
```
#!/bin/bash
GH_USERNAME="YourUsername"
git config --global gpg.format ssh
git config --global user.signingkey ~/.ssh/id_ed25519.pub
git config --global tag.gpgSign true
git config --global commit.gpgSign true
mkdir -p ~/.config/git
touch ~/.config/git/allowed_signers
echo "${GH_USERNAME} $(cat ~/.ssh/id_ed25519.pub)" > ~/.config/git/allowed_signers
git config --global gpg.ssh.allowedSignersFile ~/.config/git/allowed_signers
# Make a commit to verify
git log --show-signature -1
```

Make a commit after running those commands and then use `git log --show-signature -1`. You should see something like `Good "git" signature for <yourname> with ED25519 key SHA256:abcdef...` if it worked.



Your commits should now be verified from your account. This helps us ensure that valid users are contributing to this project. Unverified commits will be scrutinized and likely discarded.

Syncing your Fork with the Upstream ProLUG Repo



In an effort to minimize merge conflicts we strongly suggest forks remain up to date with the original repository before committing changes. This will help us reduce pull request management overhead.

> **Warning**
>
> Pull requests with substantial merge conflicts may be rejected or marked for change requests.

You can do this from the GitHub web UI easily with the `Sync Fork` button. If you want to do this from the terminal, you can add a new `git remote` called `upstream`.

```
git remote add upstream https://github.com/ProfessionalLinuxUsersGroup/course-books.git
```

Then, to sync your local fork with the original repo, do a `git pull` from the `upstream` remote.

```
git pull upstream main
```

This fork should now be up to date with the original upstream repository.

Basic Contribution Workflow

You'll create your own fork of the repository using the GitHub web UI, create a branch, make changes, push to your fork, then open a pull request.

The basic steps to contribute are outlined below.

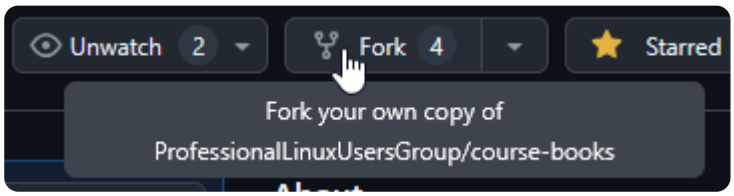
1. COMMENT FIRST

If you'd like to work on a specific worksheet or lab, please let us know first by commenting on the issue so you can be assigned to it. This way, other contributors can see that someone is already working on it.

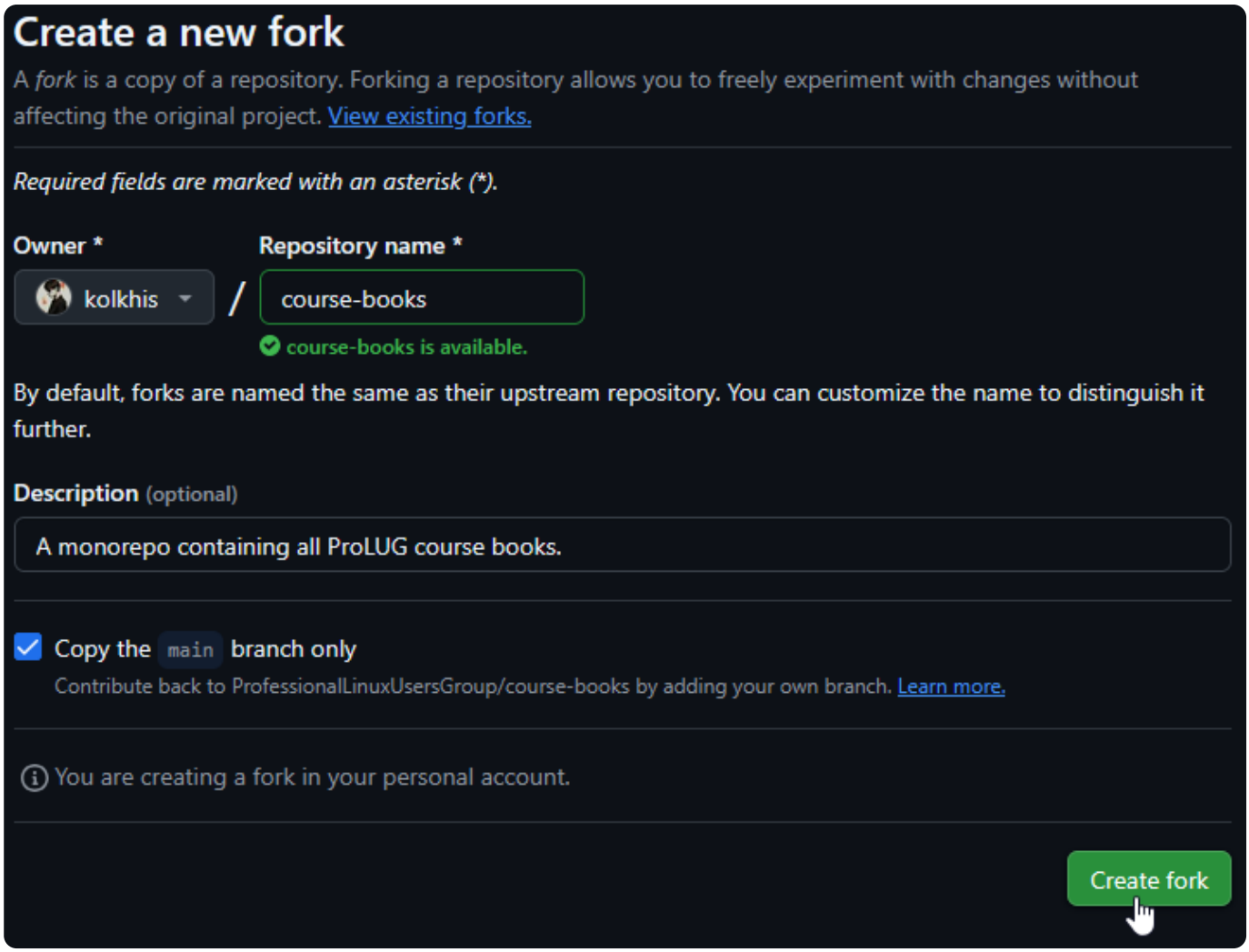
This helps the repository maintainers and contributors attain a high degree of visibility and collaboration before merging changes.

2. CREATE A FORK

Go to the original repository link. Click on "Fork" on the top right.



Then click "Create Fork" on the next page.



Now you'll have your own version of the repository tied to your GitHub account.

3. CLONE THE FORK TO YOUR LOCAL MACHINE

After creating your fork, you'll need to clone it down to your local machine in order to work on it.

```
git clone git@github.com:YOUR_USERNAME/course-books.git
# Or, with https:
git clone https://github.com/YOUR_USERNAME/course-books.git
```

4. CREATE A NEW BRANCH

Whenever making changes contributors are highly encouraged to create a branch with an appropriate name. Switch to that branch, then make changes there.

Branch Naming Convention

Our branch naming convention is as follows:

```
<book>-<unit>-<action>
```

If you are making a change to something that affects the entire project rather than just a specific book, give it a descriptive name.

For example, if you're working on adding the Unit 1 Worksheet for the Linux Admin Course book:

```
git branch lac-unit1-add-worksheet
git switch lac-unit1-add-worksheet
# Or, simply:
git switch -c lac-unit1-add-worksheet
```

5. MAKE CHANGES AND COMMIT

Once you're on your new branch, make changes to the `u1ws.md` using the editor of your choice.

```
vi u1ws.md
# Make changes
:wq
```

Once the changes are made, commit them.

```
git add u1ws.md
git commit -m "feat: Add lac unit 1 worksheet"
```

> **Commit Message Convention**
>
> Your commit message should be structured following the conventions laid out here: https://www.conventionalcommits.org/en/v1.0.0/#summary

6. PUSH THE CHANGES

After making your commit, you can now push the changes to **your fork** on the **new branch** you created earlier.

```
git push origin lac-unit1-add-worksheet
```

This will update your forked repository on GitHub to contain the new branch with the new changes.

7. CREATE A PULL REQUEST

> **Local Testing**
>
> We ask that you test your changes locally before opening a pull request. Our development page outlines how to test locally.

Now you'll be able to open a pull request.

GitHub should be smart about detecting new changes and prompt you to open a pull request upon visiting the original repository or your own fork.

You can also go to the original repository link, go to the "Pull Requests" tab, and create a new pull request.

After starting your pull request, select your branch `lac-unit1-add-worksheet`, create a description, and mention an issue by number, prefixed with `#` (e.g., `#5`).

Consider a few Useful Practices

The practices presented below are not required to contribute to the ProLUG course books but can streamline contributing to any project and are considered to some as incredibly useful when engaging in version control with Git.

GIT WORKTREES

A notably useful workflow provided by Git is the `git worktree`. This allows the instantion of multiple working directories within a repository that point to a particular branch for any number of reasons without needing to clone the base respository separately.

Suppose one worktree is created to work on feature "x" on branch 'feat', and a separate worktree is implemented for bug "y" on branch 'bug', all localized in the same repository. A git worktree could even be utilized to quickly checkout a pull request in its own separate directory within the repository facilitating downstream commands like pushing over changed files to a host for testing.

If the branch does not already exist this command will make a local branch and directory within the main repo. Otherwise you can quickly checkout a remote branch and create a directory within the main repo.

```
git worktree add -b {branch to create} {directory to be created}
```

Here we can see multiple copies of the primary repo but checked out under different branches in different directories. The utility in this is that we can work on multiple branches without disturbing the primary while maintaining a source of truth, all contained within one directory on the host. This facilitates things like easily rebasing a separate branch cleanly from the main branch, testing file changes, repo asset package changes, and more.

```
git worktree add test-branch test-branch
# Or create a branch if it doesn't exist
git worktree add -b test-branch ../test-branch-dir

├── dev-cleanup # Separate branch
│   ├── docs
│   ├── ref
│   └── scripts
├── docs        # Primary repo branch
│   ├── assets
│   ├── beginners
│   ├── lac
│   ├── pcae
│   ├── psc
│   └── stylesheets
├── fixups      # Separate branch
│   ├── docs
│   ├── ref
│   └── scripts
├── ref
├── scripts
│   └── link-storage
```

Git worktree documentation.

GIT REBASING

> **Warning**
>
> Do not rebase commits that exist outside your repository and that people may have based work on. Rebase only within your own branches and forks, never onto public branches or repos.

In sum, rebasing can be utilized to replay commits onto a currently checked-out branch if such branch is behind in commits from its upstream branch. This allows circumvention of potentially hard to read merge commits in a busy repository. As such, proper implementation of rebasing can leave a clean, and easily readable commit history for all concerned parties. Rebasing can also facilitate the management of branches and working directories in a notably active project in a myriad of other ways. Contributors are encouraged to explore the possibilties of rebasing.

```
git checkout experiment
git rebase master
First, rewinding head to replay your work on top of it...
Applying: added staged command

# Or even a simple pull
git pull --rebase # (1)
```

1. The default `git pull` behavior can be changed to rebase instead of merging which won't require a `--rebase` flag; `git config --global pull.rebase true`. Find out more here: https://git-scm.com/docs/git-config#Documentation/git-config.txt-pullrebase

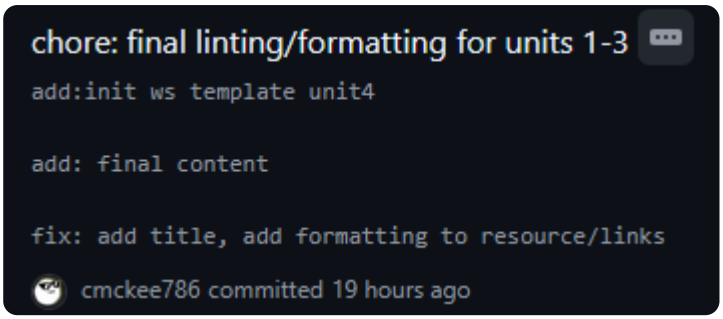
Rebasing also plays a role in facilitating any commit reverts that may need to be made in the future. More on that will follow.

Git Rebasing documentation.

COMMIT EARLY, OFTEN, AND SQUASHING COMMITS

It is great practice to commit early, and often. This however can produce hard to read commits for repo maintainers and contributors. Squashing commits, which is a type of rebasing, can be utilized to compress a large number of commits made in a local repository before being pushed upstream to a remote repository and eventual pull request.

Below is an example of 4 local commits squashed into a single commit that was pushed remotely:



Squashing commits can improve readability, but its primary utility, especially for larger projects, may be in addressing an event where rolling back several commits due to a bug or test can be done with a single commit revert.

freeCodeCamp has a great write-up on this procedure. When done appropriately this can greatly facilitate the development process. Contributors are strongly encouraged to begin exploring these types of workflows if they never have.

GIT STASHING

Another useful practice is to employ "stashing" uncommitted files in a local repository. This is useful in many contexts including stashing local changes to resolve recently introduced remote vs. local repo conflicts, or quickly switching working spaces.

Stashing effectively unstages any changes made in the local repo and saves them to be applied later. This can further help facilitate a rebase or merge before committing changes upstream for instance.

More on this here:

- https://www.atlassian.com/git/tutorials/saving-changes/git-stash
- https://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning

Adding Assets

If you'd like to add an image to one of the pages, you will need to save that image into the `docs/assets/` directory.

Each book has its own subdirectory in `assets/`:

```
assets/
├── deploy
├── downloads
├── images
├── lac
├── pcae
└── psc
```

- Notice the `lac`, `psc`, and `pcae` directories. These are where course-specific assets belong.

If you're attempting to add an image to a page in the Linux Admin Course book, you'd move that image to the `docs/assets/lac/images/` directory.

It's preferable that each unit of each course also has its own directory.

For example, if you're adding a diagram to the intro page in unit 6 of the Linux Admin Course book, this would go in `/docs/asset/lac/images/u6/`. Giving the image file a descriptive name also helps (e.g., `firewall-connection-diagram.png`).

This convention helps us keep track of assets and where they belong.

Continuing with that example, this is the path that the image should go in:

```
docs/
└── assets/
    └── lac/
        └── images/
            └── u6/
                └── firewall-connection-diagram.png
```

Once you've added the asset to the appropriate directory, you need to reference it in the page itself. We're adding a diagram to the Unit 6 Intro of the Linux Admin Course, so we'd add it to `docs/lac/u6intro.md`.

This can be done with a raw HTML `<img>` element:

```
<img src="../../assets/lac/images/u6/firewall_connection_diagram.png"></img>
```

This is a **relative path** to the image.

- When determining either the relative or absolute path of the image, use `mkdocs build` and examine where the asset file is and where the target page is.
- For example, `docs/lac/u6intro.md` will end up in `/site/lac/u6intro/index.html`
- So the relative path from here would be `../../assets/lac/image/u6/firewall_connection_diagram.png`

> **Note**
>
> The markdown syntax `![alt text](/path/to/image)` also works. However, to maintain consistency, we ask that you use the HTML version.

After adding that, your new asset should be ready to go!

To minimize the strain on maintainers, we ask that you test your change locally before opening a pull request.

Supporting Material

Below are links to the necessary materials to build out the course templates:

- Look over the template pages wiki, or directly here:
- Reference Pages: intro, bonus, lab, worksheet

1.4.2 Local Development and Testing

It is strongly encouraged that contributors test their changes before making commits.

To help facilitate this process, we have provided a local build script that can be used to test changes locally.

If the contributor prefers to configure their environment on their own, a set of instructions and guidelines are provided below. These guidelines are by no means a requirement or the only set of procedures to locally develop on this project.

The examples, code, and commands provided below were developed using such technologies as containers, bash scripts, and more.

Using the Provided Build Script

If you are on a Linux machine or another machine that has access to Bash and Python, you may use the script provided in the `scripts/` directory.

This script should be run from the root directory of the project:

```
git clone https://github.com/ProfessionalLinuxUsersGroup/course-books.git
cd course-books
./scripts/local-build
```

Use the `--help` option to see usage instructions:

```
./scripts/local-build --help
```

This is a Bash script that requires Bash version 4.4+, Python 3.10+, as well as the `venv` and `pip` Python modules.

This script will create a Python virtual environment for the project, install the necessary dependencies, and serve the site locally via HTTP. A link will be output to the terminal that you can use to see the local version of the website.

Any changes made while this script is running will reload the website in real-time.

When you are done testing, use `Ctrl-C` (`SIGINT`) to stop the script.

> **Reporting an Issue**
>
> Any issues or errors encountered with this script can be reported by opening an issue in the project's repository.

Building Manually

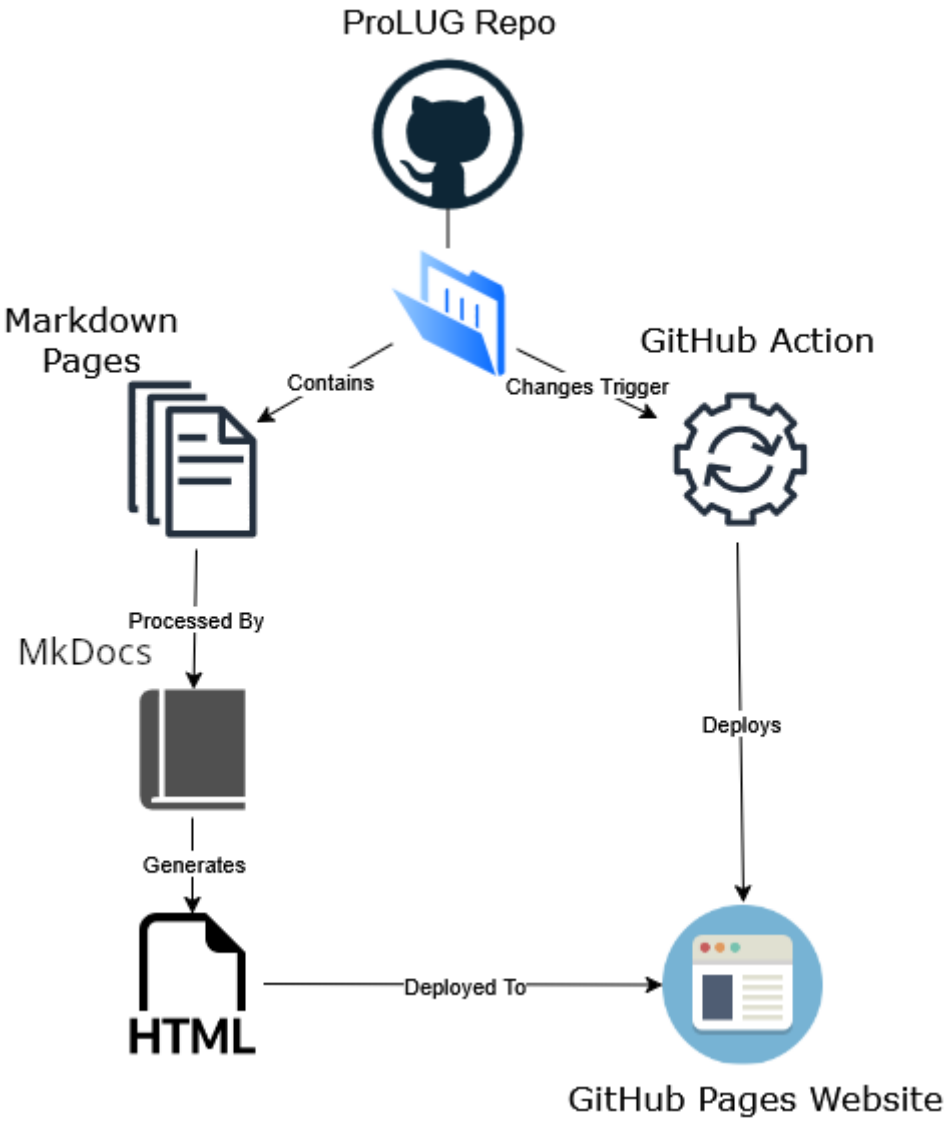
If you prefer to build out your own test environment, you can do so by following the instructions below.

BUILD DEPENDENCIES

The ProLUG course books utilize Material for MkDocs, a friendly and popular markdown utility that quickly exports static web files for documentation or general website use cases.

Utilizing Material for MkDocs, this course then deploys the exported web structure to a GitHub Pages workflow and runner that then produces an easily navigable website.

Below is the current workflow that deploys the Git Page for the courses:



To begin developing locally on this project Material for MkDocs has provided a pre-baked Docker image. Or contributors can utilize a virtual python environment with the required dependencies.

Local Dependencies

- `httpd` or `apache2`
- `git`
- `weasyprint`
- Python 3.9+
- Python virtual environment with `mkdoc-material` required pip packages
- a clone of the ProLUG course-books repository

To achieve this deployment locally without deploying the suggested Materials for MkDocs Docker image the following environment and dependencies are required: A localhost, this could be a container, virtual machine, or local machine, and the following packages installed on such machine.

More information to get started is provided by Material for MkDocs here: https://squidfunk.github.io/mkdocs-material/getting-started/

BUILDING, DEPLOYING, AND DEVELOPING LOCALLY

Below is a set of scripts that can quickly achieve this environment in an automated fashion if contributors choose to forgo utilizing the provided build script.

These commands assume a `root` user. These scripts will update host package repositories to their latest offerings and download missing dependencies, insantiate the python virtual environment required to build the project, process and produce the necessary `.html` files from the course book source files, and deploy the website either via `httpd/apache2` or served via the built-in development server.

Outside of system packages all files will be localized to the `/root/course-books` directory on the container or machine.

Tested on: Rocky 10.1 LXC (550MB of packages after install) Ubuntu 25.04 LXC (850MB of packages after install)

APT DNF

```
#!/bin/bash
apt-get update && apt-get -y install git python3-full hostname apache2 weasyprint
git clone https://github.com/ProfessionalLinuxUsersGroup/course-books
cd course-books
python3 -m venv venv
source venv/bin/activate
pip install -U pip
pip install -U mkdocs mkdocs-material mkdocs-glightbox mkdocs-to-pdf
# use for live reloading after changes:
#   mkdocs serve -a "$(hostname -I | awk '{print $1}'):8000"
# use for local webserver:
#   mkdocs build -d /var/www/html/ && systemctl enable --now apache2
```

```
#!/bin/bash
dnf install -y git python3 python-pip pango hostname httpd weasyprint
git clone https://github.com/ProfessionalLinuxUsersGroup/course-books
cd course-books
python3 -m venv venv
source venv/bin/activate
pip install -U pip
pip install -U mkdocs mkdocs-material mkdocs-glightbox mkdocs-to-pdf
# use for live reloading after changes:
#   mkdocs serve -a "$(hostname -I | awk '{print $1}'):8000"
# use for local webserver:
#   mkdocs build -d /var/www/html/ && systemctl enable --now httpd
```

The ProLUG Linux Course Books website should now be available from your web browser either at http://localhost:{assigned_port} or its designated IP address and port.

From here you can use such commands from your localhost to implement changes:

```
cd "$HOME"/course-books
source venv/bin/activate
mkdocs build -d /var/www/html
systemctl restart {httpd or apache}
```

These commands will switch your shell into the appropriate directory, activate the python virtual environment, execute the necessary mkdocs binary located in its installed virtual $PATH, build the site from the source files, and then finally restart the web server.

From there you should be able to see any changes you have made are reflected.

Or send commands over to a networked container or machine:

> **Note**
>
> To minimize complexity and given the nature of commands over SSH, these commands will need to utilize absolute paths.

```
scp {working directory}/{targeted document} {TARGET_IP}:/root/course-books/{targeted document}
ssh {TARGET_IP} "cd /root/course-books && /root/course-books/venv/bin/mkdocks build -d /var/www/html && systemctl restart httpd"
```

2. Linux Admin Course

2.1 Course Overview

2.1.1 ProLUG Systems Administration for the Enterprise

Welcome to the ProLUG Enterprise Linux Systems Administration Course Book.

This Book

Contains all materials pertaining to the course including links to external resources. It has been put together with care by a number of ProLUG group members referencing original instructional materials produced by Scott Champine @Het_Tanis .

The content is version controlled with Git and stored here: https://github.com/ProfessionalLinuxUsersGroup/course-books

COURSE DESCRIPTION

This course addresses how Linux systems work for administration level tasks inside a corporate environment. This course will explore everything from the administration of a Linux server and fundamental command line tasks to advanced topics such as patching and web administration.

PREREQUISITE(S) AND/OR CO-REQUISITE(S):

Prerequisites: None
Credit hours: N/A
Contact hours: 120 (50 Theory Hours, 70 Lab Hours)

Course Summary

MAJOR INSTRUCTIONAL AREAS

- Server build and Hardware components
- Command Line tools and Syntax
- Basic Scripting
- Linux networking
- Linux security practices
- Automation and repeating tasks
- Implement Networking in Linux
- Troubleshooting
- Benchmarking and Baselining

COURSE OBJECTIVES

- Explain the server build process and hardware system components.
- Analyze system security and implement basic hardening of system.
- Construct command line syntax to explore the system and gather resource information.
- Construct scripting structures of assigning variables, conditional tests, and recording output to generate scripts that do basic system tasks.
- Analyze and troubleshoot the Apache Web Server
- Analyze and troubleshoot the NFS/Samba File Shares.
- Analyze Docker and Kubernetes components and workflows.
- Describe and troubleshoot network services.
- Write and perform Ansible tasks to automate deployments to servers.

Learning Materials and References

Option #1 (Killercoda Machine)

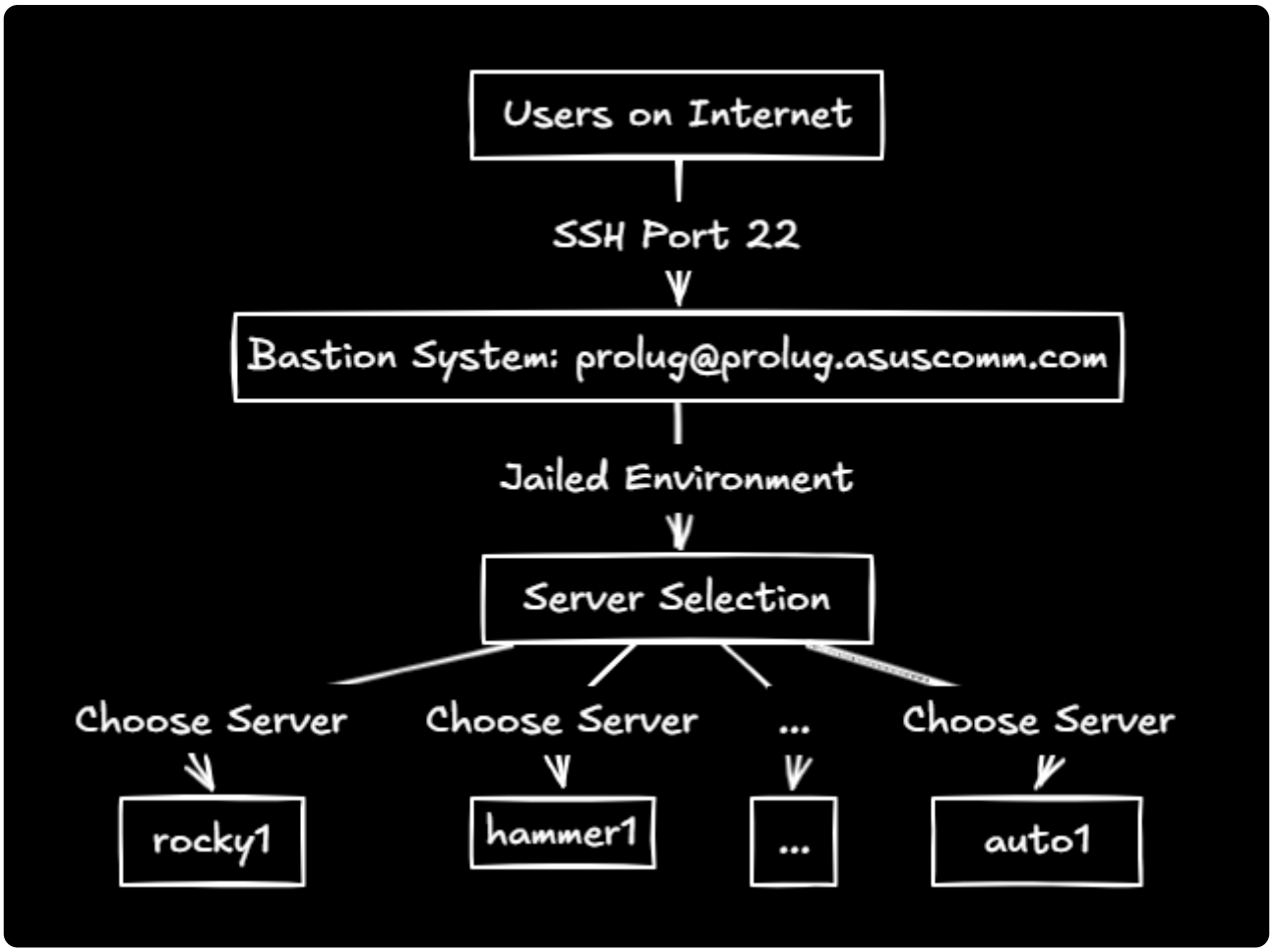
- Cloud Lab server running Ubuntu on Killercoda. Minimal resources can accomplish our tasks
  - 1 CPU
  - 2 GB Ram
  - 30 GB Hard Drive
  - Network Interface (IP already setup)

Option #2 (Home Lab)

- Local VM server running: RHEL, Fedora, Rocky Minimal resources
  - 1 CPU
  - 2GB RAM
  - 3 3-5GB Hard Drives
  - Network Interface (Bridged)

Option #3 (ProLUG Remote Lab)

- ProLUG Lab access to Rocky 9.4+ instance. Minimal resources can accomplish our tasks
  - 1 CPU
  - 4 GB RAM
  - 15 GB Hard Drive
  - 3 x 3GB hard drives (for raid and disk labs)
  - Network Interface (IP already setup)



Course Plan

INSTRUCTIONAL METHODS

This course is designed to promote learner-centered activities and support the development of fundamental Linux skills. The course utilizes individual and group learning activities, performance-driven assignments, problem-based cases, projects, and discussions. These methods focus on building engaging learning experiences conducive to development of critical knowledge and skills that can be effectively applied in professional contexts.

CLASS SIZE

This class will effectively engage 40-60 learners.

CLASS SCHEDULE

Class will meet over weekend (Brown bag) sessions. 1 time per week, for 16 weeks. There will be a total of 16 sessions.

Session	Topic
1	Get Linux Lab Access - CLI Primer - vi/vim/nano basics
2	Essential Tools - Files, Redirects, and Permissions
3	Storage - Logical Volume Management and RAID
4	Operating Running Systems
5	Security - Manage users and groups
6	Security - Firewalld/UFW
7	Security - Patching the system/ Package Management - yum, dnf, rpm
8	Scripting - System checks
9	Docker - K3s Setup and basics
10	K3s advanced w/ microservices
11	Monitoring systems
12	Engineering - System baselining/benchmarking and testing
13	System Hardening
14	Ansible Automation
15	Engineering Troubleshooting
16	Incident Response - Actual incident callout and information gathering

Suggested Learning Approach

In this course, you will be studying individually and within a group of your peers, primarily in a lab environment. As you work on the course deliverables, you are encouraged to share ideas with your peers and instructor, work collaboratively on projects and team assignments, raise critical questions, and provide constructive feedback.

2.1.2 Final Project Outline

Students aiming to complete the Linux Systems Administration course are expected to devise and complete a capstone project, to be turned in at the end of the course.

The instructions, expectations, and deliverables for the project are listed on this page.

Instructions

1. Select a topic to research about a project that you are going to build.
   Topics:
   - System Stability
   - System Performance
   - System Security
   - System monitoring
   - Kubernetes
   - Programming/Automation
2. Plan the project
   - Find documentation or similar projects and build off of what was done there.
3. Document
   - First pass, what does it take to build this?
4. Diagram
   - Draw the thing
     - Excalidraw.com
     - Draw.io
5. Build
   - Get screen shots
   - Make a video?
   - Basically prove you built it.
6. Finalize documentation
   - Redline the documentation
7. Prepare to Present (overleaf.com is a great alternative to Powerpoint)
   - Setup a 15-20 slide deck on what you did
     - Project purpose
     - Diagram
     - Build Process
     - What did you learn?
     - How are you going to apply this?
8. Do any of you want to present?
   - Let me (Scott) know and we'll get you a slot in the last few weeks.

Deliverables

1. Build Documentation for your project that works in either the ProLUG labs, or in the Killercoda environment.
2. A diagram of what you built. This should be both a physical and a logical representation of the system (if applicable).

3. Examples of the running system, screen shots, or other proof that you built it and show it in a running state.
4. A 15-20 slide presentation of the above material that you would present to a group (presenting to us is voluntary, but definitely possible.)

2.1.3 Table of Contents

2.2 Unit 1 - Linux File Operations

2.2.1 Unit 1 - Linux File Operations

Overview

This unit introduces the foundational skills needed for effective Linux system administration with an emphasis on Red Hat Enterprise Linux (RHEL). It covers:

- **Command-Line Proficiency:** Mastery of the shell environment is essential for routine tasks such as navigating the file system, managing processes, and automating scripts.
- **Text Editing with VI/Vim:** Given that many RHEL systems use VI/Vim as the default editor for configuration and scripting, learners are introduced to these tools through practical exercises like using vimtutor and exploring interactive resources (e.g., VIM Adventures).
- **Understanding the Linux File System:** The worksheet emphasizes the standard Linux file hierarchy—critical for managing files, permissions, and services in a Red Hat environment.
- **Basic Utilities and System Management:** Along with the command-line and text editors, the unit touches on fundamental utilities that are pivotal for system configuration, troubleshooting, and maintenance on enterprise systems.

Learning Objectives

1. **Master Command-Line Fundamentals:**
   - Develop proficiency in navigating the Linux command-line interface (CLI) for everyday system management tasks.
   - Learn how to execute commands to manipulate files, directories, and system processes efficiently.
2. **Understand the Linux File System:**
   - Grasp the structure and organization of the Linux file hierarchy.
   - Comprehend how the file system affects system configuration, security, and troubleshooting on Red Hat platforms.
3. **Gain Proficiency in Text Editing with VI/Vim:**
   - Acquire hands-on experience with vi/vim through guided exercises (e.g., vimtutor, VIM Adventures).
   - Learn to edit configuration files and scripts accurately, which is critical for system administration.
4. **Engage with Practical System Administration Tasks:**
   - Explore foundational utilities and commands essential for managing a Linux system.
   - Apply theoretical knowledge through real-world examples, discussion posts, and interactive resources to reinforce learning.

These objectives are designed to ensure that learners not only acquire technical competencies but also understand how these skills integrate into broader system administration practices in a Red Hat environment.

Key terms and Definitions

Linux Kernel	**Command-LineInterface(CLI)**
Shell	**Terminal**
Filesystem Hierarchy	**Package Manager(e.g.,YUM/DNF)**
Text Editors(VI/Vim)	**Sudo**
File Permissions and Ownership	**Processes and Daemons**
System Logs	**Networking Basics**
Bash Scripting	

2.2.2 Unit 1 Worksheet - Linux File Operations

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- What is Vim?
- The Linux Foundation
- Linux CLI Cheatsheets

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u1_worksheet(`.txt`)
- 📥 u1_worksheet(`.docx`)

UNIT 1 RECORDING

Link: https://www.youtube.com/watch?v=eHB8WKWz2eQ

Discussion Post #1

Using a 0-10 system, rate yourself on how well you think you know each topic in the table below. (You do not have to post this rating).

Skill	High (8-10)	Mid (4-7)	Low (0-3)	Total
Linux				
Storage				
Security				
Networking				
Git				
Automation				
Monitoring				
Database				
Cloud				
Kubernetes				
Total				

Next, answer these questions here:

1. What do you hope to learn in this course?
2. What type of career path are you shooting for?

Discussion Post #2

1. Post a job that you are interested in from a local job website. (link or image)
2. What do you know how to do in the posting?

3. What don't you know how to do in the posting?
4. What are you doing to close the gap? What can you do to remedy the difference?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

START THINKING ABOUT YOUR PROJECT IDEAS (MORE TO COME IN FUTURE WEEKS)

Topics:

1. System Stability
2. System Performance
3. System Security
4. System monitoring
5. Kubernetes
6. Programming/Automation

You will research, design, deploy, and document a system that improves your administration of Linux systems in some way.

Definitions

Kernel:

Kernel Args:

OS Version:

Modules:

Mount Points:

Text Editor:

Digging Deeper

1. Use vimtutor and see how far you get. What did you learn that you did not know about vi/vim?
2. Go to https://vim-adventures.com/ and see how far you get. What did you learn that you did not already know about vi/vim?
3. Go to https://www.youtube.com/watch?v=d8XtNXutVto and see how far you get with vim. What did you learn that you did not already know about vi/vim?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

2.2.3 Unit 1 Lab - Linux File Operations

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Top 50+ Linux CLI Commands

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u1_lab(`.pdf`)
- 📥 u1_lab(`.docx`)

Pre-Lab Warm-Up

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

```
mkdir lab_essentials
cd lab_essentials
ls
touch testfile1
ls
touch testfile{2..10}
ls

# What does this do differently?
# Can you figure out what the size of those files are in bytes? What command did you use?

touch file.`hostname`
touch file.`hostname`.`date +%F`
touch file.`hostname`.`date +%F`.`date +%s`
ls

# What do each of these values mean? `man date` to figure those values out.

# Try to set the following values in the file

# year, just two digits
# today's day of the month
# Just the century

date +%y
date +%e
date +%C
```

Lab 🧪

This lab is designed to help you get familiar with the basics of the systems you will be working on. Some of you will find that you know the basic material but the techniques here allow you to put it together in a more complex fashion.

It is recommended that you type these commands and do not copy and paste them. Word sometimes likes to format characters and they don't always play nice with Linux.

Working with files

```
# Creating empty files with touch
touch fruits.txt

ls -l fruits.txt
# You will see that fruits.txt exists and is a 0 length (bytes) file

-rw-r--r--. 1 root root 0 Jun 22 07:59 fruits.txt
# Take a look at those values and see if you can figure out what they mean.
# man touch and see if it has any other useful features you might use. If
# you've ever used tiered storage think about access times and how to keep data
# hot/warm/cold. If you haven't just look around for a bit.

rm -rf fruits.txt

ls -l fruits.txt
# You will see that fruits.txt is gone.
```

Creating files just by stuffing data in them

```
echo "grapes 5" > fruits.txt
cat fruits.txt
echo "apples 3" > fruits.txt
cat fruits.txt

echo " " > fruits.txt

echo "grapes 5" >> fruits.txt
cat fruits.txt
echo "apples 3" >> fruits.txt
cat fruits.txt
```

What is the difference between these two? Appending a file >> adds to the file whereas > just overwrites the file each write. Log files almost always are written with >>, we never > over those types of files.

Creating file with vi or vim

```
# It is highly recommended the user read vimtutor. To get vimtutor follow
# these steps:
sudo -i
yum -y install vim
vimtutor

# There are about 36 short labs to show a user how to get around inside of vi.
# There are also cheat sheets around to help.

vi somefile.txt
# type "i" to enter insert mode

# Enter the following lines
grapes 5
apples 7
oranges 3
bananas 2
pears 6
pineapples 9

# hit the "esc" key at the top left of your keyboard
# Type ":wq"
# Hit enter

cat somefile.txt
```

Copying and moving files

```
cp somefile.txt backupfile.txt
ls
cat backupfile.txt
mv somefile.txt fruits.txt
ls
cat fruits.txt
```

Look at what happened in each of these scenarios. Can you explain the difference between cp and mv? Read the manuals for cp and mv to see if there's anything that may be useful to you. For most of us -r is tremendously useful option for moving directories.

Searching/filtering through files

```
# So maybe we only want to see certain values from a file, we can filter
# with a tool called grep

cat fruits.txt
cat fruits.txt | grep apple
cat fruits.txt | grep APPLE

# read the manual for grep and see if you can cause it to ignore case.

# See if you can figure out how to both ignore case and only find the
# word apple at the beginning of the line.

# If you can't, here's the answer. Try it:
cat fruits.txt | grep -i "^apple"
```

Can you figure out why that worked? What do you think the ^ does? Anchoring is a common term for this. See if you can find what anchors to the end of a string.

Sorting files with sort

```
# Let's sort our file fruits.txt and look at what happens to the output
# and the original file

sort fruits.txt
cat fruits.txt

# Did the sort output come out different than the cat output? Did sorting
# your file do anything to your original data? So let's sort our data again
# and figure out what this command does differently

sort -k 2 fruits.txt

# You can of course man sort to figure it out, but -k refers to the "key" and
# can be useful for sorting by a specific column

# But, if we cat fruits.txt we see we didn't save anything we did. What if we
# wanted to save these outputs into a file. Could you do it? If you couldn't,
# here's an answer:

sort fruits.txt > sort_by_alphabetical.txt
sort -k 2 fruits.txt > sort_by_price.txt

# Cat both of those files out and verify their output
```

Advanced sort practice

```
# Consider the command
ps -aux

# But that's too long to probably see everything, so let's use a command
# to filter just the top few lines
ps -aux | head

# So now you can see the actual fields (keys) across the top that we could sort by

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

# So let's say we wanted to sort by %MEM
ps -aux | sort -k 4 -n -r | head -10
```

Read man to see why that works. Why do you suppose that it needs to be reversed to have the highest numbers at the top? What is the difference, if you can see any, between using the -n or not using it? You may have to use head -40 to figure that out, depending on your processes running.

Read man ps to figure out what other things you can see or sort by from the ps command. We will examine that command in detail in another lab.

Working with redirection

The good thing is that you've already been redirecting information into files. The > and >> are useful for moving data into files. We have other functionality within redirects that can prove useful for putting data where we want it, or even not seeing the data.

Catching the input of one command and feeding that into the input of another command We've actually been doing this the entire time. "|" is the pipe operator and causes the output of one command to become the input of the second command.

```
cat fruits.txt | grep apple
# This cats out the file, all of it, but then only shows the things that
# pass through the filter of grep. We could continually add to these and make
# them longer and longer

cat fruits.txt | grep apple | sort | nl | awk '{print $2}' | sort -r
pineapples
apples
cat fruits.txt | grep apple | sort | nl | awk '{print $3}' | sort -r
9
7
cat fruits.txt | grep apple | sort | nl | awk '{print $1}' | sort -r
2
1

# Take these apart by pulling the end pipe and command off to see what is
# actually happening:

cat fruits.txt | grep apple | sort | nl | awk '{print $1}' | sort -r
2
1
cat fruits.txt | grep apple | sort | nl | awk '{print $1}'
1
2
cat fruits.txt | grep apple | sort | nl
1 apples 7
2 pineapples 9
cat fruits.txt | grep apple | sort
apples 7
pineapples 9
cat fruits.txt | grep apple
apples 7
pineapples 9
```

See if you can figure out what each of those commands do. Read the manual `man command` for any command you don't recognize. Use something you learned to affect the output.

Throwing the output into a file

We've already used > and >> to throw data into a file but when we redirect like that we are catching it before it comes to the screen. There is another tool that is useful for catching data and also showing it to us, that is tee.

```
date
# comes to the screen

date > datefile
# redirects and creates a file datefile with the value

date | tee -a datefile
# will come to screen, redirect to the file.
```

Do a quick man on tee to see what the -a does. Try it without that value. Can you see any other useful options in there for tee?

Ignoring pesky errors or tossing out unwanted output

Sometimes we don't care when something errs out. We just want to see that it's working or not. If you're wanting to filter out errors (2) in the standarderr, you can do this

```
ls fruits.txt
# You should see normal output

ls fruity.txt
# You should see an error unless you made this file

ls fruity.txt 2> /dev/null
# You should no longer see the error.

# But, sometimes you do care how well your script runs against 100 servers,
# or you're testing and want to see those errors. You can redirect that to a file, just as easy

ls fruity.txt 2> error.log
cat error.log
# You'll see the error. If you want it see it a few times do the error line to see it happen.
```

In one of our later labs we're going to look at stressing our systems out. For this, we'll use a command that basically just causes the system to burn cpu cycles creating random numbers, zipping up the output and then throwing it all away. Here's a preview of that command so you can play with it.

May have to yum -y install bzip2 for this next one to work.

```
time dd if=/dev/urandom bs=1024k count=20 | bzip2 -9 >> /dev/null
```

Use “crtl + c” to break if you use that and it becomes too long or your system is under too much load. The only numbers you can play with there are the 1024k and the count. Other numbers should be only changed if you use man to read about them first.

This is the “poor man’s” answer file. Something we used to do when we needed to answer some values into a script or installer. This is still very accurate and still works, but might be a bit advanced with a lot of advanced topics in here. Try it if you’d like but don’t worry if you don’t get this on the first lab.

```
vi testscript.sh
hit “i” to enter insert mode
add the following lines:

#!/bin/bash

read value
echo "The first value is $value"
read value
echo "The second value is $value"
read value
echo "The third value is $value"
read value
echo "The fourth value is $value"

# hit “esc” key
type in :wq
# hit enter

chmod 755 testscript.sh

# Now type in this (don’t type in the > those will just be there in your shell):

[xgqa6cha@N01APL4244 ~]$ echo "yes

> no
> 10
> why" | ./testscript.sh
> yes
> no
> 10
> why
```

What happened here is that we read the input from command line and gave it, in order to the script to read and then output. This is something we do if we know an installer wants certain values throughout it, but we don’t want to sit there and type them in, or we’re doing it across 100 servers quickly, or all kinds of reasons. It’s just a quick and dirty input “hack” that counts as a redirect.

Working with permissions

Permissions have to do with who can or cannot access (read), edit (write), or execute (xecute)files.

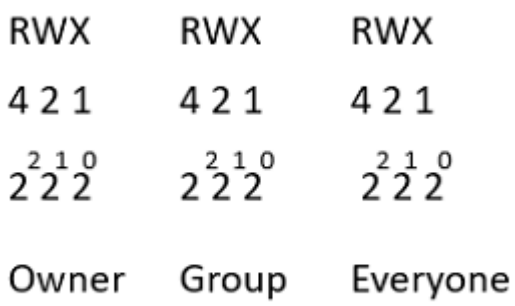
Permissions look like this.

```
ls -l
```

Permission	# of Links	UID Owner	Group Owner	Size (b)	Creation Month	Crea
-rw-r--r--.	1	Root	root	58	Jun	

The primary permissions commands we’re going to use are going to be chmod (access) and chown (ownership).

A quick rundown of how permissions break out:



Let's examine some permissions and see if we can't figure out what permissions are allowed.

```
ls -ld /root/
# drwx------. 5 root root 4096 Jun 22 09:11 /root/
```

The first character lets you know if the file is a directory, file, or link. In this case we are looking at my home directory.

`rwx` : For UID (me).

- What permissions do I have?

`---` : For group.

- Who are they?
- What can my group do?

`---` : For everyone else.

- What can everyone else do?

Go find some other interesting files or directories and see what you see there. Can you identify their characteristics and permissions?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.2.4 Unit 1 Bonus - VIM Fundamentals for Linux Sysadmins

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

Module 1: Getting Started (Days 1-2)

DAY 1: FIRST CONTACT WITH VIM

Segment 1: The Basics

1. Complete first section of `vimtutor`
2. Learn essential commands:
    - `vim filename` - Open/create file
    - `i` - Enter insert mode (before the cursor)
    - `a` - Enter insert mode (after the cursor)
    - `Esc` - Return to normal mode
    - `:w` - Save changes
    - `:q` - Quit
    - `:wq` or `ZZ` - Save and quit
    - `:q!` - Quit without saving

Segment 2: Building Muscle Memory

1. Create five different files
2. Practice mode switching 50 times
3. Write and save content in each file
4. Practice recovering from common mistakes:
    - Accidentally pressed keys in normal mode
    - Forgot to enter insert mode
    - Trying to quit without saving

Segment 3: First Real Task

1. Create a simple bash script template
2. Add standard sections:
    - Shebang line
    - Comments
    - Basic variables
    - Simple functions
3. Save and reopen multiple times

DAY 2: COMFORT ZONE

Segment 1: More Basic Operations

1. Complete second section of `vimtutor`
2. Practice quick save and exit combinations
3. Learn to read VIM messages and errors
4. Understand modes in depth:
    - Normal mode
    - Insert mode
    - Visual mode (introduction)

Segment 2: Error Recovery

1. Create deliberate errors and fix them:
    - Write without insert mode
    - Exit without saving needed changes
    - Get stuck in different modes
2. Practice until you can recover without thinking

Segment 3: Real Config Practice

1. Copy `/etc/hosts` file
2. Make various modifications:
    - Add new host entries
    - Modify existing entries
    - Add comments
    - Save different versions

Module 2: Navigation (Days 3-4)

DAY 3: BASIC MOVEMENT

Segment 1: Core Movement Commands

- Master the basics:
    - `h` - Left
    - `j` - Down
    - `k` - Up
    - `l` - Right
    - `w` - Next word
    - `b` - Previous word
    - `0` - Line start
    - `$` - Line end
    - `^` - First non-blank character of the line
    - `g_` - Last non-blank character of the line

Segment 2: Movement Drills

1. Create a "movement course" file
2. Practice moving between marked points
3. Time your navigation speed
4. Compete against your previous times

Segment 3: Applied Navigation

1. Navigate through `/etc/ssh/sshd_config`:
    - Find specific settings
    - Move between sections
    - Locate comments
    - Jump to line numbers

DAY 4: ADVANCED MOVEMENT

Segment 1: Extended Movement

- Learn efficient jumps:
    - `gg` - File start
    - `G` - File end
    - `{` - Previous paragraph
    - `}` - Next paragraph
    - `Ctrl+f` - Page down
    - `Ctrl+b` - Page up

Segment 2: Speed Training

1. Work with a large configuration file
2. Practice jumping between sections
3. Find specific lines quickly
4. Navigate through code blocks

Segment 3: Real-world Navigation

1. Work with system logs
2. Jump between error messages
3. Navigate through long configuration files
4. Practice quick file browsing

Module 3: Essential Editing (Days 5-7)

DAY 5: BASIC EDITING

Segment 1: Edit Commands

- Master core editing:
  - `x` - Delete character
  - `dd` - Delete line
  - `yy` - Copy line
  - `p` - Paste after
  - `P` - Paste before
  - `u` - Undo
  - `Ctrl + r` - Redo
  - `s` - Substitute a character
  - `r` - Replace a character
  - `c` - Change character

Segment 2: Editing Drills

1. Create practice documents
2. Delete and replace text
3. Copy and paste sections
4. Practice undo/redo chains

Segment 3: System File Editing

1. Work with `/etc/fstab` copy:
   - Add mount points
   - Remove entries
   - Comment lines
   - Fix formatting

DAY 6: INTERMEDIATE EDITING

Segment 1: Combined Commands

- Learn efficient combinations:
  - `dw` - Delete word
  - `d$` - Delete to line end
  - `d0` - Delete to line start
  - `cc` - Change whole line
  - `cw` - Change word
  - `D` - Delete to line end
  - `C` - Change to line end

Segment 2: Practical Application

1. Edit service configuration files
2. Modify system settings
3. Update network configurations
4. Clean up log files

Segment 3: Speed Challenges

1. Timed editing tasks
2. Configuration file cleanup
3. Quick text transformation
4. Error correction sprints

DAY 7: EDITING MASTERY

Segment 1: Advanced Operations

- Master text objects:
  - `ciw` - Change inner word
  - `ci"` - Change inside quotes
  - `di(` - Delete inside parentheses
  - `yi{` - Yank inside braces
  - `ca"` - Change a quotes block
  - `da{` - Delete a `{ }` block
  - `ya(` - Yank a `( )` block

Segment 2: Integration Practice

1. Combine all learned commands
2. Work with multiple files
3. Practice common scenarios
4. Time your operations

Daily Success Metrics

By end of each day, you should be able to:

- Day 1: Open, edit, save, and exit files confidently
- Day 2: Understand and recover from common errors
- Day 3: Navigate small files without arrow keys
- Day 4: Move through large files efficiently
- Day 5: Perform basic edits without hesitation
- Day 6: Combine movement and editing commands
- Day 7: Edit configuration files with confidence

Practice Tips

1. Use `vimtutor` during breaks
2. Disable arrow keys completely
3. Keep a command log of new discoveries

4. Time your editing operations
5. Practice with real system files (copies)

Remember: Focus on accuracy first, then build speed.

2.3 Unit 2 - Essential Tools

2.3.1 Unit 2 - Essential Tools

Overview

This unit centers on a focus on security and troubleshooting.

- The use of **SELinux** for implementing mandatory access controls, managing file permissions with **ACLs (Access Control Lists)**,
- Understanding operational methodologies for incident triage.

Learning Objectives

1. **Understand and Configure SELinux:**
    - Grasp the core concepts of SELinux, including security contexts, labels, and its role in enforcing mandatory access control.
    - Learn how to configure and troubleshoot SELinux settings to ensure system security and compliance.
2. **Master Access Control Lists (ACLs):**
    - Recognize the limitations of traditional Unix permissions and how ACLs provide granular control over file and directory access.
    - Develop skills in applying and managing ACLs in a complex Linux environment.
3. **Develop Effective Troubleshooting Methodologies:**
    - Acquire techniques to diagnose and resolve system access issues, particularly those arising from SELinux policies and ACL misconfigurations.
    - Apply structured troubleshooting strategies to ensure minimal downtime and maintain high availability.
4. **Integrate Theoretical Knowledge with Practical Application:**
    - Engage with interactive exercises, discussion prompts, and real-world scenarios to reinforce learning.
    - Utilize external resources, such as technical documentation and instructional videos, to supplement hands-on practice.
5. **Enhance Collaborative Problem-Solving Skills:**
    - Participate in peer discussions and reflective exercises to compare different approaches to system administration challenges.
    - Learn to articulate and document troubleshooting processes and system configurations for continuous improvement.
6. **Build a Foundation for Advanced Security Practices:**
    - Understand how SELinux and ACLs fit into the broader context of system security and operational stability.
    - Prepare for more advanced topics by reinforcing the fundamental skills needed to manage and secure Red Hat Enterprise Linux environments.

These objectives aim to ensure that learners not only acquire specific technical skills but also develop a holistic understanding of how to secure and manage Linux systems in enterprise settings.

Key terms and Definitions

SELinux(Security-EnhancedLinux)	**AccessControlLists(ACLs)**
Security Contexts	**Mandatory Access Control(MAC)**
Discretionary Access Control(DAC)	**Uptime**
Standard Streams(stdin,stdout,stderr)	**High Availability(HA)**
Service Level Objectives(SLOs)	**Troubleshooting Methodologies**

2.3.2 Unit 2 Worksheet - Essential Tools

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Bash Reference Manual
- Security Enhanced Linux

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u2_worksheet(`.txt`)
- 📥 u2_worksheet(`.docx`)

UNIT 2 RECORDING

Link: https://www.youtube.com/watch?v=miVuSoHTuP4

Unit 2 Discussion Post #1

Think about how week 1 went for you.

1. Do you understand everything that needs to be done?
2. Do you need to allocate more time to the course, and if so, how do you plan to do it?
3. How well did you take notes during the lecture? Do you need to improve this?

Unit 2 Discussion Post #2

Read a blog, check a search engine, or ask an AI about SELinux.
What is the significance of contexts? What are the significance of labels?

> **Scenario**
>
> You follow your company instructions to add a new user to a set of 10 Linux servers. They cannot access just one of the servers.
>
> When you review the differences in the servers you see that the server they cannot access is running SELINUX. On checking other users have no problem getting into the system.
>
> You find nothing in the documentation (typical) about this different system or how these users are accessing it.

What do you do?
Where do you check?

You may use any online resources to help you answer this. This is not a trick and it is not a “one answer solution”. This is for you to think through.

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

START THINKING ABOUT YOUR PROJECT IDEAS (MORE TO COME IN FUTURE WEEKS):

Topics:

1. System Stability
2. System Performance
3. System Security
4. System monitoring
5. Kubernetes
6. Programming/Automation

You will research, design, deploy, and document a system that improves your administration of Linux systems in some way.

Definitions

Uptime:

Standard input (stdin):

Standard output (stdout):

Standard error (stderr):

Mandatory Access Control (MAC):

Discretionary Access Control (DAC):

Security contexts (SELinux):

SELinux operating modes:

Digging Deeper

1. How does troubleshooting differ between system administration and system engineering? To clarify, how might you troubleshoot differently if you know a system was previously running correctly. If you're building a new system out?
2. Investigate a troubleshooting methodology, by either Google or AI search. Does the methodology fit for you in an IT sense, why or why not?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

2.3.3 Unit 2 Lab - Essential Tools

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Top 50+ Linux CLI Commands

REQUIRED MATERIALS

- Putty or other connection tool
- Rocky 9.4+ - ProLUG Lab
    - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u2_lab(`.pdf`)
- 📥 u2_lab(`.docx`)

Pre-Lab Warm-Up

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

```
cd ~
ls
mkdir evaluation
mkdir evaluation/test/round6
# This fails, can you find out why?

mkdir -p evaluation/test/round6
# This works, think about why?

cd evaluation
pwd
# What is the path you are in?

touch testfile1
ls
# What did this do?

touch testfile{2..10}
ls
# What did this do differently than earlier?
# touch .hfile .hfile2 .hfile3

ls
# Can you see your newest files? Why or why not? (man ls)
# What was the command to let you see those hidden files?

ls -l
# What do you know about this long listing? Think about 10 things this can show you.
# Did it show you all the files or are some missing?
```

Lab 🧪

This lab is designed to help you get familiar with the basics of the systems you will be working on. Some of you will find that you know the basic material but the techniques here allow you to put it together in a more complex fashion.

It is recommended that you type these commands and do not copy and paste them. Word sometimes likes to format characters and they don't always play nice with Linux.

Gathering system information

```
hostname
cat /etc/*release
# What do you recognize about this output? What version of RHEL (CENTOS) are we on?

uname
uname -a
uname -r

# man uname to see what those options mean if you don't recognize the values
```

Check the amount of RAM

```
cat /proc/meminfo
free
free -m

# What do each of these commands show you? How are they useful?
```

Check the number of processors and processor info

```
cat /proc/cpuinfo
# What type of processors do you have? How many are there? (counting starts at 0)

cat /proc/cpuinfo | grep proc | wc -l
# Does this command accurately count the processors?
```

Check Storage usage and mounted filesystems

```
df
# But df is barely readable, so find the option that makes it more readable `man df`

df -h
df -h | grep -i var
# What does this show, or search for? Can you invert this search? (hint `man grep`
# look for invert or google "inverting grep's output")

df -h | grep -i sd
# This one is a little harder, what does this one show? Not just the line, what are
# we checking for? (hint if you need it, google "what is /dev/sda in linux")

mount
# Mount by itself gives a huge amount of information. But, let's say someone is asking
# you to verify that the mount is there for /home on a system. Can you check that
# quickly with one command?

mount | grep -i home
#This works, but there is a slight note to add here. Just because something isn't
# individually mounted doesn't mean it doesn't exist. It just means it's not part of
# it's own mounted filesystem.

mount | grep -i /home/xgqa6cha
# will produce no output

df -h /home/xgqa6cha
# will show you that my home filesystem falls under /home.

cd ~; pwd; df -h .
# This command moves you to your home directory, prints out that directory,
# and then shows you what partition your home directory is on.

du -sh .
# will show you space usage of just your directory

try `du -h .` as well to see how that ouput differs
# read `man du` to learn more about your options.
```

Check the system uptime

```
uptime

man uptime
# Read the man for uptime and figure out what those 3 numbers represent.
# Referencing this server, do you think it is under high load? Why or why not?
```

Check who has recently logged into the server and who is currently in

```
last
# Last is a command that outputs backwards. (Top of the output is most recent).
# So it is less than useful without using the more command.

last | more
# Were you the last person to log in? Who else has logged in today?

w
who
whoami
# how many other users are on this system? What does the pts/0 mean on google?
```

Check who you are and what is going on in your environment

```
printenv
# This scrolls by way too fast, how would you search for your home?

printenv | grep -i home
whoami
id
echo $SHELL
```

Check running processes and services

```
ps -aux | more
ps -ef | more
ps -ef | wc -l
```

Check memory usage and what is using the memory

```
# Run each of these individually for understanding before we look at part b.
free -m
free -m | egrep “Mem|Swap”
free -m| egrep “Mem|Swap” | awk ‘{print $1, $2, $3}’
free -t | egrep "Mem|Swap" | awk '{print $1 " Used Space = " ($3 / $2) * 100"%"}'

# Taking this apart a bit:
# You’re just using free and searching for the lines that are for memory and swap
# You then print out the values $1 = Mem or Swap
# You then take $3 used divided by $2 total and multiply by 100 to get the percentage
```

Have you ever written a basic check script or touched on conditional statements or loops? (Use ctrl + c to break out of these):

```
while true; do free -m; sleep 3; done

# Watch this output for a few and then break with ctrl + c
# Try to edit this to wait for 5 seconds
# Try to add a check for uptime and date each loop with a blank line between
# each and 10 second wait:

while true; do date; uptime; free -m; echo “ “; sleep 10; done
# Since we can wrap anything inside of our while statements, let’s try adding
# something from earlier:
while true; do free -t | egrep "Mem|Swap" | awk '{print $1 " Used Space = " ($3 / $2) * 100"%"}'; sleep 3; done
```

```
seq 1 10
# What did this do?
# Can you man seq to modify that to count from 2 to 20 by 2’s?
# Let’s make a counting for loop from that sequence

for i in `seq 1 20`; do echo "I am counting i and am on $i times through the loop"; done
```

Can you tell me what is the difference or significance of the $ in the command above? What does that denote to the system?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.4 Unit 3 - Storage

2.4.1 Unit 3 - LVM and Raid

Overview

The unit focuses on understanding and implementing techniques to ensure systems remain operational with minimal downtime.

- The process of quickly assessing, prioritizing, and addressing system incidents.
- Leveraging performance indicators (KPIs, SLIs) and setting clear operational targets (SLOs, SLAs) to guide troubleshooting and recovery efforts.

Learning Objectives

1. **Understand Fundamental Concepts of System Reliability and High Availability:**
   - Explain the importance of uptime and the implications of “Five 9’s” availability in mission-critical environments.
   - Define key terms such as Single Point of Failure (SPOF), Mean Time to Detect (MTTD), Mean Time to Recover (MTTR), and Mean Time Between Failures (MTBF).
2. **Identify and Apply High Availability Architectures:**
   - Differentiate between Active-Active and Active-Standby configurations and describe their advantages and trade-offs.
   - Evaluate real-world scenarios to determine where redundancy and clustering (using tools like Pacemaker and Corosync) can improve system resilience.
3. **Develop Incident Triage and Response Skills:**
   - Outline a structured approach to incident detection, prioritization, and resolution.
   - Use performance metrics (KPIs, SLIs, SLOs, and SLAs) to guide decision-making during operational incidents.
4. **Integrate Theoretical Knowledge with Practical Application:**
   - Leverage external resources (such as AWS whitepapers, Google SRE documentation, and Red Hat guidelines) to deepen understanding of system reliability best practices.
   - Participate in interactive discussion posts and collaborative problem-solving exercises to reinforce learning.
5. **Cultivate Analytical and Troubleshooting Abilities:**
   - Apply systematic troubleshooting techniques to diagnose and resolve system issues.
   - Reflect on incident case studies and simulated exercises to improve proactive prevention strategies.

These learning objectives are designed to ensure that participants not only grasp the theoretical underpinnings of system reliability and high availability but also build the practical skills needed for effective incident management and system optimization in a professional Linux environment.

Key terms and Definitions

Resilience Engineering	**Fault Tolerance**
Proactive Monitoring	**Observability**
Incident Response	**Root Cause Analysis(RCA)**
Disaster Recovery(DR)	**Error Budgeting**
Capacity Planning	**Load Balancing**
Service Continuity	**DevOps Culture**
Infrastructureas Code(IaC)	**Configuration Management**
Preventive Maintenance	

2.4.2 Unit 3 Worksheet - LVM and Raid

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Google SRE Book - Implementing SLOs
- AWS High Availability Architecture Guide
- Red Hat High Availability Cluster Configuration

Downloads

The worksheet has been provided below. The document can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u3_worksheet(`.txt`)
- 📥 u3_worksheet(`.docx`)

UNIT 3 RECORDING

Link: https://www.youtube.com/watch?v=NYL85ndQLbc

Discussion Post #1

Scan the chapter here for keywords and pull out what you think will help you to better understand how to triage an incident.

Read the section called "Operation Security" in this same chapter: Building Secure and Reliable Systems

1. What important concepts do you learn about how we behave during an operational response to an incident?

Discussion Post #2

Ask Google, find a blog, or ask an AI about high availability. (Here's one if you need it: AWS Real-Time Communication Whitepaper

1. What are some important terms you read about? Why do you think understanding HA will help you better in the context of triaging incidents?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Five 9's:

Single Point of Failure (SPOF):

Key Performance Indicators (KPIs):

Service Level Indicator (SLI):

Service Level Objective (SLO):

Service Level Agreement (SLA):

Active-Standby:

Active-Active:

Mean Time to Detect (MTTD):

Mean Time to Recover/Restore (MTTR):

Mean Time Between Failures (MTBF):

Digging Deeper

1. If uptime is so important to us, why is it so important to us to also understand how our systems can fail? Why would we focus on the thing that does not drive uptime?
2. Start reading about SLOs: Implementing SLOs How does this help you operationally? Does it make sense that keeping systems within defined parameters will help keep them operating longer?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

2.4.3 Unit 3 Lab - LVM and Raid

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Ubuntu Documentation on LVM

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
    - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u3_lab(`.pdf`)
- 📥 u3_lab(`.docx`)

Pre-Lab Warm-Up

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

```
cd ~
mkdir lvm_lab

cd lvm_lab
touch somefile
echo “this is a string of text” > somefile
cat somefile
echo “this is a string of text” > somefile
# Repeat 3 times

cat somefile
# How many lines are there?

Echo “this is a string of text” >> somefile
# Repeat 3 times

cat somefile
# How many lines are there?

# cheat with `cat somefile | wc -l`
echo “this is our other test text” >> somefile
# Repeat 3 times

cat somefile | nl
# How many lines are there?

cat somefile | nl | grep test
# compare that with 14

cat somefile | grep test | nl
```

If you want to preserve positional lines in file (know how much you’ve cut out when you grep something, or generally be able to find it in the unfiltered file for context, always | nl | before your grep

Pre Lab - Disk Speed tests:

When using the ProLUG lab environment, you should always check that there are no other users on the system `w` or `who`.

After this, you may want to check the current state of the disks, as they retain their information even after a reboot resets the rest of the machine. `lsblk /dev/xvda`.

```
# If you need to wipe the disks, you should use fdisk or a similar partition utility.
fdisk /dev/xvda

p #print to see partitions
d #delete partitions or information
w #Write out the changes to the disk.
```

This is an aside, before the lab. This is a way to test different read or writes into or out of your filesystems as you create them. Different types of raid and different disk setups will give different speed of read and write. This is a simple way to test them. Use these throughout the lab in each mount for fun and understanding.

Write tests (saving off write data - rename /tmp/file each time)

```
# Check /dev/xvda for a filesystem
blkid /dev/xvda

# If it does not have one, make one
mkfs.ext4 /dev/xvda
mkdir /space # (If you don't have it. Lab will tell you to later as well)

mount /dev/xvda /space
```

Write Test

```
for i in `seq 1 10`; do time dd if=/dev/zero of=/space/testfile$i bs=1024k count=1000 | tee -a /tmp/speedtest1.basiclvm; done
```

Read tests

```
for i in `seq 1 10`; do time dd if=/space/testfile$i of=/dev/null; done
```

Cleanup

```
for i in `seq 1 10`; do rm -rf /space/testfile$i; done
```

If you are re-creating a test without blowing away the filesystem, change the name or counting numbers of testfile because that's the only way to be sure there is not some type of filesystem caching going on to optimize. This is especially true in SAN write tests.

Lab 🧪

start in root (#); cd /root

LVM explanation and use within the system

```
# Check physical volumes on your server (my output may vary)
fdisk -l | grep -i xvd

# Disk /dev/xvda: 15 GiB, 16106127360 bytes, 31457280 sectors
# Disk /dev/xvdb: 3 GiB, 3221225472 bytes, 6291456 sectors
# Disk /dev/xvdc: 3 GiB, 3221225472 bytes, 6291456 sectors
# Disk /dev/xvde: 3 GiB, 3221225472 bytes, 6291456 sectors
```

Looking at Logical Volume Management

Logical Volume Management is an abstraction layer that looks a lot like how we carve up SAN disks for storage management. We have Physical Volumes that get grouped up into Volume Groups. We carve Volume Groups up to be presented as Logical Volumes.

Here at the Logical Volume layer we can assign RAID functionality from our Physical Volumes attached to a Volume Group or do all kinds of different things that are "under the hood". Logical Volumes get filesystems formatting and are mounted to the OS.

There are many important commands for showing your physical volumes, volume groups, and logical volumes.

The three simplest and easiest are:

```
pvs
vgs
lvs
```

With these you can see basic information that allows you to see how the disks are allocated. Why do you think there is no output from these commands the first time you run them? Try these next commands to see if you can figure out what is happening? To see more in depth information try pvdisplay, vgdisplay, and lvdisplay.

If there is still no output, it's because this system is not configured for LVM. You will notice that none of the disk you verified are attached are allocated to LVM yet. We'll do that next.

Creating and Carving up your LVM resources

Disks for this lab are /dev/xvdb, /dev/xvdc, and /dev/xvdd. (but verify before continuing and adjust accordingly.)

We can do individual pvcreates for each disk `pvcreate /dev/xvdb` but we can also loop over them with a simple loop as below. Use your drive letters.

```
for disk in b c d; do pvcreate /dev/xvd$disk; done

# Physical volume "/dev/xvdb" successfully created.
# Creating devices file /etc/lvm/devices/system.devices
# Physical volume "/dev/xvdc" successfully created.
# Physical volume "/dev/xvde" successfully created.

# To see what we made:

pvs

# PV VG Fmt Attr PSize PFree
# /dev/xvdb lvm2 --- 3.00g 3.00g
# /dev/xvdc lvm2 --- 3.00g 3.00g
# /dev/xvde lvm2 --- 3.00g 3.00g

vgcreate VolGroupTest /dev/xvdb /dev/xvdc /dev/xvde

# Volume group "VolGroupTest" successfully created

vgs
# VG #PV #LV #SN Attr VSize VFree
# VolGroupTest 3 0 0 wz--n- <8.99g <8.99g

lvcreate -l +100%FREE -n lv_test VolGroupTest

# Logical volume "lv_test" created.

lvs

# LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
# lv_test VolGroupTest -wi-a----- <8.99g
# Formatting and mounting the filesystem

mkfs.ext4 /dev/mapper/VolGroupTest-lv_test

# mke2fs 1.42.9 (28-Dec-2013)
# Filesystem label=
# OS type: Linux
# Block size=4096 (log=2)
# Fragment size=4096 (log=2)
# Stride=0 blocks, Stripe width=0 blocks
# 983040 inodes, 3929088 blocks
# 196454 blocks (5.00%) reserved for the super user
# First data block=0
# Maximum filesystem blocks=2151677952
# 120 block groups
# 32768 blocks per group, 32768 fragments per group
# 8192 inodes per group
# Superblock backups stored on blocks:
# 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208
#
# Allocating group tables: done
# Writing inode tables: done
# Creating journal (32768 blocks): done
# Writing superblocks and filesystem accounting information: done

mkdir /space #Created earlier
vi /etc/fstab

# Add the following line
# /dev/mapper/VolGroupTest-lv_test /space ext4 defaults 0 0

# reload fstab
systemctl daemon-reload
```

If this command works, there will be no output. We use the df -h in the next command to verify the new filesystem exists. The use of mount -a and not manually mounting the filesystem from the command line is an old administration trick I picked up over the years.

By setting our mount in /etc/fstab and then telling the system to mount everything we verify that this will come back up properly during a reboot. We have mounted and verified we have a persistent mount in one step.

```
df -h

# Filesystem Size Used Avail Use% Mounted on
# devtmpfs 4.0M 0 4.0M 0% /dev
# tmpfs 2.0G 0 2.0G 0% /dev/shm
# tmpfs 2.0G 8.5M 1.9G 1% /run
# tmpfs 2.0G 1.4G 557M 72% /
# tmpfs 2.0G 0 2.0G 0% /run/shm
# 192.168.200.25:/home 44G 15G 30G 34% /home
# 192.168.200.25:/opt 44G 15G 30G 34% /opt
# tmpfs 390M 0 390M 0% /run/user/0
# /dev/mapper/VolGroupTest-lv*test 8.8G 24K 8.3G 1% /space
```

Good place to speed test and save off your data.

Removing and breaking down the LVM to raid disks

The following command is one way to comment out the line in /etc/fstab. If you had to do this across multiple servers this could be useful. (Or you can just use vi for simplicity).

```
grep lv_test /etc/fstab; perl -pi -e "s/\/dev\/mapper\/VolGroupTest/#removed \/dev\/mapper\/VolGroupTest/" /etc/fstab; grep removed /etc/fstab
# /dev/mapper/VolGroupTest-lv_test /space ext4 defaults 0 0
#removed dev/mapper/VolGroupTest-lv_test /space ext4 defaults 0 0

umount /space
lvremove /dev/mapper/VolGroupTest-lv_test

# Do you really want to remove active logical volume VolGroupTest/lv_test? [y/n]: y
# Logical volume "lv_test" successfully removed

vgremove VolGroupTest

# Volume group "VolGroupTest" successfully removed


for disk in c e f; do pvremove /dev/sd$disk; done

# Labels on physical volume "/dev/sdc" successfully wiped.
# Labels on physical volume "/dev/sde" successfully wiped.
# Labels on physical volume "/dev/sdf" successfully wiped.
```

Use your `pvs;vgs;lvs` commands to verify those volumes no longer exist.

```
pvs;vgs;lvs

# PV VG Fmt Attr PSize PFree
# /dev/sda2 VolGroup00 lvm2 a-- 17.48g 4.00m
# /dev/sdb VolGroup01 lvm2 a-- 20.00g 96.00m
# VG #PV #LV #SN Attr VSize VFree
# VolGroup00 1 9 0 wz--n- 17.48g 4.00m
# VolGroup01 1 1 0 wz--n- 20.00g 96.00m
# LV VG Attr LSize Pool Origin Data% Meta% Move Log
# LogVol00 VolGroup00 -wi-ao---- 2.50g
# LogVol01 VolGroup00 -wi-ao---- 1000.00m
# LogVol02 VolGroup00 -wi-ao---- 5.00g
# LogVol03 VolGroup00 -wi-ao---- 1.00g
# LogVol04 VolGroup00 -wi-ao---- 5.00g
# LogVol05 VolGroup00 -wi-ao---- 1.00g
# LogVol06 VolGroup00 -wi-ao---- 1.00g
# LogVol07 VolGroup00 -wi-ao---- 512.00m
# LogVol08 VolGroup00 -wi-ao---- 512.00m
# lv_app VolGroup01 -wi-ao---- 19.90g
```

More complex types of LVM

LVM can also be used to raid disks

Create a RAID 5 filesystem and mount it to the OS (For brevity's sake we will be limiting show commands from here on out, please use pvs,vgs,lvs often for your own understanding)

```
for disk in c e f; do pvcreate /dev/sd$disk; done

# Physical volume "/dev/sdc" successfully created.
# Physical volume "/dev/sde" successfully created.
# Physical volume "/dev/sdf" successfully created.

vgcreate VolGroupTest /dev/sdc /dev/sde /dev/sdf

lvcreate -l +100%FREE --type raid5 -n lv_test VolGroupTest
mkfs.xfs /dev/mapper/VolGroupTest-lv_test

vi /etc/fstab

# fix the /space directory to have these parameters (change ext4 to xfs)
/dev/mapper/VolGroupTest-lv_test /space xfs defaults 0 0

df -h
# Filesystem Size Used Avail Use% Mounted on
# /dev/mapper/VolGroup00-LogVol08 488M 34M 419M 8% /var/log/audit
# /dev/mapper/VolGroupTest-lv_test 10G 33M 10G 1% /space
```

Since we're now using RAID 5 we would expect to see the size no longer match the full 15GB, 10GB is much more of a RAID 5 value 66% of raw disk space.

To verify our raid levels we use lvs

```
lvs
# LV VG Attr LSize Pool Origin Data% Meta% Move Log Cpy%Sync
# LogVol00 VolGroup00 -wi-ao---- 2.50g
# LogVol01 VolGroup00 -wi-ao---- 1000.00m
# LogVol02 VolGroup00 -wi-ao---- 5.00g
# LogVol03 VolGroup00 -wi-ao---- 1.00g
# LogVol04 VolGroup00 -wi-ao---- 5.00g
# LogVol05 VolGroup00 -wi-ao---- 1.00g
# LogVol06 VolGroup00 -wi-ao---- 1.00g
# LogVol07 VolGroup00 -wi-ao---- 512.00m
# LogVol08 VolGroup00 -wi-ao---- 512.00m
# lv*app VolGroup01 -wi-ao---- 19.90g
# lv_test VolGroupTest rwi-aor--- 9.98g 100.00
```

Spend 5 min reading the `man lvs` page to read up on raid levels and what they can accomplish. To run RAID 5 3 disks are needed. To run RAID 6 at least 4 disks are needed.

Good place to speed test and save off your data

Set the system back to raw disks

Unmount /space and remove entry from /etc/fstab

```
lvremove /dev/mapper/VolGroupTest-lv_test
# Do you really want to remove active logical volume VolGroupTest/lv_test? [y/n]: y
# Logical volume "lv_test" successfully removed

vgremove VolGroupTest
# Volume group "VolGroupTest" successfully removed

for disk in c e f; do pvremove /dev/sd$disk; done
# Labels on physical volume "/dev/sdc" successfully wiped.
# Labels on physical volume "/dev/sde" successfully wiped.
# Labels on physical volume "/dev/sdf" successfully wiped.
```

Working with MDADM as another RAID option

There could be a reason to use MDADM on the system. For example you want raid handled outside of your LVM so that you can bring in sets of new disks already raided and treat them as their own Physical Volumes. Think, “I want to add another layer of abstraction so that even my LVM is unaware of the RAID levels.” This has special use case, but is still useful to understand.

May have to install mdadm yum: yum install mdadm

Create a raid5 with MDADM

```
mdadm --create -l raid5 /dev/md0 -n 3 /dev/sdc /dev/sde /dev/sdf

# mdadm: Defaulting to version 1.2 metadata
# mdadm: array /dev/md0 started.
```

Add newly created /dev/md0 raid to LVM

This is same as any other add. The only difference here is that LVM is unaware of the lower level RAID that is happening.

```
pvcreate /dev/md0
# Physical volume "/dev/md0" successfully created.

vgcreate VolGroupTest /dev/md0
# Volume group "VolGroupTest" successfully created

lvcreate -l +100%Free -n lv_test VolGroupTest
# Logical volume "lv_test" created.

lvs
# LV VG Attr LSize Pool Origin Data% Meta% Move Log
# LogVol00 VolGroup00 -wi-ao---- 2.50g
# LogVol01 VolGroup00 -wi-ao---- 1000.00m
# LogVol02 VolGroup00 -wi-ao---- 5.00g
# LogVol03 VolGroup00 -wi-ao---- 1.00g
# LogVol04 VolGroup00 -wi-ao---- 5.00g
# LogVol05 VolGroup00 -wi-ao---- 1.00g
# LogVol06 VolGroup00 -wi-ao---- 1.00g
# LogVol07 VolGroup00 -wi-ao---- 512.00m
# LogVol08 VolGroup00 -wi-ao---- 512.00m
# lv_app VolGroup01 -wi-ao---- 19.90g
# lv_test VolGroupTest -wi-a----- 9.99g
```

Note that LVM does not see that it is dealing with a raid system, but the size is still 10g instead of 15g.

Fix your /etc/fstab to read

/dev/mapper/VolGroupTest-lv_test /space xfs defaults 0 0

```
mkfs.xfs /dev/mapper/VolGroupTest-lv_test

# meta-data=/dev/mapper/VolGroupTest-lv_test isize=512 agcount=16, agsize=163712 blks
# = sectsz=512 attr=2, projid32bit=1
# = crc=1 finobt=0, sparse=0
# data = bsize=4096 blocks=2618368, imaxpct=25
# = sunit=128 swidth=256 blks
# naming =version 2 bsize=4096 ascii-ci=0 ftype=1
# log =internal log bsize=4096 blocks=2560, version=2
# = sectsz=512 sunit=8 blks, lazy-count=1
# realtime =none extsz=4096 blocks=0, rtextents=0

mount -a
```

Good place to speed test and save off your data

Setting the MDADM to persist through reboots:

(not in our lab environment though)

```
mdadm --detail --scan >> /etc/mdadm.conf
cat /etc/mdadm.conf
# ARRAY /dev/md0 metadata=1.2 name=ROCKY1:0 UUID=03583924:533e5338:8d363715:09a8b834
```

Verify with `df -h` ensure that your /space is mounted.

There is no procedure in this lab for breaking down this MDADM RAID.

You are root/administrator on your machine, and you do not care about the data on this RAID. Can you use the internet/man pages/or other documentation to take this raid down safely and clear those disks?

Can you document your steps so that you or others could come back and do this process again?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.4.4 Unit 3 Bonus - Storage

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

When storage issues arise, troubleshooting step by step ensures a quick resolution. This guide flows logically, covering the most common issues you might face, from slow performance to filesystem corruption.

🔍 Step 1: Is Storage Performance Slow?

If everything feels sluggish, your disk might be the bottleneck.

CHECK

```
# Monitor disk I/O, latency, and throughput
iostat -xz 1

# Identify processes consuming high I/O
pidstat -d 1

# Real-time disk activity monitoring
iostat -dx 1
```

- If I/O wait is high, it means the CPU is waiting on slow disk operations.
- If certain processes are consuming all disk bandwidth, they might be the cause.

FIX

1. Identify and stop unnecessary high I/O processes:

```
# Forcefully terminate a process (use with caution)
kill -9 <PID>
```

2. Optimize filesystem writes (for ext4):

```
# Enable writeback mode for better performance
tune2fs -o journal_data_writeback /dev/sdX
```

3. Reduce excessive metadata writes:

```
# Disable access time updates and set commit interval
mount -o noatime,commit=60 /mnt/data
```

4. If using LVM, extend the volume to reduce fragmentation:

```
# Add 5GB to volume
lvextend -L +5G /dev/examplegroup/lv_data
```

🔍 Step 2: Is the Filesystem Full? ("No Space Left on Device")

👉 Disk space exhaustion is one of the most common causes of storage failures.

CHECK

```
# Show disk usage per filesystem
df -hT

# Find the biggest files
du -ahx / | sort -rh | head -20
```

- If a filesystem is 100% full, it prevents writes and can cause application crashes.
- If there's space but files still won't write, check Step 4 (Corrupted Filesystem).

FIX

1. Find and remove large unnecessary files:

```
# Remove specific log file
rm -f /var/log/large_old_log.log
```

2. Truncate logs safely without deleting them:

```
# Clear log contents while preserving file
truncate -s 0 /var/log/syslog

# Limit journal size
journalctl --vacuum-size=100M
```

3. Expand disk space if using LVM:

```
# Extend logical volume
lvextend -L +10G /dev/examplegroup/lv_data

# Resize filesystem
resize2fs /dev/examplegroup/lv_data  # for ext4
xfs_growfs /mnt/data                 # for XFS
```

🔍 Step 3: Are Mounts Failing? (LVM, fstab, NFS, SMB)

If files suddenly disappear or applications complain about missing storage, a mount issue may be the cause.

CHECK

```
# View current mounts
mount | grep /mnt/data

# Check block devices
lsblk

# Verify permanent mount configuration
cat /etc/fstab
```

FIX

1. Manually remount the filesystem (if missing):

```
# Remount all fstab entries
mount -a
```

2. Ensure correct fstab entry for persistence:

```
# Add to /etc/fstab (replace UUID with actual value)
UUID=xxx-yyy-zzz /mnt/data ext4 defaults 0 2
```

3. If an LVM mount is missing after reboot, reactivate it:

```
# Activate volume groups
vgchange -ay

# Mount the logical volume
mount /dev/examplegroup/lv_data /mnt/data
```

4. For NFS issues, check connectivity and restart services:

```
# Check NFS exports
showmount -e <NFS_SERVER_IP>

# Restart NFS service
systemctl restart nfs-server
```

🔍 Step 4: Is the Filesystem Corrupted?

👉 Power losses, unexpected shutdowns, and failing drives can cause corruption.

CHECK

```
# Check kernel error messages
dmesg | grep -i "error"

# Check filesystem integrity (non-destructive)
fsck.ext4 -n /dev/sdX  # for ext4
xfs_repair -n /dev/sdX  # for XFS
```

FIX

1. Repair the filesystem (if unmounted):

```
# Unmount first
umount /dev/sdX

# Run filesystem repair
fsck -y /dev/sdX    # for ext4
xfs_repair /dev/sdX  # for XFS
```

2. If corruption is severe, restore from backup:

```
# Restore using rsync
rsync -av /backup/mnt_data /mnt/data/
```

🔍 Step 5: Are You Out of Inodes?

You might have disk space but still can't create files? Check your inodes!

CHECK

```
# Check inode usage
df -i

# Count files in current directory
find . -type f | wc -l
```

- If inode usage shows 100%, you can't create new files even with free space.
- This happens when you have too many small files.

FIX

1. Clean up temporary files:

```
# Remove old files in /tmp
rm -rf /tmp/*

# Clean package cache (Debian/Ubuntu)
apt-get clean
```

2. Find and remove unnecessary files:

```
# List directories with most files
du -a | sort -n -r | head -n 10
```

2.5 Unit 4 - Operating Running Systems

2.5.1 Unit 4 - Operating Running Systems

Overview

This unit concentrates on the core tasks involved in **operating running systems** in a Linux environment, particularly with Red Hat Enterprise Linux (RHEL). It covers: - Understanding resource usage CPU, memory, disk I/O. - Become familiar with service management frameworks.

Learning Objectives

1. **Monitor and Manage System Resources:**
    - Learn to track CPU, memory, disk, and network usage.
    - Understand how to troubleshoot performance bottlenecks.
2. **Master Service and Process Control:**
    - Gain proficiency with systemd for managing services and understanding dependency trees.
    - Acquire the ability to identify, start, stop, and restart services and processes as needed.
3. **Configure and Interpret System Logs:**
    - Explore journald and syslog-based logging to collect and store vital system events.
    - Develop techniques to analyze log files for troubleshooting and security assessments.
4. **Implement Scheduling and Automation:**
    - Use `cron`, `at`, and `systemd` timers to automate recurring tasks. Understand how automated job scheduling improves reliability and reduces manual intervention.

These objectives ensure learners can sustain, troubleshoot, and improve actively running Linux systems within enterprise environments, reducing downtime and increasing system reliability.

Key Terms and Definitions

Systemd	Journalctl
Cron / At / Systemd Timers	Daemon

2.5.2 Unit 4 Worksheet - Operating Running Systems

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Operations Bridge
- Security Incident Cheatsheet
- Battle Drills

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u4_worksheet(`.txt`)
- 📥 u4_worksheet(`.docx`)

UNIT 4 RECORDING

Link: https://www.youtube.com/watch?v=MulGrUKSMRg

Discussion Post #1

Read this article: https://cio-wiki.org/wiki/Operations_Bridge

1. What terms and concepts are new to you?
2. Which pro seems the most important to you? Why?
3. Which con seems the most costly, or difficult to overcome to you? Why?

Discussion Post #2

> **Scenario**
>
> Your team has no documentation around how to check out a server during an incident. Write out a procedure of what you think an operations person should be doing on the system they suspect is not working properly.

This may help, to get you started https://zeltser.com/media/docs/security-incident-survey-cheat-sheet.pdf?msc=Cheat+Sheet+Blog You may use AI for this, but let us know if you do.

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Detection:

Response:

Mitigation:

Reporting:

Recovery:

Remediation:

Lessons Learned:

After action review:

Operations Bridge:

Digging Deeper

1. Read about battle drills here https://en.wikipedia.org/wiki/Battle_drill
2. Why might it be important to practice incident handling before an incident occurs?
3. Why might it be important to understand your tools before an incident occurs?

Reflection Questions

1. What questions do you still have about this week?
2. How much better has your note taken gotten since you started? What do you still need to work on? Have you started using a different tool? Have you taken more notes?

2.5.3 Unit 4 Lab - Operating Running Systems

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Cron Wiki page
- tldp.org's cron guide

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u4_lab(`.pdf`)
- 📥 u4_lab(`.docx`)

Pre-Lab Warm-Up

1. `cd ~`
2. `ls`
3. `mkdir unit4`
4. `mkdir unit4/test/round6`
   - This fails.
5. `mkdir -p unit4/test/round6`
   - This works, think about why. (`man mkdir`)
6. `cd unit4`
7. `ps`
   - Read `man ps`
8. `ps -ef`
   - What does this show differently?
9. `ps -ef | grep -i root`
   - What is the PID of the 4th line?
10. `ps -ef | grep -i root | wc -l`
    - What does this show you and why might it be useful?
11. `top`
    - Use `q` to exit.
    - Inside top, use `h` to find commands you can use to toggle system info.

Pre-Lab - Disk Speed tests

1. Real quick check for a package that is useful.

```
rpm -qa | grep -i iostat #should find nothing
```

2. Let's find what provides iostat by looking in the YUM (we'll explore more in later lab)

```
dnf whatprovides iostat
```

   - This should tell you that `sysstat` provides `iostat`.

3. Let's check to see if we have it

```
rpm -qa | grep -i sysstat
```

4. If you don't, lets install it

```
dnf install sysstat
```

5. Re-check to verify we have it now

```
rpm -qa | grep -I sysstat
rpm -qi sysstat<version>
iostat # We'll look at this more in a bit
```

While we're working with packages, make sure that Vim is on your system.
This is the same procedure as above.

```
rpm -qa | grep -i vim  # Check if vim is installed
# If it's there, good.
dnf install vim
# If it's not, install it so you can use vimtutor later (if you need help with vi commands)
```

Lab 🧪

1. Gathering system information release and kernel information

```
cat /etc/*release
uname
uname -a
uname -r
```

Run `man uname` to see what those options mean if you don't recognize the values

```
rpm -qa | grep -i kernel
```

What is your kernel number? Highlight it (copy in putty)

```
rpm -qi <kernel from earlier>
```

What does this tell you about your kernel? When was the kernel last updated? What license is your kernel released under?

1. Check the number of disks

```
fdisk -l
ls /dev/sd*
```

- When might this command be useful?
- What are we assuming about the disks for this to work?
- How many disks are there on this system?
- How do you know if it's a partition or a disk?

```
pvs # What system are we running if we have physical volumes?
    # What other things can we tell with vgs and lvs?
```

- Use `pvdisply`, `vgdisplay`, and `lvdisplay` to look at your carved up volumes.
  Thinking back to last week's lab, what might be interesting from each of those?
- Try a command like `lvdisplay | egrep "Path|Size"` and see what it shows.
  - Does that output look useful?
  - Try to `egrep` on some other values. Separate with `|` between search items.
- Check some quick disk statistics

```
iostat -d
iostat -d 2   # Wait for a while, then use crtl + c to break. What did this do? Try changing this to a different number.
iostat -d 2 5 # Don't break this, just wait. What did this do differently? Why might this be useful?
```

- Check the amount of RAM

```
cat /proc/meminfo
free
free -m
```

  - What do each of these commands show you? How are they useful?
- Check the number of processors and processor info

```
cat /proc/cpuinfo
```

What type of processors do you have? Google to try to see when they were released. Look at the flags. Sometimes when compiling these are important to know. This is how you check what execution flags your processor works with.

```
cat /proc/cpuinfo | grep proc | wc -l
```

  - Does this command accurately count the processors?
  - Check some quick processor statistics

```
iostat -c
iostat -c 2 # Wait for a while, then use Ctrl+C to break. What did this do? Try changing this to a different number.
iostat -c 2 5 # Don't break this, just wait. What did this do differently? Why might this be useful?
```

    Does this look familiar to what we did earlier with `iostat`?
- Check the system uptime

```
uptime
man uptime
```

Read `man uptime` and figure out what those 3 numbers represent.
Referencing this server, do you think it is under high load? Why or why not?

- Check who has recently logged into the server and who is currently in

```
last
```

Last is a command that outputs backwards. (Top is most recent). So it is less than useful without using the more command.

```
last | more
```

  - Were you the last person to log in? Who else has logged in today?

```
w
who
whoami
```

    How many other users are on this system? What does the `pts/0` mean on Google?

- Check running processes and services

```
ps -aux | more
ps -ef | more
ps -ef | wc -l
```

  - Try to use what you've learned to see all the processes owned by your user
  - Try to use what you've learned to count up all of those processes owned by your user

- Looking at system usage (historical)
  - Check processing for last day

```
sar | more
```

  - Check memory for the last day

```
sar -r | more
```

Sar is a tool that shows the 10 minute weighted average of the system for the last day.

Sar is tremendously useful for showing long periods of activity and system load. It is exactly the opposite in it's usefulness of spikes or high traffic loads. In a 20 minute period of 100% usage a system may just show to averages times of 50% and 50%, never actually giving accurate info.

Problems typically need to be proactively tracked by other means, or with scripts, as we will see below. Sar can also be run interactively. Run the command `yum whatprovides sar` and you will see that it is the `sysstat` package. You may have guessed that sar runs almost exactly like `iostat`.

- Try the same commands from earlier, but with their interactive information:

```
sar 2  # Ctrl+C to break
sar 2 5
# or
sar -r 2
sar -r 2 5
```

- Check sar logs for previous daily usage

```
cd /var/log/sa/
ls
# Sar logfiles will look like: sa01 sa02 sa03 sa04 sa05 sar01 sar02 sar03 sar04
sar -f sa03 | head
sar -r -f sa03 | head #should output just the beginning of 3 July (whatever month you're in).
```

Most Sar data is kept for just one month but is very configurable. Read `man sar` for more info.

Sar logs are not kept in a readable format, they are binary. So if you needed to dump all the sar logs from a server, you'd have to output it to a file that is readable.

You could do something like this:

- Gather information and move to the right location

```
cd /var/log/sa
pwd
ls
```

We know the files we want are in this directory and all look like this `sa*`

- Build a loop against that list of files

```
for file in `ls /var/log/sa/sa??`; do echo "reading this file $file"; done
```

- Execute that loop with the output command of sar instead of just saying the filename

```
for file in `ls /var/log/sa/sa?? | sort -n`; do sar -f $file ; done
```

- But that is too much scroll, so let's also send it to a file for later viewing

```
for file in `ls /var/log/sa/sa?? | sort -n`; do sar -f $file | tee -a /tmp/sar_data_`hostname`; done
```

- Let's verify that file is as long as we expect it to be:

```
ls -l /tmp/sar_data*
cat /tmp/sar_data_<yourhostname> | wc -l
```

- Is it what you expected? You can also visually check it a number of ways

```
cat /tmp/<filename>
more /tmp/<filename>
```

Exploring Cron

Your system is running the cron daemon. You can check with:

```
ps -ef | grep -i cron
systemctl status crond
```

This is a tool that wakes up between the 1st and 5th second of every minute and checks to see if it has any tasks it needs to run. It checks in a few places in the system for these tasks. It can either read from a crontab or it can execute tasks from files found in the following locations.

`/var/spool/cron` is one location you can `ls` to check if there are any crontabs on your system.

The other locations are directories found under:

```
ls -ld /etc/cron*
```

These should be self-explanatory in their use. If you want to see if the user you are running has a crontab, use the command `crontab -l`. If you want to edit (using your default editor, probably `vi`), use `crontab -e`.

We'll make a quick crontab entry and I'll point you here if you're interested in learning more.

Crontab format looks like this picture:

```
# .------- Minute (0 - 59)
# | .------- Hour (0 - 23)
# | | .------- Day of month (1 - 31)
# | | | .------- Month (1 - 12)
# | | | | .------- Day of week (0 - 6) (Sunday to Saturday - Sunday is also 7 on some systems)
# | | | | |
# | | | | |
  * * * * *  command to be executed
```

Let's do these steps.

1. `crontab -e`
2. Add this line (using vi commands - Revisit `vimtutor` if you need help with them)

```
* * * * * echo 'this is my cronjob running at' `date` | wall
```

3. Verify with `crontab -l`.
4. Wait to see if it runs and echos out to wall.
5. `cat /var/spool/cron/root` to see that it is actually stored where I said it was.
6. This will quickly become very annoying, so I recommend removing that line, or commenting it out (`#`) from that file.

We can change all kinds of things about this to execute at different times. The one above, we executed every minute through all hours, of every day, of every month.

We could also have done some other things:

- Every 2 minutes (divisible by any number you need):

```
*/2 * * * *
```

- The first and 31st minute of each hour:

```
1,31 * * * *
```

- The first minute of every 4th hour:

```
1 */4 * * *
```

- NOTE: If you're adding system-wide cron jobs (`/etc/crontab`), you can also specify the user to run the command as.

```
* * * * * <user> <command>
```

There's a lot there to explore, I recommend looking into the Cron wiki or tldp.org's cron guide for more information.

That's all for this week's lab. There are a lot of uses for all of these tools above. Most of what I've shown here, I'd liken to showing you around a tool box. Nothing here is terribly useful in itself, the value comes from knowing the tool exists and then being able to properly apply it to the problem at hand.

I hope you enjoyed this lab.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.6 Unit 5 - Managing Users and Groups

2.6.1 Unit 5 Lab - Managing Users and Groups

Overview

This unit focuses on managing user's environments and scanning and enumerating Systems. - Become familiar with networking scanning tools - Understand the functionality systems files and customized .(dot) files.

Learning Objectives

1. **Become familiar with Networking mapping:**
    - Learn how to find your network inventory by using nmap.
    - Grasp the basics of targeted scans by scanning virtual boxes and creating a report.
2. **Explore the system files:**
    - Understand the structure of the /etc/passwd file by using the cat command.
    - Customize the /etc/skel file to create a default shell environment for the users.

Key Terms and Definitions

Footprinting	**Scanning**
Enumeration	**System Hacking**
Escalation of Privilege	**Rule of Least Privilege**
Covering Tracks	**Planting Backdoors**

2.6.2 Unit 5 Worksheet - Managing Users and Groups

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- OWASP Top Ten
- attack.mitre.org
- Attack Vectors

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u5_worksheet(`.txt`)
- 📥 u5_worksheet(`.docx`)

UNIT 5 RECORDING

Link: https://www.youtube.com/watch?v=xLv7CIJD6UI

Discussion Post #1

Review the page: https://attack.mitre.org/

1. What terms and concepts are new to you?
2. Why, as a system administrator and not directly in security, do you think it's so important to understand how your systems can be attacked? Isn't it someone else's problem to think about that?
3. What impact to the organization is data exfiltration? Even if you're not a data owner or data custodian, why is it so important to understand the data on your systems?

Discussion Post #2

Find a blog or article on the web that discusses the user environment in Linux. You may want to search for `.bashrc` or (dot) environment files in Linux.

1. What types of customizations might you setup for your environment? Why?
2. What problems can you anticipate around helping users with their dot files?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Footprinting:

Scanning:

Enumeration:

System Hacking:

Escalation of Privilege:

Rule of least privilege:

Covering Tracks:

Planting Backdoors:

Digging Deeper

Map the Internal ProLUG Network (`192.168.200.0/24`):

1. Map the network from one of the rocky nodes.
   Using a template that you build or find from the internet, provide a 1 page summary of what you find in the network.
2. Read this page: https://owasp.org/www-project-top-ten/
   - What is the OWASP Top Ten?
   - Why is this important to know as a system administrator?
3. Read this article: https://www.cobalt.io/blog/defending-against-23-common-attack-vectors
   - What is an attack vector?
   - Why might it be a good idea to keep up to date with these?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

2.6.3 Unit 5 Lab - Managing Users and Groups

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- RedHat: User and Group Management
- Rocky Linux User Admin Guide
- Killercoda lab by FishermanGuyBro

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
    - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u5_lab(`.pdf`)
- 📥 u5_lab(`.docx`)

Pre-Lab Warm-Up

Exercises (Warmup to quickly run through your system and practice commands)

1. `mkdir lab_users`
2. `cd /lab_users`
3. `cat /etc/passwd`
    - We'll be examining the contents of this file later
4. `cat /etc/passwd | tail -5`
    - What did this do to the output of the file?
5. `cat /etc/passwd | tail -5 | nl`
6. `cat /etc/passwd | tail -5 | awk -F : '{print $1, $3, $7}'`
    - What did that do and what do each of the `$#` represent?
    - Can you give the 2nd, 5th, and 6th fields?
7. `cat /etc/passwd | tail -5 | awk -F : '{print $NF}'`
    - What does this `$NF` mean? Why might this be useful to us as administrators?
8. `alias`
    - Look at the things you have aliased.
    - These come from defaults in your `.bashrc` file. We'll configure these later
9. `cd /root`
10. `ls -l`
11. `ll`
    - Output should be similar.
12. `unalias ll`
13. `ll`
    - You shouldn't have this command available anymore.
14. `ls`
15. `unalias ls`
    - How did `ls` change on your screen?

No worries, there are two ways to fix the mess you've made. Nothing you've done is permanent, so logging out and reloading a shell (logging back in) would fix this.
We just put the aliases back.

1. `alias ll='ls -l --color=auto'`
2. `alias ls='ls --color=auto'`
    - Test with `alias` to see them added and also use `ll` and `ls` to see them work properly.

Lab 🧪

This lab is designed to help you get familiar with the basics of the systems you will be working on.

Some of you will find that you know the basic material but the techniques here allow you to put it together in a more complex fashion.

It is recommended that you type these commands and do not copy and paste them. Browsers sometimes like to format characters in a way that doesn't always play nice with Linux.

The Shadow password suite

There are 4 files that comprise of the shadow password suite. We'll investigate them a bit and look at how they secure the system. The four files are `/etc/passwd`, `/etc/group`, `/etc/shadow`, and `/etc/gshadow`.

- Look at each of the files and see if you can determine some basic information about them

```
more /etc/passwd
more /etc/group
more /etc/shadow
more /etc/gshadow
```

There is one other file you may want to become familiar with:

```
more /etc/login.defs
```

Check the file permissions:

```
ls -l /etc/passwd
```

Do this for each file to see how their permissions are set.
You may note that `/etc/passwd` and `/etc/group` are readable by everyone on the system but `/etc/shadow` and `/etc/gshadow` are not readable by anyone on the system.

- Anatomy of the `/etc/passwd` file `/etc/passwd` is broken down like this, a `:` (colon) delimited file:

Username	Password	User ID (UID)	Group ID (GID)	User Info	Home Directory	Login S
`puppet`	`x`	`994`	`991`	`Puppet server daemon`	`/opt/puppetlabs/server/data/puppetserver`	`/sbin/`

`cat` or `more` the file to verify these are values you see.

Are there always 7 fields?

- Anatomy of the `/etc/group` file `/etc/group` is broken down like this, a `:` (colon) delimited file:

Groupname	Password	Group ID	Group Members
`puppet`	`x`	`991`	`foreman, foreman-proxy`

- `cat` or `more` the file to verify these are the values you see. Are there always 4 fields?

We're not going to break down the `g` files, but there are a lot of resources online that can show you this same information.

Suffice it to say, the passwords, if they exist, are stored in an md5 digest format up to RHEL 5. RHEL 6,7,8 and 9 use SHA-512 hash. We cannot allow these to be read by just anyone because then they could brute force and try to figure out our passwords.

Creating and modifying local users

We should take a second to note that the systems you're using are tied into our active directory with Kerberos. You will not be seeing your account in `/etc/passwd`, as that authentication is occurring remotely. You can, however, run `id <username>` to see user information about yourself that you have according to active directory. Your `/etc/login.defs` file is default and contains a lot of the values that control how our next commands work

Creating Users

- Create users with `useradd`:

```
useradd user1
useradd user2
useradd user3
```

- Do a quick check on our main files:

```
tail -5 /etc/passwd
tail -5 /etc/shadow
```

- What `UID` and `GID` were each of these given? Do they match up? Verify your users all have home directories. Where would you check this?

```
ls /home
```

- Your users `/home/<username>` directories have hidden files that were all pulled from a directory called `/etc/skel`. If you wanted to test this and verify you might do something like this:

```
cd /etc/skel
vi .bashrc
```

- Use `vi` commands to add the line:

```
alias dinosaur='echo "Rarw"'
```

- Your file should now look like this:

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
alias dinosaur='echo "Rarw"'
# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=
# User specific aliases and functions
```

- Save the file with `:wq`.

```
useradd user4
su - user4
dinosaur # Should roar out to the screen
```

Doing that changed the `.bashrc` file for **all new users** that have home directories created on the server.
An old trick, when users mess up their login files (all the `.` files), is to move them all to a directory and pull them from `/etc/skel` again. If the user can log in with no problems, you know the problem was something they created.

- We can test this with the same steps on an existing user. Pick an existing user and verify they don't have that command

```
su - user1
dinosaur # Command not found
exit
```

- Then, as root:

```
cd /home/user1
mkdir old_dot_files
mv .* old_dot_files          # Ignore the errors, those are directories
cp /etc/skel/.* /home/user1  # Ignore the errors, those are directories
su - user1
dinosaur # Should 'roar' now because the .bashrc file is new from /etc/skel
```

Creating Groups

- From our `/etc/login.defs` we can see that the default range for UIDs on this system, when created by `useradd` are:

```
UID_MIN 1000
UID_MAX 60000
```

- So an easy way to make sure that we don't get confused on our group numbering is to ensure we create groups outside of that range. This isn't required, but can save you headache in the future.

```
groupadd -g 60001 project
tail -5 /etc/group
```

- You can also make groups the old fashioned way by putting a line right into the `/etc/group` file.
  Try this:

```
vi /etc/group
```

  - ⇧ Shift + G to go to the bottom of the file.
  - Press O to create a new line and go to insert mode.
  - Add `project2:x:60002:user4`
  - Hit `Esc`
  - `:wq!` to write quit the file explicit force because it's a read only file.
- Check the changes:

```
id user4 # Should now see the project2 in the user's groups
```

Modifying or deleting users

- So maybe now we need to move our users into that group.

```
usermod -G project user4
tail -f /etc/group # Should see user4 in the group
```

- But, maybe we want to add more users and we want to just put them in there:

```
vi /etc/group
```

  - ⇧ Shift + G Will take you to the bottom.
  - Hit I (will put you into insert mode).
  - Add `,user1,user2` after `user4`.
  - Hit ⎋ Esc.
  - `:wq` to save and exit.
  - Verify your users are in the group now

```
id user4
id user1
id user2
```

Test group permissions

I included the permissions discussion from an earlier lab because it's important to see how permissions affect what user can see what information.

- Currently we have `user1,2,4` belonging to group `project` but not `user3`. So we will verify these permissions are enforced by the filesystem.

```
mkdir /project
ls -ld /project
chown root:project /project
chmod 775 /project
ls -ld /project
```

- If you do this, you now have a directory `/project` and you've changed the group ownership to `/project`. You've also given group `project` users the ability to write into your directory. Everyone can still read from your directory. Check permissions with users:

```
su - user1
cd /project
touch user1
exit
su - user3
cd /project
touch user3
exit
```

- Anyone not in the `project` group doesn't have permissions to write a file into that directory. Now, as the root user:

```
chmod 770 /project
```

- Check permissions with users:

```
su - user1
cd /project
touch user1.1
exit
su - user3
cd /project # Should break right about here
touch user3
exit
```

You can play with these permissions a bit, but there's a lot of information online to help you understand permissions better if you need more resources.

Working with permissions

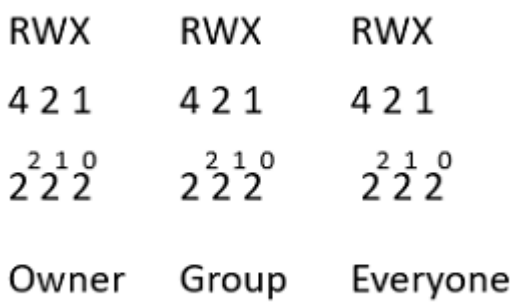
Permissions have to do with who can or cannot access (read), edit (write), or execute (xecute)files. Permissions look like this:

```
ls -l
```

Permission	# of Links	UID Owner	Group Owner	Size (b)	Creation Month	Crea
`-rw-r--r--.`	`1`	`scott`	`domain_users`	`58`	`Jun`	

The primary permissions commands we're going to use are going to be `chmod` (access) and `chown` (ownership).

A quick rundown of how permissions break out:



Let's examine some permissions and see if we can't figure out what permissions are allowed:

```
ls -ld /home/scott/
drwx------. 5 scott domain_users 4096 Jun 22 09:11 /home/scott/
```

The first character lets you know if the file is a directory, file, or link. In this case we are looking at my home directory.

1. `rwx` : For UID (me).
    - What permissions do I have?
2. `---` : For group.
    - Who are they?
    - What can my group do?
3. `---` : For everyone else.
    - What can everyone else do?

Go find some other interesting files or directories and see what you see there.
Can you identify their characteristics and permissions?

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

2.7 Unit 6 - Firewalls

2.7.1 Unit 6 - Firewalls

Overview

This unit focuses on Nohup environments and firewalls.

- We will cover Nohup tools and how to properly use Nohup environments.
- We will explore different types of firewalls and learn the use cases for each firewall type.

Learning Objectives

1. **Become familiar with the `nohup` command:**
    - Learn real-life use cases of the `nohup` command.
    - Understand the correlation between jump boxes and Nohup environments, including `screen` and `tmux`.
2. **Implement and manage Nohup environments:**
    - Learn how `nohup` allows processes to continue running after a user logs out, ensuring that long-running tasks are not interrupted.
    - Develop skills in managing background processes effectively using `nohup`, `screen`, and `tmux`.
3. **Develop effective troubleshooting methodologies:**
    - Acquire systematic approaches to diagnosing firewall misconfigurations, network connectivity issues, and unauthorized access attempts.
    - Apply structured troubleshooting strategies to minimize downtime and maintain high availability.

Key Terms and Definitions

Firewall	Zone
DMZ (Demilitarized Zone)	Proxy
Stateful Packet Filtering	WAF (Web Application Firewall)
NGFW (Next-Generation Firewall)	

2.7.2 Unit 6 Worksheet - Firewalls

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Official Firewalld Documentation
- RedHat Firewalld Documentation
- Official UFW Documentation
- Wikipedia entry for Next-Gen Firewalls

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u6_worksheet(`.txt`)
- 📥 u6_worksheet(`.docx`)

UNIT 6 RECORDING

Link: https://www.youtube.com/watch?v=wCVj3qeLTMg

Discussion Post #1

> **Scenario**
>
> A ticket has come in from an application team. Some of the servers your team built for them last week have not been reporting up to enterprise monitoring and they need it to be able to troubleshoot a current issue, but they have no data. You jump on the new servers and find that your engineer built everything correctly and the agents for node_exporter, ceph_exporter and logstash exporter that your teams use. But, they also have adhered to the new company standard of firewalld must be running. No one has documented the ports that need to be open, so you're stuck between the new standards and fixing this problem on live systems.

Next, answer these questions here:

1. As you're looking this up, what terms and concepts are new to you?
2. What are the ports that you need to expose? How did you find the answer?
3. What are you going to do to fix this on your firewall?

Discussion Post #2

> **Scenario**
>
> A manager heard you were the one that saved the new application by fixing the firewall. They get your manager to approach you with a request to review some documentation from a vendor that is pushing them hard to run a WAF in front of their web application. You are "the firewall" guy now, and they're asking you to give them a review of the differences between the firewalls you set up (which they think should be enough to protect them) and what a WAF is doing.

1. What do you know about the differences now?
2. What are you going to do to figure out more?
3. Prepare a report for them comparing it to the firewall you did in the first discussion.

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Firewall:

Zone:

Service:

DMZ:

Proxy:

Stateful packet filtering:

Stateless packet filtering:

WAF:

NGFW:

Digging Deeper

1. Read https://docs.rockylinux.org/zh/guides/security/firewalld-beginners/
   What new things did you learn that you didn't learn in the lab?
   What functionality of firewalld are you likely to use in your professional work?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

2.7.3 Unit 6 Lab - Firewalls

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Firewalld Official Documentation
- RedHat Firewalld Documentation

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u6_lab(`.pdf`)
- 📥 u6_lab(`.docx`)

Pre-Lab Warm-Up

Exercises (Warmup to quickly run through your system and practice commands)

1. `cd~`
2. `pwd (should be /home/<yourusername>)`
3. `cd /tmp`
4. `pwd (should be /tmp)`
5. `cd`
6. `pwd (should be /home/<yourusername>)`
7. `mkdir lab_firewalld`
8. `cd lab_firewalld`
9. `touch testfile1`
10. `ls`
11. `touch testfile{2..10}`
12. `ls`
13. `seq 10`
14. `seq 1 10`
15. `seq 1 2 10`
    - man seq and see what each of those values mean. It's important to know the behavior if you intend to ever use the command, as we often do with counting (for) loops.

No worries, there are two ways to fix the mess you've made. Nothing you've done is permanent, so logging out and reloading a shell (logging back in) would fix this.
We just put the aliases back.

1. `for i in` seq 1 10 `; do touch file$i; done;`
2. `ls`
    - Think about some of those commands and when you might use them. Try to change command #15 to remove all of those files (rm -rf file$i)

Lab 🧪

This lab is designed to help you get familiar with the basics of the systems you will be working on.

Some of you will find that you know the basic material but the techniques here allow you to put it together in a more complex fashion.

It is recommended that you type these commands and do not copy and paste them. Browsers sometimes like to format characters in a way that doesn't always play nice with Linux.

Check Firewall Status and settings

A very important thing to note before starting this lab. You're connected into that server on ssh via port 22. If you do anything to lockout **port 22** in this lab, you will be blocked from that connection and we'll have to reset it.

1. Check firewall status

```
systemctl status firewalld
```

Example Output:

```
firewalld.service - firewalld - dynamic firewall daemon
Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
Active: inactive (dead) since Sat 2017-01-21 19:27:10 MST; 2 weeks 6 days ago
 Main PID: 722 (code=exited, status=0/SUCCESS)

Jan 21 19:18:11 schampine firewalld[722]: 2017-01-21 19:18:11 ERROR: COMMAND....
Jan 21 19:18:13 schampine firewalld[722]: 2017-01-21 19:18:13 ERROR: COMMAND....
Jan 21 19:18:13 schampine firewalld[722]: 2017-01-21 19:18:13 ERROR: COMMAND....
Jan 21 19:18:13 schampine firewalld[722]: 2017-01-21 19:18:13 ERROR: COMMAND....
Jan 21 19:18:13 schampine firewalld[722]: 2017-01-21 19:18:13 ERROR: COMMAND....
Jan 21 19:18:14 schampine firewalld[722]: 2017-01-21 19:18:14 ERROR: COMMAND....
Jan 21 19:18:14 schampine firewalld[722]: 2017-01-21 19:18:14 ERROR: COMMAND....
Jan 21 19:18:14 schampine firewalld[722]: 2017-01-21 19:18:14 ERROR: COMMAND....
Jan 21 19:27:08 schampine systemd[1]: Stopping firewalld - dynamic firewall.....
Jan 21 19:27:10 schampine systemd[1]: Stopped firewalld - dynamic firewall ...n.
```

Hint: Some lines were ellipsized, use -l to show in full.

If necessary start the firewalld daemon

```
systemctl start firewalld
```

Set the firewalld daemon to be persistent through reboots

```
systemctl enable firewalld
```

Verify with systemctl status firewalld again from **step 1**

Check which zones exist

```
firewall-cmd --get-zones
```

Checking the values within each zone

```
firewall-cmd --list-all --zone=public
```

General Output

```
public (default, active)
interfaces: wlp4s0
sources:
services: dhcpv6-client ssh
ports:
masquerade: no
forward-ports:
icmp-blocks:
rich rules:
```

Checking the active and default zones

```
firewall-cmd --get-default
```

Example Output:

```
public
```

Next Command

```
firewall-cmd --get-active-zones
```

Example Output:

```
public
interfaces: wlp4s0
```

Note: this also shows which interface the zone is applied to. Multiple interfaces and zones can be applied

So now you know how to see the values in your firewall. Use steps 4 and 5 to check all the values of the different zones to see how they differ.

Set the firewall active and default zones

We know the zones from above, set your firewall to the different active or default zones. Default zones are the ones that will come up when the firewall is restarted.

Note: It may be useful to perform an `ifconfig -a` and note your interfaces for the next part

```
ifconfig -a | grep -i flags
```

Example Output:

```
ifconfig -a | grep -i flags
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
ens32: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
```

1. Changing the default zones (This is permanent over a reboot, other commands require --permanent switch)

```
firewall-cmd --set-default-zone=work
```

Example Output:

```
success
```

Next Command:

```
firewall-cmd --get-active-zones
```

Example Output:

```
work
    interfaces: wlp4s0
```

Attempt to set it back to the original public zone and verify. Set it to one other zone, verify, then set it back to public.

Changing interfaces and assigning different zones (use another interface from your earlier `ifconfig -a`

```
firewall-cmd --change-interface=virbr0 --zone dmz
```

Example Output:

```
success
```

Next Command:

```
firewall-cmd --add-source 192.168.11.0/24 --zone=public
```

Example Output:

```
success
```

Next Command:

```
firewall-cmd --get-active-zones
```

Example Output:

```
dmz
   interfaces: virbr0
work
   interfaces: wlp4s0
public
   sources: 192.168.200.0/24
```

Working with ports and services

We can be even more granular with our ports and services. We can block or allow services by port number, or we can assign port numbers to a service name and then block or allow those service names.

1. List all services assigned in firewalld

```
firewall-cmd --get-services
```

Example Output:

```
RH-Satellite-6 amanda-client bacula bacula-client dhcp dhcpv6 dhcpv6-client dns freeipa-ldap freeipa-ldaps freeipa-replication ftp high-availability
http https imaps ipp ipp-client ipsec iscsi-target kerberos kpasswd ldap ldaps libvirt libvirt-tls mdns mountd ms-wbt mysql nfs ntp openvpn pmcd pmproxy
pmwebapi pmwebapis pop3s postgresql proxy-dhcp radius rpc-bind rsyncd samba samba-client smtp ssh telnet tftp tftp-client transmission-client vdsm vnc-
server wbem-https
```

This next part is just to show you where the service definitions exist. They are simple xml format and can easily be manipulated or changed to make new services. This would require a restart of the firewalld service to re-read this directory.

Next Command:

```
ls /usr/lib/firewalld/services/
```

Example Output:

```
amanda-client.xml        iscsi-target.xml  pop3s.xml
bacula-client.xml        kerberos.xml      postgresql.xml
bacula.xml               kpasswd.xml       proxy-dhcp.xml
dhcpv6-client.xml        ldaps.xml         radius.xml
dhcpv6.xml               ldap.xml          RH-Satellite-6.xml
dhcp.xml                 libvirt-tls.xml   rpc-bind.xml
dns.xml                  libvirt.xml       rsyncd.xml
freeipa-ldaps.xml        mdns.xml          samba-client.xml
freeipa-ldap.xml         mountd.xml        samba.xml
freeipa-replication.xml  ms-wbt.xml        smtp.xml
ftp.xml                  mysql.xml         ssh.xml
high-availability.xml    nfs.xml           telnet.xml
https.xml                ntp.xml           tftp-client.xml
http.xml                 openvpn.xml       tftp.xml
imaps.xml                pmcd.xml          transmission-client.xml
ipp-client.xml           pmproxy.xml       vdsm.xml
ipp.xml                  pmwebapis.xml     vnc-server.xml
ipsec.xml                pmwebapi.xml      wbem-https.xml
```

Next Command:

```
cat /usr/lib/firewalld/services/http.xml
```

Example Output:

```
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>WWW (HTTP)</short>
  <description>HTTP is the protocol used to serve Web pages. If you plan to make your Web       server publicly available, enable this option. This optio
n is not required for viewing pages locally or developing Web pages.</description>
  <port protocol="tcp" port="80"/>
</service>
```

1. Adding a service or port to a zone

Ensuring we are working on a public zone

```
firewall-cmd --set-default-zone=public
```

Example Output:

```
success
```

Listing Services

```
firewall-cmd --list-services
```

Example Ouput:

```
dhcpv6-client ssh
```

Note: We have 2 services

Permanently adding a service with the `--permanent` switch

```
firewall-cmd --permanent --add-service ftp
```

Example Output:

```
success
```

Reloading

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

Example Output:

```
success
```

Verifying we are in the correct **Zone**

```
firewall-cmd --get-default-zone
```

Example Output:

```
public
```

Verifying that we have successfully added the **FTP** service

```
firewall-cmd --list-services
```

Example Output:

```
dhcpv6-client ftp ssh
```

Alternatively, we can do almost the same thing but not use a defined service name. If I just want to allow port 1147 through for TCP traffic, it is very simple as well.

```
firewall-cmd --permanent --add-port=1147/tcp
```

Example Output:

```
success
```

Reloading once again

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

Example Output:

```
success
```

Listing open ports now

```
firewall-cmd --list-ports
```

Example Output:

```
1147/tcp
```

1. Removing unwanted services or ports

To remove those values and permanently fix the configuration back we simply use remove.

Firstly, we will permanently remove ftp service

```
firewall-cmd --permanent --remove-service=ftp
```

Example Output:

```
success
```

Then we will permanently remove the ports

```
firewall-cmd --permanent --remove-port=1147/tcp
```

Example Output:

```
success
```

Now lets do a reload

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

Example Output:

```
success
```

Now we can list services again to confirm our work

```
firewall-cmd --list-services
```

Example Output:

```
dhcpv6-client ssh
```

Now we can list ports

```
firewall-cmd --list-ports
```

Example Output:

`Nothing`

Before making any more changes I recommend running the `list` commands above with `>> /tmp/firewall.orig` on them so you have all your original values saved somewhere in case you need them.

So now take this and set up some firewalls on the interfaces of your system. Change the default ports and services assigned to your different zones (at least 3 zones) Read the `man firewall-cmd` command or `firewall-cmd -help` to see if there are any other userful things you should know.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.8 Unit 7 - Package Management && Patching

2.8.1 Unit 7 - Package Management & Patching

Overview

Managing software on a Linux system involves two essential practices: package management and patching. Together, they ensure that systems remain functional, up-to-date, and secure by handling software installation, updates, and vulnerability remediation.

- **Package Management:** the system used to install, upgrade, configure, and remove software packages while automatically resolving dependencies and maintaining system consistency.
- **Patching:** the process of applying updates to software packages or the kernel to fix bugs, close security vulnerabilities, or improve performance.

-

Learning Objectives

1. **Become Familiar with Package Management:**
    - In this unit you will see what comprises a RPM, YUM, and RPM package and how professional administrators carefully choose what is installed on a system.
2. **Become Familiar with Patching:**
    - Learn about general patching cycles.
    - Understand why it matters to inpsect packages and associated dependancies.
    - Get hands on with inspecting packages.

Key terms and Definitions

Yum	**DNF**
Repo	**GPG Key**
Software dependency**	**Software version**
Semantic Version	

2.8.2 Unit 7 Worksheet - Package Management & Patching

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Semantic Versioning
- Rocky Documentation
- Rocky DNF Guidance

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u7_worksheet(`.txt`)
- 📥 u7_worksheet(`.docx`)

UNIT 7 RECORDING

Link: https://www.youtube.com/watch?v=SwrkhTwzN_4

Discussion Post #1

1. Why is software versioning so important to software security?
2. Can you find 3 reasons, from the internet, AI, or your peers?

Discussion Post #2

> **Scenario**
>
> You are new to a Linux team. A ticket has come in from an application team and has already been escalated to your manager.
>
> They want software installed on one of their servers but you cannot find any documentation. Your security team is out to lunch and not responding.
>
> You remember from some early documentation that you read that all the software in the internal repos you currently have are approved for deployment on servers. You want to also verify by checking other servers that this software exists.
>
> This is an urgent task and your manager is hovering.

1. How can you check all the repos on your system to see which are active?
2. How would you check another server to see if the software was installed there?
3. If you find the software, how might you figure out when it was installed? (Time/Date)

Discussion Post #3

> **Scenario**
>
> Looking at the concept of group install from DNF or Yum. Why do you think an administrator may never want to use that in a running system? Why might an engineer want to or not want to use that?

This is a thought exercise, so it's not a "right or wrong" answer it's for you to think about.

1. What is the concept of software bloat, and how do you think it relates?
2. What is the concept of a security baseline, and how do you think it relates?
3. How do you think something like this affects performance baselines?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Yum:

DNF:

Repo:

GPG Key:

Software dependency:

Software version:

Semantic Version:

Digging Deeper

1. What is semantic versioning? https://semver.org/

Reflection Questions

1. What questions do you still have about this week?
2. How does security as a system administrator differ from what you expected?

2.8.3 Unit 7 Lab - Package Management & Patching

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Rocky Documentation
- Rocky DNF Guidance
- Rocky Repository Guidance

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u7_lab(`.pdf`)
- 📥 u7_lab(`.docx`)

Pre-Lab Warm-Up

A couple commands to get the noodle and fingers warmed up.

For clarity:

DNF is the 'frontend' aspect of the Rocky package management apparatus and RPM (RHEL package manager) is the 'backend'. The frontend abstracts away and automates the necessary commands required to install and manipulate packages.

RPM allows for finer control compared to its related frontend and is intended for more advanced use cases. The Debian/Ubuntu equivalents are the apt frontend and dpkg backend.

Investigate the man pages for additional information.

```
cd ~
rpm -qa | more
rpm -qa | wc -l
# pick any <name of package> from the above list

rpm -qi {name of package}
rpm -qa | grep -i imagemagick

dnf install imagemagick

# What is the error here? Read it
dnf install ImageMagick

# What are some of the dependencies here? Look up the urw-base35
# and see what functionality that adds.
rpm -qa | grep -i imagemagick

# Why did this work when the other one didn't with dnf?
```

Math Practice

Some fun with the command line and basic scripting tools. I want you to see some of the capabilities that are available to you. Your system can do a lot of basic arithmetic for you and this is a very small set of examples.

```
# Check to see if you have bc tool.
rpm -q bc

#Install it if you need to
dnf install bc

for i in `seq 1 5`; do free | grep -i mem | awk '{print $3}'; done

# Collect the 5 numbers (what do these numbers represent? Use free to find out)
echo "(79 + 79 + 80 + 80 + 45) / 5" | bc

# Your numbers will vary. Is this effective? Is it precise enough?
echo "(79 + 79 + 80 + 80 + 45) / 5" | bc -l
# Is this precise enough for you?

# Read the man to see what the -l option does to bc
man bc
```

It would be astute to point out that I did not have you do bash arithmetic. There is a major limitation of using bash for that purpose in that it only wants to deal with integers (whole numbers) and you will struggle to represent statistical data with precision. There are very useful tools though, and I would highly encourage you to examine them. http://tldp.org/LDP/abs/html/arithexp.html

Lab 🧪

Log into your Rocky server and become root.

RPM

RPM is the Redhat package manager. It is a powerful tool to see what is installed on your system and to see what dependencies exist with different software packages. This is a tool set that was born of the frustration of “dependency nightmare” where system admins used to compile from source code only to find they had dependencies. RPM helps to de-conflict and save huge amounts of time and engineering headaches.

Run through these commands and read `man rpm` to see what they do.

```
# Read about the capabilities of systemd
rpm -qi systemd

# query the package given
rpm -q systemd

# query all packages on the system (is better used with | more or | grep)
rpm -qa

#for example shows all kernels and kernel tools
rpm -qa | grep -i kernel

# List out files, but only show the configuration files
rpm -qc systemd

# What good information do you see here? Why might it be good to know
# that some piece of software was installed last night, if there is now
# a problem with the system starting last night?
rpm -qi systemd

# Will list all the files in the package. Why might this be useful to you to know?
rpm -ql systemd

# List capabilities on which this package depends
rpm -qR systemd

# Probably going to scroll too fast to read. This output is in reverse order.
rpm -q -changelog systemd

# So let's make it useful with this command
rpm -q -changelog systemd | more

# What are some of the oldest entries?
# What is the most recent entry?
# Is there a newer version of systemd for you to use?

# If there isn't don't worry about it.
dnf update systemd
```

Use `rpm -qa | more` to find 3 other interesting packages and perform `rpm -qi <package>` on them to see information about them.

DNF

DNF is the front end package manager implemented into Rocky and derives its roots from Yum, a long decrepit version of Linux called Yellow dog. It is originally the Yellowdog Update Manager. It has a very interesting history surrounding the PS3, but that and other nostalgia can be found here: https://en.wikipedia.org/wiki/Yellow_Dog_Linux if you're interested.

We're going to use DNF to update our system. RHEL and CentOS systems look to repositories of software for installation and updates. We have a base set of them provided with the system, supported by the vendor or open source communities, but we can also create our own from file systems or web pages. We'll be mostly dealing with the defaults and how to enable or disable them, but there are many configurations that can be made to customize software deployment.

```
# Checking how dnf is configured and seeing it's available repositories
cat /etc/dnf/dnf.conf

# has some interesting information about what is or isn't going to be checked.
# You can include a line here called exclude= to remove packages from installation
# by name. Where a repo conflicts with this, this takes precedence.
dnf repolist
dnf history

# Checking where repos are stored and what they look like
ls /etc/yum.repos.d/

# Repos are still stored in /etc/yum.repos.d
cat /etc/yum.repos.d/rocky.repo
```

The mirror list system uses the connecting IP address of the client and the update status of each mirror to pick current mirrors that are geographically close to the client. You should use this for Rocky updates unless you are manually picking other mirrors.

If the mirrorlist does not work for you, you can try the commented out baseurl line instead.

```
[baseos]
name=Rocky Linux $releasever - BaseOS
mirrorlist=https://mirrors.rockylinux.org/mirrorlist?arch=$basearch&repo=BaseOS-$releasever$rltype
#baseurl=http://dl.rockylinux.org/$contentdir/$releasever/BaseOS/$basearch/os/
gpgcheck=1
enabled=1
countme=1
metadata_expire=6h
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-Rocky-9
#Output truncated for brevity's sake….
```

Something you'll find out in the next section looking at repos is that when they are properly defined they are enabled by default. enabled=1 is implied and doesn't need to exist when you create a repo.

```
# Let's disable a repo and see if the output changes at all
dnf config-manager --disable baseos

# Should now have the line enabled=0 (or false, turned off)
cat /etc/yum.repos.d/rocky.repo

[baseos]
name=Rocky Linux $releasever - BaseOS
mirrorlist=https://mirrors.rockylinux.org/mirrorlist?arch=$basearch&repo=BaseOS-$releasever$rltype
# baseurl=http://dl.rockylinux.org/$contentdir/$releasever/BaseOS/$basearch/os/
gpgcheck=1
enabled=0
countme=1
metadata_expire=6h
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-Rocky-9
# Output truncated for brevity's sake….

# Re-enable the repo and verify the output
dnf config-manager --enable baseos

# Should now have the line enabled=1 (or true, turned back on)
cat /etc/yum.repos.d/rocky.repo

# Output:
[baseos]
name=Rocky Linux $releasever - BaseOS
mirrorlist=https://mirrors.rockylinux.org/mirrorlist?arch=$basearch&repo=BaseOS-$releasever$rltype
#baseurl=http://dl.rockylinux.org/$contentdir/$releasever/BaseOS/$basearch/os/
gpgcheck=1
enabled=1
countme=1
metadata_expire=6h
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-Rocky-9
# Output truncated for brevity's sake….
```

Installing software you were asked by an application team

So someone has asked for some software and assured you it's been tested in similar environments, so you go to install it on their system for them.

```
# See if we already have a version.
rpm -qa mariadb

# See if dnf knows about it
dnf search mariadb
dnf search all mariadb

# What is DNF showing you? What are the differences between these commands based on the output?

# Try to install it
dnf install mariadb
# hit "N"

# Make note of any dependencies that are added on top of mariadb (there's at least one)
# What does DNF do with the transaction when you cancel it? Can you compare this
# to what you might have used before with YUM? How are they different?
# (You can look it up if you don't know.)

# Ok, install it
dnf -y install mariadb

# Will just assume yes to everything you say.
# You can also set this option in /etc/dnf/dnf.conf to always assume yes,
# it's just safer in an enterprise environment to be explicit.
```

Removing packages with dnf:

Surprise, the user calls back because that install has made the system unstable. They are asking for you to remove it and make the system back to the recent version.

```
dnf remove mariadb
# hit "N"

# this removes mariadb from your system
dnf -y remove mariadb

# But did this remove those dependencies from earlier?
rpm -q {dependency}
rpm -qi {dependency}

# How are you going to remove that if it's still there?
# Checking where something came from. What package provides something in your system

# One of the most useful commands dnf provides is the ability to know "what provides"
# something. Sar and iostat are powerful tools for monitoring your system.
# Let's see how we get them or where they came from, if we already have them.
# Maybe we need to see about a new version to work witha new tool.
dnf whatprovides iostat
dnf whatprovides sar

# Try it on some other tools that you regularly use to see where they come from.
dnf whatprovides systemd
dnf whatprovides ls
dnf whatprovides python

# Using Dnf to update your system or individual packages
# Check for how many packages need update
dnf update

# How many packages are going to update?
# Is one of them the kernel?
# What is the size in MB that is needed?
# Hit "N"

# Your system would have stored those in /var/cache/dnf
# Let's check to see if we have enough space to hold those
df -h /var/cache/dnf

# Is there more free space than there is needed size in MB from earlier?
# There probably is, but this becomes an issue. You'd be surprised.

# Let's see how that changes if we exclude the kernel
dnf update --exclude=kernel

# How many packages are going to update?
# Is one of them the kernel?
# What is the size in MB that is needed?
# Hit "N"
```

You can update your system if you like. You'd have to reboot for your system to take the new kernel. If you do that you can then redo the grubby portion and the ls /boot/ will show the new installed kernel, unless you excluded it.

Using dnf to install group packages

Maybe we don't even know what we need to get a project going. We know that we need to have a web server running but we don't have an expert around to tell us everything that may help to make that stable. We can scour the interwebs (our normal job) but we also have a tool that will give us the base install needed for RHEL or CentOS to run that server.

```
dnf grouplist
dnf group install development

# How many packages are going to update?
# Is one of them the kernel?
# What is the size in MB that is needed?
# Hit "N"
# Do you see a pattern forming?
```

If you install this you're going to have developer tools installed on the server but they won't be configured. How would you figure out what tools and versions were just installed? How might you report this for your own documentation and to a security team that keeps your security baselines?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.8.4 Unit 7 Bonus - Package Management & Patching

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

This bonus explores how you can audit and verify software integrity on your system using package tools, hashes, and file validation -- going deeper into real-world sysadmin practice.

This is more of a bonus lab. We're going beyond just *installing* packages. We're going to audit, validate, and verify that the software on our system is trustworthy and unmodified.
We'll explore how to detect unexpected changes using built-in tools, dig into package metadata, and get a taste of real-world security practices like intrusion detection and system baselining through package auditing.

In modern enterprise environments, packages may be tampered with, misconfigured, or out-of-date.

A responsible sysadmin needs tools and methods to answer questions like:

- Was this package installed from a trusted source?
- Have any of the installed files been modified?
- Which files belong to which packages?
- Can I detect and recover from unexpected changes?

Let's get into it.

Verifying Package Integrity

Start by finding a package you know is installed and used in your environment -- for example, `sshd`:

```
rpm -qi openssh-server
```

Now, check the integrity of the package's files:

```
rpm -V openssh-server
```

- `-V`: Stands for **verify**.
    - This option checks timestamps, permissions, ownership, and hashes of installed files.

If you don't see any output, that's a good thing.

`rpm -V` only reports files that have been altered in some way from what the package database expects. If there is no output, it means all files match the expected checksums, sizes, permissions, etc..

If this command **does** have output, being able to interpret the output is important. Each character in the output has its own meaning:

- `S` - Size differs.
- `M` - Mode differs (permissions).
- `5` - MD5 checksum mismatch.
- `T` - Modification time differs.

This is a great way to verify the integrity of installed packages. It's also helpful in troubleshooting when a package isn't working as expected.

Auditing a File in a Package

Let's say you suspect something has been changed or tampered with. Let's get all files from a package.

- Run `rpm -ql` to list the files that were installed with a package:

```
rpm -ql openssh-server
```

- Now pick one file and manually generate its sha256 hash:

```
sha256sum /usr/sbin/sshd
```

- Download the original `.rpm` package to compare its hash.

```
dnf download openssh-server
```

  - This will download the `openssh-server-<version>.rpm` package in the current directory.
  - These `.rpm` packages are not stored on the system by default.

- You can inspect the file of your choice with `rpm -qp --dump` :

```
rpm -qp --dump openssh-server*.rpm | grep ^/usr/sbin/sshd
```

This will output a bunch of information about the file.
The `sha256` hash will be in the fourth column, so we can use `awk` to extract that:

```
rpm -qp --dump openssh-server*.rpm | grep ^/usr/sbin/sshd | awk '{print $4}'
```

- Compare your version's hash to the original RPM file's hash:

```
sha256sum /usr/sbin/sshd
```

If the hashes are different, the file has been modified.

Bonus Challenge 💡

1. Run this one-liner to verify all installed packages:

```
rpm -Va
```

   - This will verify every file from every package and report anything suspicious.

2. Narrow the scope. Only show actual modified files:

```
rpm -Va | grep -v '^..5'
```

   - This removes lines where only the MD5 checksum differs (which could be expected in some config files).
   - You'll now see files where size, mode, owner, or timestamp changed — higher confidence indicators of real change.

3. Investigate a suspicious result. If you see something like:

```
.M....... c /etc/ssh/sshd_config
```

   That means:

   - The permissions (`M`) have changed.
   - It's a config file (`c`).

4. Check the file in question:

```
ls -l /etc/ssh/sshd_config
```

5. Compare that to what you expected:

```
rpm -q --qf '%{NAME} %{VERSION}-%{RELEASE}\n' -f /etc/ssh/sshd_config
```

Then you can reinstall the package or extract the original file from the `.rpm` file.

REFLECTION QUESTIONS

- What happens if you manually modify a file, then verify with `rpm -V`?
- Can you identify if changes were made outside of DNF/RPM?
- What types of files are typically most important to verify?

EXAMPLE OF REAL-WORLD SECURITY TOOLS

Large enterprises often use tools like AIDE (Advanced Intrusion Detection Environment) or Tripwire to baseline their systems and detect changes over time.

AIDE can be installed easily with `dnf`, so you can play around with it if you want. To set up AIDE on your system (as root):

```
dnf install aide -y

aide --init

# Copy the default database to use as your database
cp /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz

# Then, to run a check with aide (this will take a few minutes):
aide --check
```

AIDE compares the current state of the system to a known baseline.

This is foundational to change management, compliance, and intrusion detection.

Resources

- RPM Man Page
- AIDE Documentation
- Using sha256sum

2.9 Unit 8 - Scripting

2.9.1 Unit 8 - Scripting

Overview

This unit focuses on scripting and system checks in Linux environments, with particular emphasis on bash scripting for system administration tasks. It covers:

- **Bash Scripting Fundamentals:** Mastery of shell scripting is essential for automating routine administrative tasks, implementing monitoring solutions, and creating custom tools that enhance system management capabilities.
- **System Monitoring and Checks:** Linux administrators must continuously monitor system health, resource utilization, and potential security issues. This unit explores techniques for creating scripts that perform automated system checks, gather performance metrics, and alert administrators to potential problems.
- **Logical Flow and Decision Making:** The ability to implement complex decision-making logic in scripts is crucial for handling various system conditions and scenarios. Students will learn to use conditional statements, comparison operators, and truth tables to create intelligent scripts that can adapt to different situations.
- **Automation and Scheduled Tasks:** Effective system administration requires automating repetitive tasks and scheduling routine maintenance. This unit covers techniques for creating scripts that can be executed automatically through cron jobs or systemd timers, reducing manual intervention.

Learning Objectives

1. **Create and Execute Bash Scripts:**
   - Develop proficiency in writing and executing bash scripts for system administration tasks.
   - Learn to use variables, conditional statements, and loops effectively in scripts.
2. **Apply Logical Structures and Decision Making:**
   - Master the use of if/then/else statements, case statements, and logical operators.
   - Understand truth tables and how they apply to script logic.
   - Learn to implement complex decision trees that handle multiple conditions.
3. **Develop Error Handling and Logging:**
   - Implement robust error detection and handling in scripts.
   - Create comprehensive logging systems that facilitate troubleshooting.
   - Design scripts that can recover from common error conditions.
4. **Analyze and Improve Script Maintainability:**
   - Recognize patterns of poor script design and implement improvements.
   - Organize code with functions and meaningful variable names.
   - Document scripts effectively for future maintenance.

Key Terms and Definitions

Bash (Bourne Again Shell)	**Script**
Variables	**Conditional Statements**
Loops	**Exit Status**
Command Substitution	**Interpreted Program**
Compiled Program	**Truth Table**
Single/Dual/Multiple Alternative Logic	**Cron**
System Check	**Parameter Expansion**
And/Or Logic	

2.9.2 Unit 8 Worksheet - Scripting

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- TLDP Bash Beginner's Guide
- devhints.io - Bash Scripting Cheatsheet
- Bash Hacker's Wiki

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u8_worksheet(`.txt`)
- 📥 u8_worksheet(`.docx`)

UNIT 8 RECORDING

Link: https://www.youtube.com/watch?v=sn_LwfIObQw

Discussion Post #1

> **Scenario**
>
> It's a 2 week holiday in your country and most of the engineers and architects who designed the system are out of town.
>
> You've noticed a pattern of logs filling up on a set of web servers from increased traffic. Your research shows, and then you verify, that the logs are being sent off real time to Splunk. Your team has just been deleting the logs every few days, but one of the 3rd shift engineers didn't read the notes and your team suffered downtime. How might you implement a simple fix to stop gap the problem before all the engineering resources come back next week?

1. What resources helped you answer this?
2. Why can't you just make a design fix and add space in /var/log on all these systems?
3. Why can't you just make a design change and logrotate more often so this doesn't happen?
4. For 2,3 if you are ok with that, explain your answer. (This isn't a trick, maybe there is a valid reason.)

Discussion Post #2

> **Scenario**
>
> You are the only Linux Administrator at a small healthcare company. The engineer/admin before you left you a lot of scripts to untangle. This is one of our many tasks as administrators, so you set out to accomplish it. You start to notice that he only ever uses nested `if` statements in bash.
>
> You also notice that every loop is a conditional `while true`, and then he breaks the loop after a decision test each loop. You know his stuff works, but you think it could be more easily written for supportability, for you and future admins. You decide to write up some notes by reading some google, AI, and talking to your peers.

1. Compare the use of nested if versus case statement in bash.
2. Compare the use of conditional and counting loops. Under what circumstances would you use one or the other?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Variables:

Interpreted program:

Compiled program:

Truth table:

AND/OR logic:

Single/Dual/Multiple alternative logic:

Digging Deeper

1. Read:
2. https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
3. https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html
4. https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html

What did you learn about capabilities of bash that can help you in your scripting?

1. If you want to dig more into truth tables and logic, this is a good start: https://en.wikipedia.org/wiki/Truth_table

Reflection Questions

1. What questions do you still have about this week?
2. Just knowing a lot about scripting doesn't help much against actually doing it in a practical sense. What things are you doing currently at work or in a lab that you can apply some of this logic to?

2.9.3 Unit 8 Lab - Scripting

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Wikipedia Page for Compilers
- Wikipedia Page for the C Programming Lang
- Wikipedia Page for Truth Table
- CProgramming.com Introductory
- Unit #1 Bonus (VIM) Page
- Unit #8 Bonus (Bash Scripting) Page

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u8_lab(`.pdf`)
- 📥 u8_lab(`.docx`)

Pre-Lab Warm-Up

```
vi /etc/passwd
# Put a # in front of all your local users you created in a lab a few weeks back
```

Review how to use vi, if you have a problem getting out or saving your file or use Unit #1 Bonus (Vim) Page to brush up on your Vim Skills.

```
# Let us locate and inspect the GNU C Compiler Package
rpm -qa | grep -i gcc
dnf whatprovides gcc
dnf search gcc
Check out all the options of different compilers

# Now lets install it
dnf install gcc
# Look at what is going to be installed.

rpm -qi gcc
# Look at this package to see a little about what gcc does
# Repeat steps 2-6 for the software package strace
```

COMPILERS

A brief look at compilers and compiling code So we did all this just to show you a quick rundown of how compiled code works on your system. We are going to be doing scripting today, which is not compiled, but rather interpreted code according to a type of interpreter so we'll just breeze through this part. You can come back and play with this at your leisure, I will provide links for more study.

```
Let's write a C program
mkdir c_practice
cd c_practice

#Start a new file with Vi, in this case I am going with 'a'
vi a.c

#Add this to the file:
```

```
#include <stdio.h>

main()
{

printf("My first compiled program \n");

return 0;
}
```

```
#Let's use gcc to compile that program
gcc a.c
#This will create a file for you called a.out
#If there is an error, does it still work?

#Alternatively, and more correctly, use this:
gcc -o firstprogram a.c
#Which will create an executable file called first program
ls -salh
#Will show you all your files. Note how big those compiled programs are.
#Execute your programs
./a.out
./firstprogram

#Both of these should do the exact same thing, as they are the same code.
#Watch what your system is doing when you call it via strace
strace ./a.out
strace ./firstprogram
```

Lab 🧪

Log into your Rocky server and become root.

Module 2.1: Scripting

After all that pre-lab discussion, we won't be using `gcc` today—or compiling any programs, for that matter. Instead, we'll focus on scripting, where we write code that the system interprets and executes step by step. Think of it like reading lines from a play, following sheet music, or executing a script—each command is processed in order.

There are plenty of resources available to learn scripting, but the key to improving is daily practice. If you're serious about getting better, I recommend studying additional concepts over time. However, to get started, you only need to understand three fundamental ideas:

1. **Input and Output** - How to receive input and where to send the output.
2. **Conditionals** - How to test and evaluate conditions.
3. **Loops** - How to repeat actions efficiently.

2.2 GETTING INPUT

Let's use examples from our Operate Running Systems lab to see what it looks like to gather system information and store it in variables. Variables in scripting can be thought of as named boxes where we put things we want to look at or compare later. We can, by and large, stuff anything we want into these boxes.

```
# Try this:

echo $name  # No output
uname
name=`uname`
echo $name

echo $kernel  # No output
uname -r
kernel=`uname -r`
echo $kernel

echo $PATH  # This will have output because it is an environment variable
```

There will be output because this is one of those special variables that make up your environment variables. You can see them with:

```
printenv | more
```

These should not be changed, but if necessary, they can be. If you overwrite any, you can reset them by re-logging into your shell.

We can package things in variables and then reference them by their name preceded by a `$`.

```
# Try this to get numerical values from your system for later use in conditional tests.
cat /proc/cpuinfo  # Not very good as a count
cat /proc/cpuinfo | grep -i proc  # Shows processor count but not ideal for testing
cat /proc/cpuinfo | grep -i proc | wc -l  # Outputs a usable number
numProc=`cat /proc/cpuinfo | grep -i proc | wc -l`
echo $numProc

free -m  # Displays memory info
free -m | grep -i mem  # Filters output
free -m | grep -i mem | awk '{print $2}'  # Extracts total memory value
memSize=`free -m | grep -i mem | awk '{print $2}'`
echo $memSize
```

2.3 CHECKING EXIT CODES

One of the most important inputs in scripting is checking the exit code (`$?`) of a command. This allows us to determine whether a command succeeded or failed.

```
ps -ef | grep -i httpd | grep -v grep
echo $?
# Output: 1 (Nothing found, not good "0")

ps -ef | grep -i httpd
root      5514 17748  0 08:46 pts/0    00:00:00 grep --color=auto -i httpd
echo $?
# Output: 0 (Process found, exit code is 0)
```

Checking for installed packages:

```
rpm -qa | grep -i superprogram
echo $?
# Output: 1 (Program not found)

rpm -qa | grep -i gcc
libgcc-4.8.5-11.el7.x86_64
gcc-4.8.5-11.el7.x86_64
echo $?
# Output: 0 (GCC is found)
```

`$?` only holds the exit status of the last executed command, so store it immediately:

```
rpm -qa | grep -i superprogram
superCheck=$?

rpm -qa | grep -i gcc
gccCheck=$?

echo $superCheck
echo $gccCheck
```

2.4 TESTING AND EVALUATING CONDITIONS

2.4.1 Basics of Logic and Truth Tests

I commonly say that **"All engineering is the test for truth."** This is not meant as a philosophical statement but a practical one. We take input, verify it, and compare it to our expectations. If it matches, the result is `true`; otherwise, it is `false`.

Testing for what something `is` is much easier than testing for what something `is not`, as logically, there are infinite possibilities for what something could `not` be.

Continue exploring these concepts by practicing input handling, storing values in variables, and testing conditions to build efficient scripts.

2.5 EXERCISE

The `Red bunny` is tall. We look at our examples and see that this is not true, so the statement evaluates to `false`.
The `Blue bunny` is short. We look at our examples and see that this is not true, so the statement evaluates to `false`.

THE IDEA OF `AND` AND `OR`

- `AND` is a restricting test.
- `OR` is an inclusive test.

I will prove that here shortly.

- `ANDing` checks both sides for truth and evaluates to *true* only if `both` sides are true.
- `ORing` allows either side to be true, and the statement still evaluates to *true*. This makes OR a more inclusive test.

`AND` Examples

- The `right bunny` is `Red` and `Tall`.
  - This evaluates to `true` for the *Red* test but `false` for the *Tall* test.
  - The statement evaluates to `false`.
- The `left bunny` is `Blue` and `Tall`.
  - This evaluates to `true` for the *Blue* test and `true` for the *Tall* test.
  - The statement evaluates to `true`.

`OR` Examples

- The `right bunny` is `Red` or `Tall`.
  - This evaluates to `true` for the *Red* test but `false` for the *Tall* test.
  - The statement evaluates to `true`.
- The `left bunny` is `Red` or `Short`.
  - This evaluates to `false` for *Red* and `false` for *Short*.
  - The statement evaluates to `false`.

2.6 - TRUTH TABLES

Google **Truth Tables** to see engineering diagrams commonly used for testing truth in complex statements.
We will not draw them out in this lab, as there are already well-documented examples. This is a **well-known, solved, and understood concept** in the engineering world. Instead of reinventing those diagrams, refer to the following link for more details:

Truth Table - Wikipedia

2.7 - FLOW IN A PROGRAM

- All programs start at the top and run to the bottom. Data never flow back towards the start, unless on a separate path from a decision which always returns to the original path.

When we start thinking about how to lay something out and logically work through it, the idea of a formalized flow chart can help us get a handle on what we're seeing.

Some common symbols you'll see as we go through drawing out our logic. This example creates a loop in the program until some decision evaluates to yes (true).

2.8 - 3 TYPES OF DECISIONS

There are 3 primary types of decisions you'll run into with scripting, they are:

1. `Single` alternative
2. `Dual` alternative
3. `Multiple` alternative

2.8.1 - Single Alternative `if/then`

Single alternatives either occur or they do not. They only branch from the primary path if the condition occurs. They either can or cannot occur, depending on the condition, but compared to alternative paths where a decision must occur if these do not evaluate to true, they are simply passed over.

Evaluate these from earlier and look at the difference.

```
1  if [ $superCheck -eq "0" ]; then echo "super exists"; fi
```

```
1  if [ $gccCheck -eq "0" ]; then echo "gcc exists"; fi
```

You'll note that only one of them caused any output to come to the screen, the other simply ran and the condition never had to execute.

2.8.2 - Dual alternative (if/then/else)

Dual alternatives forces the code to split. A decision must be made. These are logically `if, then, else`. We test for a truth, and then, if that condition does not exist we execute the alternative. If you're a parent or if you ever had a parent, this is the dreaded `or else`. One of two things is going to happen here, the path splits.

```
1  if [ $superCheck -eq "0" ]; then echo "super exists"; else echo "super does not exist"; fi #super does not exist
```

```
1  if [ $gccCheck -eq "0" ]; then echo "gcc exists"; else echo "gcc does not exist"; fi #gcc exists
```

2.8.3 - Multiple Alternative (if/then/elif/.../else or Case)

Multiple alternatives provide a branch for any numbers of ways that a program can go. They can be structured as if, then, `else if elif` in bash, `else`. They can also be framed in the case statement, which can select any number of cases (like doors) that can be selected from. There should always be a default `else` value for case statements, that is to say, if one of the many conditions don't exist there is something that happens anyways (*) in case statements.

```
1  superCheck=4
2  if [ $superCheck -eq "0" ]; then echo "super exists"; elif [ $superCheck -gt "1" ]; then echo "something is really wrong"; else echo "super does not
   exist"; fi
```

```
1  gccCheck=5
2  if [ $gccCheck -eq "0" ]; then echo "gcc exists"; elif [ $gccCheck -gt "1" ]; then echo "something is really wrong"; else echo "gcc does not exist"; fi
```

Set those variables to the conditions of 0, 1, and `anything else` to see what happens.

Think about why greater than 1 does not hit the condition of 1. Might it be easier to think of as greater than or equal to 2? Here's a list of things you can test against. http://tldp.org/LDP/abs/html/tests.html

Also a huge concept we don't have a lot of time to cover is found here: File test operators http://tldp.org/LDP/abs/html/fto.html, do files exist and can you do operating system level things with them?

We didn't get to go into case, but they are pretty straight forward with the following examples: http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html

We didn't get to explore these much earlier, but to test `AND` and `OR` functionality use this.

`AND` condition

```
if [ $gccCheck -eq "0" -a $superCheck -eq "1" ]; then echo "We can install someprogram"; else echo "We can't install someprogram"; fi
```

We can't install someprogram

`OR` condition

```
if [ $gccCheck -eq "0" -o $superCheck -eq "1" ]; then echo "We can install someprogram"; else echo "We can't install someprogram"; fi
```

We can't install someprogram

2.8.4 - Looping around

As with everything today, this is simply a primer and there are hundreds to thousands of examples online a simple google away. There are only two types of loops; counting loops and conditional loops. At the most basic level, we use counting loops when we (or the system) know how many times we want to go through something. Some examples of this are actual hard counts, lists, or lengths of files typically by line. While loops will continue until a condition exists or stops existing. The difference is until that condition occurs there's no reasonable way of knowing how many times the loop may have to occur.

`for` loops

Counting is iteration.

We can count numbers

```
for i in 1 2 3 4 5; do echo "the value now is $i"; done
```

We can count items

```
for dessert in pie cake icecream sugar soda; do echo "this is my favorite $dessert"; done
```

But, it's impractical to count for ourselves sometimes so we let the system do it for us.

```
seq 100 \
seq 4 100 \
seq 6 2 100 \
man seq
```

What did each of those do? Let's put them in a loop we can use

Maybe we want to count our 1000 servers and connect to them by name.

```
for i in `seq 1000`; do echo "Connecting to server p01awl$i"; done
```

Maybe we need to create a list of all our servers and put it in a list

```
for i in `seq 1000`; do echo "p01awl$i" >> serverfile; done
```

Maybe someone else gave us a list of servers and we need to read from that list to connect and do work.

```
for server in `cat serverfile`; do echo "connecting to server $server"; done
```

So, while those are even just a limited set of cases those are all times when, at the start, we know how many times we're going to go through the loop. Counting or For loops always have a set number of times they're going to run. That can change, but when it starts the end number of runs is known.

`while` loops

While loops are going to test conditions and loop while that condition evaluates to true. We can invert that, as we can with all logic, but I find that testing for truth is always easiest.

It is important to remember that `CRTL + C` will break you out of loops, as that will come handy here.

Administrators often find themselves looking at data and needing to refresh that data. One of the simplest loops is an infinite loop that always tests the condition of true (which always evaluates to true) and then goes around again. This is especially useful when watching systems for capacity issues during daemon or program startups.

```
while true; do date; free -m; uptime; sleep 2; done
```

This will run until you break it with `CTRL + C`. This will loop over the date, `free -m`, `uptime`, and `sleep 2` commands until the condition evaluates to false, which it will never do.

Let's run something where we actually have a counter and see what that output is

```
counter=0
while [[ $counter -lt 100 ]]; do echo "counter is at $counter"; (( counter++ )); done
```

What numbers were counted through?

If you immediately run this again, what happens? Why didn't anything happen?

```
#Reset counter to 0
counter=0
```

Re-run the above loop. Why did it work this time?

Reset the counter and run it again. Try moving the counter to before the output. Can you make it count from 1 to 100? Can you make it count from 3 to 251? Are there challenges to getting that to work properly?

What if we wanted something to happen for every MB of free RAM on our system? Could we do that?

```
memFree=`free -m | grep -i mem | awk '{print $2}'`
counter=0
while [[ $counter -lt $memFree ]]; do echo "counter is at $counter"; (( counter++ )); done
```

3.0 - Scripting System Checks

The main thing we haven't covered is what to actually do with these things we've done.
We can put them into a file and then execute them sequentially until the file is done executing. To do that we need to know the interpreter (bash is default) and then what we want to do.

```
touch scriptfile.sh
chmod 755 scriptfile.sh #let's just make it executable now and save trouble later
vi scriptfile.sh
```

add the following lines:

```
#!/bin/bash

echo "checking system requirements"

rpm -qa | grep -i gcc > /dev/null
gccCheck=$?

rpm -qa | grep -i superprogram > /dev/null
superCheck=$?

if [ $gccCheck -eq "0" -a $superCheck -eq "1" ]
  then echo "We can install someprogram"
else
  echo "We can't install someprogram"
fi
```

Execute this with the following command and you'll have your first completed script.

```
./scriptfile.sh
```

run the strace command to see what is happening with your system when it interprets this script.

```
strace ./scriptfile.sh
```

That's a lot of output! Now try adding -c for a summary. Here you can see all the syscalls made by the script and the time is took for each one.

```
strace -c ./scriptfile.sh
```

There are a lot of ways to use these tools. There are a lot of things you can do and include with scripts. This is just meant to teach you the basics and give you some confidence that you can go out there and figure out the rest. You can develop things that solve your own problems or automate your own tasks.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.9.4 Unit 8 Bonus - Scripting

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

Bash: The Essential Skill for Any Linux Administrator

If you're planning to work with Linux, you'll use **Bash every day** -- whether troubleshooting servers, automating tasks, or managing system configurations.

WHY IS BASH IMPORTANT?

- Bash is everywhere:
  - Bash is the default shell on all **major Linux distributions** (RedHat, Debian, etc), and most other distributions
  - It automates common sysadmin tasks (backups, log analysis, deployments)
  - Bash is essential for DevOps and administrative workflows (writing scripts, configuring CI/CD pipelines).

WHY LEARN BASH?

- You can automate repetitive or complex tasks.
- You can manage anything on your system using Bash (files, processes, services, etc.).
- Bash works across almost all major Linux distributions.

Bash scripting turns **manual commands into powerful, reusable automation**.

Writing Your First Script

Let's create a simple script that prints a message.

- **Create a script file:**

```
touch first-script.sh
```

- **Make it executable:**

```
chmod +x first-script.sh
# Or, use octal
chmod 755 first-script.sh
```

- **Open it in a text editor (e.g., `vi`):**

```
vi first-script.sh
```

- **Add the following code:**

```
#!/bin/bash
echo "Hello, admin!"
```

- **Run the script:**

```
./first-script.sh
```

- **Expected output:**

```
Hello, admin!
```

- **Key Takeaways:**
  - The `#!/bin/bash` **shebang line** tells the system which interpreter to use to execute the script.
  - `chmod +x` or `chmod 755` **makes the script executable**.
  - `./` is required because the script is **not in the system's** `PATH`.

It's worth noting that including a `.sh` file extension is **completely optional**.
If you decide to replace the Bash script with an executable binary down the line, having a `.sh` file extension can hurt you (e.g., if any other programs try to execute it using the `.sh` filename).

10 Common Control Operators

These operators allow you to **chain and control command execution** in Bash.

Operator	Purpose	Example
`;`	Run multiple commands sequentially	`mkdir test; cd test`
`&&`	Run second command **only if first succeeds**	`mkdir test && cd test`
`||`	Run second command **only if first fails**	`mkdir test || echo "Failed"`
`&`	Run a command **in the background**	`sleep 60 &`
`|`	Pipe output from one command to another	`ls | grep ".txt"`
`()`	Run commands **in a subshell**	`(cd /tmp && ls)`
`{}`	Run commands **in the current shell**	`{ cd /tmp; ls; }`
`>`	Redirect output to a file (**overwrite**)	`echo "log" > file.txt`
`>>`	Redirect output (**append**)	`echo "log" >> file.txt`
`$(...)`	Capture command output (from a subshell)	`DATE=$(date)`

- **Why does this matter?**
  - These operators **control execution flow** and are fundamental to Bash scripting.

10 Common Conditionals

Bash conditionals allow scripts to **make decisions**.

Test	Meaning	Example
`[[ -f FILE ]]`	File exists	`[[ -f /etc/passwd ]]`
`[[ -d DIR ]]`	Directory exists	`[[ -d /home/user ]]`
`[[ -n STR ]]`	String is **non-empty**	`[[ -n "$USER" ]]`
`[[ -z STR ]]`	String is **empty**	`[[ -z "$VAR" ]]`
`[[ "$A" = "$B" ]]`	Strings are **equal**	`[[ "$USER" = "root" ]]`
`[[ "$A" != "$B" ]]`	Strings are **not equal**	`[[ "$USER" != "admin" ]]`
`[[ NUM1 -eq NUM2 ]]`	Numbers are **equal**	`[[ 5 -eq 5 ]]`
`[[ NUM1 -gt NUM2 ]]`	NUM1 is **greater than** NUM2	`[[ 10 -gt 5 ]]`
`[[ "$?" -eq 0 ]]`	Last command **was successful**	`command && echo "Success"`
`[[ -x FILE ]]`	File is **executable**	`[[ -x script.sh ]]`

- **Why does this matter?**
  - These tests are used in `if`-statements and loops.
- **Single brackets vs. double brackets?**
  - Single brackets `[ ... ]` are used in POSIX shell scripts. Using single brackets is equivalent to using the `test` command (see `help test` and `help [`)
    - These are generally more subject to shell injection attacks, since they allow anything to expand within them.
    - You **must** quote all variables and subshell command (i.e., `"$(...)"`) within single brackets.
  - Double brackets `[[ ... ]]` are **keywords** (also builtins) in bash (see `help [[`).
    - These do not require you to quote variables, and provide some extended functionality.
    - If you're writing a **Bash** script, there's no reason to use single brackets for conditionals.

10 Bash Scripting Scenarios

Below are 10 **real-world examples** of using bash from the command line.

Scenario	Solution	Cont'd.
Check if a file exists before deleting	`if [ -f "data.txt" ]; then rm data.txt; fi`	
Backup a file before modifying	`cp config.conf config.bak`	
Create a log entry every hour	`echo "$(date): Check OK" >> log.txt`	
Monitor disk space	`df -h`	`awk '$5 > 90 {print "Low disk: "$1}'`
Check if a service is running	`systemctl is-active nginx`	`systemctl restart nginx`
List large files in a directory	`find /var/log -size +100M -exec ls -lh {} \;`	
Change all `.txt` files to `.bak`	`for file in *.txt; do mv "$file" "${file%.txt}.bak"; done`	
Check if a user is logged in	`who`	`grep "admin"`
Kill a process by name	`pkill -f "python server.py"`	
Find and replace text in files	`sed -i 's/old/new/g' file.txt`	

- **Why does this matter?**
  - These scenarios show **how Bash automates real-world tasks**.

Debugging Bash Scripts

Debugging tools help troubleshoot Bash scripts.

Command	Purpose
`set -x`	Print each command **before execution**
`set -e`	Exit script **if any command fails**
`trap '...' ERR`	Run a custom **error handler** when a command fails
`echo "$VAR"`	Print variable values **for debugging**
`printf "%s\n" "$VAR"`	Print variable values **for debugging**
`bash -x script.sh`	Run script **with debugging enabled**

Using `set -x` and `echo` (or `printf`) are some of the most common methods of troubleshooting.

EXAMPLE DEBUGGING SCRIPT

```
#!/bin/bash
set -xe  # Enable debugging and exit on failure
mkdir /tmp/mydir
cd /tmp/mydir
rm -rf /tmp/mydir
```

Next Steps

Now that you understand the fundamentals, here's what to do next:

- Practice writing scripts:
  - Automate a daily task (e.g., installing a program, creating backups, user management)
- Master error handling:
  - Learn signals and `trap`, and learn about logging techniques.
- Explore advanced topics:
  - Look into writing functions, using arrays, and job control.
- Read `man bash`:
  - The ultimate built-in reference.
  - This resource has everything you need to know about Bash and then some!
- Join ProLUG community:
  - Learn from others, contribute, and improve your Linux skillset.

🚀 **Happy scripting!**

2.10 Unit 9 - Containerization on Linux

2.10.1 Unit 9 - Containerization on Linux

Overview

In this unit, we dive into the modern world of containerization, focusing on **Podman**—an open-source, daemon-less container engine. As Linux administrators, understanding containerization is crucial for supporting developers, managing production systems, and deploying services efficiently.

We'll explore what containers are, how to manage them, and how to build container images.

Relevance & Context

Containerization is a critical part of modern IT, powering development pipelines **(CI/CD)**, cloud deployments, and microservices. As Linux system administrators, we are expected to support and troubleshoot containers, manage container infrastructure, and ensure smooth operations across development and production environments.

This unit focuses on **Podman**, a secure, rootless, and daemon-less alternative to Docker, widely used in enterprise environments like Red Hat and Rocky Linux. Whether you work in a NOC, DevOps, or traditional SysAdmin role, understanding containerization is essential to being an effective part of any IT team.

Learning Objectives

By the end of this unit, you will be able to:

- Explain what containers are and how they fit into modern Linux system administration
- Run and manage Podman containers, including starting, stopping, and inspecting containers
- Build custom container images using Dockerfiles and Podman
- Analyze container processes, logs, and network interactions for troubleshooting

Key Terms and Definitions

Containers	**Virtual**
Machines	**Podman**
Images	**Dockerfiles**
Virtualization	**Daemon-less**

2.10.2 Unit 9 Worksheet - Containerization on Linux

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Dockerfile Reference Page
- Podman Command List

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u9_worksheet(`.txt`)
- 📥 u9_worksheet(`.docx`)
- 📥 u9_worksheet(`.pdf`)

UNIT 9 RECORDING

Link: https://www.youtube.com/watch?v=_lo50mgERJY

Discussion Post #1

It's a slow day in the NOC and you have heard that a new system of container deployments are being used by your developers. Do some reading about containers, docker, and podman.

1. What resources helped you answer this?
2. What did you learn about that you didn't know before?
3. What seems to be the major benefit of containers?
4. What seems to be some obstacles to container deployment?

Discussion Post #2

Scenario

You get your first ticket about a problem with containers. One of the engineers is trying to move his container up to the Dev environment shared server. He sends you over this information about the command he's trying to run.

```
[developer1@devserver read]$ podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
[developer1@devserver read]$ podman images
REPOSITORY                TAG                IMAGE ID      CREATED      SIZE
localhost/read_docker     latest             2c0728a1f483  5 days ago   68.2 MB
docker.io/library/python  3.13.0-alpine3.19  9edd75ff93ac  3 weeks ago  47.5 MB
[developer1@devserver read]$ podman run -dt -p 8080:80/tcp docker.io/library/httpd
```

You decide to check out the server

```
[developer1@devserver read] ss -ntulp
Netid    State    Recv-Q   Send-Q      Local Address:Port        Peer Address:Port         Process
udp      UNCONN   0        0           127.0.0.53%lo:53               0.0.0.0:*            users:(("systemd-resolve",pid=166693,fd=13))
tcp      LISTEN   0        80              127.0.0.1:3306             0.0.0.0:*            users:(("mariadbd",pid=234918,fd=20))
tcp      LISTEN   0        128               0.0.0.0:22               0.0.0.0:*            users:(("sshd",pid=166657,fd=3))
tcp      LISTEN   0        4096        127.0.0.53%lo:53               0.0.0.0:*            users:(("systemd-resolve",pid=166693,fd=14))
tcp      LISTEN   0        4096                *:8080                   *:*              users:(("node_exporter",pid=662,fd=3))
```

1. What do you think the problem might be?
    - How will you test this?
2. The developer tells you that he's pulling a local image. Do you find this to be true, or is something else happening in their `run` command?

Info

Submit your input by following the link below. The discussion posts are done in Discord forums. Link to Discussion Posts

Definitions

Container:

Docker:

Podman:

CI/CD:

Dev/Prod Environments (Development/Production Environments):

Dockerfile:

Docker/Podman images:

Repository:

Digging Deeper

1. Find a blog on deployment of some service or application in a container that interests you.
   See if you can get the deployment working in the lab.
    - What worked well?
    - What did you have to troubleshoot?
    - What documentation can you make to be able to do this faster next time?
2. Run this scenario and play with K3s: https://killercoda.com/k3s/scenario/intro

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT? If you're not in IT, how can you look to put something like this into your resume or portfolio?

2.10.3 Unit 9 Lab - Containerization on Linux

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- https://podman.io/docs
- https://docs.docker.com/build/concepts/dockerfile/
- https://docs.podman.io/en/latest/markdown/podman-exec.1.html

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u9_lab(`.pdf`)
- 📥 u9_lab(`.docx`)

Pre-Lab Warmup

1. `which podman`
2. `dnf whatprovides podman`
3. `rpm -qi podman`
   When was this installed?
   What version is it?
   Why might this be important to know?
4. `podman images`
5. `podman ps`
   What do you learn from those two commands?
   Why might it be important to know on a system?

Lab 🧪

BUILDING AND RUNNING CONTAINERS

Your tasks in this lab are designed to get you thinking about how container deployments interact with our Linux systems that we support.

1. Pull and run a container

```
podman run -dt -p 8080:80/tcp docker.io/library/httpd
```

What do you see on your screen as this happens?

2. Check your images again (from your earlier exercises)

```
podman images
```

Is there a new image, and if so, what do you notice about it?

3. Check your podman running containers

```
podman ps
```

What appears to be happening? Can you validate this with your Linux knowledge?

```
ss -ntulp
curl 127.0.0.1:8080
```

4. Inspect the running pod

```
podman inspect -l
```

What format is the output in?
What important information might you want from this in the future?

```
podman logs -l
```

What info do you see in the logs?
Do you see your connection attempt from earlier? What is the return code and why is that important for troubleshooting?

```
podman top -l
```

What processes is the pod running?
What other useful information might you find here?
Why might it be good to know the user being run within the pod?

5. Stop the pod by its name

```
podman stop <podname>
```

Can you verify it is stopped from your previous commands?

```
podman ps
ss -ntulp
curl 127.0.0.1:8080
```

Does the container still exist? Why might you want to know this?

```
podman image
```

BUILD AN APPLICATION IN A CONTAINER

The ProLUG lab will already have a version of this setup for you to copy and run. If you are in a different environment, follow https://docs.docker.com/build/concepts/dockerfile/ for the general same steps.

1. Setup your lab environment

```
cd /lab_work/
ls
mkdir scott_lab9
cd scott_lab9/
ls
cp /labs/lab9.tar.gz .
tar -xzvf lab9.tar.gz
# lab9/
# lab9/Dockerfile
# lab9/hello.py
ls
# lab9 lab9.tar.gz
cd lab9
pwd
# /lab_work/scott_lab9/lab9
ls
# Dockerfile hello.py
```

2. Create a docker image from the docker file:

```
time podman build -t scott_hello .
#Use your name
```

What output to your screen do you see as you build this?
Approximately how long did it take?
If this breaks in the lab, how might you fix it? What do you suspect?

3. Verify that you have built the container

```
podman images
```

4. Run the container as a daemon

```
podman run -dt localhost/scott_example
```

5. Verify the name and that it is running

```
podman ps
```

6. Exec into the pod and see that you are on the Ubuntu container

```
podman exec -it festive_pascal sh
cat /etc/*release
exit
```

CONCLUSION

There are a lot of ways to use these tools. There are a lot of ways you will support them. At the end of the day you're a Linux System Administrator, you're expected to understand everything that goes on in your system. To this end, we want to know the build process and run processes so we can help the engineers we support keep working in a Linux environment.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.11 Unit 10 - Kubernetes

2.11.1 Unit 10 - Kubernetes

Overview

This unit introduces **Kubernetes (K8s)**, an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications. The unit covers:

- **Understanding Kubernetes Architecture** - Nodes, Control Plane, and Cluster Components.
- **Installing K3s** - A lightweight Kubernetes distribution optimized for resource efficiency.
- **Interacting with Kubernetes** - Using `kubectl` to manage and troubleshoot clusters.
- **Deploying Applications** - Creating and managing **Pods, Deployments, and Services**.
- **Security and Best Practices** - Implementing security measures and troubleshooting issues.

Kubernetes plays a critical role in **modern enterprise infrastructure**, enabling **scalability, high availability, and automation** in cloud-native applications.

Learning Objectives

By the end of this unit, learners will:

1. **Understand the Core Concepts of Kubernetes:**
   - Define Kubernetes and explain its role in container orchestration.
   - Differentiate between **Kubernetes vs. PaaS (Platform as a Service)**.
2. **Deploy and Manage Kubernetes Clusters:**
   - Install **K3s** and verify its functionality.
   - Manage cluster resources using `kubectl`.
3. **Perform Basic Kubernetes Operations:**
   - Create and manage **Pods, Deployments, and Services**.
   - Understand the role of **Namespaces, ConfigMaps, and Secrets**.
4. **Troubleshoot Kubernetes Clusters:**
   - Identify common cluster issues and validate node status.
   - Diagnose networking and pod scheduling problems.
5. **Apply Security Best Practices in Kubernetes:**
   - Secure containerized applications using best practices.
   - Implement **Kubernetes Pod Security Standards**.

Key Terms and Definitions

Kubernetes (K8s)	**K3s**
Control Plane	**Nodes**
Pods	**Deployments**
Services	**Kubelet**
Scheduler	**ETCD**
Kube-proxy	**Static Pod**

2.11.2 Unit 10 Worksheet - Kubernetes

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Kubernetes Overview
- K3s Official Documentation
- Kubernetes Security Best Practices
- Pod Security Standards
- Interactive Kubernetes Labs

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u10_worksheet(`.txt`)
- 📥 u10_worksheet(`.docx`)
- 📥 u10_worksheet(`.pdf`)

UNIT 10 RECORDING

Link: https://www.youtube.com/watch?v=KycsHfZoAQs

Discussion Post #1

Read: Kubernetes Overview

1. What are the two most compelling reasons to implement Kubernetes in your organization?
2. The article states that Kubernetes is not a PaaS. What does that mean? How does Kubernetes compare to a traditional PaaS?

Discussion Post #2

Scenario

Your team is troubleshooting a Kubernetes cluster where applications are failing to deploy. They send you the following output:

```
[root@Test_Cluster1 ~]# kubectl version
Client Version: v1.31.6+k3s3
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.30.6+k3s1

[root@rocky15 ~]# kubectl get nodes
NAME            STATUS      ROLES                  AGE   VERSION
Test_Cluster1   Ready       control-plane,master   17h   v1.30.6+k3s1
Test_Cluster2   NotReady    worker                 33m   v1.29.6+k3s1
Test_Cluster3   Ready       worker                 17h   v1.28.6+k3s1
```

1. How would you validate the error?
2. What do you suspect is causing the problem?
3. Has someone already attempted to fix this problem? Why or why not?

Discussion Post #3

Scenario

You are the Network Operations Center (NOC) lead, and your team is responsible for monitoring development, test, and QA Kubernetes clusters.

Write a basic cluster health check procedure for new NOC personnel.

1. What online resources did you use to figure this out?
2. What did you learn during this process?

Info

Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Key Terminology & Definitions

Define the following Kubernetes terms:

- Kubernetes/K8s:
- K3s:
- Control Plane:
- Node:
- Pod:
- Deployment:
- Service:
- ETCD:
- Kubelet:
- Kube-proxy:
- Scheduler:
- API Server:

Lab and Assignment

Unit 10 Lab k3s

Continue working on your project from the Project Guide

Topics:

1. System Stability
2. System Performance
3. System Security
4. System monitoring
5. Kubernetes
6. Programming/Automation You will research, design, deploy, and document a system that improves your administration of Linux systems in some way.

Digging Deeper

1. Build a custom container and deploy it in Kubernetes securely.
2. Read about container security:
    - Docker Security Best Practices
    - Pod Security Standards
3. Complete this Kubernetes security lab:
    - KillerShell Kubernetes Security

Reflection Questions

1. What questions do you still have about Kubernetes?
2. How can you apply this knowledge in your current IT role?
3. If you're not in IT, how could this experience contribute to your resume or portfolio?

2.11.3 Unit 10 Lab - Kubernetes

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs
- Kubernetes Documentation
- K3s Official Site
- Pod Security Standards
- Kubernetes Troubleshooting Guide

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u10_lab(`.pdf`)
- 📥 u10_lab(`.docx`)

Pre-Lab: Quick Warmup and System Checks

Before installing K3s, verify system compatibility and gather initial data.

STEP 1: DOWNLOAD AND INSPECT K3S INSTALLER

```
curl -sfL https://get.k3s.io > /tmp/k3_installer.sh
more /tmp/k3_installer.sh
```

Questions

- What system checks does the installer perform?
- What environment variables does it check?

STEP 2: SYSTEM ARCHITECTURE CHECK

```
uname -m
grep -i arch /tmp/k3_installer.sh
```

Questions

- What is the variable holding the system architecture?
- How does K3s determine system compatibility?

STEP 3: SELINUX STATUS CHECK

```
grep -iE "selinux|sestatus" /tmp/k3_installer.sh
sestatus
```

Questions:

- Does K3s check if SELinux is enabled?
- What are the implications of SELinux on Kubernetes deployments?

Installing K3s and Verifying the Service

STEP 4: INSTALL K3S

```
curl -sfL https://get.k3s.io | sh -
```

STEP 5: VERIFY INSTALLATION

```
systemctl status k3s
systemctl is-enabled k3s
```

- What files and services were installed?
- Is K3s set to start on boot?

STEP 6: EXPLORE SYSTEM SERVICES

```
systemctl cat k3s
```

- What startup configurations does K3s have?
- Does it rely on any dependencies?

Exploring Kubernetes Environment

STEP 7: CHECKING KUBERNETES COMPONENTS

```
kubectl version
kubectl get nodes
kubectl get pods -A
kubectl get namespaces
kubectl get configmaps -A
kubectl get secrets -A
```

Questions

- What namespaces exist by default?
- What secrets are stored in the cluster?

Deploying Applications: Pods, Services, and Deployments

STEP 8: CREATE A SIMPLE WEB SERVER POD

```
kubectl run webpage --image=nginx
```

- Verify pod creation:

```
kubectl get pods
kubectl describe pod webpage
```

STEP 9: DEPLOY A REDIS DATABASE WITH LABELS

```
kubectl run database --image=redis --labels=tier=database
```

- Verify labels:

```
kubectl get pods --show-labels
```

STEP 10: EXPOSE THE REDIS DATABASE

```
kubectl expose pod database --port=6379 --name=redis-service --type=ClusterIP
```

- Verify service:

```
kubectl get services
```

STEP 11: CREATE A WEB DEPLOYMENT WITH REPLICAS

```
kubectl create deployment web-deployment --image=nginx --replicas=3
```

- Check status:

```
kubectl get deployments
```

STEP 12: CREATE A NEW NAMESPACE AND DEPLOY AN APP

```
kubectl create namespace my-test
kubectl create deployment redis-deploy -n my-test --image=redis --replicas=2
```

- Verify deployment:

```
kubectl get pods -n my-test
```

Troubleshooting Cluster Issues

Your team reports an issue with the cluster:

```
kubectl get nodes
```

Output:

```
NAME            STATUS      ROLES                AGE     VERSION
Test_Cluster1   Ready       control-plane,master 17h     v1.30.6+k3s1
Test_Cluster2   NotReady    worker               33m     v1.29.6+k3s1
Test_Cluster3   Ready       worker               17h     v1.28.6+k3s1
```

STEP 13: INVESTIGATE NODE HEALTH

```
kubectl describe node Test_Cluster2
kubectl get pods -A
```

- What errors do you notice?
- Is there a resource constraint or version mismatch?

STEP 14: RESTART K3S AND CHECK LOGS

```
systemctl restart k3s
journalctl -xeu k3s
```

- What errors appear in the logs?
- Does restarting resolve the issue?

Conclusion

At the end of this lab, you should:

- Have a fully operational K3s Kubernetes cluster.
- Be able to deploy and expose containerized applications.
- Know how to troubleshoot common Kubernetes errors.
- Understand security best practices for Kubernetes deployments.

Next Steps: Continue testing deployments, set up monitoring tools like Prometheus or Grafana, and explore Ingress Controllers to manage external access.

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

2.11.4 Unit 10 Bonus - Kubernetes

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

This section provides **advanced troubleshooting techniques**, **security best practices**, and **real-world challenges** to strengthen your Kubernetes knowledge.

Step 1: Troubleshooting Kubernetes Cluster Issues

When things go wrong, **systematic troubleshooting** is key. Here's how you diagnose **common Kubernetes issues**.

NODE NOT READY

Check node status

```
kubectl get nodes
kubectl describe node <node-name>
```

Investigate Kubelet logs

```
journalctl -u k3s -n 50 --no-pager
```

Verify system resources

```
free -m     # Check available memory
df -h       # Check disk space
htop        # Monitor CPU usage
```

Possible Fixes

- Restart K3s on the failing node:

```
systemctl restart k3s
```

- Ensure network connectivity:

```
ping <control-plane-ip>
```

PODS STUCK IN "PENDING" OR "CRASHLOOPBACKOFF"

Check pod status

```
kubectl get pods -A
kubectl describe pod <pod-name>
kubectl logs <pod-name>
```

Possible Fixes

- If **insufficient resources**, scale up the cluster.
- If **missing images**, check container registry authentication.
- If **misconfigured storage**, inspect volumes:

```
kubectl get pvc
```

Step 2: Securing Kubernetes Deployments

Security is crucial in enterprise environments. Here are **quick wins** for a more **secure Kubernetes cluster**.

LIMIT POD PRIVILEGES

Disable privileged containers

```
securityContext:
  privileged: false
```

Enforce read-only file system

```
securityContext:
  readOnlyRootFilesystem: true
```

RESTRICT NETWORK ACCESS

Use Network Policies to restrict pod communication

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector: {}
  policyTypes:
    - Ingress
```

Use NetworkPolicy Editor to create and edit your network policies.

USE POD SECURITY ADMISSION (PSA)

Enable PSA to enforce security levels:

```
kubectl label --overwrite ns my-namespace pod-security.kubernetes.io/enforce=restricted
```

Step 3: Performance Optimization Tips

Enhance Kubernetes efficiency with these quick optimizations:

OPTIMIZE RESOURCE REQUESTS & LIMITS

Set appropriate **CPU & Memory limits** in deployments:

```
resources:
  requests:
    cpu: "250m"
    memory: "256Mi"
  limits:
    cpu: "500m"
    memory: "512Mi"
```

Why? Prevents a single pod from consuming excessive resources.

ENABLE HORIZONTAL POD AUTOSCALING (HPA)

Auto-scale pods **based on CPU or memory usage**:

```
kubectl autoscale deployment my-app --cpu-percent=50 --min=2 --max=10
```

Step 4: Bonus Challenge - Build a Secure, Scalable App

Challenge:

- **Create a secure containerized app**
- **Deploy it in Kubernetes**
- **Implement Network Policies**
- **Apply Pod Security Standards**

Helpful Resources:

- Pod Security Standards
- Kubernetes Hardening Guide (Archived)
- Kubernetes Security Best Practices

Conclusion

This **bonus section** strengthens **your Kubernetes troubleshooting, security, and performance tuning skills**. Apply these principles in real-world deployments!

2.12 Unit 11 - Monitoring

2.12.1 Unit 11 - Monitoring

Overview

In this unit, we focus on Linux system monitoring, using modern tools like **Grafana, Prometheus, Node Exporter, and Loki**. As Linux administrators, monitoring is essential to ensure system stability, performance, and security across environments.

We will explore how to collect, analyze, and visualize system metrics, and discuss best practices for monitoring and dashboard design that can improve troubleshooting and proactive system management.

Learning Objectives

By the end of this unit, you will be able to:

- Explain core monitoring concepts like metrics, logs, SLOs, SLIs, and KPIs
- Set up Prometheus and Node Exporter to collect system metrics
- Use Grafana to create dashboards for visualizing system health and performance
- Write and execute PromQL queries to analyze system data
- Interpret monitoring data to diagnose system issues and support teams with actionable insights

Key Terms and Definitions

SLO (Service Level Objective)	**SLA (Service Level Agreement)**
SLI (Service Level Indicator)	**KPI (Key Performance Indicator)**
MTTD (Mean Time to Detect)	**MTTR (Mean Time to Repair)**

2.12.2 Unit 11 Worksheet - Monitoring

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- How to easily monitor your Linux server | Grafana Labs
- 30 Linux System Monitoring Tools Every SysAdmin Should Know
- Monitoring Linux Using SNMP - Nagios

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u11_worksheet(`.txt`)
- 📥 u11_worksheet(`.docx`)
- 📥 u11_worksheet(`.pdf`)

UNIT 11 RECORDING

Link: https://www.youtube.com/watch?v=6VOHFYkptOw

Discussion Post #1

> **Scenario**
>
> You've heard the term "loose coupling" thrown around the office about a new monitoring solution coming down the pike. You find a good resource and read the section on "Prefer Loose Coupling" https://sre.google/workbook/monitoring/.

1. What does "loose coupling" mean, if you had to summarize to your junior team members?
2. What is the advantage given for why you might want to implement this type of tooling in your monitoring? Do you agree? Why or why not?
3. They mention "exposing metrics" what does it mean to expose metrics? What happens to metrics that are exposed but never collected?

Discussion Post #2

> **Scenario**
>
> Your HPC team is asking for more information about how CPU0 is behaving on a set of servers. Your team has node exporter writing data out to Prometheus (Use this to simulate https://promlabs.com/promql-cheat-sheet/).

1. Can you see the usage of CPU0 and what is the query?
2. Can you see the usage of CPU0 for just the last 5 minutes and what is the query?
3. You know that CPU0 is excluded from Slurm, can you exclude that and only pull the user and system for the remaining CPUs and what is that query?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

SLO

SLA

SLIKPI

Span

Trace

Prometheus

Node_Exporter

Grafana

Dashboard

Heads up Display

Digging Deeper

1. Read the rest of the chapter https://sre.google/workbook/monitoring/ and note anything else of interest when it comes to monitoring and dashboarding.
2. Look up the “ProLUG Prometheus Certified Associate Prep 2024” in Resources -> Presentations in our ProLUG Discord. Study that for a deep dive into Prometheus.
3. Complete the project section of “Monitoring Deep Dive Project Guide” from the prolug-projects section of the Discord. We have a Youtube video on that project as well. https://www.youtube.com/watch?v=54VgGHr99Qg

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT? If you’re not in IT, how can you look to put something like this into your resume or portfolio?

2.12.3 Unit 11 Lab - Monitoring

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Killercoda Labs

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

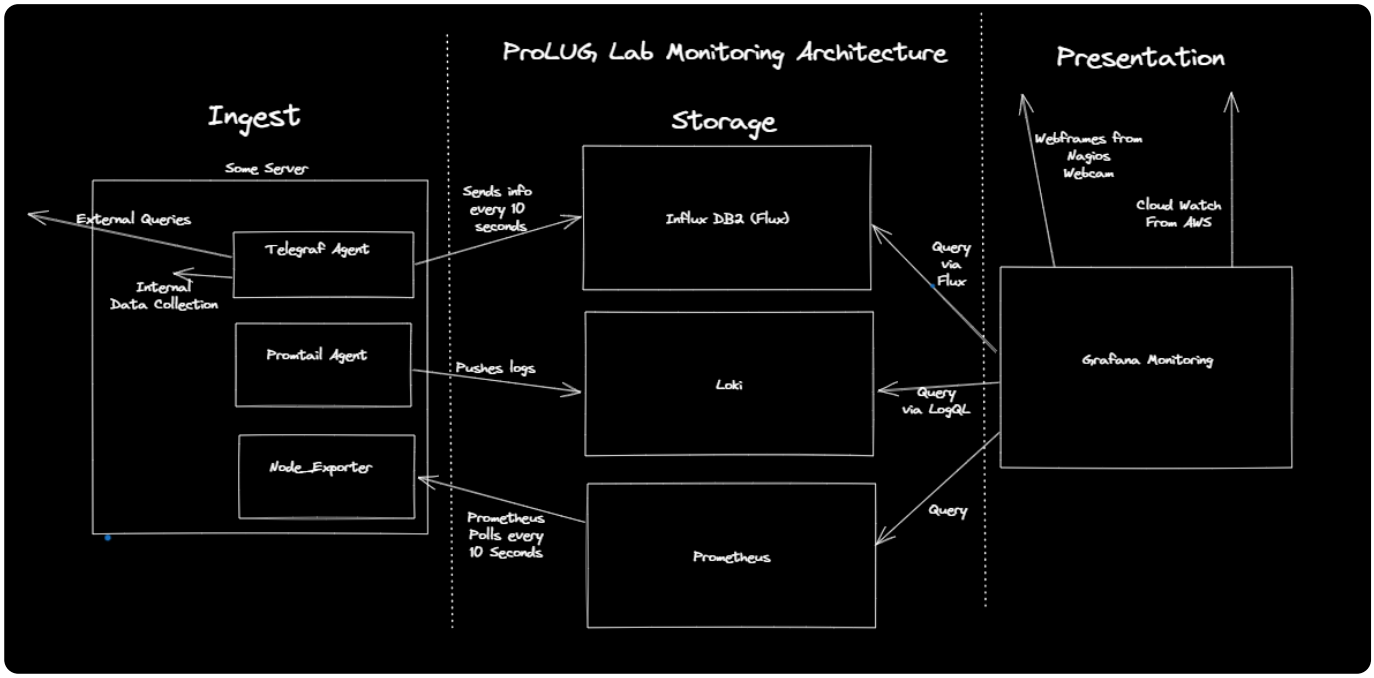
Downloads

The lab has been provided for convenience below:

- 📥 u11_lab(`.pdf`)
- 📥 u11_lab(`.docx`)

SETUP MONITORING WITH GRAFANA

1. Run through each of the three labs below in Killercoda:
   - https://killercoda.com/het-tanis/course/Linux-Labs/102-monitoring-linux-logs
   - https://killercoda.com/het-tanis/course/Linux-Labs/103-monitoring-linux-telemetry
   - https://killercoda.com/het-tanis/course/Linux-Labs/104-monitoring-linux-Influx-Grafana
2. While completing each lab think about the following:
   a. How does it tie into the diagram below?
   b. What could you improve, or what would you change based on your previous administration experience.



CONCLUSION

In the end monitoring is more an art than engineering. Sure, we can design all the systems to track all the things, but there's no equation on what is the one right answer for any of this. You have to spend time with the systems, know what is important and what is an indicator of problems. Then, you have to consider your audience and how to best show them what they need to see.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.13 Unit 12 - Baselines && Benchmarks

2.13.1 Unit 12 - Baselines & Benchmarks

Overview

In this unit, we focus on **baselining, benchmarking, testing methodology, and data analytics** — all essential skills for Linux system administrators. These topics allow us to understand the current state of our systems, measure performance under varying loads, and validate improvements with real data.

We'll explore how to gather accurate system information using tools like **iostat, sar, stress, and iperf3**, and learn how to develop test plans that can support decision-making and capacity planning. Whether you're justifying budget increases or validating a new storage solution, knowing how to gather and present performance data makes you a more effective administrator.

Learning Objectives

By the end of this unit, you will be able to:

- Define and use key concepts: **baseline, benchmark, high watermark, scope, and methodology**
- Use tools like **sar, iostat, stress**, and **iperf3** to collect performance data
- Create and execute test plans to evaluate system behavior under different loads
- Apply data analytics concepts: **descriptive, diagnostic, predictive, and prescriptive**
- Communicate system performance clearly with stakeholders through objective evidence

Key Terms and Definitions

Baseline	**Benchmark**
High Watermark	**Scope**
Methodology	**Testing**
Control	**Experiment**
Analytics	**Analytics - Descriptive**
Analytics - Predictive	

2.13.2 Unit 12 Worksheet - Baselines & Benchmarks

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Kaggle - Python and Data Science Learning

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u12_worksheet(`.txt`)
- 📥 u12_worksheet(`.docx`)
- 📥 u12_worksheet(`.pdf`)

UNIT 12 RECORDING

Link: https://www.youtube.com/watch?v=8psu0D4rSmc

Discussion Post #1

> **Scenario**
>
> Your manager has come to you with another emergency.
>
> He has a meeting next week to discuss capacity planning and usage of the system with IT upper management. He doesn't want to lose his budget, but he has to prove that the system utilization warrants spending more.

1. What information can you show your manager from your systems?
2. What type of data would prove system utilization? (Remember the big 4: compute, memory, disk, networking)
3. What would your report look like to your manager?

Discussion Post #2

Scenario

You are in a capacity planning meeting with a few of the architects. They have decided to add 2 more agents to your Linux systems, a Bacula Agent and an Avamar Agent. They expect these agents to run their work starting at 0400 every morning.

1. What do these agents do? (May have to look them up)
2. Do you think there is a good reason not to use these agents at this timeframe?
3. Is there anything else you might want to point out to these architects about these agents they are installing?

Discussion Post #3

Scenario

Your team has recently tested a proof of concept of a new storage system. The vendor has published the blazing fast speeds that are capable of being run through this storage system. You have a set of systems connected to both the old storage system and the new storage system.

1. Write up a test procedure of how you may test these two systems.
2. How are you assuring these test are objective?
    - What is meant by the term Ceteris Paribus, in this context?

Info

Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Baseline:

Benchmark:

High watermark:

Scope:

Methodology:

Testing:

Control:

Experiment:

Analytics:

Digging Deeper (optional)

1. Analyzing data may open up a new field of interest to you. Go through some of the free lessons on Kaggle, here: https://www.kaggle.com/learn
    - What did you learn?
    - How will you apply these lessons to data and monitoring you have already collected as a system administrator?
2. Find a blog or article that discusses the 4 types of data analytics.
    - What did you learn about past operations?
    - What did you learn about predictive operations?

3. Download Spyder IDE (Open source).
    - Find a blog post or otherwise try to evaluate some data.
    - Perform some Linear regression. My block of code (but this requires some additional libraries to be added. I can help with that if you need it.)

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
size = [[5.0], [5.5], [5.9], [6.3], [6.9], [7.5]]
price =[[165], [200], [223], [250], [278], [315]]
plt.title('Pizza Price plotted against the size')
plt.xlabel('Pizza Size in inches')
plt.ylabel('Pizza Price in cents')
plt.plot(size, price, 'k.')
plt.axis([5.0, 9.0, 99, 355])
plt.grid(True)
model = LinearRegression()
model.fit(X = size, y = price)
#plot the regression line
plt.plot(size, model.predict(size), color='r')
```

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT?
   If you're not in IT, how can you look to put something like this into your resume or portfolio?

2.13.3 Unit 12 Lab - Baselines & Benchmarks

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- SAR Documentation
- iostat Manual
- stress GitHub
- iperf3 Documentation

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u12_lab(`.pdf`)
- 📥 u12_lab(`.docx`)

Pre-Lab Warm-Up

1. Create a working directory

```
mkdir lab_baseline
cd lab_baseline
```

2. Verify if `iostat` is available

```
which iostat
```

If it's not there:

```
# Find which package provides iostat
dnf whatprovides iostat

# This should tell you it's sysstat
rpm -qa | grep -i sysstat

# Install sysstat if needed
dnf install sysstat

# Verify installation
rpm -qa | grep -i sysstat
```

3. Verify if `stress` is available

```
which stress
```

If it's not there:

```
# Find which package provides stress
dnf whatprovides stress

# Install stress
dnf install stress

# Verify installation
rpm -qa | grep -i stress
rpm -qi stress  # Read the package description
```

4. Verify if `iperf3` is available

```
which iperf3
```

If it's not there:

```
# Find which package provides iperf3
dnf whatprovides iperf

# Install iperf3
dnf install iperf

# Verify installation
rpm -qa | grep -i iperf
rpm -qi iperf
```

Lab 🧪

BASELINE INFORMATION GATHERING

The purpose of a baseline is not to find fault, load, or to take corrective action. A baseline simply determines what is. You must know what is so that you can test against that when you make a change to be able to objectively say there was or wasn't an improvement. You must know where you are at to be able to properly plan where you are going. A poor baseline assessment, because of inflated numbers or inaccurate testing, does a disservice to the rest of your project. You must accurately draw the first line and understand your system's performance.

Using SAR (CPU and memory statistics)

Some useful sar tracking commands. 10 minute increments.

```
# By itself, this gives the last day's processing numbers
sar

# Gives memory statistics
sar -r

# Gives swapping statistics (useful to check if system runs out of physical memory)
sar -W

# List SAR log files
ls /var/log/sa/

# View SAR data from a specific day of the month
sar -f /var/log/sa/sa28
```

For your later labs, you need to collect `sar` data in real time to compare with the baseline data.

```
# View how SAR collects data every 10 minutes
systemctl cat sysstat-collect.timer

# Collect SAR data in real time (every 2 seconds, 10 samples)
sar 2 10

# Memory statistics (every 2 seconds, 10 samples)
sar -r 2 10
```

Using IOSTAT (CPU and device statistics)

`iostat` will give you either processing or device statistics for your system.

```
# Gives all information (CPU and device)
iostat

# CPU statistics only
iostat -c

# Device statistics only
iostat -d

# 1-second CPU stats until interrupted
iostat -c 1

# 1-second CPU stats, 5 times
iostat -c 1 5
```

Using iperf3 (network speed testing)

In the ProLUG lab, `red1` is the iperf3 server, so we can bounce connections off it (`192.168.200.101`).

```
# TCP connection with 128 connections
time iperf3 -c 192.168.200.101 -n 1G -P 128

# UDP connection with 128 connections
time iperf3 -c 192.168.200.101 -u -n 1G -P 128
```

Using STRESS to generate load

`stress` will produce extra load on a system. It can run against proc, ram, and disk I/O.

```
# View stress usage information
stress

# Stress CPU with 1 process (will run indefinitely)
stress -c 1

# Stress multiple subsystems (this will do a lot of things)
stress --cpu 8 --io 4 --vm 2 --vm-bytes 128M -d 1 --timeout 10s
```

Read the usage output and try to figure out what each option does.

DEVELOPING A TEST PLAN

The company has decided we are going to add a new agent to all machines. Management has given this directive to you because of PCI compliance standards with no regard for what it may do to the system. You want to validate if there are any problems and be able to express your concerns as an engineer, if there are actual issues. No one cares what you think, they care what you can show, or prove.

Determine the right question to ask

- Do we have a system baseline to compare against?
    - No? Make a baseline.

```
iostat -xh 1 10
```

- Can we say that this system is not under heavy load?

- What does a system under no load look like performing tasks in our environment?
  - Assuming our systems are not running under load, capture SAR and baseline stats.
  - Perform some basic tasks and get their completion times.
    - Writing/deleting 3000 empty files #modify as needed for your system

```
# Speed: ~10s
time for i in `seq 1 3000`; do touch testfile$i; done

# Removing them
time for i in `seq 1 3000`; do rm -rf testfile$i; done

# Writing large files
for i in `seq 1 5`; do time dd if=/dev/zero of=/root/lab_baseline/sizetest$i bs=1024k count=1000; done

# Removing the files
for i in `seq 1 5`; do rm -rf sizetest$i ; done
```

    - Testing processor speed

```
time $(i=0; while (( i < 999999 )); do (( i ++ )); done)
# if this takes your system under 10 seconds, add a 9
```

    - Alternate processor speed test

```
time dd if=/dev/urandom bs=1024k count=20 | bzip2 -9 >> /dev/null
```

      This takes random numbers in blocks, zips them, and then throws them away.
      Tune to about ~10 seconds as needed.

- What is the difference between systems under load with and without the agent?

Run a load test (with `stress`) of what the agent is going to do against the system.

While the load test is running, do your same functions and see if they perform differently.

EXECUTE THE PLAN AND GATHER DATA

Edit these as you see fit, add columns or rows to increase understanding of system performance. This is your chance to test and record these things.

System Baseline Tests

Metric	Server 1
SAR average load (past week)	
IOSTAT test (10 min)	
IOSTAT test (2s x 10 samples)	
Disk write - small files	
Disk write - small files (retry)	
Disk write - large files	
Processor benchmark	

You may baseline more than once, more data is rarely bad.

Make 3 different assumptions for how load may look on your system with the agent and design your stress commands around them (examples):

1. I assume no load on hdd, light load on processors

```
while true; do stress --cpu 2 --io 4 --vm 2 --vm-bytes 128M --timeout 30; done #
```

2. I assume low load on hdd, light load on processors

```
while true; do stress --cpu 2-io 4 --vm 2 --vm-bytes 128M -d 1 --timeout 30; done
```

3. I just assume everything is high load and it's a mess

```
while true; do stress --cpu 4 --io 4 --vm 2 --vm-bytes 256M -d 4 --timeout 30; done
```

In one window start your load tests (YOU MUST REMEMBER TO STOP THESE AFTER YOU GATHER YOUR DATA). In another window gather your data again, exactly as you did for your baseline with `sar` and `iostat` just for the time of the test.

System Tests while under significant load

Put command you're using for load here:

Metric	Server 1
SAR average load (during test)	
IOSTAT test (10 min)	
IOSTAT test (2s x 10 samples)	
Disk write - small files	
Disk write - small files (retry)	
Disk write - large files	
Processor benchmark	

System Tests while under significant load

Put command you're using for load here:

Metric	Server 1
SAR average load (during test)	
IOSTAT test (10 min)	
IOSTAT test (2s x 10 samples)	
Disk write - small files	
Disk write - small files (retry)	
Disk write - large files	
Processor benchmark	

Continue copying and pasting tables as needed.

Reflection Questions (optional)

- How did the system perform under load compared to your baseline?
- What would you report to your management team regarding the new agent's impact?

- How would you adjust your test plan to capture additional performance metrics?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.14 Unit 13 - System Hardening

2.14.1 Unit 13 - System Hardening

Overview

In this unit, we focus on **system hardening** — the process of configuring Linux systems to meet defined security standards. As threats evolve, system administrators play a key role in ensuring confidentiality, integrity, and availability by reducing attack surfaces and enforcing secure configurations.

We will explore industry benchmarks like **STIGs and CIS**, implement hardening techniques for services like SSH, identify unneeded software, and analyze system security posture using tools like the **SCC Tool**. You'll also revisit baselining and documentation as part of security validation and compliance.

Learning Objectives

By the end of this unit, you will be able to:

- Define system hardening and understand its role in securing Linux servers
- Scan systems using the **SCC Tool** to assess security compliance
- Apply remediation steps based on **STIG** reports
- Harden services such as **SSHD**, remove unnecessary software, and lock down ports
- Rescan and verify improvements in your system's security posture
- Understand the importance of documentation and change management in security

Key Terms and Definitions

Hardening	**Pipeline**
Change Management	**Security Standard**
Security Posture	**Acceptable Risk**
NIST 800-53	**STIG**
CIS Benchmark	**OpenSCAP**
SCC Tool	**HIDS**
HIPS	

2.14.2 Unit 13 Worksheet - System Hardening

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Killercoda Lab - Server Startup Process
- Killercoda Lab - Updating a Golden Image

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u13_worksheet(`.txt`)
- 📥 u13_worksheet(`.docx`)
- 📥 u13_worksheet(`.pdf`)

UNIT 13 RECORDING

Link: https://www.youtube.com/watch?v=ESsUM0Gz8Jk

Discussion Post #1

> **Scenario**
>
> Your security team comes to you with a discrepancy between the production security baseline and something that is running on one of your servers in production. There are 5 servers in a web cluster and only one of them is showing this behavior. They want you to account for why something is different.

1. How are you going to validate that the difference between the systems?
2. What are you going to look at to explain this?
3. What could be done to prevent this problem in the future?

Discussion Post #2

> **Scenario**
>
> Your team has been giving you more and more engineering responsibilities.
> You are being asked to build out the next set of servers to integrate into the development environment. Your team is going from RHEL 8 to Rocky 9.4.

1. How might you start to plan out your migration?
2. What are you going to check on the existing systems to baseline your build?
3. What kind of validation plan might you use for your new Rocky 9.4 systems?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Hardening:

Pipeline:

Change management (IT):

Security Standard:

Security Posture:

Acceptable Risk:

NIST 800-53:

STIG:

CIS Benchmark:

OpenSCAP:

SCC Tool:

HIDS:

HIPS:

Digging Deeper (Optional)

1. Run through this lab: https://killercoda.com/het-tanis/course/Linux-Labs/107-server-startup-process
    - How does this help you better understand the discussion 13-2 question?
2. Run through this lab: https://killercoda.com/het-tanis/course/Linux-Labs/203-updating-golden-image
    - How does this help you better understand the process of hardening systems?

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT? If you're not in IT, how can you look to put something like this into your resume or portfolio?

2.14.3 Unit 13 Lab - System Hardening

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- https://killercoda.com/het-tanis/course/Linux-Labs/207-OS_STIG_Scan_with_SCC_Tool
- https://public.cyber.mil/stigs/srg-stig-tools/
- https://nvd.nist.gov/vuln/search

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u13_lab(`.pdf`)
- 📥 u13_lab(`.docx`)

Pre-Lab Warm-Up

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

1. `ss -ntulp`
   - What ports are open on this server?
   - What is open on port 9080?
   - What does this service do?
2. `systemctl --failed`
   - Are there any failed units?
3. `systemctl list-units --state=active`
   - About how many active units are there?
     - `systemctl list-units --state=active | wc -l`
4. `rpm -qa | wc -l`
   - Approximately how many software packages do you have?
5. `rpm -qa | grep -i ssh`
   - How many ssh packages do you have?
   - What is the version of openssh?
   - Do you know if there are any known vulnerabilities for that version?
     - https://nvd.nist.gov/vuln/search

Lab 🧪

There will be three basic tasks for today's labs:

1. You will scan a server for a SCC Report and get a STIG Score
2. You will remediate some of the items from the scan
3. You will rescan and verify a better score.

SCC REPORT

This lab portion can be done in the ProLUG Rocky servers, or in killercoda at this location: https://killercoda.com/het-tanis/course/Linux-Labs/207-OS_STIG_Scan_with_SCC_Tool

Testing hardening on the ProLUG Lab may take over an hour. You are welcome to perform the test there, but make sure you have some time.

`ssh` into a Rocky sever

```
cd /opt/scc
time ./cscc

# ---- Wait over an hour ------

cd /root/SCC/sessions #find the most recent run
```

Look in the results to see output.

HARDEN THE SYSTEM

1. Harden sshd

Red Hat Enterprise Linux 9 Security Technical Implementation Guide :: Version 2, Release: 1 Benchmark Date: 24 Jul 2024

Vul ID: V-258001 **Rule ID**: SV-258001r991589_rule **STIG ID**: RHEL-09-255125

Severity: CAT II **Classification**: Unclass

Group Title: SRG-OS-000480-GPOS-00227

Rule Title: RHEL 9 SSH public host key files must have mode 0644 or less permissive.

Discussion: If a public host key file is modified by an unauthorized user, the SSH service may be compromised.

Check Text: Verify the SSH public host key files have a mode of "0644" or less permissive with the following command:

Note: SSH public key files may be found in other directories on the system depending on the installation.

$ sudo stat -c "%a %n" /etc/ssh/*.pub

644 /etc/ssh/ssh_host_dsa_key.pub
644 /etc/ssh/ssh_host_ecdsa_key.pub
644 /etc/ssh/ssh_host_ed25519_key.pub
644 /etc/ssh/ssh_host_rsa_key.pub

If any key.pub file has a mode more permissive than "0644", this is a finding.

Fix Text: Change the mode of public host key files under "/etc/ssh" to "0644" with the following command:

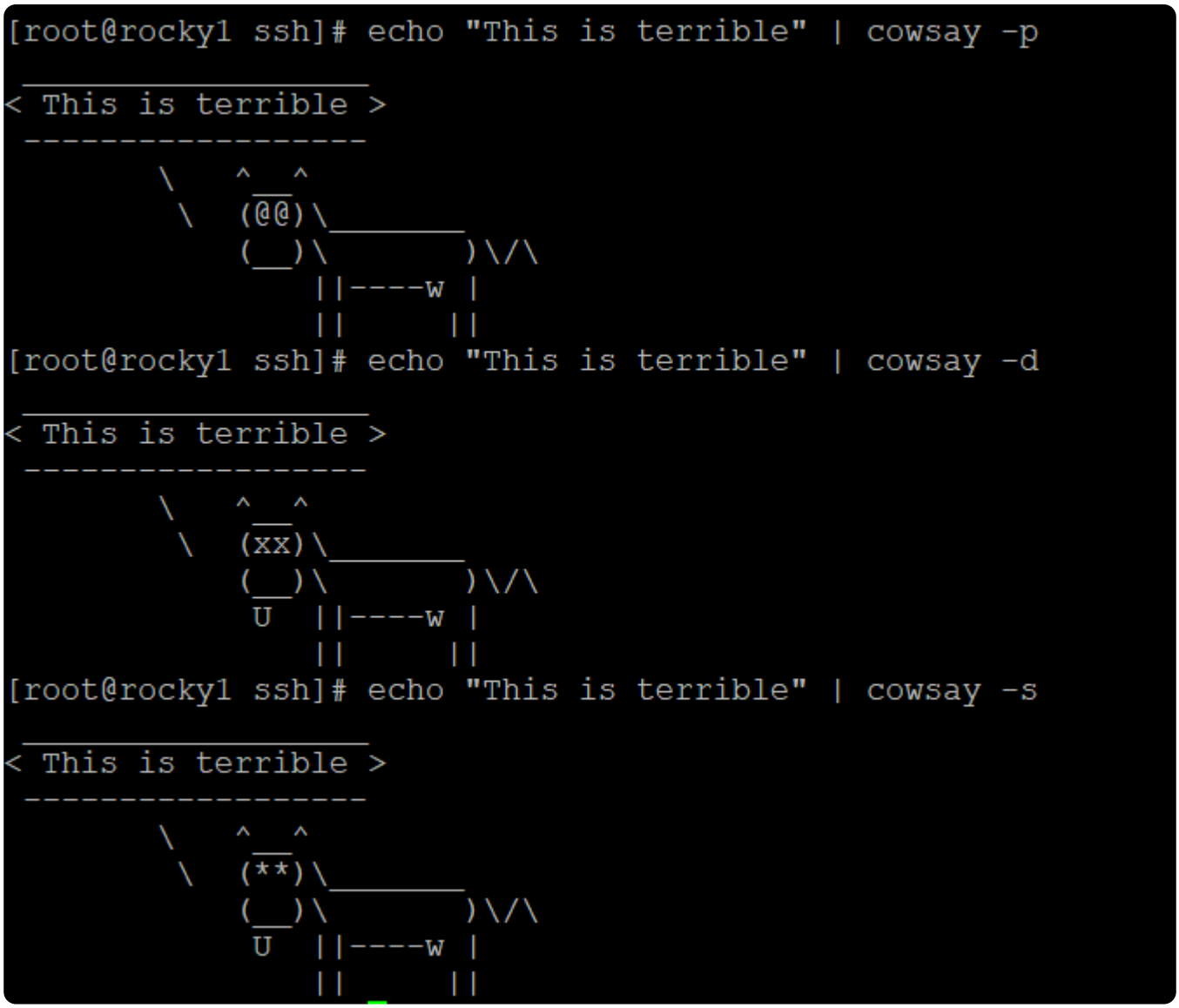
$ sudo chmod 0644 /etc/ssh/*key.pub

Restart the SSH daemon for the changes to take effect:

$ sudo systemctl restart sshd.service

   - Is your system hardened in this capacity?
   - How did you check?
   - Did the fix check work for you?
   - How did you check?

2. Remove unneeded Software
   - Read about cowsay – `man cowsay`
   - Remove cowsay – `dnf remove cowsay`



Rescan to validate change

`ssh` into a Rocky sever

```
cd /opt/scc
time ./cscc

# ---- Wait over an hour ------

cd /root/SCC/sessions #find the most recent run
```

Look in the results to see output.

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.14.4 Unit 13 Bonus - Hardening SSH and NGINX with Fail2Ban

> **Note**
>
> This is an **optional** bonus section. You **do not** need to read it, but if you're interested in digging deeper, this is for you.

This bonus lab is designed to provide material and exercises in applying the skills and principles outlined by this course.

Students will:

- Install necessary lab dependencies, `fail2ban`, `ssh`, `nginx`
- Build troubleshooting, deductive reasoning, and investigative skills
- Edit configuration files, utilize many command line tools, `systemctl`, `vi`
- Understand specific use case cyber security tools
- And more

RESOURCES / IMPORTANT LINKS

- https://github.com/fail2ban/fail2ban
- https://www.digitalocean.com/community/tutorials/how-fail2ban-works-to-protect-services-on-a-linux-server
- https://nginx.org/en/docs/beginners_guide.html
- https://www.openssh.com/features.html
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/getting-started-with-nftables_configuring-and-managing-networking#con_basics-of-nftables-tables_assembly_creating-and-managing-nftables-tables-chains-and-rules
- https://developers.cloudflare.com/waf/get-started/

REQUIRED MATERIALS

- Rocky 9.4+ host
  - Or comparable Linux box
- root or sudo command access

Bonus Pre-Lab Warm-Up

First lets identify a couple dependencies for this lab. Since many of the labs in this course are predicated on Rocky we will provide commands like `dnf` and `rpm` to accomplish this task:

```
rpm -ql openssh-server
```

This command will list configuration files and underlying dependent libraries required to run an SSH server if `openssh-server` is installed locally. But be careful, sometimes this query will list lingering configuration files and directories even though it isn't installed.

To corroborate our findings, let's query `dnf`:

```
dnf list --installed | grep openssh-server
```

All else fails you can always attempt a `find` command to verify the presence of SSH and its server side counter part SSHD (D for daemon):

```
find / -name ssh; find / -name sshd
```

You may need to use the `-xdev` flag if other filesystems are present or traversable that are external to the host that `find` executes from, this flag comes in handy with WSL subsystems or hosts residing within a hypervisor that have other host file systems mounted and accessible.

Fair warning, this command may spit out a lot of output. Familiarity with the Linux filesystem and its inner machinations should prepare you for this moment. If not, this command attempts to find every file with ssh or sshd in it from the root of the host's file system '/'.

If the piped grep or find command comes up empty `openssh-server` is likely not installed, attempt to install it. Hopefully these commands and strategies are familiar to you:

```
dnf install openssh-server
```

Now repeat this process for `nginx`, and `fail2ban`.
What are nginx and fail2ban you ask?
Well now is a great time to use the `man` pages.

If a `man` command ever fails due to a lack of 'manual entry' that means you may need to install the package or you have a path issue. However if the shell returns that the man command is not found you may need to install `man-db`.

Hopefully it's becoming clear why understanding our tools, troubleshooting, and deductive reasoning skills come in handy!

```
dnf install -y nginx fail2ban man-db

# We should now be able to man nginx and fail2ban
# if we couldn't before

man nginx
man fail2ban
```

Now that they're installed let's enable `sshd` and `nginx` from `systemctl`.

```
systemctl enable --now sshd # the --now flag along will also start the service
systemctl enable --now nginx
```

```
[root@rocky ~]# systemctl enable --now fail2ban
Created symlink /etc/systemd/system/multi-user.target.wants/fail2ban.service → /usr/lib/system
[root@rocky ~]# systemctl status fail2ban
● fail2ban.service - Fail2Ban Service
     Loaded: loaded (/usr/lib/systemd/system/fail2ban.service; enabled; preset: disabled)
    Drop-In: /run/systemd/system/service.d
             └─zzz-lxc-service.conf
     Active: active (running) since Fri 2025-04-18 19:33:39 UTC; 6s ago
       Docs: man:fail2ban(1)
    Process: 10698 ExecStartPre=/bin/mkdir -p /run/fail2ban (code=exited, status=0/SUCCESS)
   Main PID: 10699 (fail2ban-server)
      Tasks: 3 (limit: 178686)
     Memory: 10.9M
        CPU: 61ms
     CGroup: /system.slice/fail2ban.service
             └─10699 /usr/bin/python3 -s /usr/bin/fail2ban-server -xf start

Apr 18 19:33:39 rocky systemd[1]: Starting Fail2Ban Service...
Apr 18 19:33:39 rocky systemd[1]: Started Fail2Ban Service.
Apr 18 19:33:39 rocky fail2ban-server[10699]: Server ready
```

We should now be able to see these services are running in a myriad of ways:

```
ss -ntulp | grep ssh
ss -ntulp | grep nginx

systemctl status sshd
systemctl status nginx
```

Ideally these services are hosted on a student owned device/host and not the ProLUG lab boxes. With these enabled it should be possible to `ssh` into the host or visit the nginx web page via http://(localhost or {IP address}) from a web browser.

Technically you could `curl localhost` from the ProLUG Rocky box and capture the html nginx hosts and ssh into the box from a different box to verify they are enabled.

Bonus Lab 🧪

SSH, NGINX AND IMPLEMENTING FAIL2BAN

When services like SSH and NGINX (pronounced 'engine-x') are exposed to the public internet it will become exceedingly apparent (in seconds if not minutes) within their log files that they are under bombardment from web scrapers, bots, hackers, and more.

The goals of these entities are either malicious, self serving, or helpful in the case of organizations like Palo Alto who build databases for cybersecurity purposes.

An example of an nginx `access.log` file with Palo Alto's Expanse web scraping arm is shown below:

```
"Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to ident
led from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
"Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to ident
led from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
"Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to ident
led from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
" "Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to ide
luded from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
xpanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to identify
from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
"Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times per day to ident
led from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
" 404 43001 "-" "Expanse, a Palo Alto Networks company, searches across the global IPv4 space multiple times p
lke to be excluded from our scans, please send IP addresses/domains to: scaninfo@paloaltonetworks.com"
```

FAIL2BAN

Fail2Ban is a small part of a larger solution in mitigating and monitoring these threats and actors. While more robust solutions exist from providers like Cloudflare or Fortinet this lab hopes to introduce a small foray into implementing a more manageable solution for students.

Plus, it's kinda cool and pretty fun.

How Fail2Ban Works

Fail2Ban is designed to analyze log files and manipulate the host's firewall to programmatically ban offending IP addresses based on various conditions predicated by specific configuration files.

First let's briefly skim Fail2Ban's main configuration file:

```
vi /etc/fail2ban.conf
```

```
# Fail2Ban main configuration file
#
# Comments: use '#' for comment lines and ';' (following a space) for inline comments
#
# Changes:  in most of the cases you should not modify this
#           file, but provide customizations in fail2ban.local file, e.g.:
#
# [DEFAULT]
# loglevel = DEBUG
#

[DEFAULT]

# Option: loglevel
# Notes.: Set the log level output.
#         CRITICAL
#         ERROR
#         WARNING
#         NOTICE
#         INFO
#         DEBUG
# Values: [ LEVEL ]  Default: INFO
#
loglevel = INFO

# Option: logtarget
# Notes.: Set the log target. This could be a file, SYSTEMD-JOURNAL, SYSLOG, STDERR or STDOUT.
#         Only one log target can be specified.
#         If you change logtarget from the default value and you are
#         using logrotate -- also adjust or disable rotation in the
#         corresponding configuration file
#         (e.g. /etc/logrotate.d/fail2ban on Debian systems)
# Values: [ STDOUT | STDERR | SYSLOG | SYSOUT | SYSTEMD-JOURNAL | FILE ]  Default: STDERR
#
logtarget = /var/log/fail2ban.log

Remaining output redacted...
```

Generally this file can remain in its default configuration. We are however interested in its log path of `/var/log/fail2ban.log`. If we ever need to investigate or debug Fail2Ban we now know where it logs its decisions when an actionable event arises.

A further configuration file we are interested in is Fail2Ban's `jail.conf`. Let's look at Fail2Ban's `man jail.conf` entry first:

```
CONFIGURATION FILES FORMAT
 *.conf  files  are distributed by Fail2Ban.  It is recommended that *.conf files should
  remain unchanged to ease upgrades.  If needed, customizations should be provided in *.local files.
  For example, if you would like to enable the [ssh-iptables-ipset] jail specified in jail.conf,
  create jail.local containing

 jail.local
        [ssh-iptables-ipset]

        enabled = true

 In .local files specify only the settings you would like to change and the rest of the configuration
  will then come from the corresponding .conf file which is parsed first.

 jail.d/ and fail2ban.d/

        In addition to .local, for jail.conf or fail2ban.conf file there can be a corresponding .d/
        directory containing additional .conf  files.  The order e.g. for jail configuration would be:

        jail.conf
        jail.d/*.conf (in alphabetical order)
        jail.local
        jail.d/*.local (in alphabetical order).

        i.e.  all  .local  files  are  parsed after .conf files in the original configuration file and files
        under .d directory.  Settings in the file parsed later take precedence over identical entries in
        previously parsed files.  Files are ordered alphabetically, e.g.

        fail2ban.d/01_custom_log.conf - to use a different log path
        jail.d/01_enable.conf - to enable a specific jail
        jail.d/02_custom_port.conf - to change the port(s) of a jail.
```

This section is what we are most interested in. We see that Fail2Ban recommends utilizing `jail.local` files and drop-in directories to preserve initial configuration files and to facilitate updates down the road.

Further, lets now take a brief look at the `jail.conf` file and get an understanding of the options available to us. While this is a lot of information to take in it should be embraced for this is the bread and butter of becoming a better administrator. **Understanding the tools available to us**.

And for what it's worth, we don't need to know this tool down to its quantum level. "Just enough to get the job done."

```
vi /etc/fail2ban/jail.conf
```

Understanding jail configuration options as shown below are some of the many configuration details that will go a long way to intelligently protecting our services and systems.

```
#
# MISCELLANEOUS OPTIONS
#

# "bantime.increment" allows to use database for searching of previously banned ip's to increase a
# default ban time using special formula, default it is banTime * 1, 2, 4, 8, 16, 32...
#bantime.increment = true

# "bantime.maxtime" is the max number of seconds using the ban time can reach (doesn't grow further)
#bantime.maxtime =

# "bantime.factor" is a coefficient to calculate exponent growing of the formula or common multiplier,
# default value of factor is 1 and with default value of formula, the ban time
# grows by 1, 2, 4, 8, 16 ...
#bantime.factor = 1
```

Implementing Fail2Ban

Let's keep these types of options in mind and now look at an example of a basic `jail.local` file to protect our SSH server.

Based upon our observations earlier this command will make the necessary `jail.local` file we will need in its proper directory.

```
vi /etc/fail2ban/jail.local
```

And now let's define some necessary options to begin protecting SSH from broad attack.

```
[DEFAULT]
bantime            =  1h
maxretry           =  2
bantime.increment  =  true
bantime.factor     =  12
bantime.maxtime    =  5w
ignoreip           =  {IP_ADDRESS} # useful to prevent banning oneself
```

```
[sshd]
backend           =  systemd # required for systemd based systems
enabled           =  true
```

Let's make sure to save our new `jail.local` file before we restart Fail2Ban. The following command will write and persist the local jail configuration file and quit out of Vim, if you didn't know how to do this already.

```
:wq
```

It took a lot to get here but it's important to understand our firearms before we fire them. For something like Fail2Ban could potentially permanently lock us out of a system under the right circumstances and make us have a very bad day indeed.

While there are more options available for a `jail.local` file, this should suffice to ban repeat offenders who attempt to use an incorrect password on the SSH service. Any other required definitions to properly implement this `jail.local` file for SSH will be parsed before hand as was stipulated in the man page and the `jail.conf` file.

After that we must restart the Fail2Ban service to implement our changes.

```
systemctl restart fail2ban
```

Monitoring Fail2Ban

Now if everything is configured properly we should be able to input the following command to see a list of currently banned IP addresses:

```
fail2ban-client status sshd
```

```
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:  _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
```

Typically SSH services should not be exposed publicly to the internet, and if they are they should utilize public key infrastructure (PKI) with specific security constraints (see STIG or CIS Benchmark Controls). Generally It is a far better solution to lock SSH access behind an internal or Virtual Private Network (VPN) where the potential of a "slow brute-force" attack will be mitigated almost entirely.

However you still might institute Fail2Ban even behind a VPN in the event a threat actor does gain access to the VPN and attempts to brute force or slow brute force attack hosts with SSH services exposed that utilize passwords instead of PKI. However in this day and age password protected SSH access could be considered a security controls finding. Use PKI whenever possible.

NGINX

If you've made it this far I appreciate your tenacity or perhaps deep captivation of this subject.

Now where Fail2Ban really shines is when integrated with an application like nginx. As we'll see Fail2Ban allows us to configure what they call "filters" that can further protect `nginx` and our website, if we had one. Since this lab has ran long I will keep it brief.

Suffice it say for students this is theory, but as I will mention later these filters can be quite powerful.

Let's look at what is available to us for nginx:

```
ls /etc/fail2ban/filter.d/ | grep nginx
```

```
nginx-bad-request.conf
nginx-botsearch.conf
nginx-error-common.conf
nginx-forbidden.conf
```

```
nginx-http-auth.conf
nginx-limit-req.conf
```

Well, what do we have here?

I encourage you to explore all of these available filters however in my experience the `nginx-bad-request.conf` filter has been most useful in banning bad actors.

```
cat /etc/fail2ban2/filter.d/nginx-bad-request.conf
```

```
# Fail2Ban filter to match bad requests to nginx
#

[Definition]

# The request often doesn't contain a method, only some encoded garbage
# This will also match requests that are entirely empty
failregex = ^<HOST> - \S+ \[\] "[^"]*" 400

datepattern = {^LN-BEG}%%ExY(?P<_sep>[-/.])%%m(?P=_sep)%%d[T ]%%H:%%M:%%S(?:[.,]%%f)?(?:\s*%%z)?
              ^[^\[]*\[({DATE})
              {^LN-BEG}

journalmatch = _SYSTEMD_UNIT=nginx.service + _COMM=nginx

# Author: Jan Przybylak
```

Whenever nginx receives a "bad request", think an HTTP method request for a link, file, or API call that is malformed or improperly formatted from a client, Fail2Ban will recognize it from the `nginx` access.log by an HTTP Status Code 400 returned by `nginx` and act appropriately based upon our default action options.

(After 2 bad requests, you get banned for an hour. After that timeout, if you send 2 more bad requests the timeout is extended for 12 hours, growing exponentially for each occurrence afterwards up to a 5 week ban. It is typically advised against implementing permanent bans as over time this could lead to potentially significant overhead for the host or lockout legitimate users unintentionally.)

When running this filter on my own personal nginx web server over the course of 30 days I will have banned over 40,000 IP addresses with varying ban times. But note that this filter can and will match requests that are malformed from the site itself. If a link or API implementation hosted from the nginx web server is incorrect Fail2Ban could be banning legitimate traffic unintentionally. Be sure to observe due diligence in log files or web traffic when first implementing such a filter.

As of rebooting my publicly available website not minutes ago this is how many IP addresses are already banned.

```
Status for the jail: nginx-bad-request
├─ Filter
│  ├─ Currently failed: 0
│  ├─ Total failed:     3
│  └─ File list:        /var/log/nginx/access.log
└─ Actions
   ├─ Currently banned: 10
   ├─ Total banned:     10
   └─ Banned IP list:   147.139.145.197 170.39.218.2
```

Ideally Fail2Ban is configured to utilize `nftables`, a modernized and significantly improved Linux firewall backend solution, and some type of firewall backend wrapper like `firewalld` which can be configured for specific firewall backend architectures like `iptables`, and `nftables`.

Configuring Fail2Ban with `firewalld` or `nftables` will look like such:

```
[DEFAULT]
..etc
banaction          =  firewallcmd-rich-rules[actiontype=<multiport>]
banaction_allports  =  firewallcmd-rich-rules[actiontype=<allports>]

# Or

[DEFAULT]
..etc
```

```
banaction          =  nftables
banaction_allports  =  nftables[type=allports]
```

And here is how Fail2Ban modifies an `iptables` chain.

```
Chain f2b-nginx-bad-request (1 references)
target     prot opt source               destination
REJECT     all  --  154.83.103.210       anywhere             reject-with icmp-port-unreachable
REJECT     all  --  edinburgh.scan.bufferover.run  anywhere           reject-with icmp-port-unreachable
REJECT     all  --  104.248.241.110      anywhere             reject-with icmp-port-unreachable
REJECT     all  --  server.com           anywhere             reject-with icmp-port-unreachable
REJECT     all  --  rnd.group-ib.com     anywhere             reject-with icmp-port-unreachable
REJECT     all  --  45.95.169.130        anywhere             reject-with icmp-port-unreachable
REJECT     all  --  hostglobal.plus      anywhere             reject-with icmp-port-unreachable
REJECT     all  --  45.148.10.35         anywhere             reject-with icmp-port-unreachable
REJECT     all  --  45.148.10.34         anywhere             reject-with icmp-port-unreachable
REJECT     all  --  195.178.110.164      anywhere             reject-with icmp-port-unreachable
REJECT     all  --  195.178.110.163      anywhere             reject-with icmp-port-unreachable
REJECT     all  --  tube-hosting.com     anywhere             reject-with icmp-port-unreachable
REJECT     all  --  185.100.87.136       anywhere             reject-with icmp-port-unreachable
REJECT     all  --  170.39.218.2         anywhere             reject-with icmp-port-unreachable
REJECT     all  --  147.139.145.197      anywhere             reject-with icmp-port-unreachable
RETURN     all  --  anywhere             anywhere
```

I highly, highly encourage students have a working understanding of Linux firewalls and their components like tables and chains.

It's important to be stated that this should be understood as a **mitigation**, not a perfect solution. Primarily to minimize automated bots scraping a domain, usually for nefarious purposes.

And I will iterate that there are far better solutions. Some for free and others, (Cloudflare WAF), for a modest fee that protect more intelligently and robustly than Fail2Ban. Migrating your domain over to Cloudflare or equivalent service is probably the far smarter and less work intensive task than a comprehensive Fail2Ban setup. But I like to think we're building our muscles... you know, putting in reps.

Filter Implementation

Remembering our previous configurations we know that if a user sends more than two malformed, either malicious or unintentional, requests to our website they will be temporarily banned.

```
vi /etc/fail2ban/jail.local
```

Here is what the configuration file should look like after we implement this filter:

```
[nginx-bad-request]
enabled  =  true
port     =  http,https
logpath  =  %(nginx_access_log)s
```

Append these lines to the earlier file we implemented after our SSHD section and let the regex do the work for us.

Be sure to save the file:

```
:wq
```

Then finally restart Fail2Ban again:

```
systemctl restart fail2ban
```

It's as simple as that.

You can use `fail2ban-client status nginx-bad-request` to monitor any actions Fail2Ban has taken to ban bad actors or investigate the `/var/log/fail2ban.log` file.

Well that about wraps it up. There is far more to consider when it comes to protecting web servers and SSH, but hopefully this was a good primer and helped exercise those Linux muscles.

2.15 Unit 14 - Ansible Automation

2.15.1 Unit 14 - Ansible Automation

Overview

This unit introduces **Ansible Automation**, a powerful open-source tool used for IT automation, configuration management, and application deployment. By the end of this unit, you will understand how to implement Ansible in enterprise environments to manage Linux infrastructure efficiently.

1. **Configuration Management:** Automate system configurations across multiple hosts.
2. **Infrastructure as Code (IaC):** Define infrastructure using Ansible playbooks.
3. **Automation:** Execute tasks across multiple systems in an efficient, repeatable manner.

Learning Objectives

By the end of this unit, you should be able to:

- Set up and configure Ansible on a Linux system.
- Understand Ansible inventory and playbooks.
- Automate common administrative tasks.
- Use ad-hoc commands and Ansible modules effectively.

Key Terms and Definitions

Playbook	**Task**
Inventory	**Ad-hoc Commands**
Roles	

2.15.2 Unit 14 Worksheet - Ansible Automation

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Official Ansible Documentation
- Ansible GitHub Repository
- YAML Syntax Guide

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u14_worksheet(`.txt`)
- 📥 u14_worksheet(`.docx`)
- 📥 u14_worksheet(`.pdf`)

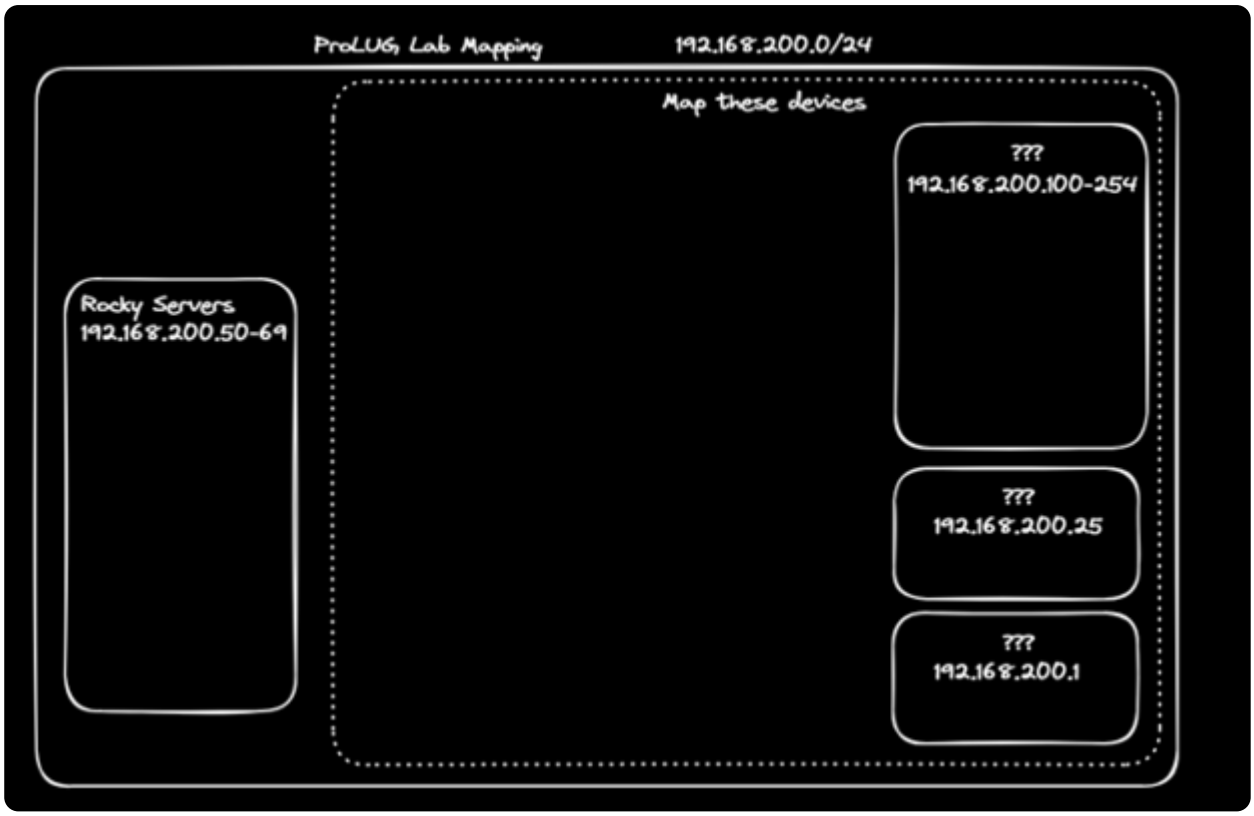
UNIT 14 RECORDING

Link: https://www.youtube.com/watch?v=-tk73llgt3E

Discussion Post 1

Refer to your Unit 5 scan of the systems.

> **Scenario**
>
> You know that Ansible is a tool that you want to maintain in the environment. Review this online documentation: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html



1. Make an inventory of the servers, grouped any way you like.
2. What format did you choose to use for your inventory?
3. What other things might you include later in your inventory to make it more useful?

Discussion Post 2

Scenario

You have been noticing drift on your server configurations, so you want a way to generate a report on them every day to validate the configurations are the same.

Use any lab in here to find ideas: https://killercoda.com/het-tanis/course/Ansible-Labs

Discussion Post 3

Using ansible module for git, pull down this repo: https://github.com/het-tanis/HPC_Deploy.git

1. How is the repo setup?
2. What is in the roles directory?
3. How are these playbooks called, and how do roles differ from tasks?

Info

Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Automation:

Consistency:

Dev/Ops:

Timelines:

Git:

Repository:

Ad-hoc:

Playbook:

Task:

Role:

SSH (Secure Shell):

WinRM (Windows Remote Management):

Digging Deeper (Optional)

1. I have a large amount of labs to get you started on your Ansible Journey (all free): https://killercoda.com/het-tanis/course/Ansible-Labs
2. Find projects from our channel Ansible-Code, in Discord and find something that is interesting to you.
3. Use Ansible to access secrets from Hashicorp Vault: https://killercoda.com/het-tanis/course/Hashicorp-Labs/004-vault-read-secrets-ansible

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT?
3. If you're not in IT, how can you look to put something like this into your resume or portfolio?

2.15.3 Unit 14 Lab - Ansible Automation

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Ansible Documentation
- Killercoda - Ansible Labs
- HPC_Deploy Repo

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u14_lab(`.pdf`)
- 📥 u14_lab(`.docx`)

WARMUP EXERCISES

Quickly run through your system and familiarize yourself:

```
mkdir /root/ansible_madness
cd /root/ansible_madness
dnf whatprovides ansible   # Where is Ansible installed from?
ansible --version          # What version of Ansible is installed?
ansible-<TAB>              # What other ansible-* tools are available?
ansible localhost -m shell -a uptime  # Compare with standalone `uptime`
ansible -vvv localhost -m shell -a uptime  # What extra info does -vvv show?
```

LAB EXERCISES

Create an Inventory File

While in `/root/ansible_madness`, create a file called `hosts`:

```
vi /root/ansible_madness/hosts
```

Add the following contents:

```
[servers]
192.168.200.101
192.168.200.102
192.168.200.103
```

RUN AD HOC COMMANDS

Test connectivity into the servers

```
ansible servers -i hosts -u inmate -k -m shell -a uptime
```

- Use password: `LinuxR0cks1!`

Verbose version:

```
ansible -vvv servers -i hosts -u inmate -k -m shell -a uptime
```

Create a Playbook to Push Files

1. Create a test file:

```
echo "This is my file <yourname>" > somefile
```

1. Create `deploy.yaml`:

```
---
- name: Start of push playbook
  hosts: servers
  vars:
  gather_facts: True
  become: False
  tasks:
    - name: Copy somefile over at {{ ansible_date_time.iso8601_basic_short }}
      copy:
        src: /root/ansible_madness/somefile
        dest: /tmp/somefile.txt
```

1. Run the playbook:

```
ansible-playbook -i hosts -k deploy.yaml
```

1. Verify the file was pushed everywhere:

```
ansible servers -i hosts -u inmate -k -m shell -a "ls -l /tmp/somefile"
```

Pull Down a GitHub Repo

```
git clone https://github.com/het-tanis/HPC_Deploy.git
cd HPC_Deploy
```

Then reflect:

- What do you see in here?
- What do you need to learn more about to deploy some of these tools?
- Can you execute some of these? Why or why not?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

2.16 Unit 15 - Troubleshooting

2.16.1 Unit 15 - Engineering Troubleshooting

Overview

In this unit, we focus on **incident management, root cause analysis, and troubleshooting frameworks**. These are foundational skills for Linux administrators who are responsible for maintaining system reliability and responding effectively to issues.

You'll explore structured approaches like the **Scientific Method**, **5 Whys**, **FMEA**, and **PDCA**, as well as methodologies like **Six Sigma**, **TQM**, and systems thinking. We'll also look at tools for visual problem solving, including the **Fishbone Diagram** and **Fault Tree Analysis**, and discuss how data types play a role in investigations.

Learning Objectives

By the end of this unit, you will be able to:

- Apply the **Scientific Method** to real-world troubleshooting scenarios
- Understand and use structured methods like **FMEA**, **5 Whys**, and **PDCA**
- Differentiate between **continuous and discrete data** in diagnostics
- Use visual tools like **Fishbone Diagrams** and **Fault Tree Analysis** to trace causes
- Explain the **OSI model** as it applies to layered troubleshooting
- Leverage concepts from **Six Sigma** and **5S methodology** to organize your workflows
- Document and communicate incidents effectively with post-mortem writeups

Key Terms and Definitions

Incident	**Problem**
FMEA	**Six Sigma**
TQM	**Post Mortem**
ScientificMethod	**Iterative**
Discrete Data	**Discrete Data** - Ordinal
Discrete Data - Nominal(binary - attribute)	**Continuousdata**
Risk Priority Number(RPN)	**5 Whys**
Fishbone Diagram (Ishikawa)	**Fault Tree Analysis(FTA)**
PDCA	**SIPOC**

2.16.2 Unit 15 Worksheet - Engineering Troubleshooting

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Six Sigma Intro

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u15_worksheet(`.txt`)
- 📥 u15_worksheet(`.docx`)
- 📥 u15_worksheet(`.pdf`)

UNIT 15 RECORDING

Link: https://www.youtube.com/watch?v=UFEH3w1U_rc

Unit 15 Discussion Post #1

> **Scenario**
>
> Your management is all fired up about implementing some Six Sigma processes around the company. You decide to familiarize yourself and get some basic understanding to pass along to your team. Six Sigma Intro

1. Page 56 – What about the “5S” methodology might help us as a team of system administrators? (Think of your virtual or software workspaces)
2. Page 94 - What are the four layers of process definition? How would you explain them to your junior engineers?

Unit 15 Discussion Post #2

> **Scenario**
>
> Your team looks at a lot of visual data. You decide to write up an explanation for them to explain what they look at.

1. What is a high water mark? Why might it be good to know in utilization of systems?
2. What is an upper and lower control limit in a system output? While this isn’t exactly what we’re looking at, why might it be good to explain to your junior engineers

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Incident:

Problem:

FMEA:

Six Sigma:

TQM:

Post Mortem:

Scientific Method:

Iterative:

Discrete data:

Ordinal:

Nominal (binary – attribute):

Continuous data:

Risk Priority Number (RPN):

5 Whys:

Fishbone Diagram (Ishikawa):

Fault Tree Analysis (FTA):

PDCA:

SIPOC:

Digging Deeper

1. Spend more time in Six Sigma Intro a. Page 243 – Starts looking at visual data analysis.
2. Get your White belt (Free) Six Sigma Certification.

Reflection Questions

1. What questions do you still have about this week?
2. How can you apply this now in your current role in IT? If you're not in IT, how can you look to put something like this into your resume or portfolio?

2.16.3 Unit 15 Lab - Engineering Troubleshooting

No lab for unit 15. Work on your projects.

2.17 Unit 16 - Incident Response

2.17.1 Unit 16 - Incident Response

Overview

This unit introduces Incident Response, a critical discipline in cybersecurity and systems administration focused on identifying, containing, eradicating, recovering from, and learning from system incidents. This unit also demonstrates the crucial need for policies and procedures in the likely event an incident occurs which facilitate successful remedies for administrators during stressful events.

By the end of this unit, you'll understand the key phases and practices for developing and executing an effective incident response plan within enterprise environments to minimize the impact of system incidences.

Learning Objectives

By the end of this unit, you should be able to:

- Define the stages of the incident response lifecycle.
- Understand the roles and responsibilities within an incident response team.
- Outline steps for initial incident detection and triage.
- Describe methods for system recovery and post-incident analysis.

2.17.2 Unit 16 Worksheet - Incident Response

Instructions

No worksheet or discussion for this unit, course wrap up and lecture only.

UNIT 16 RECORDING

Link: https://www.youtube.com/watch?v=VJ0Z9O_7j48

2.17.3 Unit 16 Lab - Incident Response

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

The unit 16 lab will be done during the lecture. Students will troubleshoot problems on systems in the ProLUG lab environment.

REQUIRED MATERIALS

- Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u16_lab(`.pdf`)
- 📥 u16_lab(`.txt`)
- 📥 u16_lab(`.docx`)

Lab

You have the answers here, if they ask, you may give them hints. Otherwise, you can help them find the right solution any way you want to.

SCENARIO 1

- Connect to `tshoot1@prolug.asuscomm.com`
- Password:

A ticket has come in that the web server is not running on the web server.

To complete this event the following three must be correct.

1. Web server must be running.
   - HINT:

```
systemctl status httpd
```

2. Web server must respond on port 80.
   - HINT: Can you check the open ports?
3. Ensure that the server can be reached by external connection attempts on port 80.
   - HINT: Is the firewall running?

```
systemctl status firewalld
```

Reboot the lab machine when finished.

SCENARIO 2

- Connect to `tshoot2@prolug.asuscomm.com`
- Password:

A ticket has come in that a mount point, `/space`, is not working correctly. The team expected a 9GB partition to be built there on the 3 attached disks, but found that it was not a separate partition.

Verify that `/space` is not set up correctly.

- HINT: You may want to revisit lab 3 of the course for this one. This is a challenge here.

To complete this event the following four must be correct.

1. The three disks must be properly set up in LVM.
    - HINT: use `mkfs` to make a filesystem.
2. EXT4 or XFS must be installed on the logical volume.
    - HINT: Use your `pvs`, `vgs`, `lvs` tools
3. `/space` must be created and mounted off on this filesystem.
    - Hint: Make the directory
4. `/etc/fstab` or `systemd` must have an entry for `/space` (do not reboot during the lab, as this will not work).

Same way as above.

SCENARIO 3

- Connect to `tshoot3@prolug.asuscomm.com`
- Password:

Your team is trying to update your servers during a maintenance window. Your junior administrator kicks you over a server that they cannot get to update.

To complete this event the following two must be correct.

1. Fix the system to be able to update via `dnf`.
    - HINT: DNF isn't updating, so where are the repos that it looks for?
2. Verify that kernel updates are happening.
    - HINT: Where can updates be excluded in DNF or Yum?

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

2.18 Course Resources

2.18.1 Course Resources

This is a comprehensive list of all external resources used in this course.

Unit 1 - Linux File Operations

- https://www.youtube.com/watch?v=d8XtNXutVto
- https://vim-adventures.com/
- https://www.youtube.com/watch?v=eHB8WKWz2eQ
- Linux CLI Cheatsheets
- The Linux Foundation
- What is Vim?

Unit 2 - Essential Tools

- https://www.youtube.com/watch?v=miVuSoHTuP4
- Security Enhanced Linux
- Bash Reference Manual
- Top 50+ Linux CLI Commands

Unit 3 - Storage

- Ubuntu Documentation on LVM
- Implementing SLOs
- AWS Real-Time Communication Whitepaper
- Building Secure and Reliable Systems
- here
- https://www.youtube.com/watch?v=NYL85ndQLbc
- Red Hat High Availability Cluster Configuration
- AWS High Availability Architecture Guide
- Google SRE Book - Implementing SLOs

Unit 4 - Operating Running Systems

- the Cron wiki
- here
- tldp.org's cron guide
- Cron Wiki page
- Killercoda Labs
- https://en.wikipedia.org/wiki/Battle_drill
- https://zeltser.com/media/docs/security-incident-survey-cheat-sheet.pdf?msc=Cheat+Sheet+Blog
- https://cio-wiki.org/wiki/Operations_Bridge
- https://www.youtube.com/watch?v=MulGrUKSMRg
- Battle Drills

- Security Incident Cheatsheet
- Operations Bridge

Unit 5 - Managing Users and Groups

- https://www.cobalt.io/blog/defending-against-23-common-attack-vectors
- https://owasp.org/www-project-top-ten/
- https://attack.mitre.org/
- https://www.youtube.com/watch?v=xLv7ClJD6UI
- Attack Vectors
- attack.mitre.org
- OWASP Top Ten
- Killercoda lab by FishermanGuyBro
- Rocky Linux User Admin Guide
- RedHat: User and Group Management

Unit 6 - Firewalls

- Firewalld Official Documentation
- https://docs.rockylinux.org/zh/guides/security/firewalld-beginners/
- https://www.youtube.com/watch?v=wCVj3qeLTMg
- Wikipedia entry for Next-Gen Firewalls
- Official UFW Documentation
- RedHat Firewalld Documentation
- Official Firewalld Documentation
- Zellij Site
- GNU Screen Manual
- GNU Screen Site
- Tmux Cheatsheet
- Tmux Wiki

Unit 7 - Package Management & Patching

- https://en.wikipedia.org/wiki/Yellow_Dog_Linux
- http://tldp.org/LDP/abs/html/arithexp.html
- dpkg backend.
- the apt frontend
- Rocky Repository Guidance
- https://semver.org/
- https://www.youtube.com/watch?v=SwrkhTwzN_4
- Rocky DNF Guidance
- Rocky Documentation
- Semantic Versioning
- Using sha256sum
- AIDE Documentation
- RPM Man Page

Unit 8 - Scripting

- https://en.wikipedia.org/wiki/Truth_table
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
- https://www.youtube.com/watch?v=sn_LwfIObQw
- Bash Hacker's Wiki
- devhints.io - Bash Scripting Cheatsheet
- TLDP Bash Beginner's Guide
- http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html
- http://tldp.org/LDP/abs/html/tests.html
- Truth Table - Wikipedia
- Operate Running Systems lab
- Unit #8 Bonus (Bash Scripting) Page
- Unit #1 Bonus (VIM) Page
- CProgramming.com Introductory
- Wikipedia Page for Truth Table
- Wikipedia Page for the C Programming Lang
- Wikipedia Page for Compilers

Unit 9 - Containerization on Linux

- https://docs.podman.io/en/latest/markdown/podman-exec.1.html
- https://docs.docker.com/build/concepts/dockerfile/
- https://podman.io/docs
- https://killercoda.com/k3s/scenario/intro
- https://www.youtube.com/watch?v=_lo50mgERJY
- Podman Command List
- Dockerfile Reference Page

Unit 10 - Kubernetes

- Kubernetes Troubleshooting Guide
- K3s Official Site
- Kubernetes Documentation
- KillerShell Kubernetes Security
- Docker Security Best Practices
- https://www.youtube.com/watch?v=KycsHfZoAQs
- Interactive Kubernetes Labs
- K3s Official Documentation
- Kubernetes Overview
- Kubernetes Security Best Practices
- Kubernetes Hardening Guide (Archived)
- Pod Security Standards

- NetworkPolicy Editor

Unit 11 - Monitoring

- https://www.youtube.com/watch?v=54VgGHr99Qg
- https://promlabs.com/promql-cheat-sheet/
- https://sre.google/workbook/monitoring/
- https://www.youtube.com/watch?v=6VOHFYkptOw
- Monitoring Linux Using SNMP - Nagios
- 30 Linux System Monitoring Tools Every SysAdmin Should Know
- How to easily monitor your Linux server | Grafana Labs
- https://killercoda.com/het-tanis/course/Linux-Labs/104-monitoring-linux-Influx-Grafana
- https://killercoda.com/het-tanis/course/Linux-Labs/103-monitoring-linux-telemetry
- https://killercoda.com/het-tanis/course/Linux-Labs/102-monitoring-linux-logs

Unit 12 - Baselines & Benchmarks

- https://www.youtube.com/watch?v=8psu0D4rSmc
- Kaggle - Python and Data Science Learning
- iperf3 Documentation
- stress GitHub
- iostat Manual
- SAR Documentation

Unit 13 - System Hardening

- https://nvd.nist.gov/vuln/search
- https://public.cyber.mil/stigs/srg-stig-tools/
- https://killercoda.com/het-tanis/course/Linux-Labs/207-OS_STIG_Scan_with_SCC_Tool
- https://killercoda.com/het-tanis/course/Linux-Labs/203-updating-golden-image
- https://killercoda.com/het-tanis/course/Linux-Labs/107-server-startup-process
- https://www.youtube.com/watch?v=ESsUM0Gz8Jk
- Killercoda Lab - Updating a Golden Image
- Killercoda Lab - Server Startup Process
- Cloudflare WAF),
- architectures
- a modernized and significantly improved Linux firewall backend solution,
- "bad request",
- nginx.
- public key infrastructure
- https://developers.cloudflare.com/waf/get-started/
- https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/configuring_and_managing_networking/getting-started-with-nftables_configuring-and-managing-networking#con_basics-of-nftables-tables_assembly_creating-and-managing-nftables-tables-chains-and-rules
- https://www.openssh.com/features.html
- https://nginx.org/en/docs/beginners_guide.html
- https://www.digitalocean.com/community/tutorials/how-fail2ban-works-to-protect-services-on-a-linux-server

- https://github.com/fail2ban/fail2ban

Unit 14 - Ansible Automation

- HPC_Deploy Repo
- Killercoda - Ansible Labs
- Ansible Documentation
- https://killercoda.com/het-tanis/course/Hashicorp-Labs/004-vault-read-secrets-ansible
- https://github.com/het-tanis/HPC_Deploy.git
- https://killercoda.com/het-tanis/course/Ansible-Labs
- https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html
- https://www.youtube.com/watch?v=-tk73llgt3E
- YAML Syntax Guide
- Ansible GitHub Repository
- Official Ansible Documentation

Unit 15 - Troubleshooting

- https://www.youtube.com/watch?v=UFEH3w1U_rc
- Six Sigma Intro

Unit 16 - Incident Response

- https://www.youtube.com/watch?v=VJ0Z9O_7j48

Misc

- Killercoda.
- https://github.com/ProfessionalLinuxUsersGroup/course-books
- @Het_Tanis
- overleaf.com
- Draw.io
- Excalidraw.com

2.18.2 Course Work Downloads

This page contains the downloads for all original course material (labs and worksheets) used in this course.

Unit 1

WORKSHEET

- Download Unit 1 Worksheet (`.txt`)
- Download Unit 1 Worksheet (`.docx`)

LAB

- Download Unit 1 Lab (`.txt`)
- Download Unit 1 Lab (`.docx`)
- Download Unit 1 Lab (`.pdf`)

Unit 2

WORKSHEET

- Download Unit 2 Worksheet (`.txt`)
- Download Unit 2 Worksheet (`.docx`)

LAB

- Download Unit 2 Lab (`.txt`)
- Download Unit 2 Lab (`.docx`)

Unit 3

WORKSHEET

- Download Unit 3 Worksheet (`.txt`)
- Download Unit 3 Worksheet (`.docx`)

LAB

- Download Unit 3 Lab (`.pdf`)
- Download Unit 3 Lab (`.txt`)
- Download Unit 3 Lab (`.docx`)

Unit 4

WORKSHEET

- Download Unit 4 Worksheet (`.txt`)
- Download Unit 4 Worksheet (`.docx`)

LAB

- Download Unit 4 Lab (`.pdf`)
- Download Unit 4 Lab (`.txt`)
- Download Unit 4 Lab (`.docx`)

Unit 5

WORKSHEET

- Download Unit 5 Worksheet (`.txt`)
- Download Unit 5 Worksheet (`.docx`)

LAB

- Download Unit 5 Lab (`.pdf`)
- Download Unit 5 Lab (`.docx`)
- Download Unit 5 Lab (`.txt`)

Unit 6

WORKSHEET

- Download Unit 6 Worksheet (`.txt`)
- Download Unit 6 Worksheet (`.docx`)

LAB

- Download Unit 6 Lab (`.txt`)
- Download Unit 6 Lab (`.docx`)
- Download Unit 6 Lab (`.pdf`)

Unit 7

WORKSHEET

- Download Unit 7 Worksheet (`.txt`)
- Download Unit 7 Worksheet (`.docx`)

LAB

- Download Unit 7 Lab (`.pdf`)
- Download Unit 7 Lab (`.docx`)
- Download Unit 7 Lab (`.txt`)

Unit 8

WORKSHEET

- Download Unit 8 Worksheet (`.txt`)
- Download Unit 8 Worksheet (`.docx`)

LAB

- Download Unit 8 Lab (`.docx`)
- Download Unit 8 Lab (`.txt`)

Unit 9

WORKSHEET

- Download Unit 9 Worksheet (`.pdf`)
- Download Unit 9 Worksheet (`.docx`)
- Download Unit 9 Worksheet (`.txt`)

LAB

- Download Unit 9 Lab (`.pdf`)
- Download Unit 9 Lab (`.docx`)
- Download Unit 9 Lab (`.txt`)

Unit 10

WORKSHEET

- Download Unit 10 Worksheet (`.docx`)
- Download Unit 10 Worksheet (`.pdf`)

LAB

- Download Unit 10 Lab (`.docx`)
- Download Unit 10 Lab (`.pdf`)

Unit 11

WORKSHEET

- Download Unit 11 Worksheet (`.docx`)
- Download Unit 11 Worksheet (`.pdf`)

LAB

- Download Unit 11 Lab (`.pdf`)
- Download Unit 11 Lab (`.docx`)

Unit 12

WORKSHEET

- Download Unit 12 Worksheet (`.pdf`)
- Download Unit 12 Worksheet (`.docx`)

LAB

- Download Unit 12 Lab (`.docx`)
- Download Unit 12 Lab (`.pdf`)

Unit 13

WORKSHEET

- Download Unit 13 Worksheet (`.pdf`)
- Download Unit 13 Worksheet (`.docx`)

LAB

- Download Unit 13 Lab (`.pdf`)
- Download Unit 13 Lab (`.docx`)

Unit 14

WORKSHEET

- Download Unit 14 Worksheet (`.pdf`)

- Download Unit 14 Worksheet (`.docx`)

LAB

- Download Unit 14 Lab (`.pdf`)
- Download Unit 14 Lab (`.docx`)

Unit 15

WORKSHEET

- Download Unit 15 Worksheet (`.pdf`)
- Download Unit 15 Worksheet (`.docx`)

LAB

- Download Unit 15 Lab (`.pdf`)
- Download Unit 15 Lab (`.docx`)

Unit 16

LAB

- Download Unit 16 Lab (`.docx`)
- Download Unit 16 Lab (`.pdf`)

3. Linux Security Course

3.1 Course Overview

3.1.1 ProLUG Security Engineering Course

Welcome to the ProLUG Security Engineering Course Book.

This Book

Contains all materials pertaining to the course including links to external resources. It has been put together with care by a number of ProLUG group members referencing original instructional materials produced by Scott Champine @Het_Tanis .

The content is version controlled with Git and stored here: https://github.com/ProfessionalLinuxUsersGroup/course-books/

COURSE DESCRIPTION

This course addresses how to secure Linux a corporate environment. This course will focus on adhering to regulations, best practices, and industry standards. This course will expose the concepts of controls, their implementation, and how they fit into overall security posture. The learner will practice securely building, deploying, integrating, and monitoring Linux systems. Standard security documentation and reporting will be practiced throughout, to better prepare the learner for the industry.

PREREQUISITE(S) AND/OR COREQUISITE(S):

Prerequisites: None
Credit hours: N/A
Contact hours: 100 (40 Theory Hours, 60 Lab Hours)

Course Summary

MAJOR INSTRUCTIONAL AREAS

- Build Standards and Compliance
- Securing the Network Connection
- User Access and System Integration
- Bastion Hosts and Air-Gaps
- Updating Systems and Patch Cycles
- Monitoring and Parsing Logs
- Monitoring and Alerting
- Configuration drift and Remediation
- Certificate and Key Madness

COURSE OBJECTIVES

- Build and configure a Linux system to adhere to compliance frameworks
- Integrating Linux to a network in a secure fashion
- Integrating Linux with Enterprise Identity and Access Management (IAM) frameworks
- Implement User ingress controls to a system/network with bastion frameworks
- Updating Linux to resolve security vulnerabilities and reporting out to security teams
- Design logging workflows to move event logging off of systems for real time monitoring
- Monitoring and alerting on events in Linux
- Maintaining system configuration drift and remediation

WRITTEN DISCUSSIONS

Are assigned as 'Discussion Posts' within each unit. Discussions generally take place within the Discord Server under #prolug-projects. More specifically, each unit will contain links to particular discussion posts within #prolug-projects.

COMPLETING THE COURSE

In order to complete this course students must participate in group discussions and complete provided labs. Additionally, students are to propose and complete a final project involving skills learned from the course.

RECOMMENDED TOOLS, RESOURCES, AND FRAMEWORKS

- Killercoda: https://killercoda.com/
- STIG Resources: https://public.cyber.mil/stigs/srg-stig-tools/
    - Recommended (but not required) STIG Viewer: v2.18
- NIST: https://www.nist.gov/
- Open Worldwide Application Security Project Top 10: https://owasp.org/www-project-top-ten/
- CIS Controls and Benchmarks: https://www.cisecurity.org/cis-benchmarks

REQUIRED RESOURCES

Option #1 (Killercoda Machine)

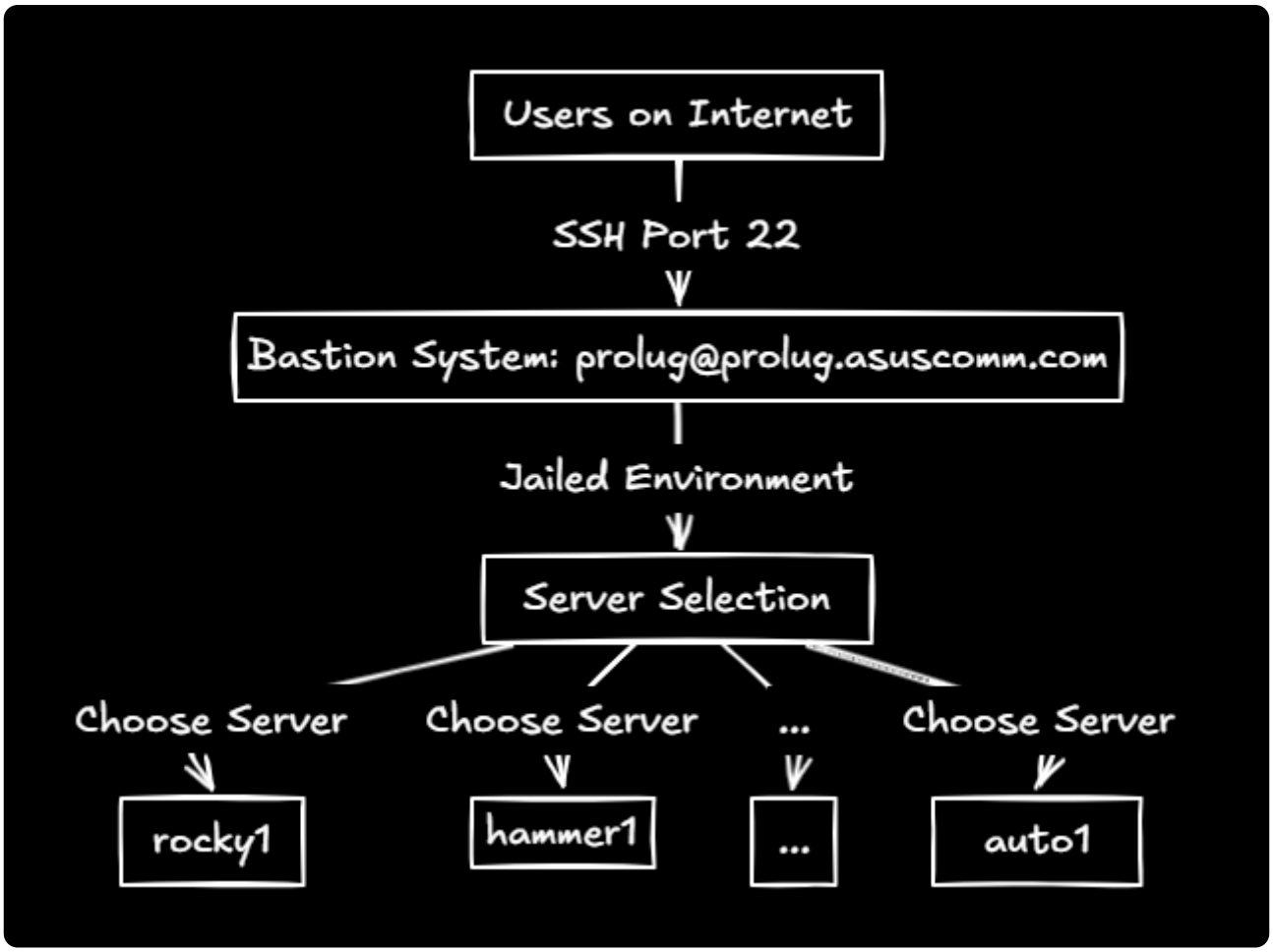
1. Cloud Lab server running Ubuntu on Killercoda. Minimal resources can accomplish our tasks
    - 1 CPU
    - 2 GB Ram
    - 30 GB Hard Drive
    - Network Interface (IP already setup)

Option #2 (Home Lab)

1. Local VM server running: RHEL, Fedora, Rocky Minimal resources
    - 1 CPU
    - 2GB RAM
    - Network Interface (Bridged)

Option #3 (ProLUG Remote Lab)

1. ProLUG Lab access to Rocky 9.4+ instance. Minimal resources can accomplish our tasks
    - 1 CPU
    - 4 GB RAM
    - Network Interface (IP already setup)



COURSE PLAN

INSTRUCTIONAL METHODS

This course is designed to promote learner-centered activities and support the development of Linux security skills. The course utilizes individual and group learning activities, performance-driven assignments, problem-based cases, projects, and discussions. These methods focus on building engaging learning experiences conducive to development of critical knowledge and skills that can be effectively applied in professional contexts.

CLASS SIZE

This class will effectively engage 40-60 learners.

CLASS SCHEDULE

https://discord.com/events/611027490848374811/1353330418669326407

Class will meet over weekend (Brown bag) sessions. 1 time per week, for 10 weeks. There will be a total of 10 sessions.

Session	Topic
1	Unit 1 - Build Standards and Compliance
2	Unit 2 - Securing the network connection
3	Unit 3 - User Access and system integration
4	Unit 4 - Bastion hosts and airgaps
5	Unit 5 - Updating systems and patch cycles
6	Unit 6 - Monitoring and parsing logs
7	Unit 7 - Monitoring and alerting
8	Unit 8 - Configuration drift and remediation
9	Unit 9 - Certificate and key madness
10	Unit 10 - Recap and final project

SUGGESTED LEARNING APPROACH

In this course, you will be studying individually and within a group of your peers, primarily in a lab environment. As you work on the course deliverables, you are encouraged to share ideas with your peers and instructor, work collaboratively on projects and team assignments, raise questions, and provide constructive feedback.

3.1.2 ProLUG Security Engineering - Final Project

Students wishing to complete the Security Engineering course are expected to devise and complete a capstone project, to be turned in at the end of the course.

The instructions, expectations, and deliverables for the project are listed on this page.

Instructions

1. We have picked up a new client. They are requesting we help them adhere to the HIPAA compliance standard.
    - Review an explanation of the standard here: https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html
    - If you are in the EU and want to substitute GDPR, you may do so. https://gdpr.eu/what-is-gdpr/
2. Build the documentation for HIPAA Compliance.
    - How are we implementing Risk analysis and management?
    - What are our safeguards?
        a. Administrative
        b. Physical
        c. Technical
    - How do we form Business Associate Agreements
    - What are our documentation practices?
        a. Policies
        b. Procedures
        c. Update and review cadence
3. Prepare to Present (https://www.overleaf.com/ is a great alternative to Powerpoint)
    - Setup a 15-20 slide deck on what you did
        a. Project purpose
        b. Diagram
        c. Build Process
        d. What did you learn?
        e. How are you going to apply this?
4. Do any of you want to present?
    - Let Scott know (@het_tanis) and we'll get you a slot in the last few weeks.

Deliverables

1. A 15-20 slide presentation of the above material that you would present to a group (presenting to us is voluntary, but definitely possible.)
    - This can be done with Microsoft PowerPoint, LibreOffice Impress, or overleaf.com.

3.1.3 Table of Contents

3.2 Unit 1 - Build Standards and Compliance

3.2.1 Unit 1 - Build Standards and Compliance

Overview

Building standards and compliance in cybersecurity engineering ensures that systems adhere to industry best practices, regulatory requirements, and security frameworks, reducing risks and vulnerabilities.

By implementing structured guidelines through tools and frameworks like STIGs (Security Technical Implementation Guides) and the NIST CS (National Institute of Standards and Technology Cyber Security) framework, organizations can maintain resilience against evolving threats while ensuring accountability and regulatory alignment.

This chapter will present critical knowledge in implementing security controls in information systems.

Learning Objectives

By the end of Unit 1 students will have foundational knowledge and skills of the concepts below:

1. Security Frameworks such as STIGs, CIS Controls, NIST Cybersecurity Framework
2. Regulatory Compliance and Industry Standards when administering and building systems
3. Skills and concepts in interacting with STIG remediation processes
4. Understanding Risk Management and concepts surrounding risk vectors to organizations
5. STIG Remediation and documentation skills

Key terms and Definitions

CIA Triad	**Regulatory Compliance**
HIPAA	**Industry Standard**
PCI/DS	**Security Framework**
CIS	**STIG**

3.2.2 Unit 1 Worksheet - Build Standards and Compliance

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://public.cyber.mil/stigs/downloads/
- https://excalidraw.com
- https://www.open-scap.org
- https://www.sans.org/information-security-policy
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u1_worksheet(`.txt`)
- 📥 u1_worksheet(`.docx`)

UNIT 1 RECORDING

Link: https://www.youtube.com/watch?v=F6loIx9WwGM

Discussion Post #1

The first question of this course is, "What is Security?"

1. Describe the CIA Triad.
2. What is the relationship between Authority, Will, and Force as they relate to security?
3. What are the types of controls and how do they relate to the above question?

Discussion Post #2

Find a STIG or compliance requirement that you do not agree is necessary for a server or service build.

1. What is the STIG or compliance requirement trying to do?
2. What category and type of control is it?
3. Defend why you think it is not necessary. (What type of defenses do you think you could present?)

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

CIA Triad:

Regulatory Compliance:

HIPAA:

Industry Standards:

PCI/DSS:

Security Frameworks:

CIS:

STIG:

Digging Deeper

1. Research a risk management framework. https://csrc.nist.gov/projects/risk-management/about-rmf
    - What are the areas of concern for risk management?
2. Research the difference between quantitative and qualitative risks.
    - Why might you use one or the other?
3. Research ALE, SLE, and ARO.
    - What are these terms in relation to?
    - How do these help in the risk discussion?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

3.2.3 Unit 1 Lab - Build Standards and Compliance

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u1_lab(`.txt`)
- 📥 u1_lab(`.docx`)

Module 1: Exploring System Information

EXERCISE 1.1: FAMILIARIZING OURSELVES WITH THE SYSTEM

```
mount | grep -i noexec

mount | grep -i nodev

mount | grep -i nosuid

# Approximately how many of your mounted filesystems have each of these values?
```

EXERCISE 1.2: CHECKING MOUNTED SYSTEMS

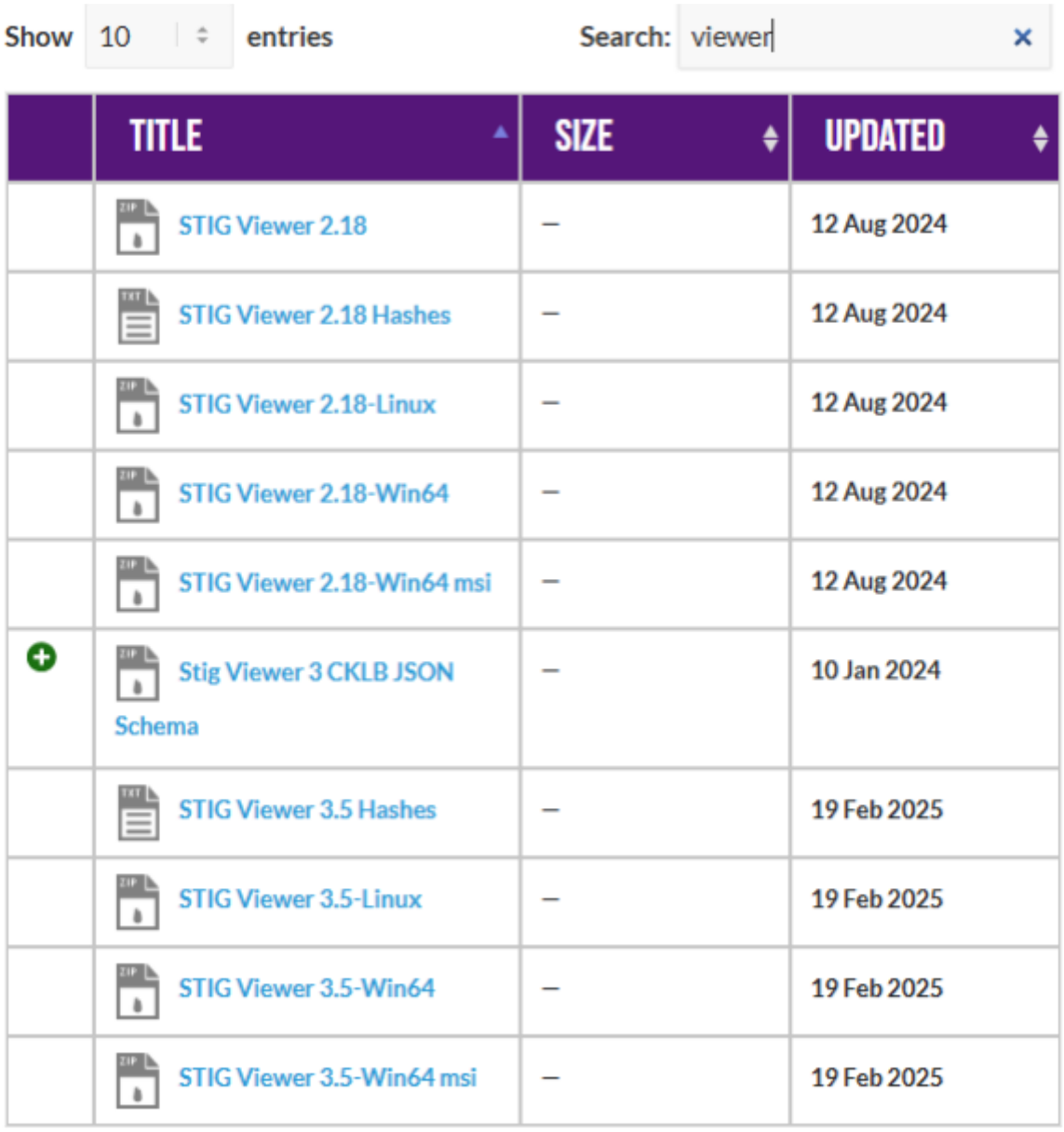
```
sysctl -a | grep -i ipv4

sysctl -a | grep -i ipv6

# How many of each are there?
```

```
sysctl -a | grep -i ipv4 | grep -i forward

# Does IPv4 forward on interfaces?
```

```
lsmod | grep -i tables

# What type of tables exist?
```

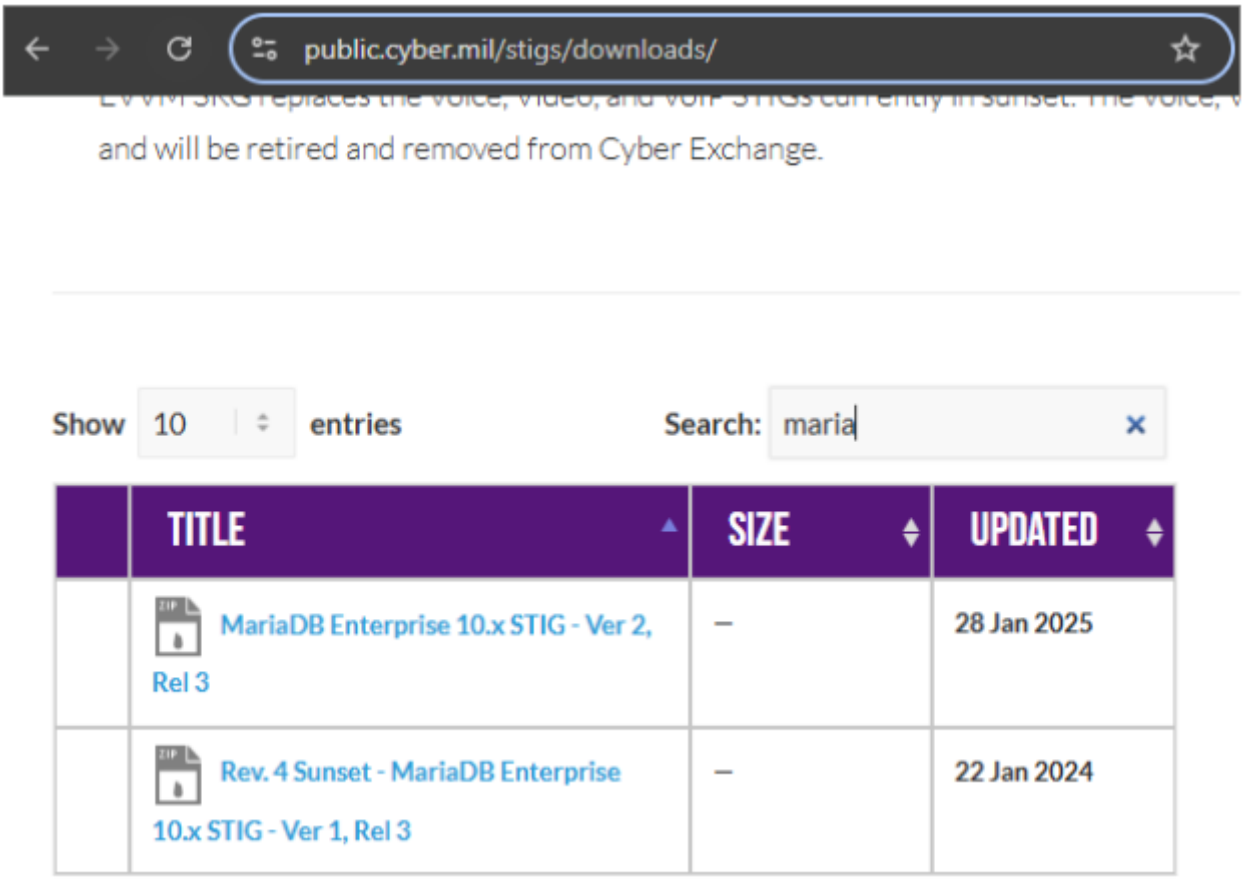
Module 2: PreLAB

1. Download the STIG Viewer 2.18 from - https://public.cyber.mil/stigs/downloads/



	TITLE	SIZE	UPDATED
	STIG Viewer 2.18	–	12 Aug 2024
	STIG Viewer 2.18 Hashes	–	12 Aug 2024
	STIG Viewer 2.18-Linux	–	12 Aug 2024
	STIG Viewer 2.18-Win64	–	12 Aug 2024
	STIG Viewer 2.18-Win64 msi	–	12 Aug 2024
	Stig Viewer 3 CKLB JSON Schema	–	10 Jan 2024
	STIG Viewer 3.5 Hashes	–	19 Feb 2025
	STIG Viewer 3.5-Linux	–	19 Feb 2025
	STIG Viewer 3.5-Win64	–	19 Feb 2025
	STIG Viewer 3.5-Win64 msi	–	19 Feb 2025

2. Download the STIG for Mariadb and the import it into your STIG viewer.



Module 3: Lab

This lab is designed to have the engineer practice securing a Linux server or service against a set of configuration standards. These standards are sometimes called benchmarks, checklists, or guidelines.
The engineer will be using STIG Viewer 2.18 to complete this lab.

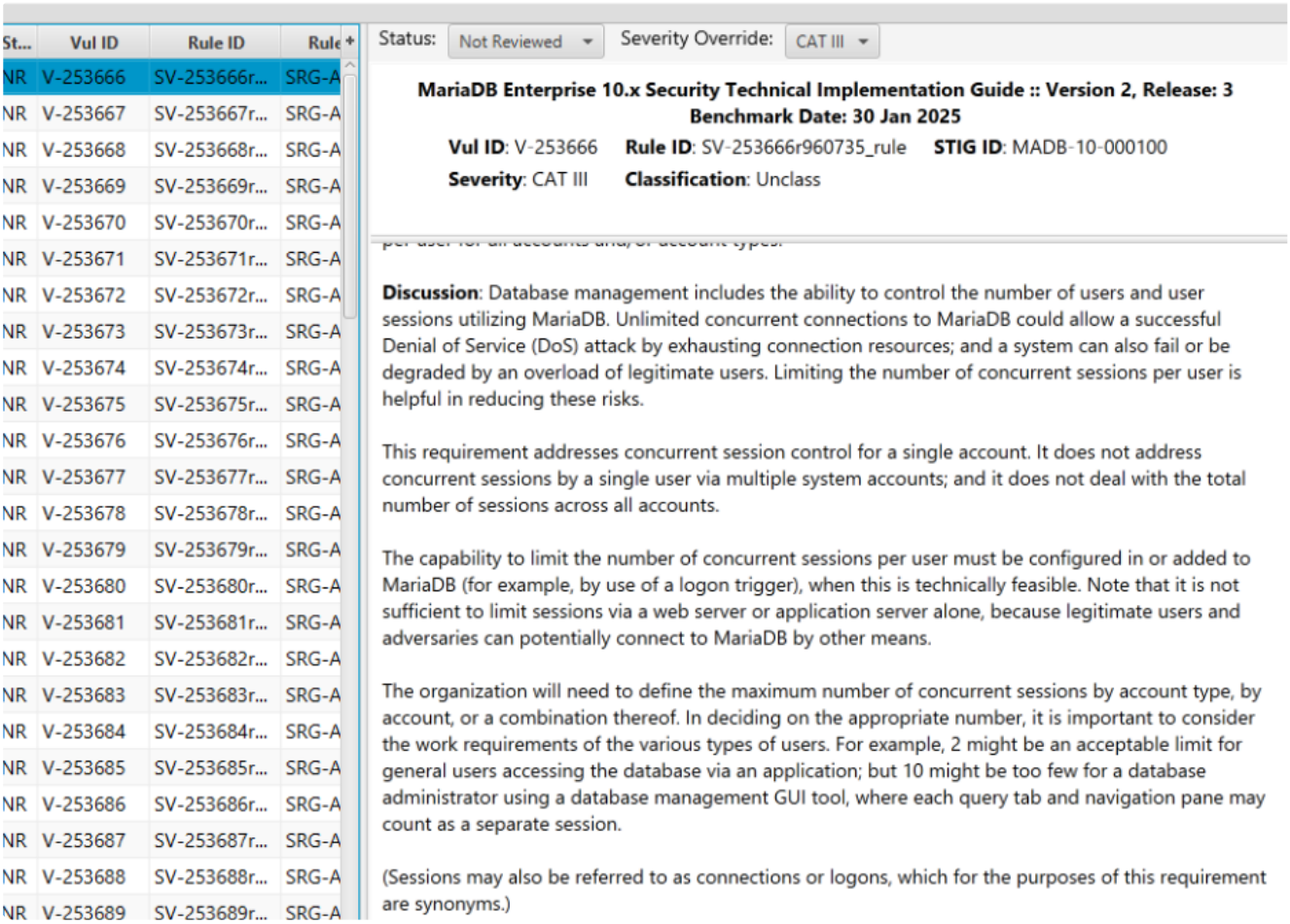
MARIADB SERVICE CONFIGURATION:

1. Connect to a hammer server.

2. Install MariaDB.

```
dnf install mariadb-server

# Ensure that it is running

systemctl start mariadb

systemctl status mariadb

ss -ntulp | grep 3306
```

• Check and remediate v-253666 STIG.



St...	Vul ID	Rule ID	Rule
NR	V-253666	SV-253666r...	SRG-A
NR	V-253667	SV-253667r...	SRG-A
NR	V-253668	SV-253668r...	SRG-A
NR	V-253669	SV-253669r...	SRG-A
NR	V-253670	SV-253670r...	SRG-A
NR	V-253671	SV-253671r...	SRG-A
NR	V-253672	SV-253672r...	SRG-A
NR	V-253673	SV-253673r...	SRG-A
NR	V-253674	SV-253674r...	SRG-A
NR	V-253675	SV-253675r...	SRG-A
NR	V-253676	SV-253676r...	SRG-A
NR	V-253677	SV-253677r...	SRG-A
NR	V-253678	SV-253678r...	SRG-A
NR	V-253679	SV-253679r...	SRG-A
NR	V-253680	SV-253680r...	SRG-A
NR	V-253681	SV-253681r...	SRG-A
NR	V-253682	SV-253682r...	SRG-A
NR	V-253683	SV-253683r...	SRG-A
NR	V-253684	SV-253684r...	SRG-A
NR	V-253685	SV-253685r...	SRG-A
NR	V-253686	SV-253686r...	SRG-A
NR	V-253687	SV-253687r...	SRG-A
NR	V-253688	SV-253688r...	SRG-A
NR	V-253689	SV-253689r...	SRG-A

Status: Not Reviewed Severity Override: CAT III

MariaDB Enterprise 10.x Security Technical Implementation Guide :: Version 2, Release: 3
Benchmark Date: 30 Jan 2025

Vul ID: V-253666 **Rule ID**: SV-253666r960735_rule **STIG ID**: MADB-10-000100
Severity: CAT III **Classification**: Unclass

Discussion: Database management includes the ability to control the number of users and user sessions utilizing MariaDB. Unlimited concurrent connections to MariaDB could allow a successful Denial of Service (DoS) attack by exhausting connection resources; and a system can also fail or be degraded by an overload of legitimate users. Limiting the number of concurrent sessions per user is helpful in reducing these risks.

This requirement addresses concurrent session control for a single account. It does not address concurrent sessions by a single user via multiple system accounts; and it does not deal with the total number of sessions across all accounts.

The capability to limit the number of concurrent sessions per user must be configured in or added to MariaDB (for example, by use of a logon trigger), when this is technically feasible. Note that it is not sufficient to limit sessions via a web server or application server alone, because legitimate users and adversaries can potentially connect to MariaDB by other means.

The organization will need to define the maximum number of concurrent sessions by account type, by account, or a combination thereof. In deciding on the appropriate number, it is important to consider the work requirements of the various types of users. For example, 2 might be an acceptable limit for general users accessing the database via an application; but 10 might be too few for a database administrator using a database management GUI tool, where each query tab and navigation pane may count as a separate session.

(Sessions may also be referred to as connections or logons, which for the purposes of this requirement are synonyms.)

Check Text: To check the number of connections allowed for each user, as the database administrator, run the following SQL:

```
MariaDB> SELECT user, max_user_connections FROM mysql.user;
```

If any users have more connections configured than documented, this is a finding. A value of 0 indicates unlimited and is a finding.

Fix Text: To limit the number of connections allowed by a specific user, as a user with appropriate privileges, run the following SQL:

```
MariaDB> GRANT USAGE ON *.* TO  'username'@'host'  WITH MAX_USER_CONNECTIONS number_of_connections;
```

- What is the problem?
- What is the fix?
- What type of control is being implemented?
- Is it set properly on your system?

Connect to MariaDB locally.

```
mysql
```

Run the SQL command in the STIG's Fix Text section:

```
SELECT user, max_user_connections FROM mysql.user;
```

```
MariaDB [(none)]>  SELECT user, max_user_connections FROM mysql.user;
+-------------+----------------------+
| User        | max_user_connections |
+-------------+----------------------+
| mariadb.sys |                    0 |
| root        |                    0 |
| mysql       |                    0 |
+-------------+----------------------+
3 rows in set (0.005 sec)
```

Can you remediate this finding?

```
MariaDB [(none)]> GRANT USAGE ON *.* TO  'username'@'host' WITH MAX_USER_CONNECTIONS 1
;
ERROR 1133 (28000): Can't find any matching row in the user table
MariaDB [(none)]> GRANT USAGE ON *.* TO  'root'@'localhost' WITH MAX_USER_CONNECTIONS
1;
Query OK, 0 rows affected (0.001 sec)

MariaDB [(none)]>  SELECT user, max_user_connections FROM mysql.user;
+-------------+----------------------+
| User        | max_user_connections |
+-------------+----------------------+
| mariadb.sys |                    0 |
| root        |                    1 |
| mysql       |                    0 |
+-------------+----------------------+
3 rows in set (0.002 sec)
```

- Check and remediate `v-253677 STIG`
  - What is the problem?
  - What is the fix?
  - What type of control is being implemented?
  - Is it set properly on your system?
- Check and remediate `v-253678 STIG`
  - What is the problem?
  - What is the fix?
  - What type of control is being implemented?
  - Is it set properly on your system?
- Check and remediate `v-253734 STIG`
  - What is the problem?
  - What is the fix?
  - What type of control is being implemented?
  - Is it set properly on your system?

Info

Be sure to `reboot` the lab machine from the command line when you are done.

3.3 Unit 2 - Securing the Network

3.3.1 Unit 2 - Securing the Network Connection

Overview

Understanding and implementing network standards and compliance measures can make security controls of critical importance very effective.

This unit introduces foundational knowledge on analyzing, configuring, and hardening networking components using tools and frameworks like STIGs, OpenSCAP, and DNS configurations.

Learning Objectives

By the end of Unit 2 students will have foundational knowledge and skills of the concepts below:

1. Identifying and analyzing STIGs related to Linux networking.
2. Understand and configure secure name resolution using nsswitch.conf and DNS.
3. Utilizing tools like tcpdump, ngrep, ss, and netstat to monitor network behavior.
4. Applying OpenSCAP and SCC tools for network compliance assessments.
5. Exploring known network-based exploits and understanding their anatomy via the Diamond Model of Intrusion Analysis.

Key Terms and Definitions

sysctl	**nsswitch.conf**
DNS	**Openscap**
CIS Benchmarks	**ss/netstat**
tcpdump	**ngrep**

3.3.2 Unit 2 Worksheet - Securing the Network Connection

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://docs.rockylinux.org/gemstones/core/view_kernel_conf/
- https://ciq.com/blog/demystifying-and-troubleshooting-name-resolution-in-rocky-linux/
- https://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u2_worksheet(`.txt`)
- 📥 u2_worksheet(`.docx`)

UNIT 2 RECORDING

Link: https://www.youtube.com/watch?v=x1kgXOWv-eM

Discussion Post #1

There are 401 stigs for RHEL 9. If you filter in your STIG viewer for `sysctl` there are 33 (mostly network focused), ssh - 39, and network - 58. Now there are some overlaps between those, but review them and answer these questions

1. As systems engineers why are we focused on protecting the network portion of our server builds?
2. Why is it important to understand all the possible ingress points to our servers that exist?
    - Why is it so important to understand the behaviors of processes that are connecting on those ingress points?

Discussion Post #2

Read this: https://ciq.com/blog/demystifying-and-troubleshooting-name-resolution-in-rocky-linux/ or similar blogs on DNS and host file configurations.

1. What is the significance of the nsswitch.conf file?
2. What are security problems associated with DNS and common exploits? (May have to look into some more blogs or posts for this)

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

sysctl:

nsswitch.conf:

DNS:

Openscap:

CIS Benchmarks:

ss/netstat:

tcpdump:

ngrep:

Digging Deeper

1. See if you can find any DNS exploits that have been used and written up in the diamond model of intrusion analysis format. If you can, what are the primary actors and actions that made up the attack?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

3.3.3 Unit 2 Lab - Securing the Network Connection

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.docx` could be transposed to a `.md` file.

- 📥 u2_lab(`.txt`)
- 📥 u2_lab(`.docx`)

Pre-Lab Warm-Up

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

```
sysctl -a | grep -i ipv4 | grep -i forward
# Does this system appear to be set to forward? Why or why not?

sysctl -a | grep -i ipv4 | grep -i martian
# What are martians and is this system allowing them?

sysctl -a | grep -i panic
# How does this system handle panics?

sysctl -a | grep -i crypto
# What are the settings you see? Is FIPS enabled?

cat /proc/cmdline
fips-mode-setup --check
sestatus
cat /etc/selinux/config
```

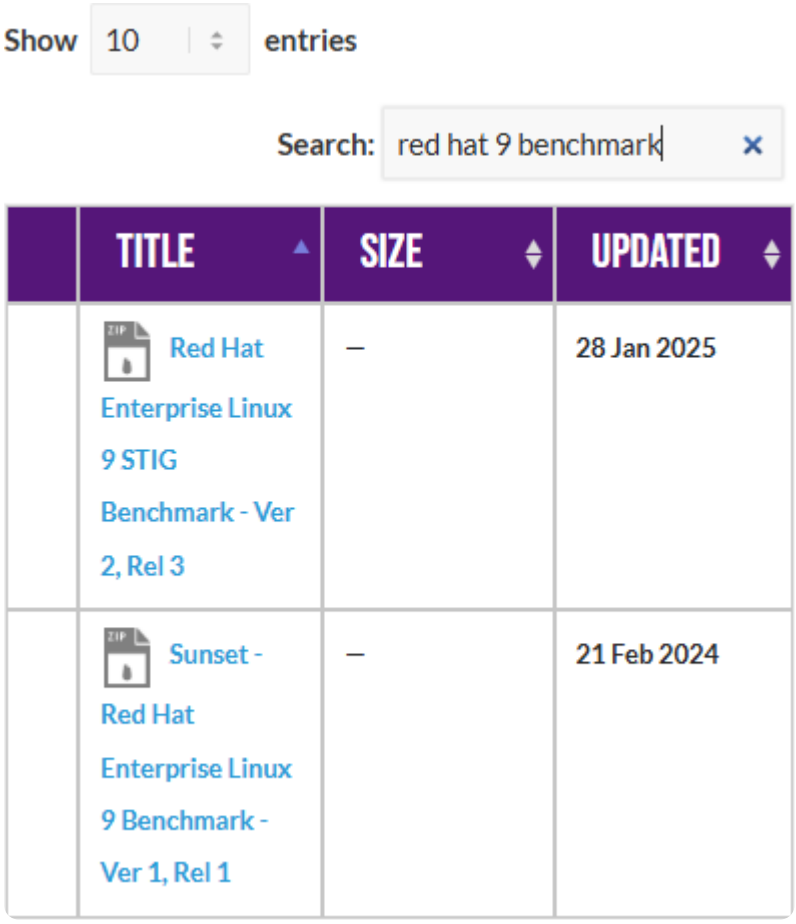
What information about the security posture of the system can you see here?
Can you verify SELINUX status?
Can you verify FIPS status?

Download the STIG Viewer 2.18 from - https://public.cyber.mil/stigs/downloads/

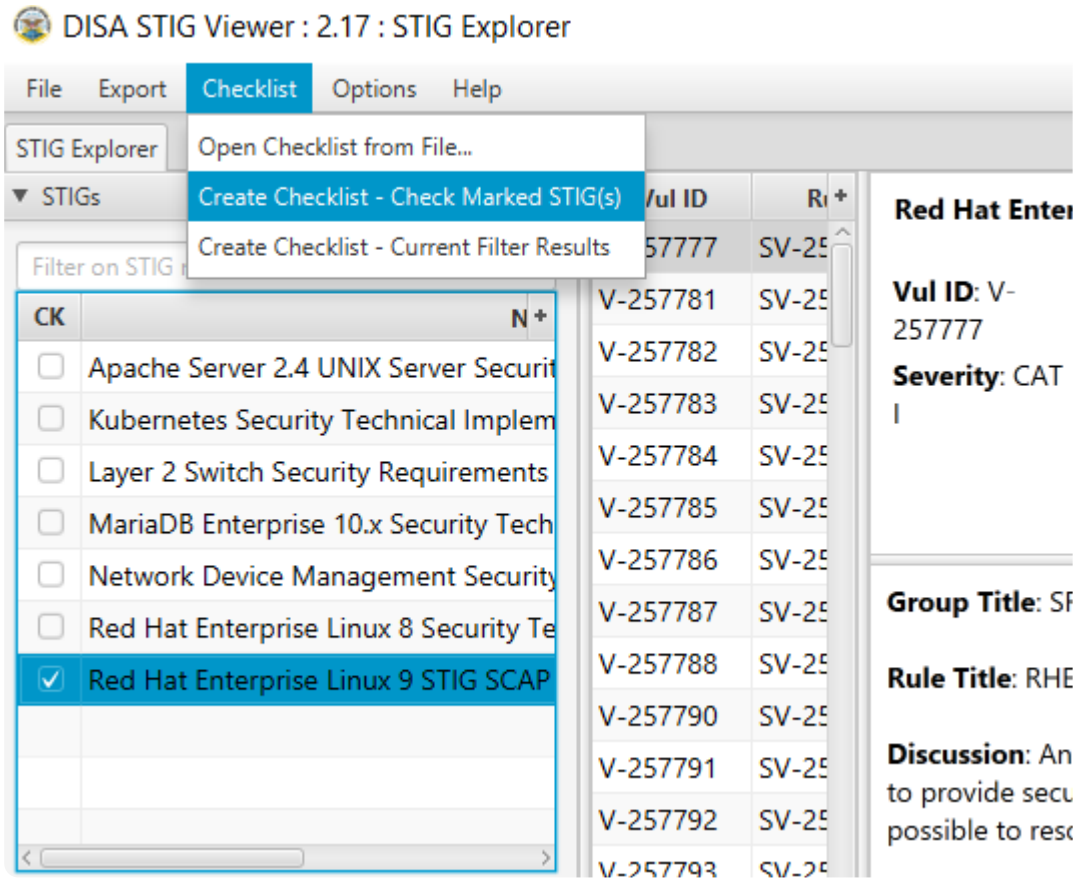
Show 10 entries Search: viewer

	TITLE	SIZE	UPDATED
	STIG Viewer 2.18	–	12 Aug 2024
	STIG Viewer 2.18 Hashes	–	12 Aug 2024
	STIG Viewer 2.18-Linux	–	12 Aug 2024
	STIG Viewer 2.18-Win64	–	12 Aug 2024
	STIG Viewer 2.18-Win64 msi	–	12 Aug 2024
	Stig Viewer 3 CKLB JSON Schema	–	10 Jan 2024
	STIG Viewer 3.5 Hashes	–	19 Feb 2025
	STIG Viewer 3.5-Linux	–	19 Feb 2025
	STIG Viewer 3.5-Win64	–	19 Feb 2025
	STIG Viewer 3.5-Win64 msi	–	19 Feb 2025

Download the STIG for RHEL 9 and the import it into your STIG viewer



Create a checklist from the opened STIG for RHEL 9

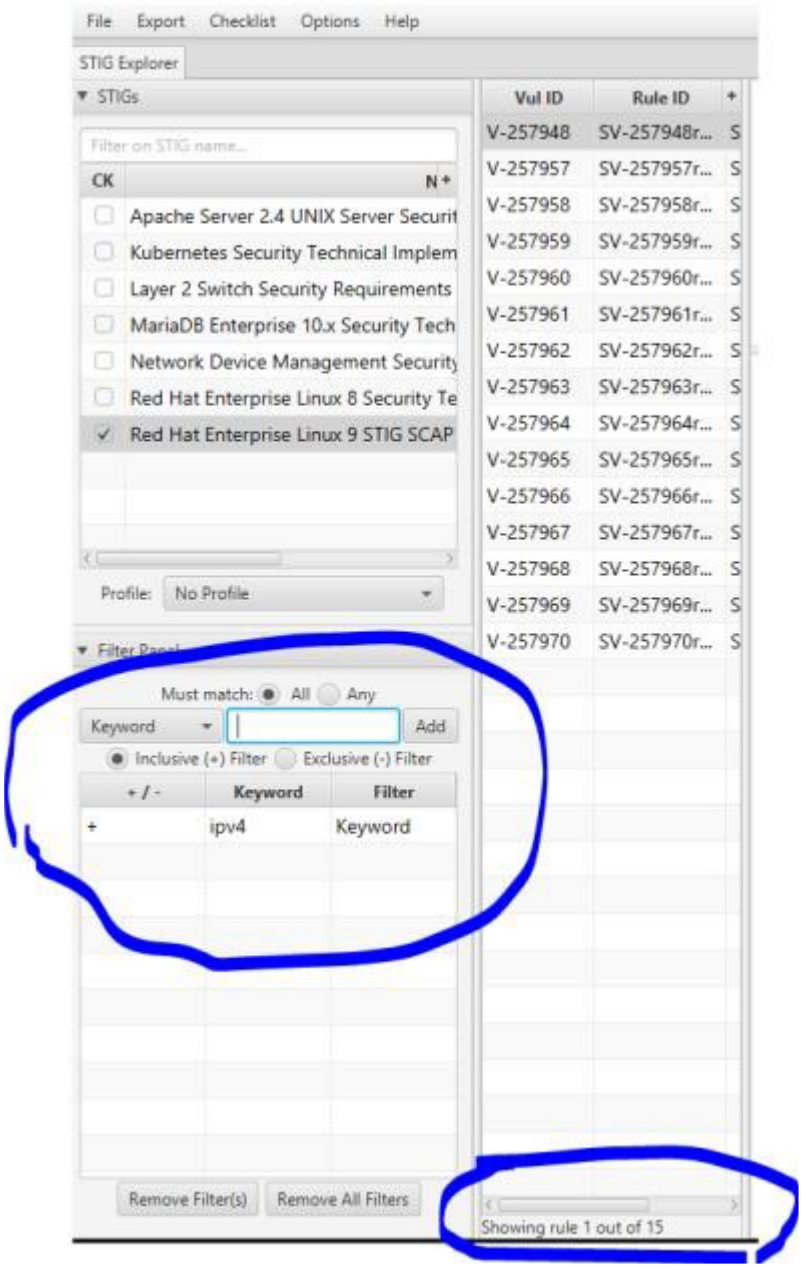


Lab 🧪

This lab is designed to have the engineer practice securing a Linux server or service against a set of configuration standards. These standards are sometimes called benchmarks, checklists, or guidelines. The engineer will be using STIG Viewer 2.18 to complete this lab.

NETWORK SERVICE CONFIGURATION

Connect to a hammer server
Filter by ipv4 and see how many STIGs you have.



Examine STIG V-257957

What is the problem?
What is the fix?
What type of control is being implemented?
Is it set properly on your system?

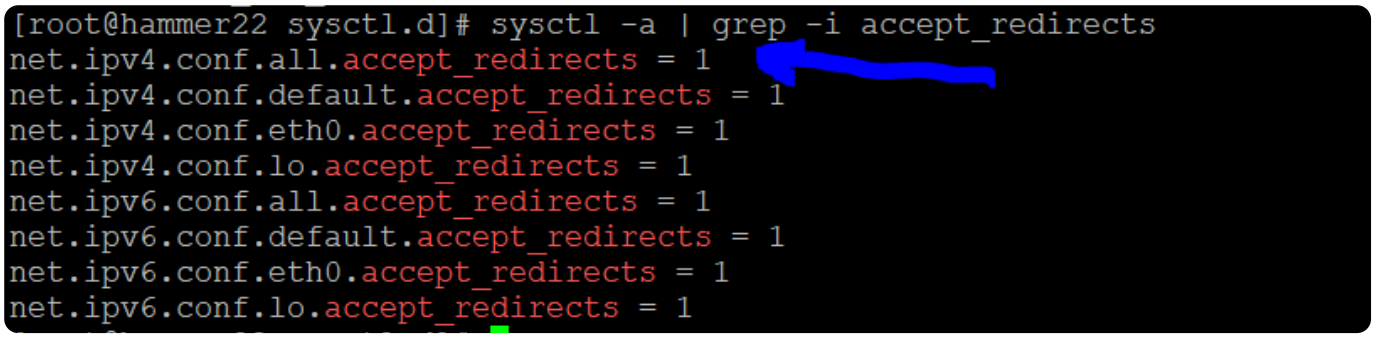
```
sysctl -a | grep -i ipv4 | grep -i syncookies
```

```
[root@hammer22 ~]# sysctl -a | grep -i ipv4 | grep -i syncookies
net.ipv4.tcp_syncookies = 1
```

Can you remediate this finding?
In this case it's already correctly set.
But if we needed to, we would set that value in /etc/sysctl.d/00- remediate.conf
And then reload sysctl with `sysctl --system`

Check and remediate V-257958 STIG

What is the problem?
What is the fix?
What type of control is being implemented?
Is it set properly on your system?



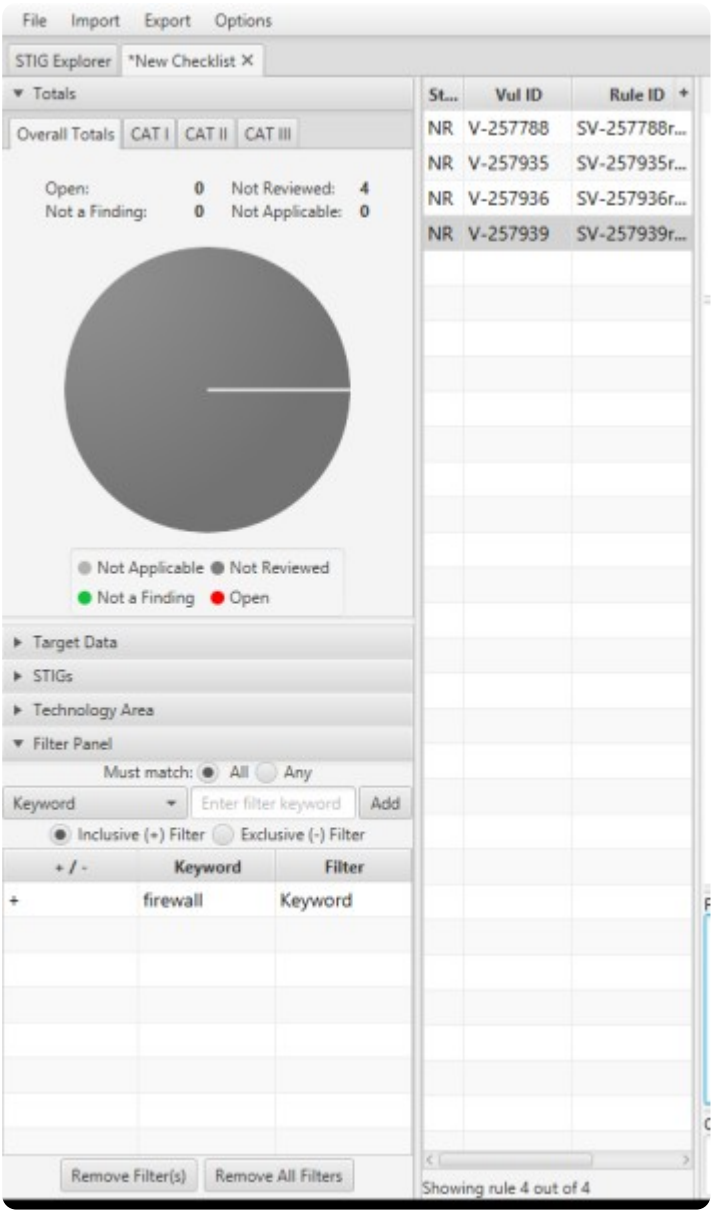
How would you go about remediating this on your system?

Check and remediate V-257960 and V-257961 STIGs

What is the problem? How are they related?
What is the fix?
What type of control is being implemented?
Is it set properly on your system?

Filter by firewall

How many STIGS do you see?



What do these STIGS appear to be trying to do? What types of controls are they?

FIREWALL PORT EXPOSURE

Scenario:

> Your team needs to use node_exporter with Prometheus to allow scraping of system information back to your network monitoring solution. You are running a firewall, so you need to expose the port that node_exporter runs on to the network outside of your system.

Expose a network port through your firewall

```
# Verify that your firewall is running
systemctl status firewalld

# Verify that your firewall has the service defined
firewall-cmd --get-services | grep -i node
ls /usr/lib/firewalld/services | grep -i node

# Verify that the service is not currently enabled for node_exporter
firewall-cmd --list-services

# Examine the structure of the firewall .xml file
cat /usr/lib/firewalld/services/prometheus-node-exporter.xml

# Enable the service through your firewall
firewall-cmd --permanent --add-service=prometheus-node-exporter

# Reload so the changes take effect
firewall-cmd --reload

# Verify that the service is currently enabled for node_exporter
firewall-cmd --list-services
```

AUTOMATE STIG REMEDIATION ON A SYSTEM

There are many options and the STIG remediation steps are well known. Here the learner will examine a few ways to generate Ansible and Shell fixes to your system. Then one can apply all of them, or just some of them. This is the real value of a security engineer focused Linux engineer, the trade-off between security and productivity.

Download and extract a STIG remediation tool

Note: If any lab download does not work, check the `/labs` folder on the server for a `[course]_[unit#].zip` file to complete the activities.

```
cd /root
mkdir stigs
cd stigs
wget -O U_RHEL_9_V2R4_STIG_Ansible.zip https://dl.dod.cyber.mil/wp-content/uploads/stigs/zip/U_RHEL_9_V2R4_STIG_Ansible.zip
unzip U_RHEL_9_V2R4_STIG_Ansible.zip
mkdir ansible
cp rhel9STIG-ansible.zip ansible/
cd ansible
unzip rhel9STIG-ansible.zip
```

Examine the default values for STIGS

```
cd /root/stigs/ansible/roles/rhel9STIG/defaults/
vim main.yml
```

Search for a few of the STIG numbers you used earlier and see their default values.

- use /257784 to search

Examine the playbook to see how those are applied in a running system.

```
vim /root/stigs/ansible/roles/rhel9STIG/tasks/main.yml
```

- use /257784 to search for the STIG from above and see how it is fixed in the playbook.

Create an Ansible playbook from OpenSCAP

```
dnf -y install openscap-scanner openscap-utils openscap-scanner scap-security-guide
cd /root
mkdir openscap
cd openscap

# Generate the Ansible
oscap xccdf generate fix --profile ospp --fix-type ansible /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml > draft-disa-remediate.yml

# Examine the file
vim draft-disa-remediate.yml

# Generate a BASH version
oscap xccdf generate fix --profile ospp --fix-type bash /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml > draft-disa-remediate.sh

# Examine the file
vim draf-disa-remediate.sh
```

Info

Be sure to `reboot` the lab machine from the command line when you are done.

3.4 Unit 3 - User Access and System Integration

3.4.1 Unit 3 - User Access and System Integration

Overview

User access in an larger organizations requires more sophisticated controls. For this purpose Active Directory (AD) and Lightweight Directory Access Protocol (LDAP) have become popular choices as they offer more sophisticated and robust ways of controlling access. In this chapter, you will learn why AD and LDAP are popular choices.

Learning Objectives

1. Understand how LDAP or AD works and why it is beneficial.
2. High level understanding of hardening Rocky Linux, a RHEL adjacent distro.
3. Gain a basic understanding of PAM.

Key Terms and Definitions

PAM	**AD**
LDAP	**sssd**
oddjob	**krb5**
realm/realmd	**wheel (system group in RHEL)**

3.4.2 Unit 3 Worksheet - User Access and System Integration

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://docs.rockylinux.org/guides/security/pam/
- https://docs.rockylinux.org/guides/security/authentication/active_directory_authentication/
- https://docs.rockylinux.org/books/admin_guide/06-users/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u3_worksheet(`.txt`)
- 📥 u3_worksheet(`.pdf`)

UNIT 3 RECORDING

Link: https://www.youtube.com/watch?v=-4FhZ2q4RSo

Discussion Post #1

There are 16 Stigs that involve PAM for RHEL 9. Read the guide from Rocky Linux here: https://docs.rockylinux.org/guides/security/pam/

1. What are the mechanisms and how do they affect PAM functionality?
    - Review `/etc/pam.d/sshd` on a Linux system.
      What is happening in that file relative to these functionalities?
2. What are the common PAM modules?
    - Review `/etc/pam.d/sshd` on a Linux system.
      What is happening in that file relative to these functionalities?
3. Look for a blog post or article about PAM that discusses real world application.
   Post it here and give us a quick synopsis. (Bonus arbitrary points if you find one of our ProLUG members blogs on the subject.)

Discussion Post #2

Read about active directory (or LDAP) configurations of Linux via `sssd` here: https://docs.rockylinux.org/guides/security/authentication/active_directory_authentication

1. Why do we not want to just use local authentication in Linux? Or really any system?
2. There are 4 SSSD STIGS.
    - What are they?
    - What do they seek to do with the system?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

PAM:

AD:

LDAP:

sssd:

oddjob:

krb5:

realm/realmd:

wheel (system group in RHEL):

Digging Deeper

1. How does `/etc/security/access.conf` come into play with pam_access? Read up on it here: https://man7.org/linux/man-pages/man8/pam_access.8.html
    - Can you find any other good resources?
    - What is the structure of the access.conf file directives?
2. What other important user access or user management information do you learn by reading this? https://docs.rockylinux.org/books/admin_guide/06-users/
    - What is the contents of the `/etc/login.defs` file? Why do you care?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

3.4.3 Unit 3 Lab - User Access and System Integration

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.docx` could be transposed to a `.md` file.

- 📥 u3_lab(`.pdf`)
- 📥 u3_lab(`.docx`)

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

```
ls -l /etc/pam.d/
# What are the permissions and names of files? Can everyone read them?

cat /etc/pam.d/sshd

# What information do you see in this file?
# Does any of it look familiar to you?
```

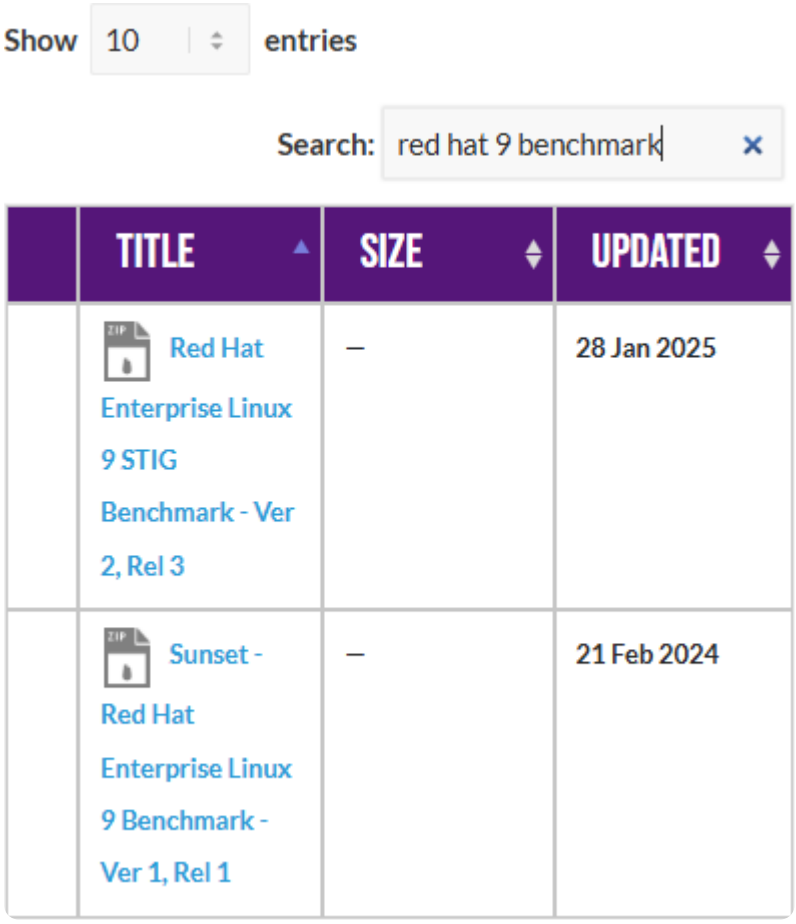
Pre-Lab Warm-Up

Download the STIG Viewer 2.18 from - https://public.cyber.mil/stigs/downloads/

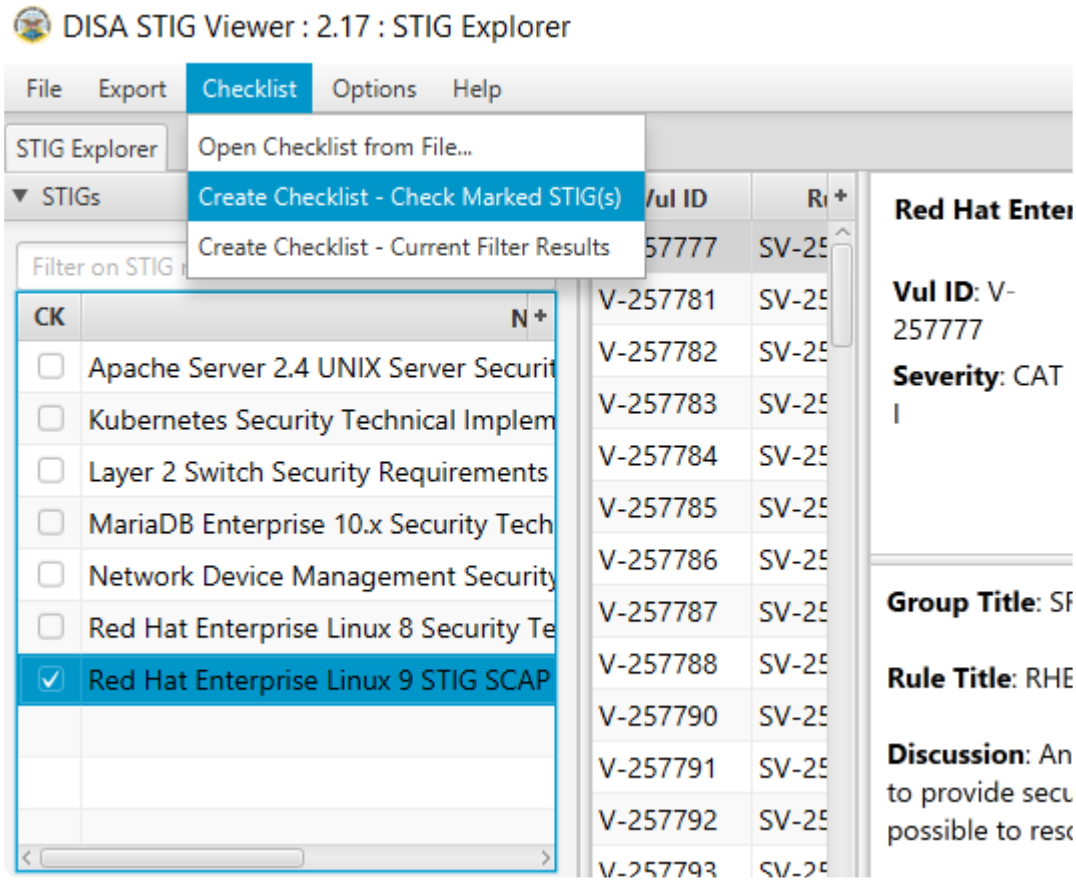
Show 10 entries Search: viewer

	TITLE	SIZE	UPDATED
	STIG Viewer 2.18	–	12 Aug 2024
	STIG Viewer 2.18 Hashes	–	12 Aug 2024
	STIG Viewer 2.18-Linux	–	12 Aug 2024
	STIG Viewer 2.18-Win64	–	12 Aug 2024
	STIG Viewer 2.18-Win64 msi	–	12 Aug 2024
	Stig Viewer 3 CKLB JSON Schema	–	10 Jan 2024
	STIG Viewer 3.5 Hashes	–	19 Feb 2025
	STIG Viewer 3.5-Linux	–	19 Feb 2025
	STIG Viewer 3.5-Win64	–	19 Feb 2025
	STIG Viewer 3.5-Win64 msi	–	19 Feb 2025

Download the STIG for RHEL 9 and the import it into your STIG viewer



Create a checklist from the opened STIG for RHEL 9

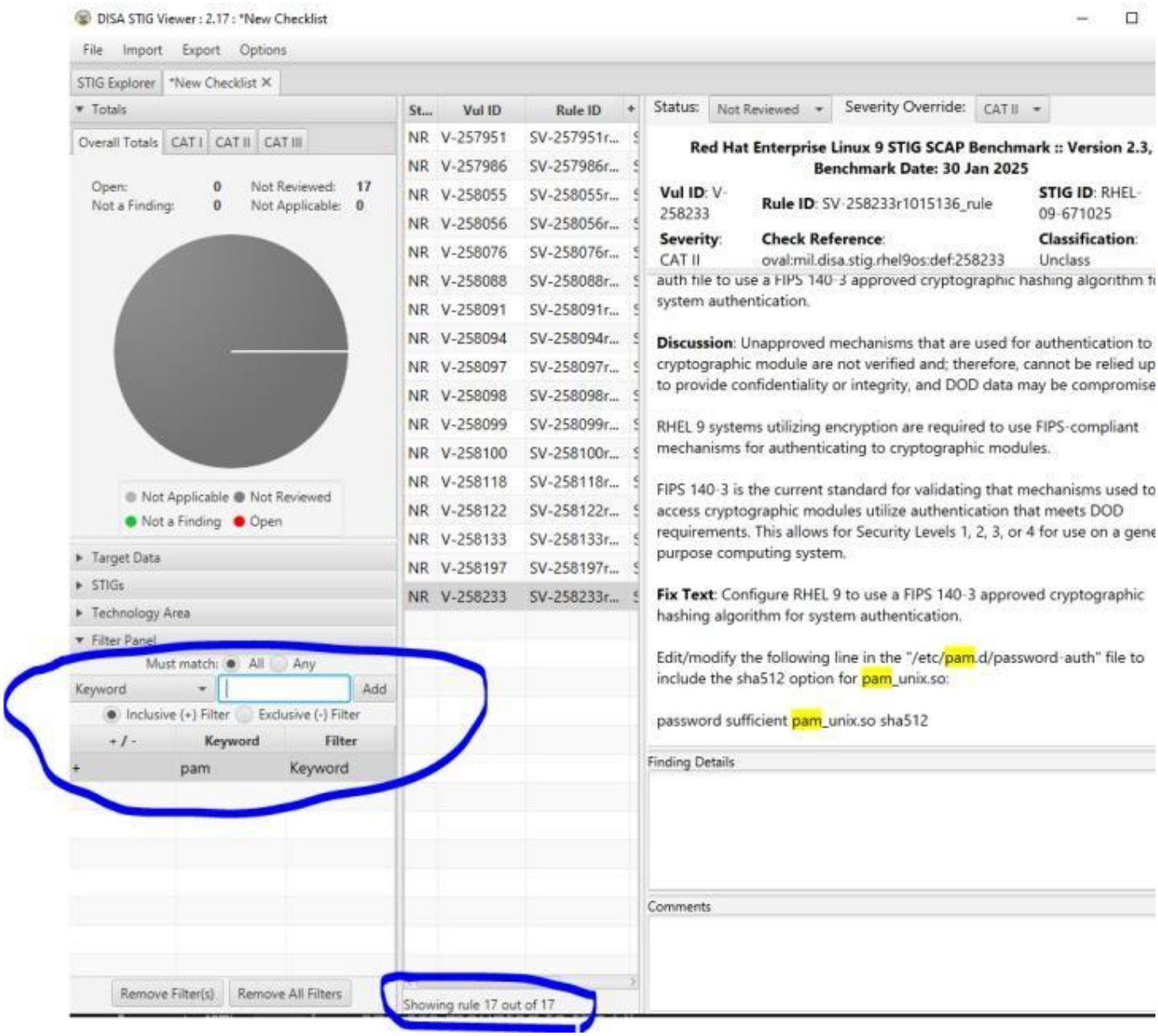


Lab 🧪

This lab is designed to have the engineer practice securing a Linux server or service against a set of configuration standards. These standards are sometimes called benchmarks, checklists, or guidelines. The engineer will be using STIG Viewer 2.18 to complete this lab.

PAM CONFIGURATION

Connect to a hammer server
Filter by pam and see how many STIGS you have. (Why is it really only 16?)



Examine STIG V-257986

What is the problem?
What is the fix?
What type of control is being implemented?
Is it set properly on your system?

```
grep -i pam /etc/ssh/sshd_config
```

```
[root@hammer1 ~]# grep -i pam /etc/ssh/sshd_config
# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# PasswordAuthentication.  Depending on your PAM configuration,
# PAM authentication via KbdInteractiveAuthentication may bypass
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# WARNING: 'UsePAM no' is not supported in RHEL and may cause several
#UsePAM no
```

Can you remediate this finding?

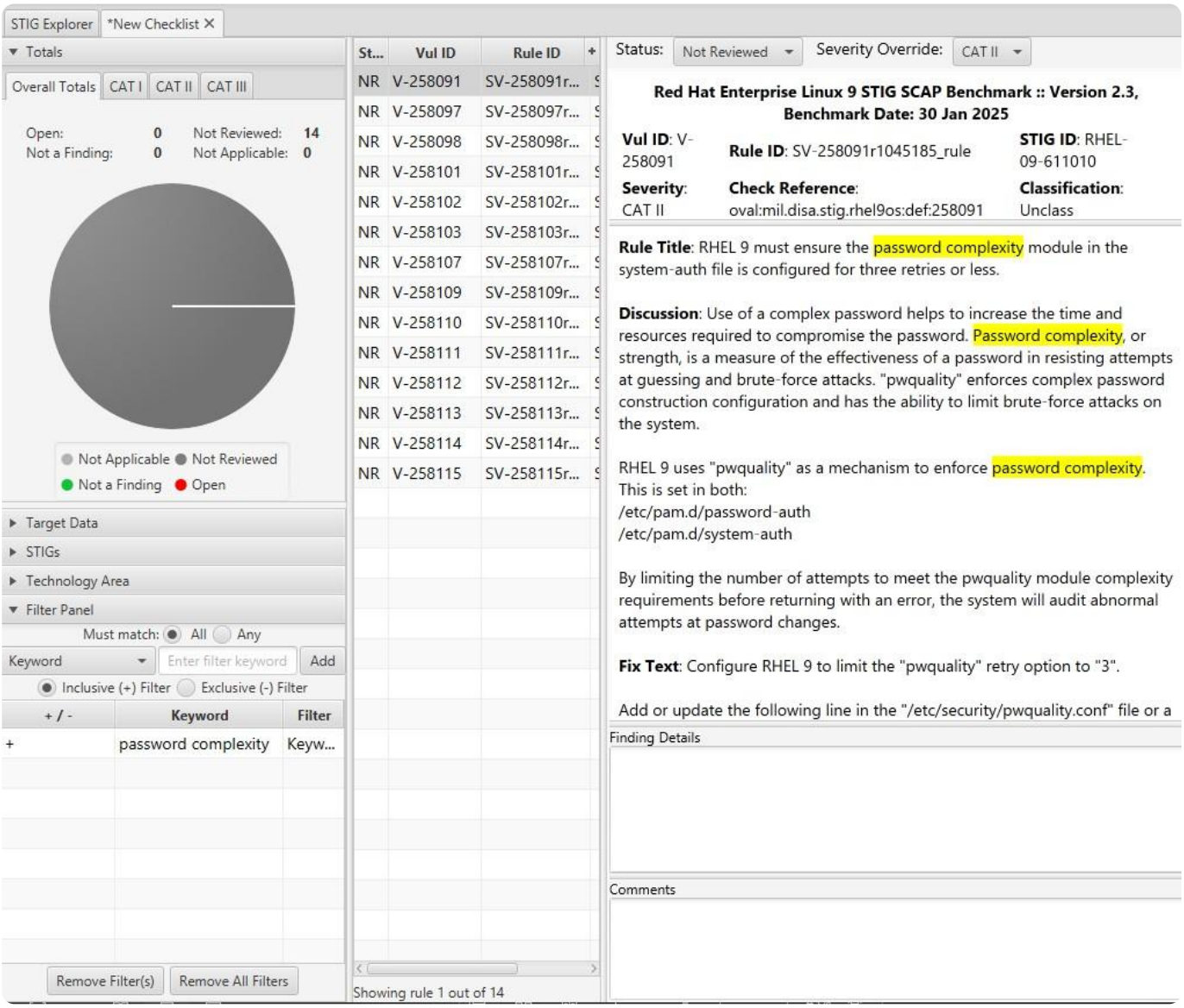
Check and remediate STIG V-258055

What is the problem?
What is the fix?
What type of control is being implemented?
Are there any major implications to think about with this change on your system? Why or why not?
Is it set properly on your system?
How would you go about remediating this on your system?

Check and remediate STIG V-258098

What is the problem?
What is the fix?
What type of control is being implemented?
Is it set properly on your system?

Filter by "password complexity"



How many are there?
What are the password complexity rules?
Are there any you haven't seen before?

Filter by sssd

How many STIGS do you see?
What do these STIGS appear to be trying to do? What types of controls are they?

OPENLDAP SETUP

You will likely not build an LDAP server in a real world environment. We are doing it for understanding and ability to complete the lab. In a normal corporate environment this is likely Active Directory.

To simplify some of the typing in this lab, there is a file located at `/lab_work/identity_and_access_management.tar.gz` that you can pull down to your system with the correct `.ldif` files.

```
cp /lab_work/identity_and_access_management.tar.gz .
tar -xzvf identity_and_access_management.tar.gz
```

INSTALL AND CONFIGURE OPENLDAP

1. Stop the warewulf client

```
systemctl stop wwclient
```

2. Edit your /etc/hosts file

Look for and edit the line that has your current server

```
vi /etc/hosts
```

Entry for hammer1 for example:
`192.168.200.151 hammer1 hammer1-default ldap.prolug.lan ldap`

3. Setup dnf repo

```
dnf config-manager --set-enabled plus
dnf repolist
dnf -y install openldap-servers openldap-clients openldap
```

4. Start slapd systemctl

```
systemctl start slapd
ss -ntulp | grep slapd
```

5. Allow ldap through the firewall

```
firewall-cmd --add-service={ldap,ldaps} --permanent
firewall-cmd --reload
firewall-cmd --list-all
```

6. Generate a password

(Our example uses `testpassword`) This will return a salted SSHA password. *Save this password and salted hash for later input*

```
slappasswd
```

Output:

> New password: Re-enter new password: {SSHA}wpRvODvIC/EPYf2GqHUlQMDdsFIW5yig

7. Change the password

```
vi changerootpass.ldif
```

```
dn: olcDatabase={0}config,cn=config
changetype: modify
```

```
replace: olcRootPW
olcRootPW: {SSHA}vKobSZO1HDGxp2OElzli/xfAzY4jSDMZ
```

```
ldapadd -Y EXTERNAL -H ldapi:/// -f changerootpass.ldif
```

Output:

> SASL/EXTERNAL authentication started SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth SASL SSF: 0 modifying entry "olcDatabase={0}config,cn=config"

8. Generate basic schemas

```
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/cosine.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/nis.ldif
ldapadd -Y EXTERNAL -H ldapi:/// -f /etc/openldap/schema/inetorgperson.ldif
```

9. Set up the domain

(USE THE PASSWORD YOU GENERATED EARLIER)

```
vi setdomain.ldif
```

```
dn: olcDatabase={1}monitor,cn=config
changetype: modify
replace: olcAccess
olcAccess: {0}to * by dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth"
read by dn.base="cn=Manager,dc=prolug,dc=lan" read by * none

dn: olcDatabase={2}mdb,cn=config
changetype: modify
replace: olcSuffix
olcSuffix: dc=prolug,dc=lan

dn: olcDatabase={2}mdb,cn=config
changetype: modify
replace: olcRootDN
olcRootDN: cn=Manager,dc=prolug,dc=lan

dn: olcDatabase={2}mdb,cn=config
changetype: modify
add: olcRootPW
olcRootPW: {SSHA}s4x6uAxcAPZN/4e3pGnU7UEIiADY0/Ob

dn: olcDatabase={2}mdb,cn=config
changetype: modify
add: olcAccess
olcAccess: {0}to attrs=userPassword,shadowLastChange by
dn="cn=Manager,dc=prolug,dc=lan" write by anonymous auth by self write by * none
olcAccess: {1}to dn.base="" by * read
olcAccess: {2}to * by dn="cn=Manager,dc=prolug,dc=lan" write by * read
```

10. Run it

```
ldapmodify -Y EXTERNAL -H ldapi:/// -f setdomain.ldif
```

Output:

> SASL/EXTERNAL authentication started SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth SASL SSF: 0 modifying entry "olcDatabase={1}monitor,cn=config" modifying entry "olcDatabase={2}mdb,cn=config" modifying entry "olcDatabase={2}mdb,cn=config" modifying entry "olcDatabase={2}mdb,cn=config" modifying entry "olcDatabase={2}mdb,cn=config"

11. Search and verify the domain is working.

```
ldapsearch -H ldap:// -x -s base -b "" -LLL "namingContexts"
```

Output:

> dn: namingContexts: dc=prolug,dc=lan

12. Add the base group and organization.

```
vi addou.ldif
```

```
dn: dc=prolug,dc=lan
objectClass: top
objectClass: dcObject
objectclass: organization
o: My prolug Organisation
dc: prolug

dn: cn=Manager,dc=prolug,dc=lan
objectClass: organizationalRole
cn: Manager
description: OpenLDAP Manager

dn: ou=People,dc=prolug,dc=lan
objectClass: organizationalUnit
ou: People

dn: ou=Group,dc=prolug,dc=lan
objectClass: organizationalUnit
ou: Group
```

```
ldapadd -x -D cn=Manager,dc=prolug,dc=lan -W -f addou.ldif
```

13. Verifying

```
ldapsearch -H ldap:// -x -s base -b "" -LLL "+"
ldapsearch -x -b "dc=prolug,dc=lan" ou
```

14. Add a user

Generate a password (use testuser1234)

```
slappasswd
```

```
vi adduser.ldif
```

```
dn: uid=testuser,ou=People,dc=prolug,dc=lan
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
cn: testuser
sn: temp
userPassword: {SSHA}yb6e0ICSdlZaMef3zizvysEzXRGozQOK
loginShell: /bin/bash
uidNumber: 15000
gidNumber: 15000
homeDirectory: /home/testuser
shadowLastChange: 0
shadowMax: 0
shadowWarning: 0

dn: cn=testuser,ou=Group,dc=prolug,dc=lan
objectClass: posixGroup
cn: testuser
gidNumber: 15000
memberUid: testuser
```

```
ldapadd -x -D cn=Manager,dc=prolug,dc=lan -W -f adduser.ldif
```

16. Verify that your user is in the system.

```
ldapsearch -x -b "ou=People,dc=prolug,dc=lan"
```

17. Secure the system with TLS (accept all defaults)

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/pki/tls/ldapserver.key -out /etc/pki/tls/ldapserver.crt
chown ldap:ldap /etc/pki/tls/{ldapserver.crt,ldapserver.key}
```

```
ls -l /etc/pki/tls/ldap*
```

Output:

> -rw-r--r--. 1 ldap ldap 1224 Apr 12 18:23 /etc/pki/tls/ldapserver.crt -rw-------. 1 ldap ldap 1704 Apr 12 18:22 /etc/pki/tls/ldapserver.key

```
vi tls.ldif
```

```
dn: cn=config
changetype: modify
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/pki/tls/ldapserver.crt

add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/pki/tls/ldapserver.key

add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/pki/tls/ldapserver.crt
```

```
ldapadd -Y EXTERNAL -H ldapi:/// -f tls.ldif
```

18. Fix the /etc/openldap/ldap.conf to allow for certs

```
vi /etc/openldap/ldap.conf
```

```
#
# LDAP Defaults
#

# See ldap.conf(5) for details
# This file should be world readable but not world writable.

#BASE dc=example,dc=com
#URI ldap://ldap.example.com ldap://ldap-master.example.com:666

#SIZELIMIT 12
#TIMELIMIT 15
#DEREF never

# When no CA certificates are specified the Shared System Certificates
# are in use. In order to have these available along with the ones specified # by TLS_CACERTDIR one has to include them explicitly:

TLS_CACERT /etc/pki/tls/ldapserver.crt
TLS_REQCERT never

# System-wide Crypto Policies provide up to date cipher suite which should
# be used unless one needs a finer grinded selection of ciphers. Hence, the
# PROFILE=SYSTEM value represents the default behavior which is in place
# when no explicit setting is used. (see openssl-ciphers(1) for more info)
#TLS_CIPHER_SUITE PROFILE=SYSTEM

# Turning this off breaks GSSAPI used with krb5 when rdns = false
SASL_NOCANON on
```

```
systemctl restart slapd
```

SSSD CONFIGURATION AND REALMD JOIN TO LDAP

SSSD can connect a server to a trusted LDAP system and authenticate users for access to local resources. You will likely do this during your career and it is a valuable skill to work with.

1. Install sssd, configure, and validate that the user is seen by the system

```
dnf install openldap-clients sssd sssd-ldap oddjob-mkhomedir authselect
authselect select sssd with-mkhomedir --force
systemctl enable --now oddjobd.service
systemctl status oddjobd.service
```

2. Uncomment and fix the lines in /etc/openldap/ldap.conf

```
vi /etc/openldap/ldap.conf
```

Output:

> BASE dc=prolug,dc=lan URI ldap://ldap.ldap.lan/

3. Edit the sssd.conf file

```
vi /etc/sssd/sssd.conf
```

```
[domain/default]
id_provider = ldap
autofs_provider = ldap
auth_provider = ldap
chpass_provider = ldap
ldap_uri = ldap://ldap.prolug.lan/
ldap_search_base = dc=prolug,dc=lan
#ldap_id_use_start_tls = True
#ldap_tls_cacertdir = /etc/openldap/certs
cache_credentials = True
#ldap_tls_reqcert = allow

[sssd]
services = nss, pam, autofs
domains = default

[nss]
homedir_substring = /home
```

```
chmod 0600 /etc/sssd/sssd.conf
systemctl start sssd
systemctl status sssd
```

4. Validate that the user can be seen

```
id testuser
```

Output:

> uid=15000(testuser) gid=15000 groups=15000

Congratulations! Look at you, doing all the Linux.

```
reboot
```

Info

Be sure to `reboot` the lab machine from the command line when you are done.

3.5 Unit 4 - Bastion Hosts and Airgaps

3.5.1 Bastion Hosts & Air-Gaps

Overview

Bastions and airgaps are strategies for controlling how systems connect—or don't connect—to the outside world. They focus on limiting exposure, creating strong boundaries that support a broader security design. In this unit, we look at how we can seperate systems and create safe disconnects should a problem arise.

Learning Objectives

1. Understand the role and importance of air-gapped systems.
2. Recognize how to balance strong security with operational efficiency.
3. Learn how bastion hosts can help control and limit system access.
4. Understand methods for automating the jailing and restriction of users.
5. Gain a foundational understanding of `chroot` environments and diversion techniques.

Key Terms and Definitions

Air-gapped	**Bastion**
Jailed process	**Isolation**
Ingress	**Egress**
Exfiltration	**Cgroups**
Namespaces - Mount, PID, IPC, UTS	

3.5.2 Unit 4 Worksheet - Bastions and Jailing Users

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://aws.amazon.com/search/?searchQuery=air+gapped#facet_type=blogs&page=1
- https://aws.amazon.com/blogs/security/tag/bastion-host/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u4_worksheet(`.txt`)
- 📥 u4_worksheet(`.pdf`)

UNIT 4 RECORDING

Link: https://www.youtube.com/watch?v=dnz92v71Tr4

Discussion Post #1

Review some of the blogs here:

- https://aws.amazon.com/search/?searchQuery=air+gapped#facet_type=blogs&page=1
- https://aws.amazon.com/blogs/security/tag/bastion-host/

Or find some on your own about air-gapped systems.

1. What seems to be the theme of air-gapped systems?
2. What seems to be their purpose?
3. If you use google, or an AI, what are some of the common themes that come up when asked about air-gapped or bastion systems?

Discussion Post #2

Do a Google or AI search of topics around jailing a user or processes in Linux.

1. Can you enumerate the methods of jailing users?
2. Can you think of when you've been jailed as a Linux user?
   If not, can you think of the useful ways to use a jail?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Air-gapped

Bastion

Jailed process

Isolation

Ingress

Egress

Exfiltration

Cgroups

Namespaces

- Mount
- PID
- IPC
- UTS

Digging Deeper

1. While this isn't, strictly speaking, an automation course there is some value in looking at automation of the bastion deployments. Check out this ansible code:
   https://github.com/het-tanis/stream_setup/blob/master/roles/bastion_deploy/tasks/main.yml
   - Does the setup make sense to you with our deployment?
   - What can improve and make this better?
2. Find a blog or github where someone else deploys a bastion. Compare it to our process.
3. Knowing what you now know about bastions, jails, and air-gapped systems. Reflect on the first 3 weeks, all the STIGs you've reviewed and touched. Do any of them seem moot, or less necessary if applied in an air-gapped environment?
   - Does your answer change if you read about Zero Trust and know how much of a hot topic that is in the security world now?
     a. Why or why not?
4. Think of a Linux system where you would like to deploy a bastion (If you cannot think of one, use ProLUG Lab). Draw out how you think the system works in excalidraw.com.

Reflection Questions

1. Does it matter if the user knows that they are jailed? Why or why not?
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.5.3 Unit 4 Lab - Bastions

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

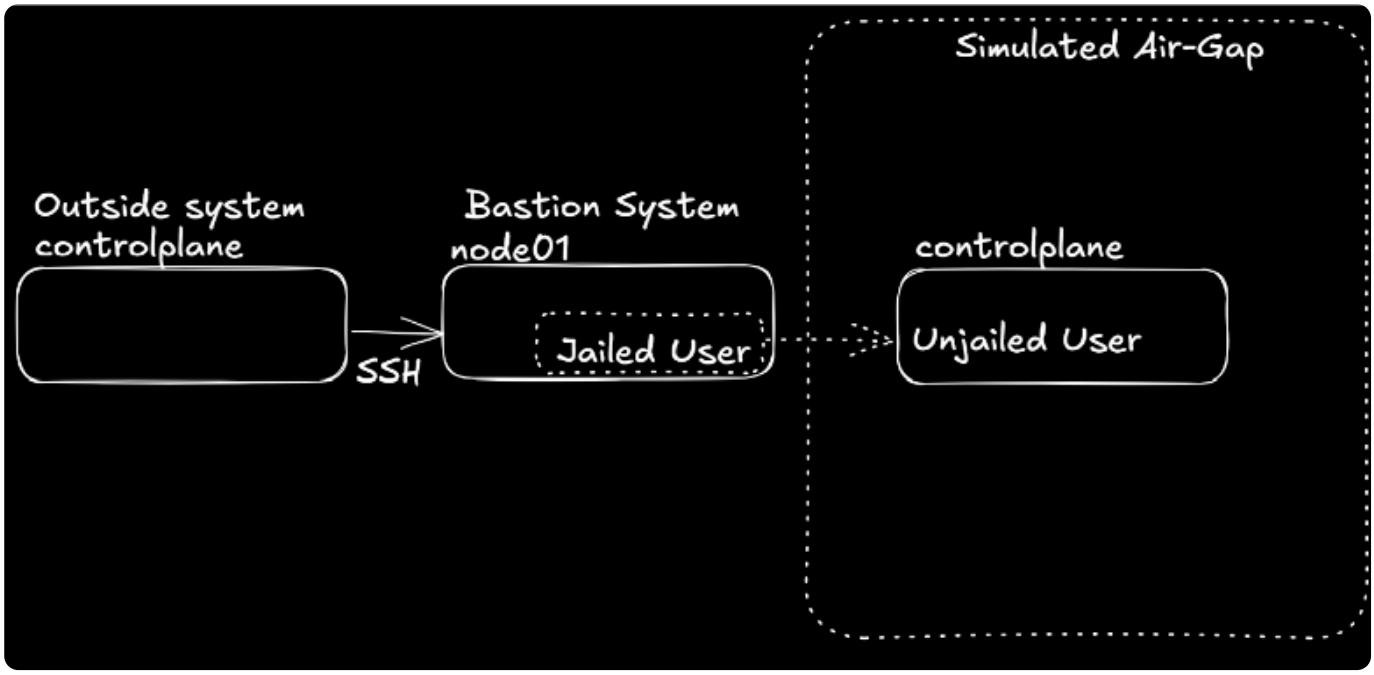
Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u4_lab(`.pdf`)

Pre-Lab Warm-Up

Review lab diagram for the Bastion design.



Lab 🧪

This lab is designed to have the engineer practice securing a Linux environment by the use of bastion hosts and jailing users as they enter an air-gapped environment.

JAILING A USER

1. Follow the lab here answering the questions below as you progress: https://killercoda.com/het-tanis/course/Linux-Labs/204-building-a-chroot-jail
2. If you were to write out the high level steps of building a chroot jail, what would they be?

3. Think about what you did in the lab and what extra (or less) you might give a user/process.
    - What directories are needed?
    - What executables might you give the jailed user/process?
    - If you give an executable, why is it important to give the link libraries that it uses?
    - What are the special files that you made with mknod and why must they be there? (try removing them or redoing the lab without them. How does it break?)

BUILDING A BASTION

1. Follow the lab here: https://killercoda.com/het-tanis/course/Linux-Labs/210-building-a-bastion-host
2. If you were to write out the high level steps of building a bastion host, what would they be?
3. When you jump into the bastion host, do you have any options other than the one you have given yourself?
4. How did you test that you couldn't leave the jailed environment?
    - How effective do you think this is as a technical preventative control against user breakout in the jail, having a 20 second timeout?

Digging Deeper challenge (not required for finishing lab)

1. Fix the drawing from the lab with excalidraw and properly replace it here: https://github.com/het-tanis/prolug-labs/tree/main/Linux-Labs/210-building-a-bastion-host
2. Do a pull request and get some github street cred or something.

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.6 Unit 5 - Updating Systems and Patch Cycles

3.6.1 Repos & Patching

Overview

Where software originates—and how and when it is updated (patched)—is essential to maintaining system stability and security. Every patch applied to a system must come from a known and trusted source, as introducing changes into a stable environment can have significant consequences. Administrators and engineers ensure that patching is planned and scheduled using verified, trackable repositories and resources.

In this unit, we will examine how this process is implemented in adjacent distributions, where administrators can apply granular control to Red Hat Package Manager (RPM) packages and maintain internal repositories of vetted packages.

Learning Objectives

1. Understand the importance of package integrity.
2. Understand patching techniques and routines.
3. Understanding automated methods of patching.
4. Understanding custom internal package repositories.

Key Terms and Definitions

Patching	Repos
Software - EPEL, BaseOS v. Appstream (in RHEL/Rocky) - Other types you can find?	**httpd**
patching	**GPG Key**
DNF/YUM	

3.6.2 Unit 5 Worksheet - Repos & Patching

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://public.cyber.mil/stigs/downloads/
- https://httpd.apache.org/
- https://docs.rockylinux.org/books/admin_guide/13-softwares/
- https://sig-core.rocky.page/documentation/patching/patching/
- https://wiki.rockylinux.org/rocky/repo/
- https://www.sans.org/information-security-policy/
- https://www.redhat.com/en/blog/whats-epel-and-how-do-i-use-it/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u5_worksheet(`.txt`)
- 📥 u5_worksheet(`.pdf`)

UNIT 5 RECORDING

Link: https://www.youtube.com/watch?v=YyK5doWENY8

Discussion Post #1

Review the rocky documentation on Software management in Linux.

- https://docs.rockylinux.org/books/admin_guide/13-softwares/
- What do you already understand about the process?
- What new things did you learn or pick up?
- What are the DNF plugins? What is the use of the versionlock plugin?
- What is an EPEL? Why do you need to consider this when using one?

Discussion Post #2

Do a google search for "patching enterprise Linux" and try to wade through all of the noise.

1. What blogs (or AI) do you find that enumerates a list of steps or checklists to consider?
2. After looking at that, how does patching a fleet of systems in the enterprise differ from pushing "update now" on your local desktop? What seems to be the major considerations? What seems to be the major roadblocks?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Patching

Repos

Software

EPEL

BaseOS v. Appstream (in RHEL/Rocky)

Other types you can find?

- httpd
- patching
- GPG Key
- DNF/YUM

Digging Deeper

1. After completing the lab and worksheet, draw out how you would deploy a software repository into your system. How are you going to update it? What tools do you find that are useful in this space?

Reflection Questions

1. Why is it that repos are controlled by root/admin functions and not any user, developer, or manager?
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.6.3 Unit 5 Lab - Repos and Patching

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u5_lab(`.txt`)
- 📥 u5_lab(`.pdf`)

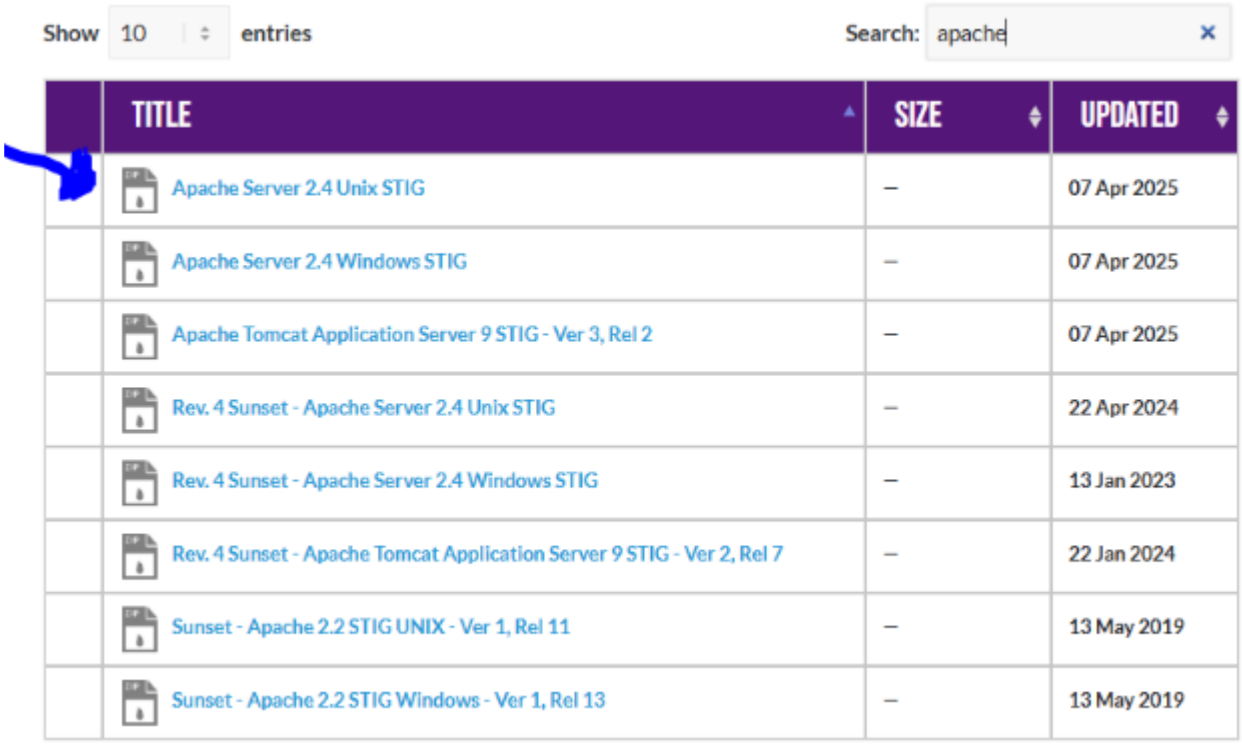
Pre-Lab Warm-Up

Download the STIG Viewer 2.18 from - https://public.cyber.mil/stigs/downloads/

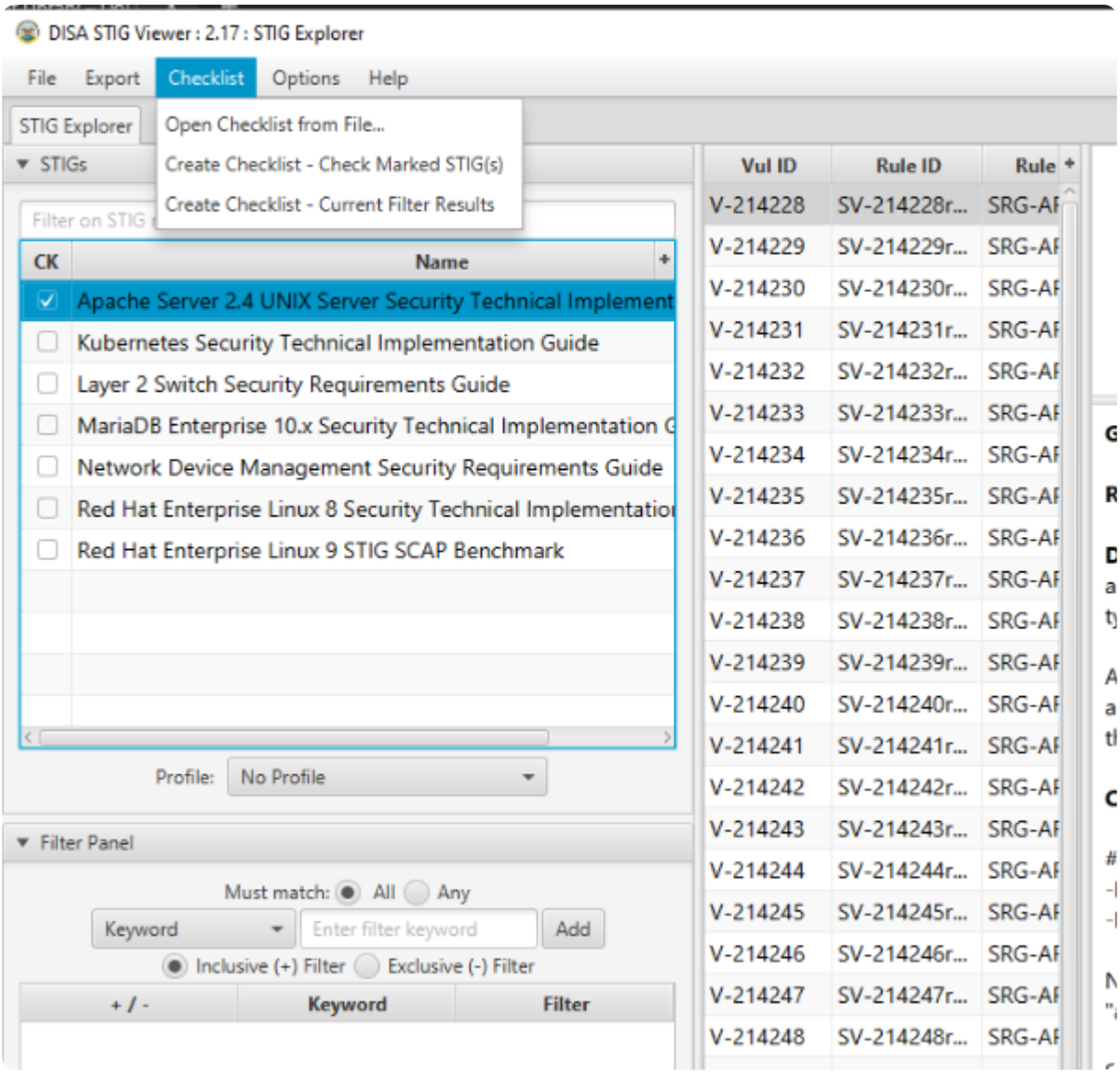
Show 10 entries Search: viewer

	TITLE	SIZE	UPDATED
	STIG Viewer 2.18	–	12 Aug 2024
	STIG Viewer 2.18 Hashes	–	12 Aug 2024
	STIG Viewer 2.18-Linux	–	12 Aug 2024
	STIG Viewer 2.18-Win64	–	12 Aug 2024
	STIG Viewer 2.18-Win64 msi	–	12 Aug 2024
	Stig Viewer 3 CKLB JSON Schema	–	10 Jan 2024
	STIG Viewer 3.5 Hashes	–	19 Feb 2025
	STIG Viewer 3.5-Linux	–	19 Feb 2025
	STIG Viewer 3.5-Win64	–	19 Feb 2025
	STIG Viewer 3.5-Win64 msi	–	19 Feb 2025

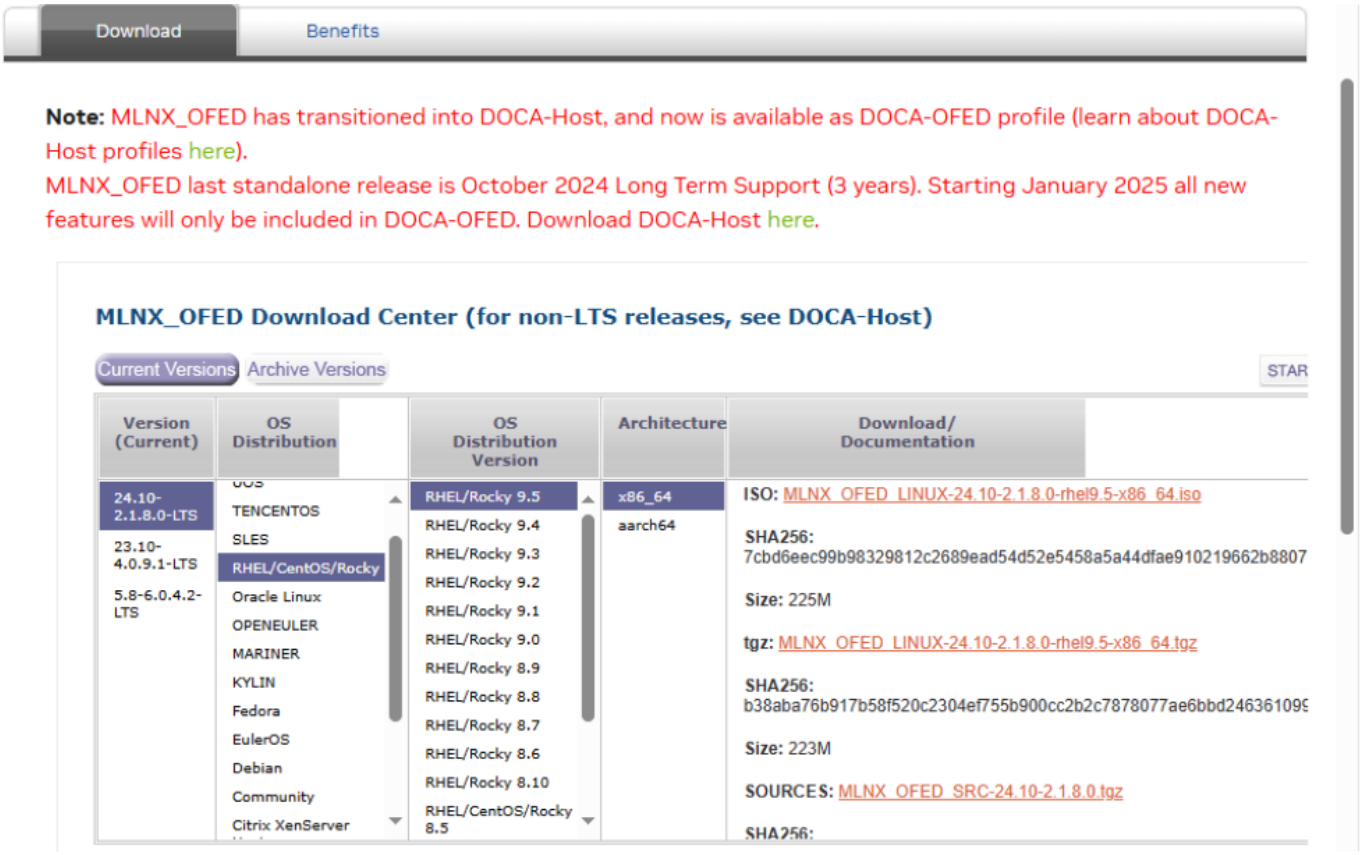
Download the STIG for Apache 2.4 and the import it into your STIG viewer



Create a checklist from the opened STIG for Apache 2.4



Review the software download process for Mellanox drivers:
Linux InfiniBand Drivers



Look through the available downloads and see if you can find the currently available LTS for Rocky 9.5 x86_64.
After that find a distribution of your choice and play with their tool.

You do not have to download or move this into our environment, I have already provided them in our lab.

Lab 🧪

This lab is designed to have the engineer practice deploying patches in a Linux environment. The engineer will create repos and then deploy patches through an automated enterprise level Ansible playbook. But first, the engineer will review some of the Apache 2.4 STIG requirements if they want to run their own repo on their network.

APACHE STIGS REVIEW

1. Look at the 4 STIGs for "tls"
    - What file is fixed for all of them to be remediated?
2. Install httpd on your Hammer server

```
systemctl stop wwclient
dnf install -y httpd
systemctl start httpd
```

3. Check STIG V-214234
    - What is the problem?
    - What is the fix?
    - What type of control is being implemented?
    - Is it set properly on your system?

4. Check STIG V-214248

    - What is the problem?
    - What is the fix?
    - What type of control is being implemented?
    - Is it set properly on your system?
    - How do you think SELINUX will help implement this control in an enforcing state? Or will it not affect it?

BUILDING REPOS

1. Start out by removing all your active repos

```
cd /etc/yum.repos.d
mkdir old_archive
mv *.repo old_archive
dnf repolist
```

2. Mount the local repository and make a local repo

```
mount -o loop /lab_work/repos_and_patching/Rocky-9.5-x86_64-dvd.iso /mnt
df -h  # Should see the mount point
ls -l /mnt
touch /etc/yum.repos.d/rocky9.repo
vi /etc/yum.repos.d/rocky9.repo
```

Add the repo configuration:

```
[BaseOS]
name=BaseOS Packages Rocky Linux 9
metadata_expire=-1
gpgcheck=1
enabled=1
baseurl=file:///mnt/BaseOS/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[AppStream]
name=AppStream Packages Rocky Linux 9
metadata_expire=-1
gpgcheck=1
enabled=1
baseurl=file:///mnt/AppStream/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

Save with `esc :wq` or "shift + ZZ"

- Do the paths you're using here make sense to you based off what you saw when using the `ls -l`? Why or why not?

```
chmod 644 /etc/yum.repos.d/rocky9.repo
dnf clean all
```

3. Test the local repository.

```
dnf repolist
dnf --disablerepo="*" --enablerepo="AppStream" list available
```

- Approximately how many are available?

```
dnf --disablerepo="*" --enablerepo="AppStream" list available | nl
dnf --disablerepo="*" --enablerepo="AppStream" list available | nl | head
```

Now use BaseOS.

```
dnf --disablerepo="*" --enablerepo="BaseOS" list available
```

- Approximately how many are available?

```
dnf --disablerepo="*" --enablerepo="BaseOS" list available | nl
dnf --disablerepo="*" --enablerepo="BaseOS" list available | nl | head
```

- Try to install something

```
dnf --disablerepo="*" --enablerepo="BaseOS AppStream" install gimp
# hit "n"
```

- How many packages does it want to install?
- How can you tell they're from different repos?

4. Share out the local repository for your internal systems (tested on just this one system)

```
rpm -qa | grep -i httpd
systemctl status httpd
ss -ntulp | grep 80
lsof -i :80
cd /etc/httpd/conf.d
vi repos.conf
```

Edit the file:

```
<Directory "/mnt">
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
Alias /repo /mnt
<Location /repo>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Location>
```

Restart the service.

```
systemctl restart httpd
vi /etc/yum.repos.d/rocky9.repo
```

Edit the file with your lab's name in the `baseurl`:

```
###USE YOUR HAMMER MACHINE IN BASEURL###
[BaseOS]
name=BaseOS Packages Rocky Linux 9
metadata_expire=-1
gpgcheck=1
enabled=1
#baseurl=file:///mnt/BaseOS/
baseurl=http://hammer25/repo/BaseOS/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[AppStream]
name=AppStream Packages Rocky Linux 9
metadata_expire=-1
gpgcheck=1
enabled=1
#baseurl=file:///mnt/AppStream/
```

```
baseurl=http://hammer25/repo/AppStream/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

- Do the paths you've modified at `baseurl` make sense to you? If not, what do you need to better understand?

```
dnf clean all
dnf repolist
Try to install something
dnf --disablerepo="*" --enablerepo="BaseOS AppStream" install gimp
```

ENTERPRISE PATCHING

1. Complete the killercoda lab found here: https://killercoda.com/het-tanis/course/Ansible-Labs/102-Enterprise-Ansible-Patching
   - Look at the roles, in the order the are run in the playbook.
     - Does it make sense how the custom facts are used? What other custom facts might you use?
     - What are the prechecks doing? What other ones might you add?
     - What does the reboot task do, and how does it check for reboot to be needed?

Digging Deeper challenge (not required for finishing lab)

1. You've set up a local repository and you've shared that repo out to other systems that might want to connect. Why might you need this if you're going to fully air-gap systems?
   Is it still necessary even if your enterprise patching solution is well designed? Why or why not?
2. Can you add the Mellanox ISO that is included in the `/lab_work/repos_and_patching` section to be a repository that your systems can access? If you have trouble, troubleshoot and ask the group for help.
3. Make a pull request to improve the enterprise patching tool that you just used. Write up, for the group, why you need that change and how it improves the efficacy of the patching.

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.7 Unit 6 - Monitoring and Parsing Logs

3.7.1 Unit 6 - Monitoring and Parsing Logs

Overview

Monitoring and parsing logs is one of the most essential security engineering practices in any production environment.

This unit explores how logs are generated, formatted, collected, and analyzed across various layers of the infrastructure stack, from applications to operating systems to networks.

Students will gain an operational understanding of how to identify log sources, use modern tools for log aggregation and search (such as Loki), and develop awareness of log structure, integrity, and retention requirements.

Learning Objectives

By the end of Unit 6, students will:

1. Understand the different types of logs and their role in system and security monitoring.
2. Identify log structures (e.g., RFC 3164, RFC 5424, `journald`) and apply appropriate parsing techniques.
3. Explore and configure log aggregation pipelines using modern tools like Grafana Loki.
4. Analyze real-world security events using log data and query languages.
5. Learn how log immutability and integrity contribute to reliable forensics and compliance.

Key terms and Definitions

Types of Logs	**Application Logs**
Host Logs	**Network Logs**
Database Logs	**Log Structure**
RFC 3164 BSD Syslog	**RFC 5424 IETF Syslog**
Systemd Journal	**Log Rotation**
Log Aggregation	**ELK Stack**
Splunk	**Loki**
Graylog	**SIEM (Security Information and Event Management)**

3.7.2 Unit 6 Worksheet - Monitoring and Parsing Logs

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://grafana.com/docs/loki/latest/query/analyzer/
- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://public.cyber.mil/stigs/downloads/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u6_worksheet(`.txt`)
- 📥 u6_worksheet(`.pdf`)

UNIT 6 RECORDING

Link: https://www.youtube.com/watch?v=pnCC-FX-aag

Discussion Post #1

Review chapter 15 of the SRE book: https://google.github.io/building-secure-and-reliable-systems/raw/ch15.html#collect_appropriate_and_useful_logs. There are 14 references at the end of the chapter. Follow them for more information. One of them: https://jvns.ca/blog/2019/06/23/a-few-debugging-resources/ should be reviewed for question "c".

- a. What are some concepts that are new to you?
- b. There are 5 conclusions drawn, do you agree with them? Would you add or remove anything from the list?
- c. In Julia Evan's debugging blog, which shows that debugging is just another form of troubleshooting, what useful things do you learn about the relationship between these topics? Are there any techniques you already do that this helps solidify for you?

Discussion Post #2

Read https://sre.google/sre-book/monitoring-distributed-systems/.

- What interesting or new things do you learn in this reading? What may you want to know more about?
- What are the "4 golden signals"?
- After reading these, why is immutability so important to logging? What do you think the other required items are for logging to be effective?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Types of logs

- Application
- Host
- Network
- DB

Immutable

Structure of Logs

- RFC 3164 BSD Syslog
- RFC 5424 IETF Syslog
- Systemd Journal

Log rotation

Rsyslog

Log aggregation

- ELK
- Splunk
- Graylog
- Loki

SIEM

Digging Deeper

1. Find a cloud service and see what their logging best practices are for security incident response. Here is AWS: https://aws.amazon.com/blogs/security/logging-strategies-for-security-incident-response/
    - What are the high level concepts mentioned?
    - What are the tools available and what actions do they take?
    - What are the manual and automated query capabilities provided, and how they help you rapidly get to a correct assessment of the logged events?
2. Open up that STIG Viewer and filter by "logging" for any of the previous STIGs we've worked on. (Mariadb has some really good ones.)
    - What seems to be a common theme?
    - What types of activities MUST be logged in various applications and operating systems?
        - Does it make sense why all logins are tracked?
        - Does it make sense why all admin actions, even just attempted admin actions, are logged?

Reflection Questions

1. What architectures have you used in your career?
    - If you haven't yet worked with any of these, what do you think you would architect in the ProLUG lab (~60 virtual machines, 4 physical machines, 1 NFS share, and 2 Windows laptops?)
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.7.3 Unit 6 Lab - Monitoring and Parsing Logs

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u6_lab(`.pdf`)

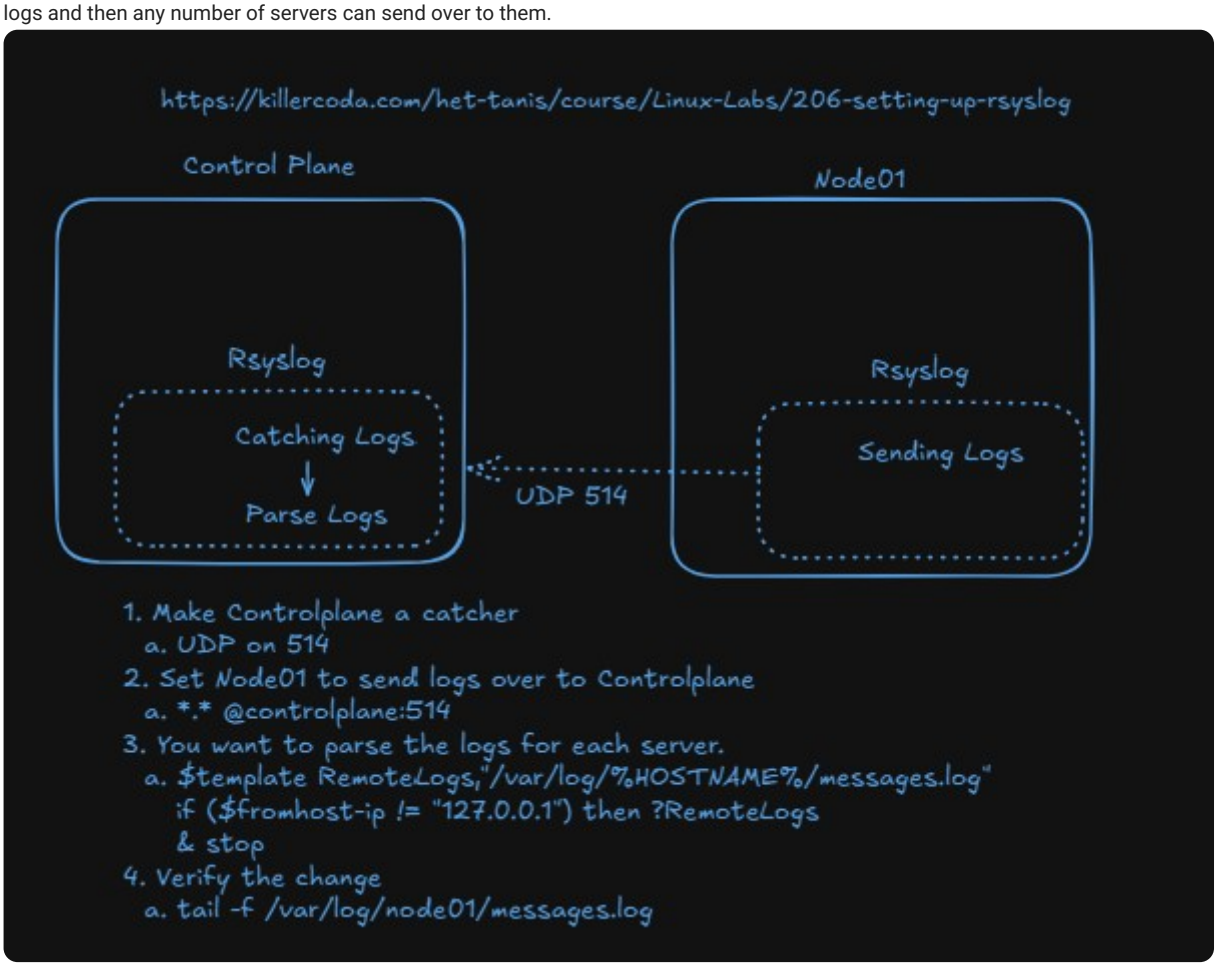
Lab 🧪

In keeping with the lab for this week, there are 4 major architectures for collecting and storing logs. Within these architectures exist many mutations from the archetype that solve different problems addressed in the scale, reliability, real-time analysis, budget, expertise, compliance, and existing infrastructure of the systems being logged.

This lab will touch 3 of the 4 types of architectures, so that the learner understands the deployment and capabilities. The 4th, cloud, architecture type will be optionally completed by the learner for their cloud deployment of choice. The learner can then reflect on the tradeoff of why one or another of these tools may be the right choice in their organization or not.

RSYSLOG FORWARDING AND COLLECTION

1. Consider this architecture, where all modern Linux systems have built in rsyslog capabilities. One of them can be set to "catch" or aggregate all logs and then any number of servers can send over to them.



2. Complete the lab: https://killercoda.com/het-tanis/course/Linux-Labs/206-setting-up-rsyslog
    - Why do we split out the logs in this lab? Why don't we just aggregated them to one place?
        - What do we split them out by?
        - How does that template configuration work?
    - Are we securing this communication in any way, or do we still need to configure that?
3. We will revisit this lab in Unit 10, with security involved via certificates, so make sure you are comfortable with the base components you are configuring.

AGENTS FORWARD TO A CENTRALIZED PLATFORM

1. Review the base architecture here: https://grafana.com/docs/loki/latest/get-started/architecture/

2. Complete the lab here: https://killercoda.com/het-tanis/course/Linux-Labs/102-monitoring-linux-logs

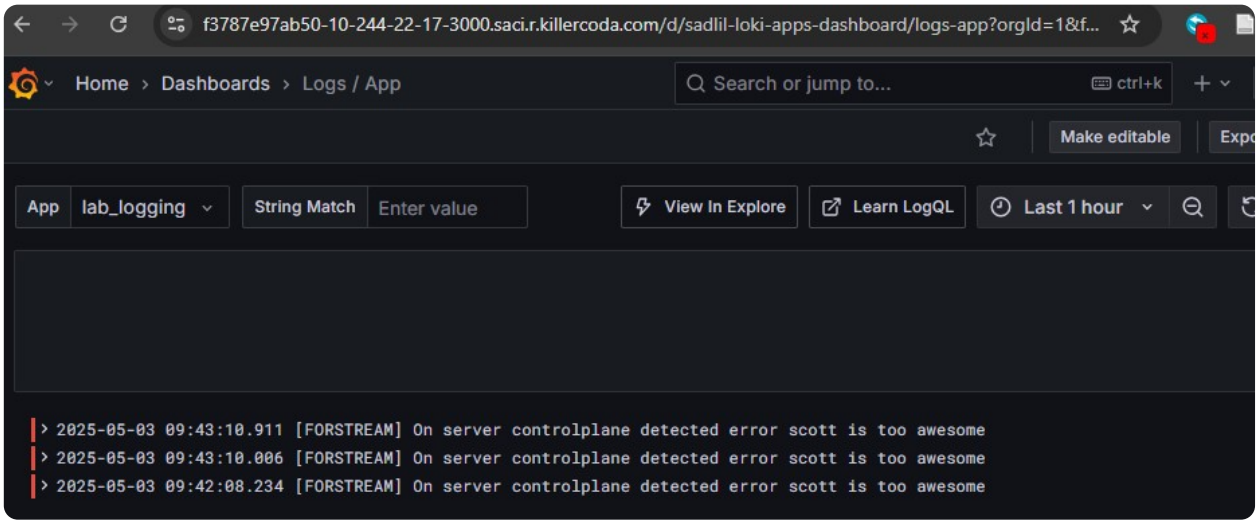
    • Does the lab work correctly, and do you understand the data flow?

    • While still in the lab

```
cd /answers
python3 loki-write.py  # Do this a few times
```

        • Refresh your Grafana and change the app to lab_logging

        • Can you see it in your Grafana?



    • Can you modify the file `loki-write.py` to say something related to your name?

    • Run this bash snippet and see if you can see your loki-writes

```
curl -G -s "http://localhost:3100/loki/api/v1/query_range" \
--data-urlencode 'query=sum(rate({job="lab_logging"}[10m])) by (level)' \
--data-urlencode 'step=300' | jq
```

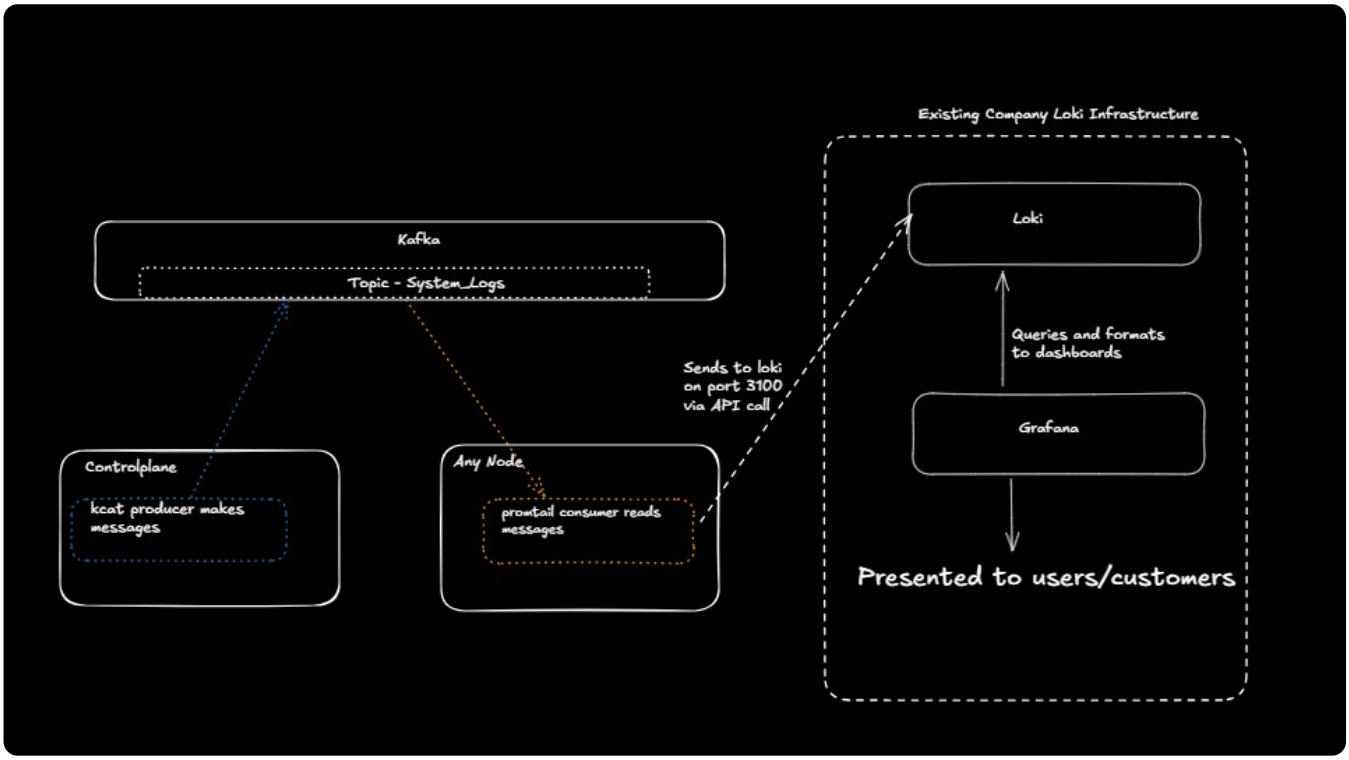
    • Can you modify that to see the actual entires? https://grafana.com/docs/loki/latest/reference/loki-http-api/#query-logs-within-a-range-of-time

3. We will revisit this lab in Unit 10, with security involved via certificates, so make sure you are comfortable with the base components you are configuring.

MESSAGE QUEUES (EVENT BUS) FOR LOG AGGREGATION AND PROPAGATION

1. Apache Kafka is not the only message queue, but it is extremely popular (found in 80% for Fortune 100 companies… or 80 of them). Read about the use cases here: https://kafka.apache.org/uses

2. Review our diagram here. Maybe we're testing kafka and want to integrate it to the existing infrastructure. Maybe we have a remote location that we need to reliably catch logs in real time and then move them remote. There are many reasons to use this.



3. Complete the killercoda lab found here: https://killercoda.com/het-tanis/course/Linux-Labs/108-kafka-to-loki-logging

    - Did you get it all to work?
        - Does the flow make sense in the context of this diagram?
    - Can you find any configurations or blogs that describe why you might want to use this architecture or how it has been used in the industry?

(OPTIONAL) CLOUD-NATIVE LOGGING SERVICES

1. OPTIONAL: Setup VPC flow logs in your AWS environment: https://catalog.workshops.aws/well-architected-security/en-US/3-detection/40-vpc-flow-logs-analysis-dashboard/1-enable-vpc-flow-logs
2. OPTIONAL: Even if not completing these labs, why might it be useful to understand the fields of a VPC flow log even if you're not setting up logging in AWS environments (but your organization does use AWS)? https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs-records-examples.html

Digging Deeper challenge (not required for finishing lab)

1. For Architecture 3, using message queues. This is an excellent write-up of how disparate systems can be connected with a message queues or event bus to enhance metrics pipelining. https://get.influxdata.com/rs/972-GDU-533/images/Customer%20Case%20Study_%20Wayfair.pdf
    - They're not necessarily doing logs, but rather metric data, but can you see how they solved their latency and connectivity problems on page 14 and 15?
2. Review some of the anti-patterns for cloud, but really any logging patterns. https://docs.aws.amazon.com/wellarchitected/latest/framework/sec_detect_investigate_events_app_service_logging.html
    - How do these relate to your current understanding of logging?
    - Do they show anything that you need to think about in the future of how you look at enterprise logging?

3. Go to https://landscape.cncf.io/guide#observability-and-analysis--observability

   - Which of these have you used and which have you not used?
   - How do many of these plug into existing observability patterns (logging)?
   - What is Fluentd trying to solve? How does it work? https://www.fluentd.org/architecture

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.8 Unit 7 - Monitoring and Alerting

3.8.1 Monitoring and Alerting

Overview

Monitoring systems and alerting when issues arise are critical responsibilities for system operators. Effective observability ensures that system health, performance, and security can be continuously assessed. In this unit, we will explore how to design reliable monitoring infrastructures through sound architectural decisions. We will also examine how alerts can be tuned and moderated to minimize noise, prioritize actionable events, and ensure timely response to real issues.

Learning Objectives

1. Understand robust monitoring architecture.
2. Understand what comprises a well architected monitoring pipeline.
3. Understand alert fatigue and how to focus on pertinent, actionable alerts.
4. Understand the trade off between information flow and security.
5. Get hands on with Fail2Ban, Prometheus, and Grafana.

Key terms and Definitions

Tracing	**Span**
Label	**Time Series Database (TSDB)**
Queue	**Upper control limit / Lower control limit (UCL/LCL)**
Aggregation	**SLO, SLA, SLI**
Push v. Pull of data	**Alerting rules**
Alertmanager	**Alert template**
Routing	**Throttling**
Monitoring for defensive operations	**SIEM**
Intrusion Detection Systems - IDS	**Intrusion Prevention Systems - IPS**

3.8.2 Unit 7 Worksheet - Monitoring and Alerting

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://promlabs.com/promql-cheat-sheet/
- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u7_worksheet(`.txt`)
- 📥 u7_worksheet(`.pdf`)

UNIT 7 RECORDING

Link: https://www.youtube.com/watch?v=2yhCRQ-QmlE

Discussion Post #1

Read about telemetry, logs, and traces. Ther are many good sources, even from Microsoft: https://microsoft.github.io/code-with-engineering-playbook/observability/log-vs-metric-vs-trace/

1. How does the usage guidance of that blog (at bottom) align with your understanding of these three items?
2. What other useful blogs or AI write-ups were you able to find?
3. What is the usefulness of this in securing your system?

Discussion Post #2

When we think of our systems, sometimes an airgapped system is simple to think about because everything is closed in. The idea of alerting or reporting is the opposite. We are trying to get the correct, timely, and important information out of the system when and where it is needed.

Read the summary at the top of: https://docs.google.com/document/d/199PqyG3UsyXlwieHaqbGiWVa8eMWi8zzAn0YfcApr8Q/edit?tab=t.0

1. What is the litmus test for a page? (Sending something out of the system?)
2. What is over-monitoring v. under-monitoring? Do you agree with the assessment of the paper? Why or why not, in your experience?
3. What is cause-based v. symptom-based and where do they belong? Do you agree?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

Telemetry

Tracing

- Span
- Label

Time Series Database (TSDB)

Queue

Upper control limit / Lower control limit (UCL/LCL)

Aggregation

SLO, SLA, SLI

Push v. Pull of data

Alerting rules

Alertmanager

- Alert template
- Routing
- Throttling

Monitoring for defensive operations

- SIEM
- Intrusion Detection Systems - IDS
- Intrusion Prevention Systems - IPS

Digging Deeper

1. Look into Wazuh: Security Information and Event Management (SIEM). Real Time Monitoring | Wazuh
    a. What are their major capabilities and features (what they advertise)?
    b. What are they doing with logs that increases visibility and usefulness in the security space? Log data analysis - Use cases · Wazuh documentation

Reflection Questions

1. What do I mean when I say that security is an art and not an engineering practice?
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.8.3 Unit 7 Lab - Monitoring and Alerting

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

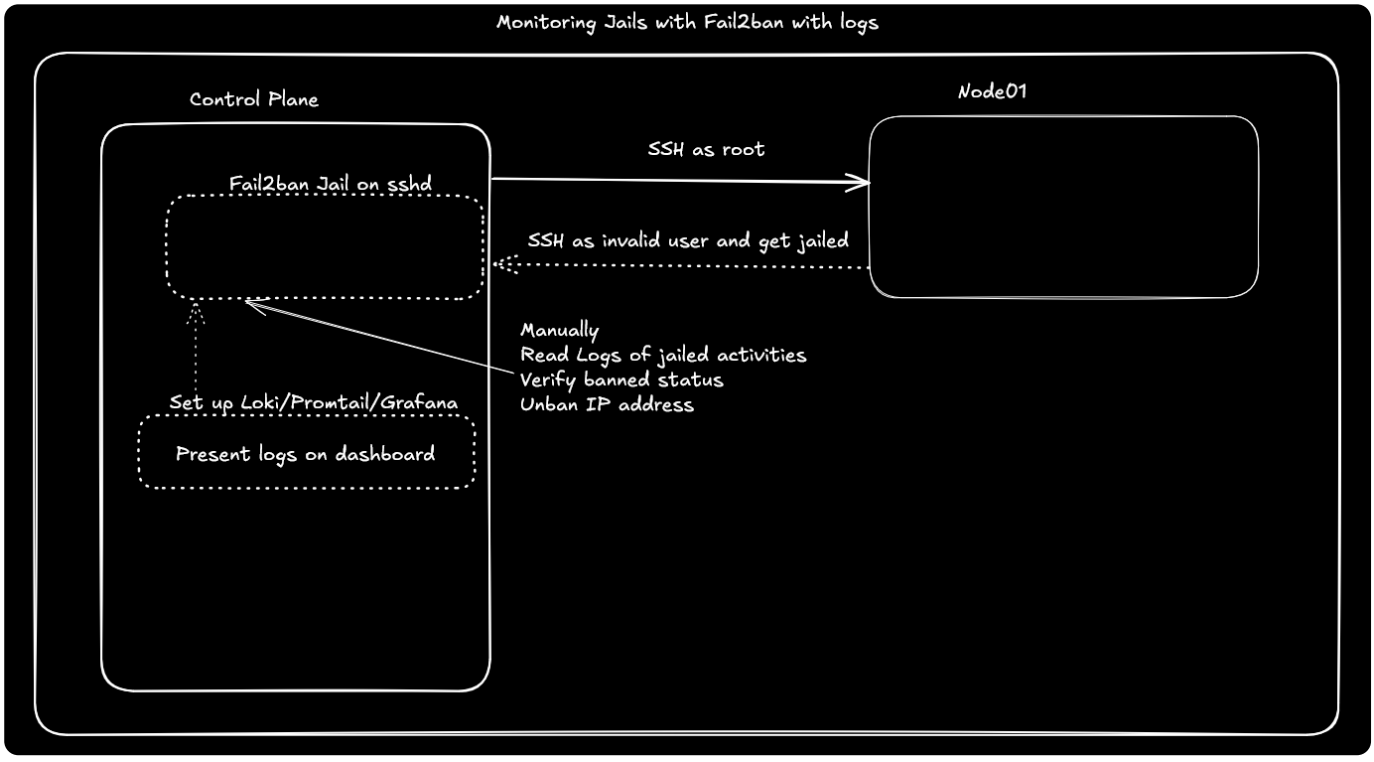
The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u7_lab(`.pdf`)

Lab 🧪

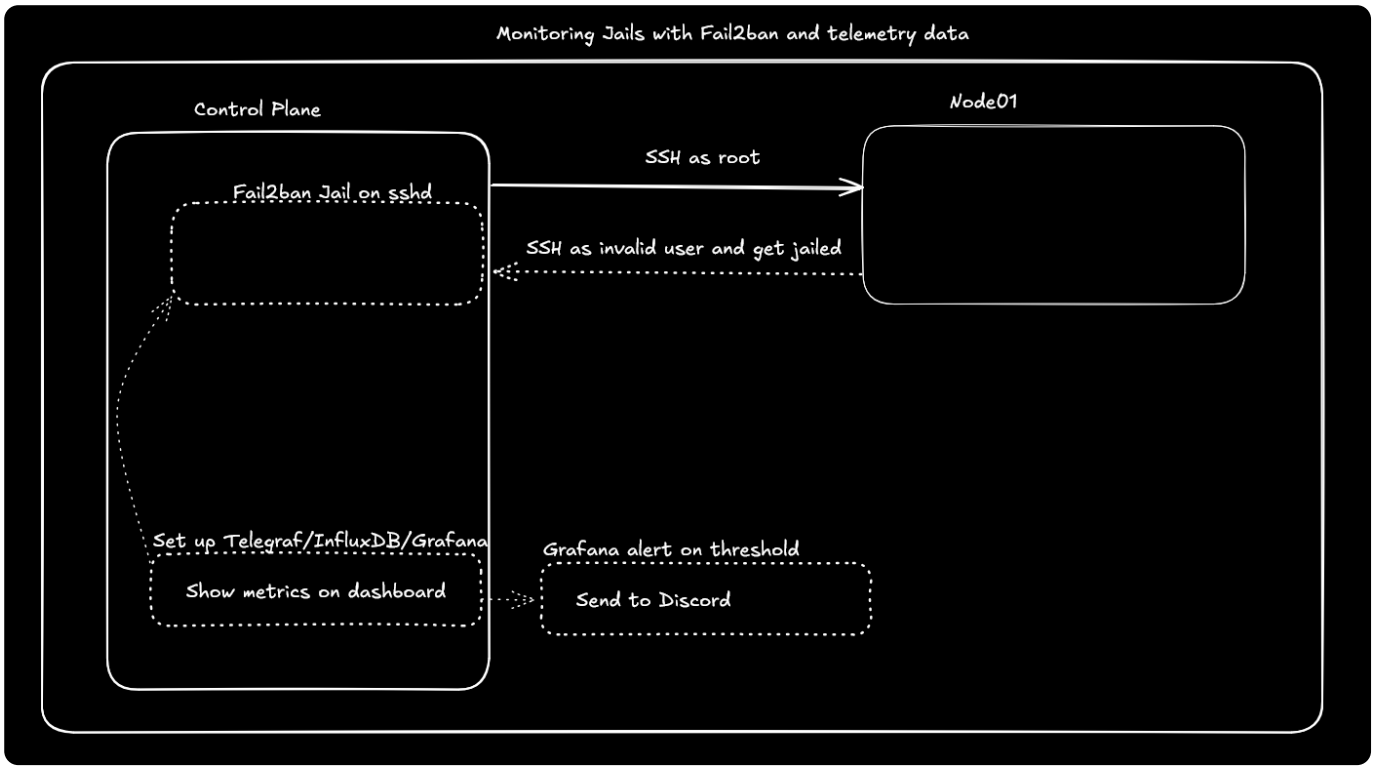
These labs focus on pulling metric information and then visualizing that data quickly on dashboards for real time analysis.

MONITORING JAILS WITH FAIL2BAN LOGS



1. Complete the lab: https://killercoda.com/het-tanis/course/Linux-Labs/109-fail2ban-with-log-monitoring
    - Were you able to see the IP address that was banned and unban it?
    - Were you able to see all the NOTICE events in Grafana?
    - What other questions do you have about this lab, and how might you go figure them out?

MONITORING JAILS WITH FAIL2BAN AND TELEMETRY DATA



1. Complete the lab here: https://killercoda.com/het-tanis/course/Linux-Labs/110-fail2ban-with-metric-alerting
    - Do you see `fail2ban` in the Grafana Dashboard? If not, how are you going to troubleshoot it?
    - Did you get your test alert and then real alert to trigger into the Discord channel?
    - What other applications or uses for this could you think of? Do you have other places you could send alerts that would help you professionally?

Digging Deeper challenge (not required for finishing lab)

1. Review the alert manager documentation: https://prometheus.io/docs/alerting/latest/configuration/
    - What are all the types of receivers you see?
    - Which of the receivers do you have experience with?
2. Review the Grafana alert thresholds: https://grafana.com/docs/grafana/latest/panels-visualizations/configure-thresholds/
    - Can you modify one of the thresholds from the lab to trigger into the discord?
    - What is the relationship between critical and warning by default?

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.9 Unit 8 - Configuration Drift and Remediation

3.9.1 Unit 8 - Configuration Drift and Remediation

Overview

Configuration drift is the silent enemy of consistent, secure infrastructure.
When systems slowly deviate from their intended state, whether that be through manual changes, failed updates, or misconfigured automation, security risks increase and reliability suffers.

In this unit, we focus on identifying, preventing, and correcting configuration drift.
Students will explore concepts like Infrastructure as Code (IaC), immutable infrastructure, and centralized configuration management.
We will also look at how drift can be detected through tools like AIDE and remediated through automation platforms like Ansible.

Students will not only understand why drift happens, but also learn how to build resilient systems that can identify and self-correct unauthorized changes.

Learning Objectives

1. Define configuration drift and understand its impact on security and operations.
2. Explore change management frameworks, including CMDBs and baselines.
3. Implement detection tools like AIDE to monitor file system integrity.
4. Use Ansible to remediate drift and enforce configuration state.
5. Connect drift management to compliance, auditability, and incident response.

Key terms and Definitions

Configuration Drift	**System Lifecycle**
Change Management	**CMDB (Configuration Management Database)**
CI (Configuration Item)	**Baseline**
Build Book / Run Book	**Immutable Infrastructure**
**Hashing **	**IaC (Infrastructure as Code)**
Orchestration	**Automation**
AIDE (Advanced Intrusion Detection Environment)	

3.9.2 Unit 8 Worksheet - Configuration Drift and Remediation

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://google.github.io/building-secure-and-reliable-systems/raw/ch14.html#treat_configuration_as_code
- https://en.wikipedia.org/wiki/Configuration_management
- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u8_worksheet(`.txt`)
- 📥 u8_worksheet(`.pdf`)

UNIT 8 RECORDING

Link: https://www.youtube.com/watch?v=C5F6i9zDgJ4

Discussion Post #1

Read about configuration management here: https://en.wikipedia.org/wiki/Configuration_management

- What overlap of terms and concepts do you see from this week's meeting?
- What are some of the standards and guidelines organizations involved with configuration management?
    - Do you recognize them from other IT activities?

Discussion Post #2

Review the SRE guide to treating configurations as code.
Read as much as you like, but focus down on the "Practical Advice" section: https://google.github.io/building-secure-and-reliable-systems/raw/ch14.html#treat_configuration_as_code

- What are the best practices that you can use in your configuration management adherence?
- What are the security threats and how can you mitigate them?
- Why might it be good to know this as you design a CMDB or CI/CD pipeline?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

System Lifecycle

Configuration Drift

Change management activities

- CMDB
- CI
- Baseline

Build book

Run book

Hashing

- `md5sum`
- `sha<x>sum`

IaC

Orchestration

Automation

AIDE

Digging Deeper

1. Review more of the SRE books from Google: https://sre.google/books/ to try to find more useful change management practices and policies.

Reflection Questions

1. How does the idea of control play into configuration management? Why is it so important?
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.9.3 Unit 8 Lab - Configuration Drift and Remediation

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Downloads

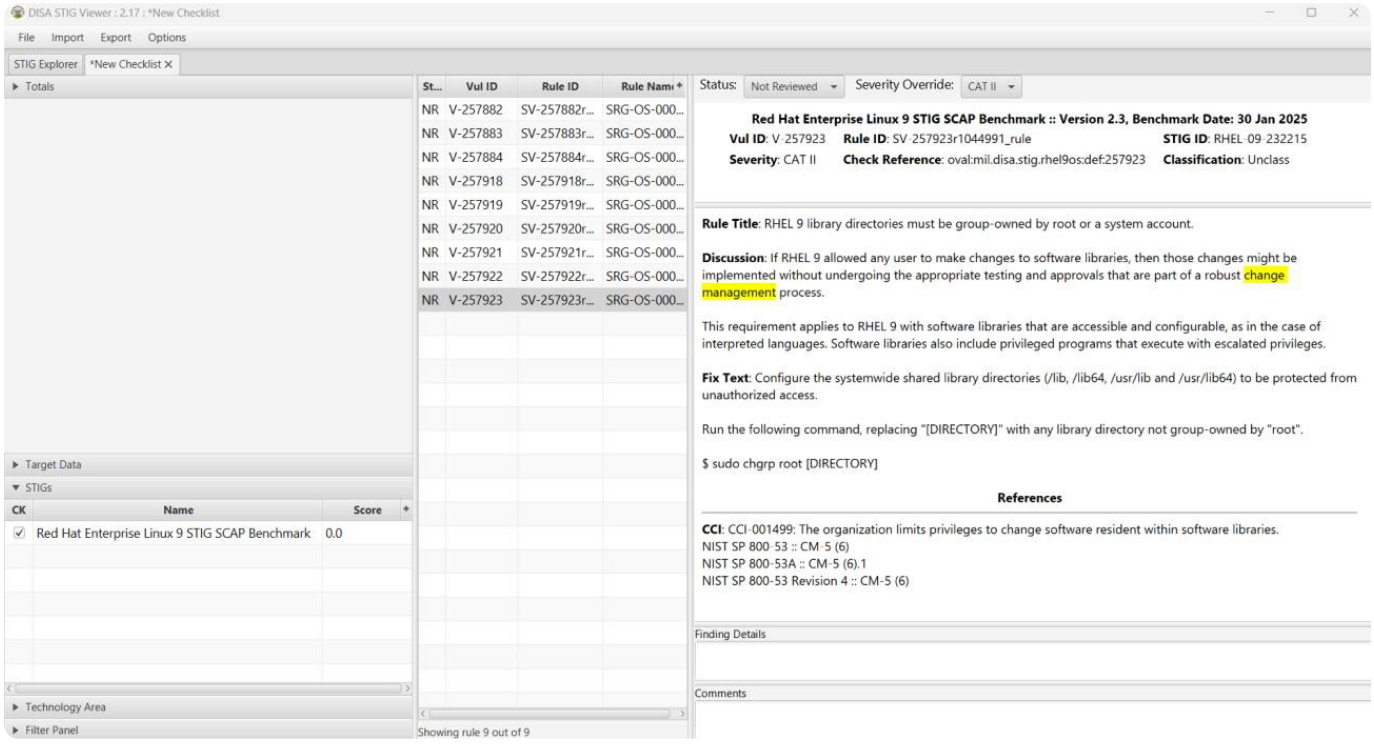
The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u8_lab(`.txt`)
- 📥 u8_lab(`.pdf`)

Lab 🧪

These labs focus on configuration drift tracking and remediation.

OPERATIONAL ACTIVITIES



1. Check your stig viewer and go to RHEL 9 stigs.
2. Set a filter for “change management”.
    - How many STIGs do you see?
3. Review the wording, what is meant by a robust change management process?
    - Do you think this can be applied in just one STIG? Why or why not?
    - What type of control is being implemented with change management in these STIGS?
        - Is it different across the STIGs or all the same?

MONITORING CONFIGURATION DRIFT WITH AIDE

1. Go into the sandbox lab: https://killercoda.com/playgrounds/scenario/ubuntu
2. Install aide and watch the installation happen.

```
apt -y install aide
```

    - What is being put in the path `/etc/aide/aide.conf.d/` ?
        - How many files are in there?
3. Check your version of aide

```
aide -v
```

4. Read the man page (first page).

```
man aide
```

    - What does aide try to do, and how does it do it?

5. What is the configuration of cron found in `/etc/cron.daily/dailyaidecheck` ?
    - What does this attempt to do?
    - What checks are there before execution?
    - Read the man for `capsh`, what is it used for?

6. Set up aide according to the default configuration

```
time aide -i -c /etc/aide/aide.conf
```

- How long did that take?

```
ubuntu:/etc/aide$ time aide -i -c /etc/aide/aide.conf
WARNING: hash calculation: '/var/log/sysstat/sa17' has been changed (change
d attributes: s+c+m, hash could not be calculated)
Start timestamp: 2025-05-17 19:25:46 +0000 (AIDE 0.18.6)
AIDE successfully initialized database.
New AIDE database written to /var/lib/aide/aide.db.new
Ignored e2fs attributes: EINV

Number of entries:      127859

---------------------------------------------------
The attributes of the (uncompressed) database(s):
---------------------------------------------------

/var/lib/aide/aide.db.new
 MD5         : /BHOfczZOeTzr6qNUdBwTA==
 SHA1        : tlAWorse1nvDevfDsP59dWuhqS8=
 SHA256      : lm4X/bVB3/IX38ixIqN8E/WKP1XaqSKn
               5NL2GbVz/hI=
 SHA512      : gOXEgLMBYtWf+ZkB9eGn/FeUb0O2ch2N
               tJB83epzCDA1tmdm1n23MHkcCntUPa2u
               xiQiXnUpt97k4YKHsVm0Hg==
 RMD160      : hww+wWCzWr1dCkvZWxmOXfSC02o=
 TIGER       : N0EomNROFRdS2cOZX9U9yJquDOWjoCyQ
 CRC32       : 5y8tQw==
 CRC32B      : zeksOg==
 HAVAL       : F04BChe+pFN382bcdmEF6xacOyYD36Gt
               7P6lFFkYMGE=
 WHIRLPOOL   : 7tq+7zYBwf5w7D2dVic0LGHxa1Xo5GUQ
               3xqPzZXcHGJ9IipoAGS0XIWwSx4bu3nc
               m9Rpge7tIOJbsC1aSUz28g==
 GOST        : 5Si7UkZYKuNSXfTZazTZWGHTWiAEe5hK
               H17TD2gQDL4=
```

(Mine took 5 minutes 8 seconds to run on the lab system)

- How much time was wall clock v. system/user time?
- Why might you want to know this on your systems?
- What do you notice about the output?
    a. What do you need to go read about?

7. Set the database up properly

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

8. Test aide by making files in a tracked directory

```
mkdir /root/prolug
touch /root/prolug/test1
touch /root/prolug/test2
time aide -c /etc/aide/aide.conf --check
```

- Did you see your new files created?
- How long did this take to run?

a. What type of usage do you see against user/system space?

```
ubuntu:/etc/aide$ aide -c /etc/aide/aide.conf --check
Entry /var/log/sysstat/sa17 in databases has different attributes: +md5+sha
1+rmd160+tiger+crc32+haval+gost+crc32b+sha256+sha512+whirlpool
Start timestamp: 2025-05-17 19:48:21 +0000 (AIDE 0.18.6)
AIDE found differences between database and filesystem!!
Ignored e2fs attributes: EINV

Summary:
  Total number of entries:      127866
  Added entries:                3
  Removed entries:              0
  Changed entries:              5

---------------------------------------------------
Added entries:
---------------------------------------------------

d++++++++++++++++++: /root/prolug
f++++++++++++++++++: /root/prolug/test1
f++++++++++++++++++: /root/prolug/test2

---------------------------------------------------
Changed entries:
---------------------------------------------------

d =.... mc.n .. . : /root
f >b... mc..H.. . : /var/log/aide/aideinit.errors
f >b... mc..H.. . : /var/log/aide/aideinit.log
f >.... mc..H.. . : /var/log/killercoda/kc-terminal.stderr
f >b... mc..+.. . : /var/log/sysstat/sa17
```

USING ANSIBLE TO FIX DRIFT

1. Complete the lab here: https://killercoda.com/het-tanis/course/Ansible-Labs/16-Ansible-Web-Server-Env-Deploy
2. When you finish ensure that you see broken output for 8081, as required.

```
curl node01:8081
```

3. One of the dev teams figured out they could modify the `test` and `qa` environments because a previous engineer left them in the sudoers file. You can address that separately with the security team, but for now you need to get those environments back to working. Run your original deployment command to see if it sets mhe environment back properly.

```
ansible-playbook -i /root/hosts /root/web_environment.yaml
```

```
controlplane:~$ ansible-playbook -i /root/hosts /root/web_environment.yaml

PLAY [Web Environment] *********************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************
ok: [node01]

TASK [Install Apache2 Server] **************************************************************************************************************
ok: [node01]

TASK [Create directories for environments] **************************************************************************************************
ok: [node01] => (item=dev)
changed: [node01] => (item=test)
ok: [node01] => (item=qa)

TASK [Add the Listener ports to /etc/apache2/ports.conf] ***********************************************************************************
ok: [node01] => (item=Listen 8080)
changed: [node01] => (item=Listen 8081)
ok: [node01] => (item=Listen 8082)

TASK [Push the Virtual Directives files into the correct place] *****************************************************************************
ok: [node01] => (item=dev_virtual_host.conf)
ok: [node01] => (item=qa_virtual_host.conf)
ok: [node01] => (item=test_virtual_host.conf)

TASK [Push the html for each page over] ****************************************************************************************************
ok: [node01] => (item={'env': 'dev', 'name': 'dev_index.html'})
changed: [node01] => (item={'env': 'test', 'name': 'test_index.html'})
ok: [node01] => (item={'env': 'qa', 'name': 'qa_index.html'})

RUNNING HANDLER [Restart apache] ***********************************************************************************************************
changed: [node01]

PLAY RECAP *********************************************************************************************************************************
node01                     : ok=7    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

controlplane:~$
```

4. Did this force the system back into a working configuration?
    - If it worked, would it always work, or would they (the systems) need to be manually intervened?
    - What is your test? (hint: `curl` the ports `8080`, `8081`, and `8082` from previous commands)
5. Could this cause potential problems in the environment?
    - If so, is that problem based on technology or operational practices? Why?

DIGGING DEEPER CHALLENGE (NOT REQUIRED FOR FINISHING LAB)

1. Complete this lab: https://killercoda.com/het-tanis/course/Ansible-Labs/19-Ansible-csv-report
    - Can you think about how you'd use this to verify that a system was stamped according to your build process?
        - You may have to tie it in with something like this lab and add some variables to your custom facts files, maybe the date of deployment: https://killercoda.com/het-tanis/course/Ansible-Labs/12-Ansible-System-Facts-Grouping

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.10 Unit 9 - Certificate and Key Madness

3.10.1 Unit 9 - Certificate and key madness

Overview

In today's interconnected world, the integrity and security of transmitted data are paramount. As systems grow in complexity and interdependence, it's crucial to verify the identity of those we communicate with and to protect the data in transit. Certificates and keys form the backbone of this trust. By securely exchanging and validating cryptographic keys and digital certificates, we establish a system where data can be encrypted, identities can be authenticated, and communications can be trusted.

Learning Objectives

1. Define the purpose and function of digital certificates and cryptographic keys.
2. Understand the differences between symmetric and asymmetric encryption.
3. Learn how TLS uses certificates for secure communication.
4. Explore the process of certificate signing and validation (PKI).
5. Use tools like `openssl` to generate keys and inspect certificates.

Key Terms & Definitions

TLS	**Symmetric Keys**
Asymmetric Keys	**Non-Repudiation**
Anti-Replay	**Plaintext**
Cypher-Text	**Fingerprints**
Passphrase (in key generation)	

3.10.2 Unit 9 Worksheet - Certificate and Key Madness

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
- https://www.sans.org/information-security-policy/
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u9_worksheet(`.txt`)
- 📥 u9_worksheet(`.pdf`)

UNIT 9 RECORDING

Link: https://www.youtube.com/watch?v=nbq_kKty1gw

Discussion Post #1

Read the Security Services section, pages 22-23 of https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf and answer the following questions.

1. How do these topics align with what you already know about system security?
2. Were any of the terms or concepts new to you?

Discussion Post #2

Review the TLS Overview section, pages 4-7 of https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf and answer the following questions.

1. What are the three subprotocols of TLS?
2. How does TLS apply
3. Confidentiality
4. Integrity
5. Authentication
6. Anti-replay

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

- TLS
- Symmetric Keys

- Asymmetric Keys
- Non-Repudiation
- Anti-Replay
- Plaintext
- Cyphertext
- Fingerprints
- Passphrase (in key generation)

Digging Deeper

1. Finish reading about TLS in the publication and think about where you might apply it.

Reflection Questions

1. What were newer topics to you, or alternatively what was a new application of something you already had heard about?
2. What questions do you still have about this week?
3. How are you going to use what you've learned in your current role?

3.10.3 Unit 9 Lab - Certificate and Key Madness

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other connection tool Lab Server
- Root or sudo command access
- STIG Viewer 2.18 (download from https://public.cyber.mil/stigs/downloads/)

Lab 🧪

These labs focus on pulling metric information and then visualizing that data quickly on dashboards for real time analysis.

Downloads

The lab has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u9_lab(`.txt`)
- 📥 u9_lab(`.pdf`)

SETTING UP RSYSLOG WITH TLS

1. Complete the lab: https://killercoda.com/het-tanis/course/Linux-Labs/211-setting-up-rsyslog-with-tls

REVIEW SOLVING THE BOTTOM TURTLE

1. Review pages 41-48 of https://spiffe.io/pdf/Solving-the-bottom-turtle-SPIFFE-SPIRE-Book.pdf
    - Does the diagram on page 44 make sense to you for what you did with a certificate authority in this lab?

SSH – PUBLIC AND PRIVATE KEY PAIRS

1. Complete the lab: https://killercoda.com/het-tanis/course/Linux-Labs/212-public-private-keys-with-ssh
    - What is the significance of the permission settings that you saw on the generated public and private key pairs?

Digging Deeper challenge (not required for finishing lab)

1. Complete the following labs and see if they reinforce any of your understanding of certificates with the use of Kubernetes.
    - https://killercoda.com/killer-shell-cks/scenario/certificate-signing-requests-sign-manually
    - https://killercoda.com/killer-shell-cks/scenario/certificate-signing-requests-sign-k8s
2. Read the rest of https://spiffe.io/pdf/Solving-the-bottom-turtle-SPIFFE-SPIRE-Book.pdf
    - How does that align with your understanding of zero-trust? if you haven't read about zero-trust, start here:
    - https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

> **Info**
>
> Be sure to `reboot` the lab machine from the command line when you are done.

3.11 Unit 10 - Recap and Final Project

3.11.1 Unit 10 - Recap and Final Project

Overview

This final unit serves as a reflection point for the course, providing students the opportunity to step back, assess what they've learned, and think deeply about how these skills apply to real-world systems and career goals.

Unit 10 is less about introducing new tools or frameworks and more about consolidating your knowledge into a cohesive security engineering mindset. Whether through discussion posts, project finalization, or self-assessment, this unit is designed to help you articulate your growth and prepare to present yourself as a capable security professional.

Learning Objectives

1. Reflect on key topics covered throughout the course and identify strengths and weaknesses.
2. Practice articulating technical security concepts and processes in your own words.
3. Prepare for technical interviews or resume reviews through self-explanation of security workflows.
4. Finalize and polish your capstone project deliverables.
5. Connect course topics to real industry expectations in security engineering.

Key terms and Definitions

This unit's terms and definitions are to be drawn from the lesson or recording.

As you watch the recording, take note of terms you're not familiar with and take the time to research them.

3.11.2 Unit 10 Worksheet - Recap and Final Project

Instructions

Fill out this sheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u10_worksheet(`.txt`)
- 📥 u10_worksheet(`.pdf`)

UNIT 10 RECORDING

Link: https://www.youtube.com/watch?v=LqvoE0UXiOo

Discussion Post #1

Capture all the terms and concepts that we talk about in this week's recording.

1. How many new topics or concepts do you have to go read about now?
2. What was completely new to you?
3. What is something you heard before, but need to spend more time with?

Discussion Post #2

1. Think about how the course objectives apply to the things you've worked on.
    - How would you answer if I asked you for a quick rundown of how you would secure a Linux system?
    - How would you answer if I asked you why you are a good fit as a security engineer in my company?
    - Think about what security concepts you think bear the most weight as you put these course objectives onto your resume.
        a. Which would you include?
        b. Which don't you feel comfortable including?

> **Info**
>
> Submit your input by following the link below. The discussion posts are done in Discord Forums. Link to Discussion Forum

Definitions

- Capture terms and definitions from this week's lesson or recording

Digging Deeper

1. Review more of the SRE books from Google: https://sre.google/books/ to try to find more useful change management practices and policies.

3.12 Course Resources

3.12.1 Course Resources

This is a comprehensive list of all external resources used in this course.

Unit 1 - Build Standards and Compliance

- https://csrc.nist.gov/projects/risk-management/about-rmf
- https://www.youtube.com/watch?v=F6loIx9WwGM
- https://www.open-scap.org
- https://excalidraw.com

Unit 2 - Securing the Network Connection

- https://www.youtube.com/watch?v=x1kgXOWv-eM
- https://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf
- https://ciq.com/blog/demystifying-and-troubleshooting-name-resolution-in-rocky-linux/
- https://docs.rockylinux.org/gemstones/core/view_kernel_conf/

Unit 3 - User Access and System Integration

- https://man7.org/linux/man-pages/man8/pam_access.8.html
- https://www.youtube.com/watch?v=-4FhZ2q4RSo
- https://docs.rockylinux.org/books/admin_guide/06-users/
- https://docs.rockylinux.org/guides/security/authentication/active_directory_authentication/
- https://docs.rockylinux.org/guides/security/pam/

Unit 4 - Bastion Hosts and Airgaps

- https://github.com/het-tanis/prolug-labs/tree/main/Linux-Labs/210-building-a-bastion-host
- https://killercoda.com/het-tanis/course/Linux-Labs/210-building-a-bastion-host
- https://killercoda.com/het-tanis/course/Linux-Labs/204-building-a-chroot-jail
- https://public.cyber.mil/stigs/downloads/
- https://github.com/het-tanis/stream_setup/blob/master/roles/bastion_deploy/tasks/main.yml
- https://www.youtube.com/watch?v=dnz92v71Tr4
- https://aws.amazon.com/blogs/security/tag/bastion-host/
- https://aws.amazon.com/search/?searchQuery=air+gapped#facet_type=blogs&page=1
- https://www.sans.org/blog/the-ultimate-list-of-sans-cheat-sheets/
- https://www.sans.org/information-security-policy/

Unit 5 - Updating Systems and Patch Cycles

- https://www.youtube.com/watch?v=YyK5doWENY8
- https://www.redhat.com/en/blog/whats-epel-and-how-do-i-use-it/
- https://wiki.rockylinux.org/rocky/repo/
- https://sig-core.rocky.page/documentation/patching/patching/

- https://docs.rockylinux.org/books/admin_guide/13-softwares/
- https://httpd.apache.org/
- https://killercoda.com/het-tanis/course/Ansible-Labs/102-Enterprise-Ansible-Patching
- Linux InfiniBand Drivers

Unit 6 - Monitoring and Parsing Logs

- https://www.fluentd.org/architecture
- https://landscape.cncf.io/guide#observability-and-analysis--observability
- https://docs.aws.amazon.com/wellarchitected/latest/framework/sec_detect_investigate_events_app_service_logging.html
- https://get.influxdata.com/rs/972-GDU-533/images/Customer%20Case%20Study_%20Wayfair.pdf
- https://catalog.workshops.aws/well-architected-security/en-US/3-detection/40-vpc-flow-logs-analysis-dashboard/1-enable-vpc-flow-logs
- https://killercoda.com/het-tanis/course/Linux-Labs/108-kafka-to-loki-logging
- https://kafka.apache.org/uses
- https://grafana.com/docs/loki/latest/reference/loki-http-api/#query-logs-within-a-range-of-time
- https://killercoda.com/het-tanis/course/Linux-Labs/102-monitoring-linux-logs
- https://grafana.com/docs/loki/latest/get-started/architecture/
- https://killercoda.com/het-tanis/course/Linux-Labs/206-setting-up-rsyslog
- https://aws.amazon.com/blogs/security/logging-strategies-for-security-incident-response/
- https://sre.google/sre-book/monitoring-distributed-systems/
- https://jvns.ca/blog/2019/06/23/a-few-debugging-resources/
- https://google.github.io/building-secure-and-reliable-systems/raw/ch15.html#collect_appropriate_and_useful_logs
- https://www.youtube.com/watch?v=pnCC-FX-aag
- https://grafana.com/docs/loki/latest/query/analyzer/

Unit 7 - Monitoring and Alerting

- https://grafana.com/docs/grafana/latest/panels-visualizations/configure-thresholds/
- https://prometheus.io/docs/alerting/latest/configuration/
- https://killercoda.com/het-tanis/course/Linux-Labs/110-fail2ban-with-metric-alerting
- https://killercoda.com/het-tanis/course/Linux-Labs/109-fail2ban-with-log-monitoring
- Log data analysis - Use cases · Wazuh documentation
- Security Information and Event Management (SIEM). Real Time Monitoring | Wazuh
- https://docs.google.com/document/d/199PqyG3UsyXlwieHaqbGiWVa8eMWi8zzAn0YfcApr8Q/edit?tab=t.0
- https://microsoft.github.io/code-with-engineering-playbook/observability/log-vs-metric-vs-trace/
- https://www.youtube.com/watch?v=2yhCRQ-QmlE
- https://promlabs.com/promql-cheat-sheet/

Unit 8 - Configuration Drift and Remediation

- https://www.youtube.com/watch?v=C5F6i9zDgJ4
- https://en.wikipedia.org/wiki/Configuration_management
- https://google.github.io/building-secure-and-reliable-systems/raw/ch14.html#treat_configuration_as_code
- https://killercoda.com/het-tanis/course/Ansible-Labs/12-Ansible-System-Facts-Grouping
- https://killercoda.com/het-tanis/course/Ansible-Labs/19-Ansible-csv-report
- https://killercoda.com/het-tanis/course/Ansible-Labs/16-Ansible-Web-Server-Env-Deploy

- https://killercoda.com/playgrounds/scenario/ubuntu

Unit 9 - Certificate and Key Madness

- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf
- https://killercoda.com/killer-shell-cks/scenario/certificate-signing-requests-sign-k8s
- https://killercoda.com/killer-shell-cks/scenario/certificate-signing-requests-sign-manually
- https://killercoda.com/het-tanis/course/Linux-Labs/212-public-private-keys-with-ssh
- https://spiffe.io/pdf/Solving-the-bottom-turtle-SPIFFE-SPIRE-Book.pdf
- https://killercoda.com/het-tanis/course/Linux-Labs/211-setting-up-rsyslog-with-tls
- https://www.youtube.com/watch?v=nbq_kKty1gw
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r2.pdf
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

Unit 10 - Recap and Final Project

- https://sre.google/books/
- https://www.youtube.com/watch?v=LqvoE0UXiOo

Misc

- https://www.cisecurity.org/cis-benchmarks
- https://owasp.org/www-project-top-ten/
- https://www.nist.gov/
- https://public.cyber.mil/stigs/srg-stig-tools/
- https://killercoda.com/
- https://github.com/ProfessionalLinuxUsersGroup/course-books/
- @Het_Tanis
- overleaf.com.
- https://www.overleaf.com/
- https://gdpr.eu/what-is-gdpr/
- https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html

4. Linux Automation Course

4.1 Course Overview

4.1.1 Linux Automation for the Enterprise

Welcome to the ProLUG Enterprise Linux Automation Course Book.

This Book

Contains all materials pertaining to the course including links to external resources. It has been put together with care by a number of ProLUG group members referencing original instructional materials produced by Scott Champine @Het_Tanis .

The content is version controlled with Git and stored here: https://github.com/ProfessionalLinuxUsersGroup/course-books

COURSE DESCRIPTION

This course addresses how to automate Linux in a corporate environment. This course will focus on taking design, build, deploy, administration, and remediation tasks and turning them into automated processes. The emphasis will be on integration during the build, run, and end of life phase so that CI/CD can occur and deployments as code are utilized. The learner will practice securely building, deploying, integrating, and monitoring Linux systems in an automated fashion.

PREREQUISITE(S) AND/OR COREQUISITE(S):

Prerequisites: None
Credit hours: N/A
Contact hours: 160 (60 Theory Hours, 100 Lab Hours)

Course Summary

MAJOR INSTRUCTIONAL AREAS

- Automation Tools
- Automation Mindset
    - Systems Engineering Focus
    - Continuous Integration/Continuous Deployment (**CI/CD**) Focus
    - Operations/Administration Focus
- System Building (Standardization)
    - Prebaking
    - Personalization
- Updating Systems and Patch Management
- Software and Package Deployment
- Networking Configurations
- System Checking and Reporting

COURSE OBJECTIVES

- Deploy various automation tools for engineering and operations activities.
- Retrieve information with API calls to facilitate automation activities.
- Maintain inventories of systems for automation activities.
- Map environment variables to systems and utilize them in automations.
- Describe the activities of the Automation Mindset.

- Map where automation works within CI/CD frameworks.
- Deploy and configure a Linux system with automated tools.
- Automate updating Linux to resolve security vulnerabilities and report out to security teams.
- Configure networking devices with automation tools.
- Maintaining system configuration and remediating drift via automation.
- Troubleshooting Automation and CI/CD pipelines when there are issues
- Utilize the testing methodologies for environments and CI/CD Deployments

Learning Materials and References

Option #1 (Killercoda Machine)

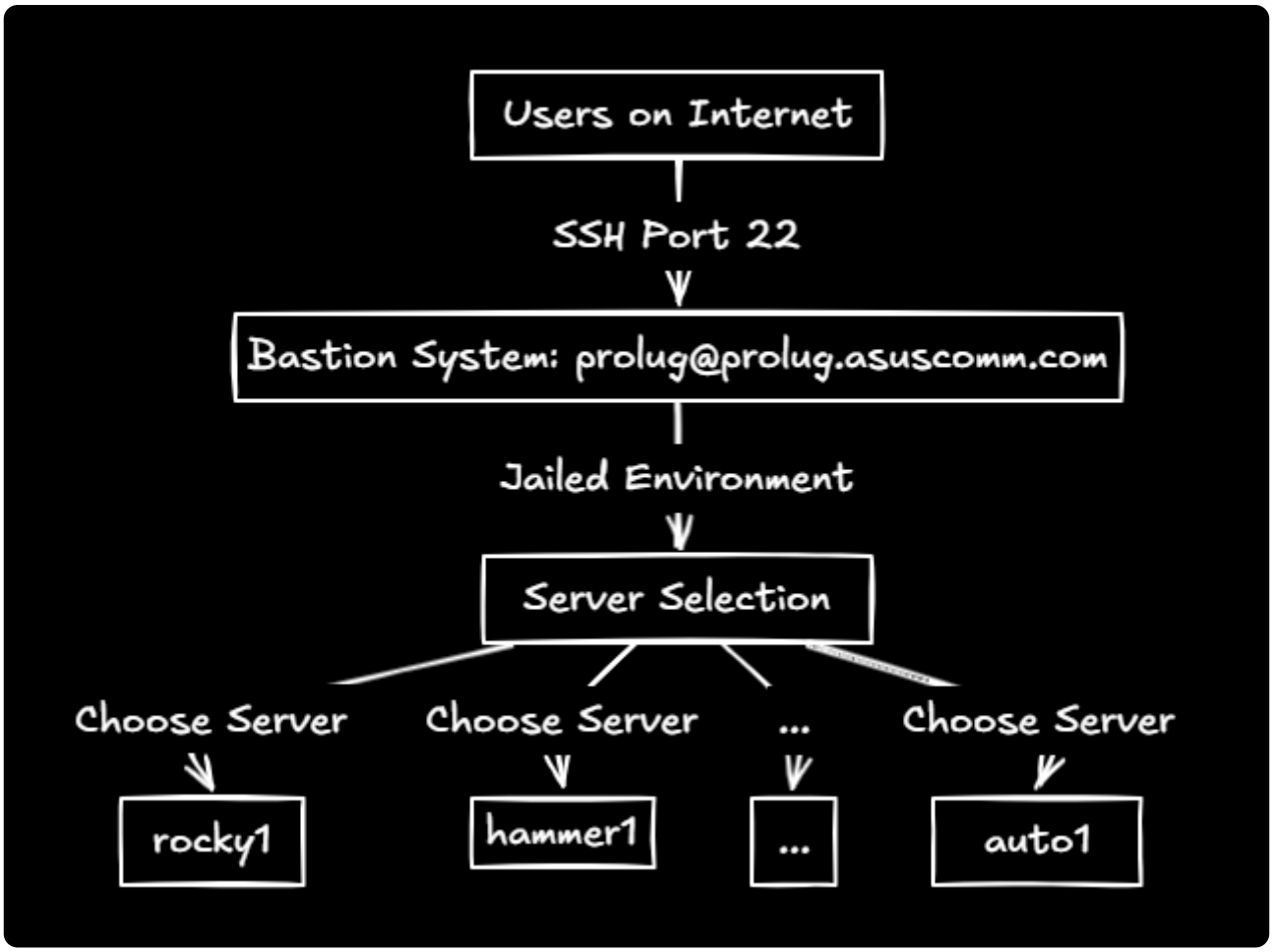
- Cloud Lab server running Ubuntu on Killercoda (must be 2 node environment). Minimal resources:
  - 1 CPU
  - 2 GB Ram
  - 30 GB Hard Drive
  - Network Interface (IP already setup)

Option #2 (Home Lab)

- Local VM server running: RHEL, Fedora, Rocky Minimal resources:
  - 1 CPU
  - 2GB RAM
  - Network Interface (Bridged)

Option #3 (ProLUG Remote Lab)

- ProLUG Lab access to Rocky 9 instances (auto servers). Minimal resources:
  - 1 CPU
  - 8 GB RAM
  - Network Interface (IP already setup)



Course Plan

INSTRUCTIONAL METHODS

This course is designed to promote learner-centered activities and support the development of Linux automation skills. The course utilizes individual and group learning activities, performance-driven assignments, problem-based cases, projects, and discussions. These methods focus on building engaging learning experiences conducive to development of critical knowledge and skills that can be effectively applied in professional contexts.

The "**Explore** → **Practice** → **Apply**" acquisition path will be integrated into this course.

CLASS SIZE

This class will effectively engage 100-200 learners.

CLASS SCHEDULE

Class will meet over weekend (brown bag) sessions. One (1) time per week, for 16 weeks. There will be a total of 16 sessions.

SESSION OVERVIEW

Session	Topic
1	Automation Tools (Installation and Execution)
2	Interacting with the Operating System
3	Making and Using Inventories
4	Admin Commands and One-Offs
5	Environment and Local Variables in Systems
6	Automating Docker Builds
7	Automating Docker Environments
8	Automating K8s Environments
9	Build and Deploy Linux Systems
10	Harden Linux Systems
11	Update and Patch Systems
12	Configure Network Devices
13	Remediating and Reporting on Drift
14	CI/CD Pipelines and Make v. Buy v. Adopt Decisions.
15	Troubleshooting/Testing 1
16	Troubleshooting/Testing 2

Suggested Learning Approach

In this course, you will be studying individually and within a group of your peers, primarily in a lab environment. As you work on the course deliverables, you are encouraged to share ideas with your peers and instructor, work collaboratively on projects and team assignments, raise critical questions, and provide constructive feedback.

4.1.2 ProLUG Linux Automation

Lab Guide

Students will be granted access to the ProLUG lab environment upon request.
If you would like access, join us on Discord and ask in the `#prolug_lab_environment` channel.

Outlined below is the Acceptable Use Policy and layout of the lab used for the Linux Automation Course.

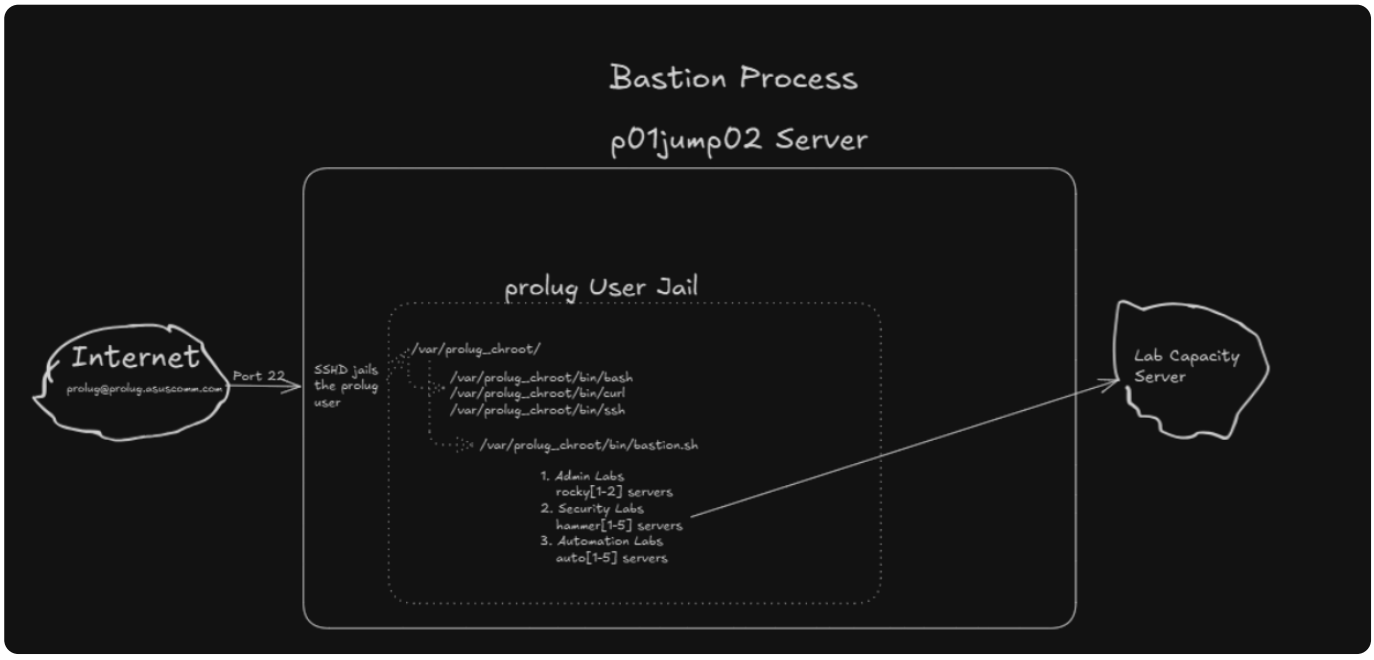
Acceptable Use Policy

The acceptable use policy is presented during login through the Bastion node:

"You are entering the ProLUG lab environment. All activities may be monitored. By entering you agree to use the lab for the stated purpose of learning Linux."

> **Warning**
>
> Willful violations of this AUP, or intentional destruction will result in removal from ProLUG.

Access to Lab



Auto server mapping to target nodes is as follows.

```
Auto1 -> user: svc_ansible, password: ansible1234 -> target nodes: target1-1, target1-2
Auto2 -> user: svc_ansible, password: ansible1234 -> target nodes: target2-1, target2-2
Auto3 -> user: svc_ansible, password: ansible1234 -> target nodes: target3-1, target3-2
Auto4 -> user: svc_ansible, password: ansible1234 -> target nodes: target4-1, target4-2
Auto5 -> user: svc_ansible, password: ansible1234 -> target nodes: target5-1, target5-2
```

For additional lab capacity, use Killercoda.com.

That lab mapping is as follows:

```
Controlplane -> user: root, key -> target node: node01
```

Public Keys

Public keys are dropped in the `#prolug_lab_environment` channel in Discord, where they can be added to the jump server for bastion access.

> **Note**
>
> The password method is unreliable, as it changes often.

Lab Best Practices

Reboot servers when you are done with them.
This will ensure a clean server for the next user to log in.

4.1.3 Table of Contents

4.2 Unit 1 - Automation Tools

4.2.1 Unit 1

Automation tools Installation and Execution

Overview

This unit introduces the types of triggers used in automation.

- Timed Triggers
- Event Triggers
- Alert Triggers

The concept of an automation mindset is introduce and explained in detail. This helps us understand what type of automation we are going to use as system administrators and system engineers.

This unit addresses the course objective to deploy various automation tools for engineering and operations activities.

Learning Objectives

1. Scientific Method
    - As engineers we use these six steps as a real/ repeatable practice.
    - "Engineering puts the scientific method into repeatable practice."
2. Eight function cycle of operation
    - The engineering process explained using an analogy of the firearm cycle of operation.
    - "Any deviation from our normal acceptable process is the absolute enemy of engineering."
3. Automation
    - Using triggers to start a set of steps in a process.
    - "You have to have things work through [specific steps] to completion to complete a cycle.."
4. Triggers
    - Time, Events & Alerts
5. The concept of an automation mindset for engineers.
    - Consistency / repeatability
    - Timeliness
6. DevOps Methodology
    - Dev is always 'spinning' in a constant cycle.

Key Terms and Definitions

Terms	Definitions
EOL	End of Life
CI/CD	Continuous Implementation / Continuous Deployment
SDLC	Software Delivery Life Cycle

4.2.2 Unit 1 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Event Driven Architecture
- Monitoring
- Kafka (event bus) Blogs
- https://killercoda.com/het-tanis/course/Ansible-Labs
- https://serverlessland.com/patterns

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u1_worksheet(`.md`)
- 📥 u1_worksheet(`.txt`)
- 📥 u1_worksheet(`.pdf`)

UNIT 1 RECORDING

Link: https://youtu.be/wyVhGtFFYIQ

Discussion Post #1

The first question of this course is, "What is automation?"

1. If you've done the previous courses, how would you define administration? If you haven't find a blog (link for us) and explain how they define administration of Linux systems.
2. If you've done the previous course, how would you define security? If you haven't find a blog (link for us) and explain how they define administration of Linux systems.
3. When you think about automation, how does it tie into things you do on a daily basis, inside or outside of computer systems?

Discussion Post #2

What is meant by a trigger in automation?

1. What is your definition of a trigger?
2. What are the types of triggers you read or can define?
3. Where would you place these triggers to positively affect your ability to build or administer Linux systems?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

- Engineering
- Automation

- Triggers
- Scientific Method
- Deviation
- Manual Intervention
- Code Commits
- Event Driven Systems
- Alerts

Digging Deeper

1. Go to serverless land: https://serverlessland.com/patterns
    a. Can you implement one of the serverless architectures in the cloud via one of the automation tools we have talked about?
    b. Do you see any other automation tools you may use in your career?
2. While we will be going over many concepts in this course, reviewing https://killercoda.com/het-tanis/course/Ansible-Labs and ensuring you have a strong understanding of Ansible will help absorb this information.

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.2.3 Unit 1 Lab

Automation Tools - Installation and Execution

> **Info**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

REQUIRED MATERIALS

- Putty or other terminal/connection tool
- Lab Server (Killercoda recommended)
  - https://killercoda.com/het-tanis/course/Automation-Labs/Unit1_Tools
- Root or sudo command access

Downloads

The lab has been provided for convenience below:

- 📥 u1_lab(`.txt`)
- 📥 u1_lab(`.md`)

Lab

This lab is designed to have the engineer verify and execute their automation tools in a controlled environment.

BASH EXECUTION

Execute some simple commands within your bash shell.

1. Verify your location and version of bash

```
which bash
/usr/bin/bash --version
```

2. Verify your shell PID

```
echo $$
```

3. Verify your shell variable

```
echo $SHELL
```

4. Loop over your target servers (use your target servers).

```
for server in target1-1 target1-2; do timeout 10 ssh svc_ansible@$server 'uptime'; done
```

Enter your password for the `svc_ansible` user from the lab guide.

- If doing the lab on Killercoda, your target servers are `controlplane` and `node01`.
- If in the ProLUG lab environment, check the lab guide for your target servers.

PYTHON EXECUTION

Test and execute Python.

1. Verify your version of Python

```
python3 --version
```

2. Test that you can import modules

```
python3      # Will drop you into interactive shell
import os    # Should work with no output
import numpy # Should not work as you don't have numpy on the system
exit()       # Will exit the interactive python3 environment.
```

ANSIBLE EXECUTION

Test and execute Ansible.

1. Verify your version of Ansible

```
ansible –version
```

2. Check other ansible tools

```
ansible- <tab><tab>
```

The Tab ⇥ + Tab ⇥ will trigger autocompletion, showing what commands are available that start with `ansible-`.

3. Check modules

```
ansible-doc -l
ansible-doc -l | wc -l
ansible-doc -l | grep -i copy
```

Info

If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

4.3 Unit 2 - Interacting with the OS

4.3.1 Unit 2

Interacting with the operating system

Overview

This unit explains the automation cycle and how it functions in an enterprise. It guides learners on how to build all eventualities on many different types of servers. To accomplish the above learners are shown how to check on individual systems; as well as how wht is needed to be able to differentiate between various systems.

This unit continues to address the course objective to deploy various automation tools for engineering and operations activities. This unit specifically guides learners on interacting with the operating system and making sure we can do things on the system in a certain way.

Learning Objectives

1. Automation Cycle
    - Situations where you want automation to finish to completion
    - Situations where you want automation to fail gracefully
2. System Checking
    - Validating that the system is in a good state to move forward
    - Grouping systems by server function, variables, software packages, et. al.
3. Pareto Rule
    - 80 / 20
    - 80% of your effort will address 20% of your final result
    - 20% of your effort will allow you to complete 80% of your final result

Key Terms and Definitions

Terms	Definitions
Input Validation	
System Validation	
Jinja Templates	
Plinko	

4.3.2 Unit 2 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Lab: Interacting with the OS

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

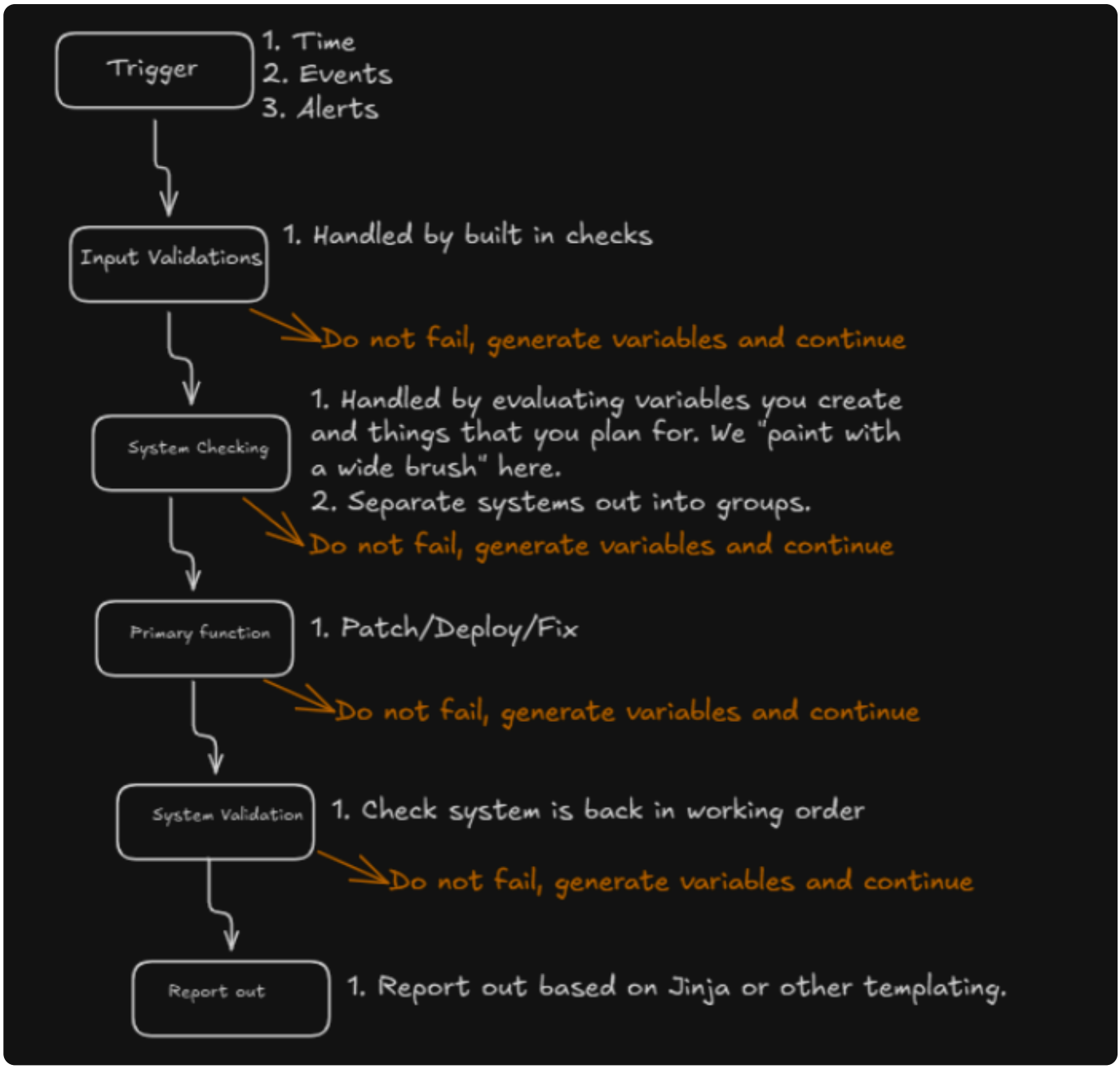
- 📥 u2_worksheet(`.md`)
- 📥 u2_worksheet(`.txt`)
- 📥 u2_worksheet(`.pdf`)

UNIT 2 RECORDING

Link: https://youtu.be/Xsz7MXJPU58

Discussion Post #1

Review the automation cycle as it is presented below and answer the following questions.



1. What would you add or take away from this drawing?
2. When might you want something to fail during an automation?
3. When might you not want an automation to fail?

Discussion Post #2

Read about the **Pareto Rule** or the **Rule of 80/20**.

1. What is the general stated rule, as you understand it?
2. Do you agree with the rule? Has this been your experience?
    a. What examples can you find where this has proven true?
    b. What examples can you find where this has not proven true?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

Pareto's Rule or Principle

80/20 Rule

Input validation

System Checking

System Validation

Reporting

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.3.3 Unit 2 Lab

Interacting with the Operating System

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

If you do the Killercoda Lab 2, just answer these questions.

If you are doing the lab in the ProLUG environment, find the scripts in `/labs/automation/unit2`.

> **Note**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Unit 2 Lab on Killercoda

REQUIRED MATERIALS

- Lab Server (Killercoda recommended)
  - Rocky 9.4+ - ProLUG Lab
  - Or comparable Linux box
- `root` or `sudo` command access

Downloads

The lab has been provided for convenience below:

- 📥 u2_lab(`.txt`)
- 📥 u2_lab(`.md`)

Pre-Lab (Lab Setup)

If you're not doing the lab on Killercoda, make sure to follow the setup guide below.

PROLUG LAB SETUP

If you're using the ProLUG lab environment, run the following commands.

```
cd /root
cp -r /labs/automation/unit2/* /root
chmod 755 /root/*.sh
chmod 755 /root/*.py
```

PERSONAL LAB SETUP

Alternatively, for your own lab environment, clone down the necessary files from GitHub.

```
cd /root
git clone https://github.com/het-tanis/prolug-labs.git
cp prolug-labs/AutomationLabs/Unit2_Interacting_with_OS/assets/* /root
```

Lab 🧪

In this lab, we will go through the process of executing our automation tools.

BASH EXECUTION

1. Run the `u2_script1.sh` and look at what it shows you.

```
/root/u2_script1.sh
```

   - What are you shown?

2. Inspect the file and see if you can modify it to show the first 15 lines.

```
cat /root/u2_script1.sh
```

   -

> **Note**
>
> Modify with vi or vim. You may have to RTFM to continue.

3. Run the u2_script2.sh and look at what it shows you.

```
/root/u2_script2.sh
```

   - What happened in the script?
   - Did it work correctly?

```
ls -l /root
```

4. Inspect the file and see if you can make it use a different date format. You may have to read the man pages for date command.

```
cat /root/u2_script2.sh
```

As you're interacting with the OS, are there any observations you have about how the scripts are set up, their structure and their output.

- Is there anything you would add for your scripts?
- If you would add something, how does it improve the code?

PYTHON EXECUTION

1. Run the u2_script1.py and look at what it shows you.

```
/root/u2_script1.py
```

   - What are you shown?

2. Inspect the file and see if you can modify it to show the first and last 15 lines.

```
cat /root/u2_script1.py
```

   -

> **Note**
>
> Modify with `vi` or `vim`. You may have to RTFM to continue.

3. Run the u2_script2.py and look at what it shows you.

```
/root/u2_script2.py
```

   - What are you shown?

4. Inspect the file and see if you can make it use a different user shell, maybe one you've seen from other output in this lab.

```
cat /root/u2_script2.py
```

- > **Note**
  >
  > Modify with vi or vim. You may have to RTFM to continue.

As you're interacting with the OS, are there any observations you have about how the scripts are set up, their structure and their output.

- Is there anything you would add for your scripts?
- If you would add something, how does it improve the code?

ANSIBLE EXECUTION

1. Run the u2_script1.yml and look at what it shows you.

   ```
   ansible-playbook /root/u2_script1.yml
   ```

   - What are you shown?
2. Inspect the file and see if you can modify it to show the first and last 15 lines.

   ```
   cat /root/u2_script1.yml
   ```

   - > **Note**
     >
     > Modify with vi or vim. You may have to RTFM to continue.
3. Run the u2_script2.yml and look at what it shows you.

   ```
   ansible-playbook /root/u2_script2.yml
   ```

   - What are you shown?
4. Inspect the file and see if you can make it name the file differently or populate different content.

   ```
   cat /root/u2_script2.yml
   ```

   - > **Note**
     >
     > Modify with vi or vim. You may have to RTFM to continue.
5. Do one final `ls -l` against the `/root` directory. What is a difference between the `.sh`, `.py`, and `.yml` files?

   ```
   ls -l
   ```

As you're interacting with the OS, are there any observations you have about how the scripts are set up, their structure and their output.

- Is there anything you would add for your scripts?
- If you would add something, how does it improve the code?

> **Note**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

4.4 Unit 3 - Creating && Using Inventories

4.4.1 Unit 3

Making and Using Inventories

Overview

This unit describes best practices for inventory management. It will clarify details about our inventory sources such as flat files and API calls from other programs. We learn that we cannot do anything properly without a good inventory and are guided on how best to achieve this goal.

We learn that a good inventory is complete, accurate and properly formatted. Inventories must use consistent identifiers of the same type such as IP vs FQDN values. The semi-structured nature of our inventory must be readable and properly formatted for use by our tool of choice (e.g. bash, python, ansible).

Learning Objectives

1. Inventory Sources
    - Flat files
    - API calls
2. Aspects of a good inventory
    - Accurate
    - Readable
    - Useful
3. Inventories are structured and useful lists of items.
    - With the goal of being 100% complete (otherwise the missing 2% will become your focus)
    - Commonly maintained with bash, python & ansible. (this course focuses on ansible)
    - Understanding the various types of semi-structured lists
        - flat files
        - CSV files
        - json files
        - yaml files

Key Terms and Definitions

Terms	Definitions		
Flat files	Just a list of servers or a basic list of tab/space separated values		
IP address	Internet Protocol address	[RFC 1918]	
FQDN	Fully Qualified Domain Name	[RFC 1034]	(note: RFC 1035 and many later updates)
CSV	Comma Separated Values	[RFC 4180]	
JSON	JavaScript Object Notation	[RFC 8259]	(note: language independent)
YAML	Yet Another Markup Language	[RFC 9512]	(note: language independent)

4.4.2 Unit 3 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://killercoda.com/het-tanis/course/Ansible-Labs/02-Ansible-Host-File
- https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u3_worksheet(`.md`)
- 📥 u3_worksheet(`.txt`)

UNIT 3 RECORDING

Link: https://www.youtube.com/watch?v=acKjnxuGjVI

Discussion Post #1

Find a blog post or open source tool that focuses on IT inventory management.

1. What is meant by the term “inventory” in an IT context?
2. What are some of the issues with inventories in a company?
    - How have people attempted to overcome these issues?
3. What different formats of inventories can you find for IT management?
    - Why would formatting matter?

Discussion Post #2

You are a system administrator for a small company with ~100 total Linux systems. The security engineer approaches you and shows you vulnerabilities in your 110 total Linux systems. He then asserts, "without a good inventory, you cannot have security in the system."

1. Do you agree with him, why or why not?
    a. How do you plan to start to “true up” your inventories?
    b. How can you prevent this type of problem (if you think it is one) in the future?
2. We often say in engineering, “Or you can do nothing”. This speaks to the possibility of just accepting the situation and allowing a system to keep running.
    a. Can you do that in this situation, or must this be corrected? Why or why not?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

- IT Inventory

- File formats (be able to identify and parse them with your tools)
  - `.csv`
  - `.ini`
  - `.yaml`
- Grouping
- Variables (in relation to inventories)
- Ranges (and their usefulness)
  - `[01:50]`
  - `[01:50:2]`

Digging Deeper

1. Build some inventories like the ones here: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html
   a. Parse these down using ansible-inventory to see if you understand the syntax and formatting.
   b. Run through this lab for understanding: https://killercoda.com/het-tanis/course/Ansible-Labs/02-Ansible-Host-File

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.4.3 Unit 3 Lab

Making and Using Inventories

> **Note**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Unit 3 Lab on Killercoda

REQUIRED MATERIALS

- Lab Server Killercoda recommended
  - Rocky 9.6+ - ProLUG Lab
  - Or comparable Linux box
- `root` or `sudo` command access

Downloads

The lab has been provided for convenience below:

- 📥 u3_lab(`.txt`)
- 📥 u3_lab(`.md`)

Pre-Lab (Lab Setup)

If you're not doing the lab on Killercoda, make sure to follow the setup guide below.

PROLUG LAB SETUP

If you're using the ProLUG lab environment, run the following commands.

```
cd /root
cp -r /labs/automation/unit3/* /root
chmod 755 /root/*.sh
chmod 755 /root/*.py
```

PERSONAL LAB SETUP

Alternatively, for your own lab environment, clone down the necessary files from GitHub.

```
cd /root
git clone https://github.com/het-tanis/prolug-labs.git
cp prolug-labs/AutomationLabs/Unit3_Inventories/assets/* /root
```

Lab 🧪

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

If you do the killercoda Lab 3, just answer these questions. If you are doing the lab in the ProLUG environment, find the scripts in /labs/automation/unit3.

If you are doing the lab in the ProLUG environment, find the scripts in `/labs/automation/unit3`.

BASH EXECUTION

1. Execute script u3_script.sh and see if you can read the data

```
/root/u3_script.sh
```

- What values are shown?

2. What does the script look like in bash?

```
cat /root/u3_script.sh
```

- Could you modify this script to hit a different API endpoint and place the file in a different location?

3. Read the provided users.csv file

```
cat /root/users.csv
```

4. Parse out the data for just the first and the third fields

```
cat /root/users.csv | awk -F , '{print $1,$3}'
```

- Could you parse out only the first and second fields?
- Can you remove the header?

5. Regenerate the data for /root/users.csv

```
/root/u3_script_user_generator.sh
```

PYTHON EXECUTION

1. Run the u3_script.py and look at what it shows you.

```
/root/u3_script.py
```

- What are you shown?

2. Inspect the file and see if you can figure out what it was doing.

```
cat /root/u3_script.py
```

- Note: Modify with vi or vim. Can you make this show the lowest 10 items, ordered by magnitude?
- Can you generate a python script that parses the /root/users.csv file? (What resources might you use to do this?)

ANSIBLE EXECUTION

1. Run the u3_script.yml and look at what it shows you.

```
ansible-playbook /root/u3_script.yml
```

- What are you shown?
- Can you modify this output so show other interesting parts of the API calls?
- If you had to pull a specific field, could you do it

Again, you don't know how data might come to you in your organization, so this is an exercise in parsing things different ways.

2. Inspect your current inventory files.

```
cat /root/hosts
cat /root/hosts_example2
cat /root/hosts_example3
```

- What file type are these? (https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html)?
- What other file types might you use for inventories?
- Do you have a preference on how the data are formatted, or where the variables are located on these?
- Do you think some of this looks better formatted or do you prefer it as yaml?

3. Check the yaml versions of these files.

```
ansible-inventory -i /root/hosts --list -y
ansible-inventory -i /root/hosts_example2 --list -y
ansible-inventory -i /root/hosts_example3 --list -y
```

This is a very high level review of the many ansible-inventory commands. It is recommended that you parse and play with these files more, as the concepts will continue to be built on in later labs.

Note

If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

4.5 Unit 4 - Admin Commands && Ad-Hoc Commands

4.5.1 Unit 4

Admin Commands and One-Offs

Overview

This unit explains administration and one-off/ad-hoc commands. We will learn what triggers automation administration commands. We also clarify how we get into a state where we need to interact with our systems. Once we are in this state, we will learn how to interact with our automation, using one-off commands.

This unit continues to address the course objective to deploy various automation tools for engineering and operations activities. Our main responsibility on the job, is to perform admin and engineering tasks, but we ultimately want to use automation to properly address our roles. A second course objective being addressed is to maintain system configuration and remediate drift via automation. Remediation of drift is essentially keeping the computing environment in-line with user expectations.

Learning Objectives

1. What types of admin commands will we use : Commands that change existing and running systems with the goal of keeping systems at 99.999% uptime.
    - Observe - Check any system and note exactly how it is configured. Later, ensure nothing has changed, since the last check.
    - Benchmark - Perform a load lest to check how the system behaves / handles itself.
    - Tune - Change the system configuration then restart the related services for the changes to take effect.
2. What triggers our need to perform admin commands
    a. A system event occurs
        - the server stops responding
        - a log event triggers an alert
        - an event is triggered from an SIEM configured system (e.g. Kafka, SQS, Splunk)
    b. A user request is received
        - give access / change permissions
        - fix / configure the environment (as a one-off instead of redeploying)
    c. A security event occurs
        - Security findings from an GRC/ISO trigger patching
        - Security incidents requiring fixes to protect CIA
3. PPDIOO Deployment methodology
    - focus on Optimize & Operate stages

Key Terms and Definitions

Terms	Definitions
SIEM	Security Information and Event Monitoring
PPDIOO	Prepare, Plan, Design, Implement, Operate, Optimize
KTLO	Keeping the lights on
GRC	Governance, Risk, and Compliance teams
ISO	Information Security Officer
CIA	Confidentiality, Integrity, and Availability

4.5.2 Unit 4 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html
- https://killercoda.com/het-tanis/course/AutomationLabs/Unit4_Admin_Commands

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u4_worksheet(`.md`)
- 📥 u4_worksheet(`.txt`)
- 📥 u4_worksheet(`.pdf`)

UNIT 4 RECORDING

Link: https://www.youtube.com/watch?v=acKjnxuGjVI

Discussion Post #1

> **Scenario**
>
> Your company is on a normal 3 year refresh cycle for hardware.
> This means that they purchase ~1/3 of hardware in each year budget. You have 6 months until the next purchase but have been having storage capacity issues in some of your servers. Your team sits down and works out a plan to put in place some "Stop gap" measures to keep the system running until the next deployment.

1. Find an article that discusses what "stop gap" measures are.
2. What is your understanding of the term "stop gap".
3. What are some things you would be doing to help a system that has no way to add capacity before a certain time to continue operations between here and there?
    - What is meant by the term "draconian measures" in this context?

Discussion Post #2

> **Scenario**
>
> You and your security team have an accurate inventory after last week's misunderstanding. You have 110 servers currently in your inventory. What are some methods you can use to verify those systems are operational?

1. How might you "touch" those servers every day?
2. How might you plan to keep that inventory updated automatically?
3. How might you monitor those servers? (What tools can you find that would do this?)
4. How might you present a report for these servers (to your team or others?)

Info

Submit your input by following the link. The discussion posts are done in Discord Forums.
Link to Discussion Posts

Definitions

- One-off
- Ad-hoc
- Admin Commands
- Stop gap fix
    - How does this relate to a full implementation?
    - How does this relate to a systemic system problem?
    - How does this relate to a systemic capacity problem?
- System load (and utilization ((always as a percentage)))
    - Averages
    - High water mark
    - Low water mark
    - Spiking
    - Capacity

Digging Deeper

1. Read this article about Ansible ad-hoc commands:
   https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html
    a. What did you learn about this that you didn't know?
    b. How are you going to use this in your current or future automations?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.5.3 Unit 4 Lab

Admin commands and one-offs

> **Note**
>
> If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

RESOURCES / IMPORTANT LINKS

- Unit 4 on Killercoda Lab

REQUIRED MATERIALS

- Lab Server Killercoda recommended
  - Rocky 9.6+ - ProLUG Lab
  - Or comparable Linux box
- `root` or `sudo` command access

Downloads

The lab has been provided for convenience below:

- 📥 u4_lab(`.txt`)
- 📥 u4_lab(`.md`)

Pre-Lab (Lab Setup)

If you're not doing the lab on Killercoda, make sure to follow the setup guide below.

PROLUG LAB SETUP

If you're using the ProLUG lab environment, run the following commands.

```
cd /root
cp -r /labs/automation/unit3/* /root
chmod 755 /root/*.sh
chmod 755 /root/*.py
```

Lab 🧪

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

If you do the killercoda Lab 4, just answer these questions. If you are doing the lab in the ProLUG environment, find the scripts in `/labs/automation/unit4`.

In the ProLUG lab, you must edit `/root/hosts` to point at your correct environment based on which “auto{x}” server you have connected to.

Server	Hostgroup	TargetNodes
auto1	[webservers]	target1-1,target1-2
auto2	[webservers]	target2-1,target2-2
auto3	[webservers]	target3-1,target3-2
auto4	[webservers]	target4-1,target4-2
auto5	[webservers]	target5-1,target5-2

Warning

This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

Note

If you are unable to finish the lab in the ProLUG lab environment we ask you `reboot` the machine from the command line so that other students will have the intended environment.

4.6 Unit 5 - Environment and Local Variables

4.6.1 Unit 5 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Inventories: https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html
- Special Variables: https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html
- Variable Precedence: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#understanding-variable-precedence
- Templates (Jinja2): https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_templating.html

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u5_worksheet(`.md`)
- 📥 u5_worksheet(`.txt`)
- 📥 u5_worksheet(`.pdf`)

UNIT 5 RECORDING

Link: https://www.youtube.com/watch?v=LQ9aXRU3vts

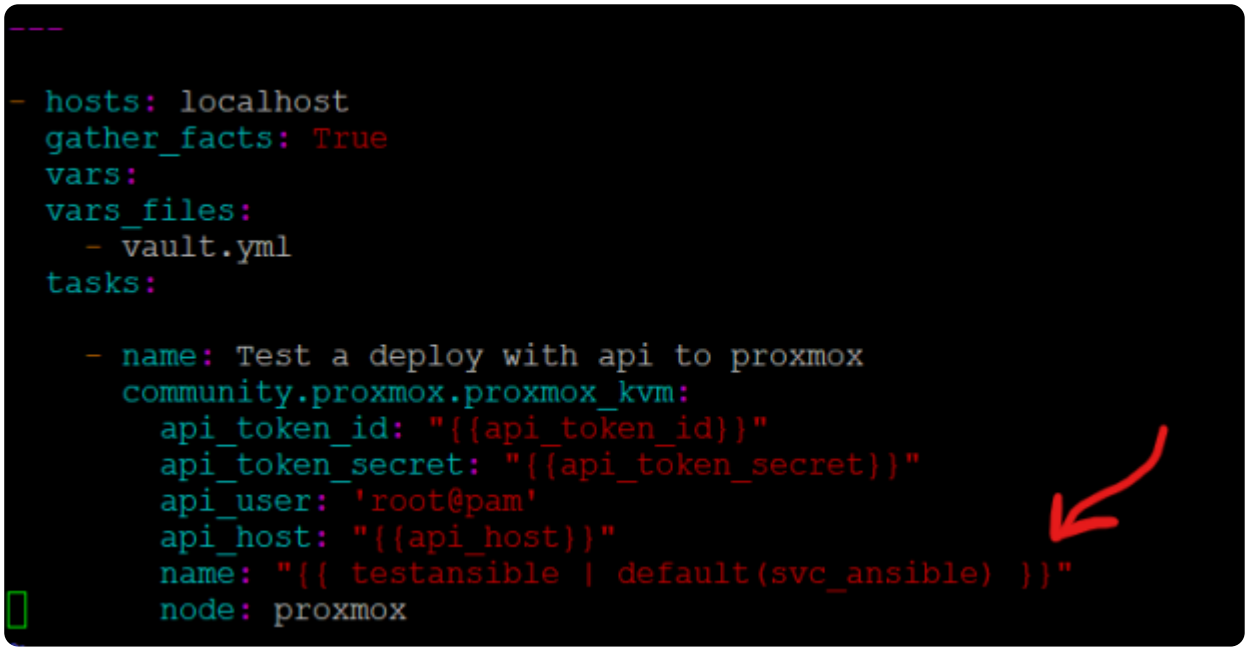
Discussion Post #1

You know about variable precedence and have decided to study it for your Ansible playbooks. Read https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html and answer the following questions:

1. What is variable precedence and why should it matter?
2. What does it mean to register a variable, and how is that variable used in the playbook?
3. How might variables be useful at the end of an automation, in relation to reporting out what happened in the playbook?

Discussion Post #2

You've stumbled on a playbook and you're trying to figure out what the following line means:



```
---

- hosts: localhost
  gather_facts: True
  vars:
  vars_files:
    - vault.yml
  tasks:

    - name: Test a deploy with api to proxmox
      community.proxmox.proxmox_kvm:
        api_token_id: "{{api_token_id}}"
        api_token_secret: "{{api_token_secret}}"
        api_user: 'root@pam'
        api_host: "{{api_host}}"
        name: "{{ testansible | default(svc_ansible) }}"
        node: proxmox
```

You have reviewed Jinja2 filters https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_filters.html#providingdefault-values and think you have a good handle on what is happening.

1. What is the variable name being called?
2. What is the default value if that variable does not exist or is not populated?
3. What is the reason this might be nice in your executions if you want them always to complete?
4. Is there a danger to always setting default values?
    a. Or another way to ask that, is there a tradeoff between always finishing and sometimes having incorrectly set values?
    b. Where will you use these in your automations?

Info

Submit your input by following the link. The discussion posts are done in Discord Forums.
Link to Discussion Posts

Definitions

- Workflow - Execution
    a. Before
    b. During
    c. After
- Variables
    - Special Variables
    - Precedence
- Environment Files
- Jinja2
- Templates

Digging Deeper

Work through the following labs to practice the topics from this week's presentation.

1. Ansible Facts: https://killercoda.com/het-tanis/course/Ansible-Labs/12-Ansible-System-Facts-Grouping
2. Ansible Vault: https://killercoda.com/het-tanis/course/Ansible-Labs/10-Ansible-Vaulting-Password-and-Variables
3. Ansible API Calls: https://killercoda.com/het-tanis/course/HashicorpLabs/004-vault-read-secrets-ansible
4. Stamping servers: https://killercoda.com/het-tanis/course/Ansible-Labs/07-Ansible-Playbook-Jinja-Templates
5. Reporting back values: https://killercoda.com/het-tanis/course/AnsibleLabs/08-Ansible-Playbook-Jinja-Reporting
6. Generating CSV for management: https://killercoda.com/het-tanis/course/Ansible-Labs/19-Ansible-csv-report

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.6.2 Unit 5 Lab

Environment and Local Variables in Systems

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 5 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.7 Unit 6 - Automating Docker Builds

4.7.1 Unit 6

Automating Docker Builds

Overview

This unit introduces containerization and the tools used to build, deploy, and manage containers in an automated fashion. We learn the principles of release engineering and how they apply to infrastructure automation. Learners will understand the relationship between container image creation, container orchestration, and infrastructure-as-code tools. The core challenge we address is this: infrastructure engineering teams frequently struggle with reliably recreating identical environments and doing so in a timely manner. Through automation, we solve this problem using tools like Docker, Packer, Terraform, and Apptainer.

This unit continues to address the course objective to deploy various automation tools for engineering and operations activities. Specifically, this unit focuses on how we can reliably and repeatably build container images and deploy containerized environments across infrastructure at scale. We apply Google SRE (Site Reliability Engineering) principles to ensure consistency, reliability, and automation in our release pipelines. By the end of this unit, you will understand how to codify infrastructure requirements, automate environment creation, implement version control for infrastructure, and enable repeatable deployments.

Learning Objectives

1. Container fundamentals and use cases
    - Docker images and container runtime environments
    - Container images as versioned artifacts using semantic versioning
    - The relationship between containers and infrastructure automation
    - Why containers matter for repeatable deployments and use cases (BYOE, reproducible science, static environments, legacy code, custom software)
    - Understand isolated execution environments with all required dependencies
2. Release Engineering Principles
    - Understanding release engineering as the discipline of building and managing software releases in a controlled, reproducible manner
    - Code base management, code changes, and version control strategies
    - Build configuration and the build process (building, branching, and testing)
    - How release engineering principles apply directly to infrastructure automation
3. Container image building tools and decision-making
    - Packer: Building Docker images, VM images, and cloud machine images in an automated fashion
    - Apptainer (formerly Singularity): Lightweight container runtime for HPC and scientific computing
    - When and why to use each tool based on use cases (general-purpose vs. HPC/scientific workloads)
    - Comparison framework for evaluating container technologies
4. Infrastructure-as-Code integration and CI/CD pipelines
    - Using Terraform to deploy container infrastructure
    - Terraform providers and their role in managing platforms and services
    - CI/CD pipelines and build triggers: GitHub Actions, manual triggers, and event-driven systems (Kafka, Event Bridge, Opensearch, Splunk)
    - Testing container images for function and security
    - Integration with version control and automation workflows
    - Tagging and versioning artifacts for deployment readiness

5. Release strategies and deployment patterns
    - Release engineering practices for containers (consistency, reliability, automation)
    - Canary deployments and safe rollout patterns
    - Maintaining consistency across environments
    - Implementing reliable and repeatable releases at scale

Key Terms and Definitions

Terms	Definitions
Docker	Container platform for building, shipping, and running applications
Container Image	A lightweight, standalone, executable package containing code and dependencies
Packer	Tool for creating machine images and container images in an automated fashion
Apptainer	Container platform designed for HPC and scientific computing workloads
Terraform	Infrastructure-as-Code tool for provisioning and managing infrastructure
Release Engineering	Discipline focused on building and deploying software reliably and repeatedly
Release	A versioned software package ready for deployment and distribution
Code base	The complete source code for a project or application
Build Configuration	Rules and instructions defining how code is compiled, packaged, and tested
CI/CD	Continuous Integration / Continuous Deployment automated build and deployment pipelines
GitHub Actions	CI/CD automation platform for triggering builds on code changes and automating workflows
Provisioning	Configuring and setting up systems and infrastructure, including provisioning containers during builds
Canary Deployment	Deployment strategy that rolls out changes to a small subset before full release
Container Artifact	A versioned, immutable container image ready for deployment and distribution
Semantic Versioning	Versioning scheme (Major.Minor.Patch) for tracking container image changes and compatibility
Build Trigger	Event or action that initiates an automated container image build (GitHub Actions, manual, event-driven systems like Kafka)
Event-Driven Architecture	Systems triggered by events from external sources (Kafka, Event Bridge, Opensearch, Splunk) rather than on-demand
Image Testing	Validation of container images for functional correctness and security vulnerabilities before deployment

4.7.2 Unit 6 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- Apptainer
  - Intro to Apptainer
- Intro to Packer
  - Packer Tutorial Library
  - Packer with GitHub Actions
  - Provisioning
- Terraform with Docker
- Release Engineering
- "Canarying" Releases

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u6_worksheet(`.md`)
- 📥 u6_worksheet(`.txt`)
- 📥 u6_worksheet(`.pdf`)

UNIT 6 RECORDING

Link: https://www.youtube.com/watch?v=1OQXN5_oyu0

Discussion Post #1

Your infrastructure engineering teams have been experiencing problems re-creating environments. The main problems have been around reliably building the exact same environment and also making those builds happen in a timely manner. Read https://sre.google/sre-book/release-engineering/ and https://sre.google/workbook/canarying-releases/ to answer the following questions.

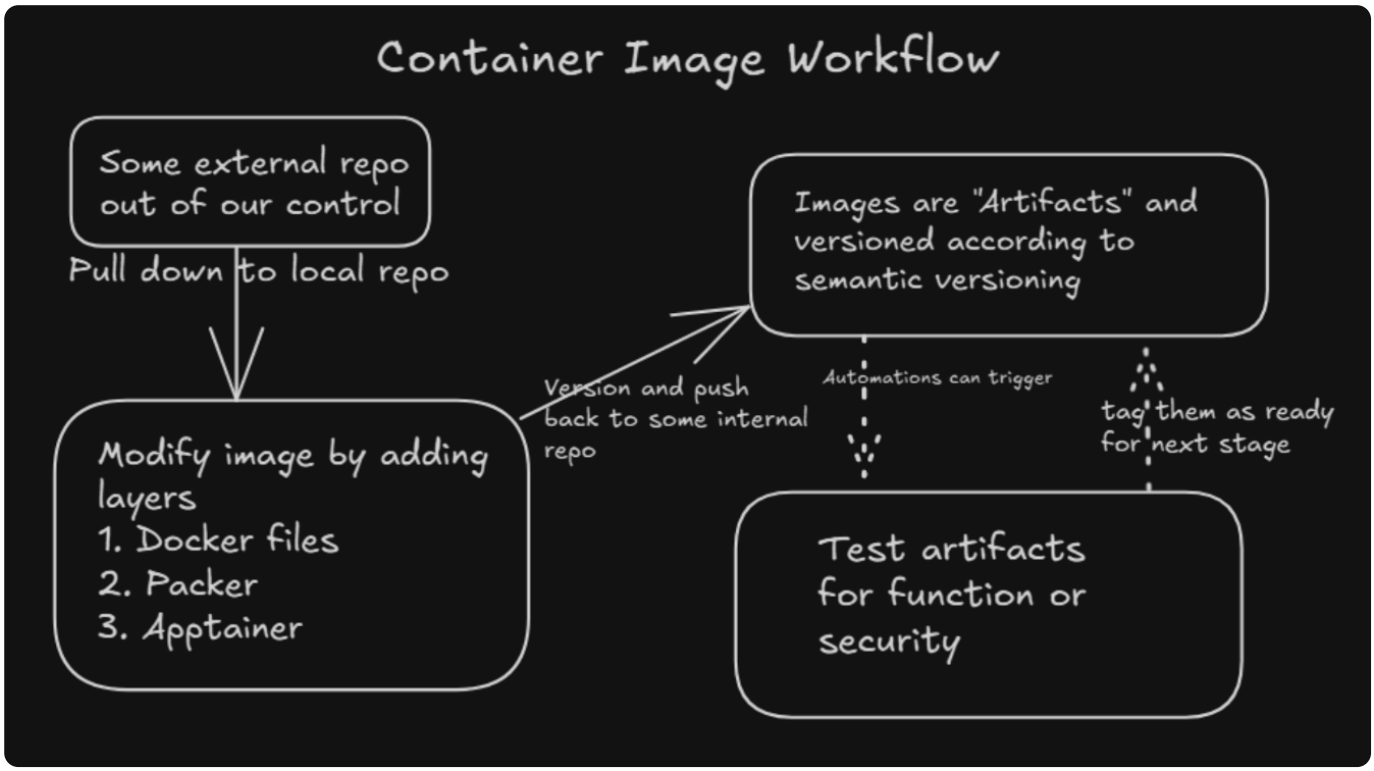
1. What is release engineering?
2. What are the release engineering principles?
3. How do the tools we've discussed this week, Apptainer, Packer, Terraform, or even Ansible fit into these topics?

Discussion Post #2

Your team is trying to decide between the Apptainer and Packer tools for container deployments. You've been tasked with making the decision between the two packages.

Read the following: https://developer.hashicorp.com/packer/docs/intro and https://apptainer.org/docs/user/latest/introduction.html#why-use-apptainer

1. Can you describe Apptainer and Packer?
2. How would you make the decision between the two of these tools? (You may want to make a table)
    a. What do they both do?
    b. What do only one or the other do?
    c. What are the strengths and weaknesses of each?
3. Modify or fix the drawing to show how your team will deploy containers.



Info

Submit your input by following the link. The discussion posts are done in Discord Forums.
Link to Discussion Posts

Definitions

- Docker Images
- Docker processes
- Container/Runtime Environment
- CI/CD

- Release engineering
    - Releases
    - Code base
    - Code changes
    - Build configuration
        - Building
        - Branching
        - Testing

Digging Deeper

Read about Terraform providers here: https://developer.hashicorp.com/terraform/language/providers

1. What are Terraform providers?
2. How would we find a specific provider?

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.7.3 Unit 6 Lab

Automating Docker Builds

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 6 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.8 Unit 7 - Automating Docker Environments

4.8.1 Unit 7 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://sre.google/sre-book/release-engineering/
- https://sre.google/workbook/canarying-releases/
- https://docs.docker.com/build/building/best-practices/

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- u7_worksheet(`.md`)
- u7_worksheet(`.txt`)
- u7_worksheet(`.pdf`)

UNIT 7 RECORDING

Link: https://www.youtube.com/watch?v=dfQsb8GJA3k

Discussion Post #1

You are the team lead of a small Linux team maintaining 700 servers. Your management is always pushing for getting more from the systems and has been asking you to explore container environments, especially in the cloud. You read some blog posts about services and decide to write out your notes: https://aws.amazon.com/blogs/containers/amazon-ecs-vs-amazon-eks-making-sense-of-aws-container-services/

1. What are the major differences between container environments and Kubernetes orchestrated environments?
    a. Why might you just want a containerized environment?
    b. Why might you want an orchestrated environment?
    c. Can you compare and contrast them?

Discussion Post #2

Your team is having problems with a deployment. This is the code snippet they are using.

1. What is the provider they are using?
2. How many docker instance are they trying to run, and what are their names?
    a. What ports are they going to be running on?
3. Your team is having problems executing this and have brought it to you. What might you check, or do with terraform to try to resolve the issue?
    a. If it's telling you there are no providers?
    b. If it's saying there's a syntax problem (how can you find it)?
    c. If there are no resources created?

```
terraform {
    required_providers {
        docker = {
            source = "kreuzwerker/docker"
            version = "~> 2.13.0"
        }
    }
}

provider "docker" {}

resource "docker_image" "nginx" {
    name = "nginx:latest"
    keep_locally = false
}

resource "docker_container" "nginx8080" {
    image = docker_image.nginx.latest
    name = "nginx8080"
    ports {
        internal = 80
        external = 8080
    }
}
resource "docker_container" "nginx8081" {
    image = docker_image.nginx.latest
    name = "nginx8081"
    ports {
        internal = 80
        external = 8081
    }
}
resource "docker_container" "nginx8082" {
    image = docker_image.nginx.latest
    name = "nginx8082"
    ports {
        internal = 80
        external = 8082
    }
}
```

Info

Submit your input by following the link. The discussion posts are done in Discord Forums.
Link to Discussion Posts

Definitions

- Pipeline
- Inotify-tools

Digging Deeper

1. What are some of the best practices around container deployments? https://docs.docker.com/build/building/best-practices/
2. Why might we not want to ever run the "latest" tag in production?

3. Why should an application be run as non-root?
4. What is it to be an immutable container?
5. What is it to be a sandboxed container?
    a. What does this mean from the kernel standpoint

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.8.2 Unit 7 Lab

Automating Docker Environments

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 7 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.9 Unit 8 - Automating K8s Environments

4.9.1 Unit 8: Automating Kubernetes Environments

Overview

You've reached a critical juncture in the ProLUG Linux Automation Engineering Course. At approximately the halfway point of the course, you're now ready to tackle one of the most significant domains in modern infrastructure automation: **Kubernetes orchestration**.

So far, you've built a solid foundation in automation fundamentals - from Bash scripting to Ansible playbooks, from container builds to Docker environment automation. These skills have prepared you for this moment. This unit represents the convergence of all your previous learning: applying comprehensive automation principles to Kubernetes, the de facto standard for container orchestration in enterprises, cloud environments, and increasingly, mid-sized organizations.

The critical insight of this unit is this: **Kubernetes doesn't automate itself.** Organizations require reliable, repeatable mechanisms to stand up environments, deploy applications, test infrastructure changes, manage resource lifecycles, and maintain compliance records. This unit teaches you how to provide these mechanisms using Ansible - the same tool your team already knows and trusts. By mastering this unit, you become capable of solving real infrastructure problems that directly impact your organization's reliability and velocity.

Learning Objectives

1. **Distinguish Between Container Deployment Approaches**
    - Understand the differences between running containers directly on Linux vs. full Kubernetes orchestration
    - Recognize when simpler container solutions suffice vs. when Kubernetes is warranted
    - Consider resource constraints, operational maturity, and team expertise in architectural decisions
2. **Automate Kubernetes Operations Using Ansible**
    - Install and configure Ansible Kubernetes modules for API interaction
    - Write idempotent Ansible playbooks that interact with the Kubernetes API
    - Replicate `kubectl` operations programmatically at scale
    - Create dynamic Kubernetes resources (namespaces, pods, services) via Ansible
    - Implement proper error handling and state verification
3. **Design and Implement Modern Deployment Strategies**
    - Understand ephemeral pod architecture and why IP addresses are ephemeral but non-critical
    - Leverage Kubernetes services as reverse proxies for reliable routing to temporary resources
    - Implement blue/green deployments for zero-downtime application updates
    - Implement canary deployments for staged, risk-mitigated application rollouts
    - Understand the release engineering principles underlying each deployment pattern
4. **Build Self-Service Infrastructure Platforms**
    - Design "push-button" deployment solutions using Ansible Automation Platform
    - Enable development teams to provision test environments without direct cluster access
    - Implement parameterized automation driven by environment variables
5. **Automate Complete Resource Lifecycles**
    - Create Kubernetes resources dynamically within automation workflows
    - Test deployed resources programmatically to verify functionality
    - Implement comprehensive resource cleanup and deletion procedures
    - Log and audit all infrastructure changes for compliance and capacity planning
    - Generate reports on resource creation, modification, and destruction for organizational records

Key Concepts

Concept	Definition
Ephemeral Pods	Kubernetes pods with temporary IP addresses assigned at runtime; the non-permanence of IPs is mitigated by Kubernetes Services acting as stable endpoints. This architecture enables safe, frequent deployments without infrastructure instability.
Kubernetes Service	An abstraction that defines a logical set of pods and a policy by which to access them; acts as an internal load balancer and stable reverse proxy, decoupling clients from ephemeral pod IP addresses.
Blue/Green Deployment	A release engineering strategy where two identical, fully provisioned environments (blue and green) exist simultaneously; traffic switches completely from one to the other, enabling instant rollback if issues occur. Kubernetes Services make this pattern practical by routing traffic to either environment.
Canary Deployment	A staged rollout strategy where new application versions are deployed to a small percentage of traffic first, allowing validation before full rollout; leverages ephemeral pod characteristics to gradually replace old pods with new ones.
Idempotent Automation	Automation that produces the same result regardless of how many times it executes; critical for Kubernetes where rerunning playbooks must be safe and verify current state before making changes.
Ansible Kubernetes Module	Ansible collection modules (kubernetes.core) that interact directly with the Kubernetes API, enabling declarative infrastructure-as-code for Kubernetes resource management without shell commands.
Namespace	A logical cluster subdivision in Kubernetes providing isolation of resources, networking policies, and RBAC controls; enables multi-tenant environments and environment separation (dev/staging/production).
Resource Manifest	YAML specification defining a Kubernetes resource (pod, service, deployment, etc.); Ansible modules convert these manifests into API calls to instantiate resources.
Release Engineering	The practice of managing how software versions move through environments, including build processes, testing phases, deployment strategies, and rollback procedures; Kubernetes automation implements release engineering practices at infrastructure scale.
Infrastructure as Code (IaC)	The practice of managing infrastructure through version-controlled, executable code rather than manual procedures; Ansible playbooks treating Kubernetes as code resources enable reproducibility, auditability, and team collaboration.

4.9.2 Unit 8 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://sre.google/sre-book/release-engineering/
- https://sre.google/workbook/canarying-releases/
- https://docs.ansible.com/projects/ansible/latest/collections/kubernetes/core/k8s_module.html
- https://killercoda.com/het-tanis/course/Automation-Labs/Unit8_Kubernetes_Automation
- https://killercoda.com/het-tanis/course/Kubernetes-Labs/blue-green-deployments
- https://killercoda.com/het-tanis/course/Kubernetes-Labs/canary-deployments

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- u8_worksheet(`.md`)
- u8_worksheet(`.txt`)
- u8_worksheet(`.pdf`)

UNIT 8 RECORDING

Link: https://www.youtube.com/watch?v=C4dzRI_bFCQ

Discussion Post #1

Your infrastructure engineering teams have been experiencing problems re-creating environments. The main problems have been around reliably building the exact same environment and also making those builds happen in a timely manner. Last week you read https://sre.google/sre-book/release-engineering/ and https://sre.google/workbook/canarying-releases/ to answer the following questions.

1. How is a Kubernetes environment different, in release context from a virtual or physical server environment?
    a. What does the term ephemeral mean, and are all Kubernetes environments ephemeral? Why or why not?
    b. How does blue/green or canary deployments help maintain uptime in release management?
        i. What is the use case of each?

Discussion Post #2

Your team needs to develop a “push button” solution for deploying Kubernetes for a number of different development teams. You have a development cluster that they can use and capacity is not a consideration for this discussion.

1. What pieces of information will you need to supply on your side for this type of automation?
2. What pieces of information will the team need to supply on their side for each deployment?
    a. How can the different dev teams feed this into your automation?
    b. What do you prefer?
3. What are some potential default variables you would want to use for this deployment?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

- Kubernetes
  - Namespaces
  - Pods
  - Deployments
  - Labels

Digging Deeper

1. Read other use cases for the Kubernetes module in Ansible: https://docs.ansible.com/projects/ansible/latest/collections/kubernetes/core/k8s_module.html
   a. How might you use these other use cases in your environments?
   b. Do you have another tool you'd use instead of this? Does it have the same, or more functionality?
2. Deployment practices with Kubernetes blue/green and canary labs:
   a. https://killercoda.com/het-tanis/course/Kubernetes-Labs/blue-green-deployments
   b. https://killercoda.com/het-tanis/course/Kubernetes-Labs/canary-deployments

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.9.3 Unit 8 Lab

Automating K8s Environments

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 8 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.10 Unit 9 - Build && Deploy Linux Systems

4.10.1 Unit 9

Build and Deploy Linux Systems

Overview

You've reached an inflection point in the ProLUG Linux Automation Engineering Course. The units up to this point have prepared you to automate individual components: containers, orchestration platforms, configuration files, and infrastructure resources. This unit brings these concepts together to address one of the most critical challenges in enterprise infrastructure: **building and deploying complete Linux systems at scale with consistency, repeatability, and security**. This unit introduces bare-metal provisioning, immutable infrastructure patterns, and the tools that enable organizations to manage hundreds or thousands of Linux systems from centralized, version-controlled configurations.

The core problem this unit solves is configuration drift and inconsistent environments. When systems are managed manually or through ad-hoc scripts, they diverge over time - package versions differ, configuration files are modified, services are started or stopped inconsistently. These differences create reliability problems, security vulnerabilities, and operational complexity. This unit teaches you how to eliminate drift by treating system images as immutable artifacts, applying infrastructure-as-code principles to entire operating systems, and deploying nodes that are identical, ephemeral, and disposable. These practices align directly with NIST Special Publication 800-223 guidance on High-Performance Computing security and the immutable infrastructure principles championed by Google SRE.

By the end of this unit, you will understand how to use Warewulf for bare-metal provisioning, Apptainer for containerized OS images, and Ansible for automated system configuration within chroot environments. You'll be able to build reproducible Linux system images, deploy them across physical or virtual infrastructure, and implement immutable deployment patterns that enable rapid rollback, consistent security posture, and operational confidence at scale. This is infrastructure automation applied to the operating system itself - the foundation that everything else depends on.

Learning Objectives

1. **Understand Bare-Metal Provisioning and Diskless Boot Architecture**
    - Comprehend how PXE (Preboot Execution Environment) and iPXE enable network-based operating system delivery
    - Understand the Warewulf provisioning framework architecture: image repository, provisioning server, overlay system, and boot process
    - Recognize when diskless booting is appropriate (HPC clusters, ephemeral compute nodes, high-security environments)
    - Implement network boot infrastructure for delivering OS images to bare metal servers
2. **Build and Manage Containerized Linux System Images**
    - Use Apptainer to create lightweight, single-file container images optimized for system deployment
    - Import OS images from container registries (Docker Hub, GitHub Container Registry) using Warewulf
    - Extract and prepare container filesystems as root OS images for deployment
    - Understand the differences between Docker (application containers) and Apptainer (system containers)
    - Manage image versioning and distribution for large-scale deployments
    - Locate and inspect Warewulf image storage locations for troubleshooting and customization
3. **Automate System Configuration Using Ansible and Chroot Environments**
    - Configure Ansible to interact with chroot environments using the `chroot` connection plugin
    - Write idempotent Ansible playbooks for package installation, service configuration, and system state management
    - Test and validate system configurations locally before deploying to production nodes
    - Implement repeatable, version-controlled system customization workflows
    - Troubleshoot Ansible execution within chroot environments (DNS resolution, package repositories, network access)

4. **Implement Immutable Infrastructure Patterns**
    - Treat deployed nodes as ephemeral resources that are rebuilt rather than patched
    - Separate base OS images from environment-specific overlays (system overlays and runtime overlays)
    - Implement the "build new, deploy, destroy old" pattern for system updates
    - Understand Warewulf terminology: image repository, system overlays, runtime overlays, and wwctl command language
    - Design deployment architectures where configuration changes trigger new image builds
    - Maintain image versioning and rollback capabilities for rapid recovery

Key Terms and Definitions

Term	Definition
Warewulf	Open-source bare-metal provisioning and lifecycle management framework for HPC and enterprise Linux systems; delivers operating system images via network boot and manages node configuration through overlays
Apptainer	Lightweight container format (formerly Singularity) optimized for scientific computing and system deployment; provides single-file, immutable containers without requiring daemon processes
PXE Boot (Preboot Execution Environment)	Industry-standard network booting protocol (Intel specification) enabling systems to boot operating systems from network-delivered images rather than local disks; foundation for diskless computing
iPXE	Enhanced open-source PXE implementation supporting HTTP/HTTPS boot sources, scripting capabilities, and advanced network protocols; extends standard PXE functionality
Chroot (Change Root)	Unix/Linux operation that changes the apparent root directory for a process and its children, creating an isolated filesystem namespace; enables system configuration testing without deployment
Ansible Chroot Connection	Ansible connection plugin (`ansible_connection=chroot`) enabling playbook execution within chroot environments; allows configuration management of system images before deployment
System Overlay	Warewulf concept for persistent modifications applied to base OS images; includes configuration files, packages, and customizations shared across node groups
Runtime Overlay	Warewulf concept for ephemeral, node-specific configurations applied at boot time; enables per-node customization while maintaining shared base images
Immutable Infrastructure	Infrastructure paradigm where deployed systems are never modified in place; updates require building new images and redeploying nodes, enabling consistent state and rapid rollback
Diskless Booting	Architecture where compute nodes run operating systems entirely from RAM without local disk storage; OS image delivered via network, enabling stateless, disposable nodes
wwctl	Warewulf control utility providing command-line interface for image management, node provisioning, overlay configuration, and cluster administration
Configuration Drift	Gradual divergence of system configurations from their intended state due to manual changes, ad-hoc patches, and inconsistent deployments; eliminated by immutable infrastructure
Bare-Metal Provisioning	Process of installing and configuring operating systems directly on physical hardware without virtualization; requires network boot infrastructure and automated deployment tools
Idempotent Automation	Automation that produces identical results regardless of how many times it executes; critical for reliable system deployment where playbooks must safely rerun without unintended side effects

4.10.2 Unit 9 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf
- https://en.wikipedia.org/wiki/Chroot
- https://docs.ansible.com/projects/ansible/latest/collections/community/general/chroot_connection.html

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u9_worksheet(`.md`)
- 📥 u9_worksheet(`.txt`)
- 📥 u9_worksheet(`.pdf`)

UNIT 9 RECORDING

Link: https://www.youtube.com/watch?v=ef8Dzk5wVm8

Discussion Post #1

Read sections “Container Usage” and “Diskless Booting” on pages 8 and 9 of this doc: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf

1. What are the main security concerns given about using containers in HPC?
2. What is the PXE or IPXE protocol defined for diskless booting?

Discussion Post #2

Read about chrooted environments here: https://en.wikipedia.org/wiki/Chroot and the Ansible connector here: https://docs.ansible.com/projects/ansible/latest/collections/community/general/chroot_connection.html

1. What are the uses of a chrooted environment? (there are 5 on Wikipedia)
2. How might using Ansible to build chrooted images help improve your build process?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

- Warewulf terminology
  - Image repository
  - System Overlays
  - Runtime Overlays
  - Wwctl language

Digging Deeper

1. Read other parts of this doc for more HPC understanding: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf
    a. What are the components on the drawing on page 3 of doc (pg. 11 in the webviewer)

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.10.3 Unit 9 Lab

Build and Deploy Linux Systems

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 9 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.11 Unit 10 - Harden Linux Systems

4.11.1 Unit 10 Worksheet

Instructions

Fill out the worksheet as you progress through the lab and discussions. Hold your worksheets until the end to turn them in as a final submission packet.

RESOURCES / IMPORTANT LINKS

- https://github.com/ansible-lockdown
- https://www.mindpointgroup.com/blog/stig-vs-cis-part-1-the-anatomy-of-baselines-and-compliance?lockdownenterprise
- https://www.mindpointgroup.com/blog/stig-vs-cis-part-2-selecting-the-best-baseline-for-your-business?lockdownenterprise
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf

Downloads

The worksheet has been provided below. The document(s) can be transposed to the desired format so long as the content is preserved. For example, the `.txt` could be transposed to a `.md` file.

- 📥 u10_worksheet(`.md`)
- 📥 u10_worksheet(`.txt`)
- 📥 u10_worksheet(`.pdf`)

UNIT 10 RECORDING

Link: https://www.youtube.com/watch?v=FKpkx4-lBuI

Discussion Post #1

> **Scenario**
>
> Your team uses Ansible and needs to secure one of the images you have been working to build in Rocky 9. You have decided to use https://github.com/ansible-lockdown ansible Lockdown for STIG remediation.

1. Where is this tool designed to be run?
    a. Why is this going to cause you a problem?
2. How will you go about remediating problems in your environment, if you have to run this in a chrooted environment?

Discussion Post #2

Read these blog posts about CIS and STIG compliance: https://www.mindpointgroup.com/blog/stig-vs-cis-part-1-the-anatomy-of-baselines-and-compliance?lockdownenterprise https://www.mindpointgroup.com/blog/stig-vs-cis-part-2-selecting-the-best-baseline-for-your-business?lockdownenterprise

1. Why might you want to choose one over the other?
2. Which version of baseline tool aligns with your current industry, or the industry you're wanting to work in?

> **Info**
>
> Submit your input by following the link. The discussion posts are done in Discord Forums.
> Link to Discussion Posts

Definitions

- Warewulf terminology
    - Images
    - Overlays
        - System
        - Runtime
- Chrooted Enviroment
- Stigs
    - Ansible Lockdown (what is this?)
    - OpenSCAP tooling

Digging Deeper

1. Read other parts of this doc for more HPC understanding: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf
2. What are the components on the drawing on page 3 of doc (pg. 11 in the web viewer)

Reflection Questions

1. What questions do you still have about this week?
2. How are you going to use what you've learned in your current role?

4.11.2 Unit 10 Lab

Harden Linux Systems

This lab is designed to have the engineer verify and execute their automation tools to interact with the OS in a controlled fashion.

Follow this link and find the lab for Unit 10 to complete it.

> **Warning**
>
> This lab is designed to be run in the killercoda environment and will take significant user tooling to change over to their own environment. This is not supported in this run of the course but the learner is welcome to work with it and tool it over for that purpose as time permits.

4.12 Course Resources

4.12.1 Course Resources

This is a comprehensive list of all external resources used in this course.

Unit 1 - Automation tools installation and execution

- https://killercoda.com/het-tanis/course/Automation-Labs/Unit1_Tools
- https://youtu.be/wyVhGtFFYIQ
- https://serverlessland.com/patterns
- https://killercoda.com/het-tanis/course/Ansible-Labs
- Kafka (event bus) Blogs
- Monitoring
- Event Driven Architecture

Unit 2 - Interacting with the Operating system

- https://youtu.be/Xsz7MXJPU58
- Lab: Interacting with the OS
- Killercoda recommended)
- Unit 2 Lab on Killercoda

Unit 3 - Making and using inventories

- Killercoda recommended
- Unit 3 Lab on Killercoda
- https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html
- https://killercoda.com/het-tanis/course/Ansible-Labs/02-Ansible-Host-File

Unit 4 - Admin commands and one-offs

- Killercoda recommended
- Unit 4 on Killercoda Lab
- https://www.youtube.com/watch?v=acKjnxuGjVI
- https://killercoda.com/het-tanis/course/AutomationLabs/Unit4_Admin_Commands
- https://docs.ansible.com/ansible/latest/command_guide/intro_adhoc.html

Unit 5 - Environment and Local Variables in systems

- https://killercoda.com/het-tanis/course/Ansible-Labs/19-Ansible-csv-report
- https://killercoda.com/het-tanis/course/AnsibleLabs/08-Ansible-Playbook-Jinja-Reporting
- https://killercoda.com/het-tanis/course/Ansible-Labs/07-Ansible-Playbook-Jinja-Templates
- https://killercoda.com/het-tanis/course/HashicorpLabs/004-vault-read-secrets-ansible
- https://killercoda.com/het-tanis/course/Ansible-Labs/10-Ansible-Vaulting-Password-and-Variables
- https://killercoda.com/het-tanis/course/Ansible-Labs/12-Ansible-System-Facts-Grouping
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_filters.html#providingdefault-values

- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html
- https://www.youtube.com/watch?v=LQ9aXRU3vts
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_templating.html
- https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_variables.html#understanding-variable-precedence
- https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

Unit 6 - Automating Docker Builds

- https://developer.hashicorp.com/terraform/language/providers
- https://apptainer.org/docs/user/latest/introduction.html#why-use-apptainer
- https://developer.hashicorp.com/packer/docs/intro
- https://www.youtube.com/watch?v=1OQXN5_oyu0
- "Canarying" Releases
- Release Engineering
- Terraform with Docker
- Provisioning
- Packer with GitHub Actions
- Packer Tutorial Library
- Intro to Packer
- Intro to Apptainer
- Apptainer

Unit 7 - Automating Docker environments

- https://aws.amazon.com/blogs/containers/amazon-ecs-vs-amazon-eks-making-sense-of-aws-container-services/
- https://www.youtube.com/watch?v=dfQsb8GJA3k
- https://docs.docker.com/build/building/best-practices/
- https://sre.google/workbook/canarying-releases/
- https://sre.google/sre-book/release-engineering/

Unit 8 - Automating K8s environments

- https://www.youtube.com/watch?v=C4dzRI_bFCQ
- https://killercoda.com/het-tanis/course/Kubernetes-Labs/canary-deployments
- https://killercoda.com/het-tanis/course/Kubernetes-Labs/blue-green-deployments
- https://killercoda.com/het-tanis/course/Automation-Labs/Unit8_Kubernetes_Automation
- https://docs.ansible.com/projects/ansible/latest/collections/kubernetes/core/k8s_module.html
- this link

Unit 9 - Build and Deploy Linux systems

- https://www.youtube.com/watch?v=ef8Dzk5wVm8
- https://docs.ansible.com/projects/ansible/latest/collections/community/general/chroot_connection.html
- https://en.wikipedia.org/wiki/Chroot
- https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-223.ipd.pdf

Unit 10 - Harden Linux systems

- https://www.youtube.com/watch?v=FKpkx4-lBuI
- https://www.mindpointgroup.com/blog/stig-vs-cis-part-2-selecting-the-best-baseline-for-your-business?lockdownenterprise
- https://www.mindpointgroup.com/blog/stig-vs-cis-part-1-the-anatomy-of-baselines-and-compliance?lockdownenterprise
- https://github.com/ansible-lockdown

Unit 11 - Update and patch systems

Unit 12 - Configure Network Devices

Unit 13 - Remediating and Reporting on Drift

Unit 14 - CI/CD Pipelines and Make v. Buy v. Adopt Decisions

Unit 15 - Troubleshooting/Testing 1

Unit 16 - Troubleshooting/Testing 2

Misc

- Killercoda
- https://github.com/ProfessionalLinuxUsersGroup/course-books
- @Het_Tanis
- Killercoda.com.

4.12.2 Automation Course Downloads

Unit 1 - Automation Tools

WORKSHEET

- u1_worksheet.md
- u1_worksheet.txt
- u1_worksheet.pdf

LAB

- u1_lab.md
- u1_lab.txt

Unit 2 - Interacting with the OS

WORKSHEET

- u2_worksheet.md
- u2_worksheet.txt
- u2_worksheet.pdf

LAB

- u2_lab.md
- u2_lab.txt

Unit 3 - Building Inventories

WORKSHEET

- u_worksheet.md
- u3_worksheet.txt

LAB

Unit 3 Lab: Coming soon.

Unit 4 - Admin Commands and One-Offs

WORKSHEET

- u_worksheet.md
- u4_worksheet.txt
- u4_worksheet.pdf

LAB

- Lab on Killercoda: https://killercoda.com/het-tanis/course/AutomationLabs/Unit4_Admin_Commands

Unit 5

WORKSHEET

- u_worksheet.md
- u5_worksheet.txt

- u5_worksheet.pdf

LAB

Unit 5 lab coming soon.

Unit 6

WORKSHEET

- u_worksheet.md
- u6_worksheet.txt
- u6_worksheet.pdf

LAB

Unit 6 lab coming soon.