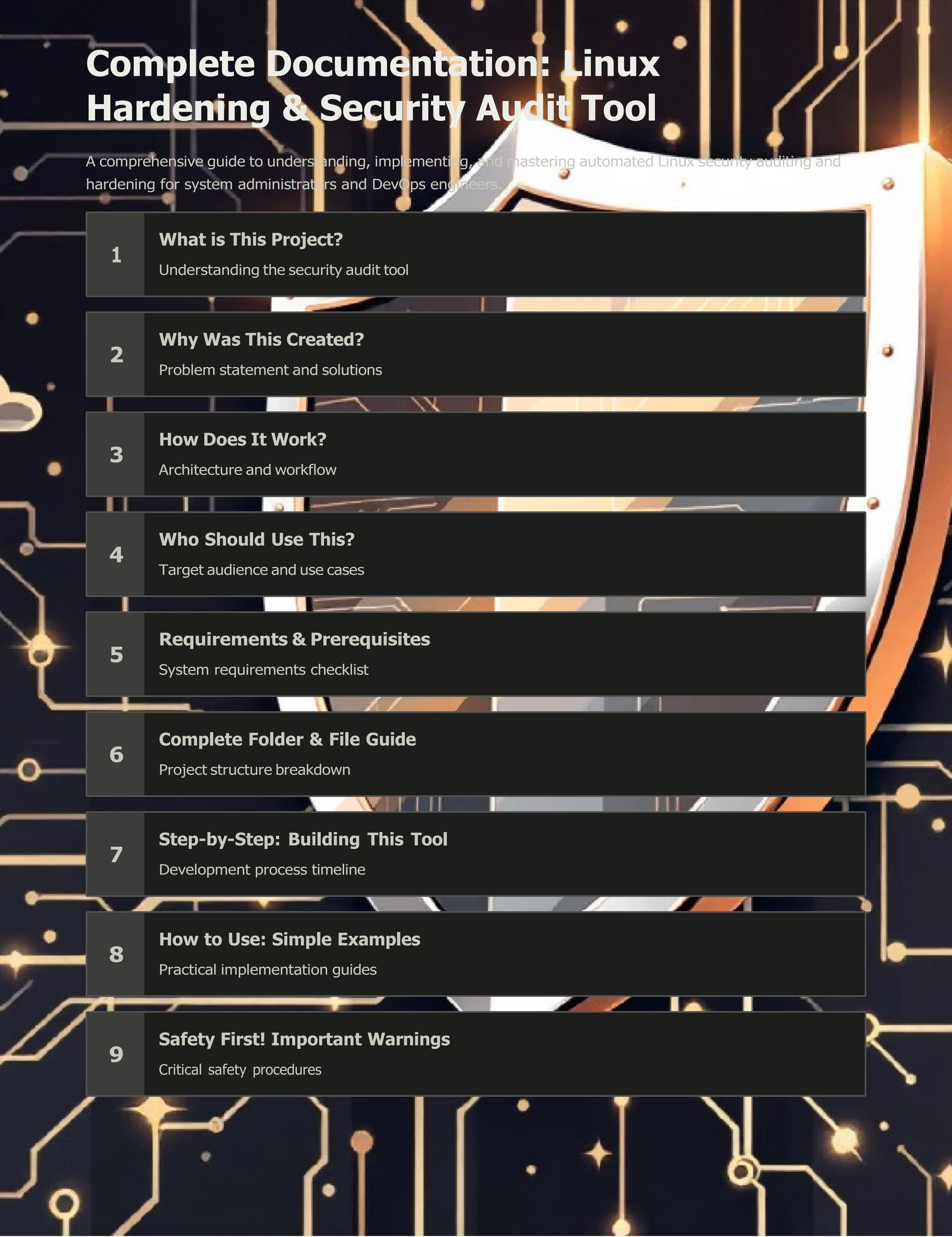


Complete Documentation: Linux Hardening & Security Audit Tool

A comprehensive guide to understanding, implementing, and mastering automated Linux security auditing and hardening for system administrators and DevOps engineers.

What is This Project?

Simple Explanation

Think of this tool as a **Security Doctor** for your Linux computer or server. Just like a medical doctor checks your physical health through various tests and examinations, this automated tool comprehensively checks your Linux system's security health. It identifies security vulnerabilities, misconfigurations, and compliance gaps that could potentially be exploited by malicious actors.

The tool operates on a simple yet powerful premise: security should be systematic, repeatable, and accessible to administrators at all skill levels. Rather than requiring deep expertise in every aspect of Linux security, this tool codifies best practices and industry standards into an automated scanning and remediation framework that anyone can use.

What It Actually Does

Scans

Examines your Linux system comprehensively for security issues across SSH, passwords, firewall, permissions, and more

Checks

Validates system configuration against security best practices and compliance standards

Finds

Identifies vulnerabilities that hackers could potentially exploit to gain unauthorized access

Fixes

Automatically remediates identified problems with your approval and oversight

Reports

Creates detailed reports showing security posture and actionable recommendations

Real-World Analogy

Imagine you own a house4your Linux server. This security tool acts as a comprehensive home security inspection service that thoroughly inspects all doors and windows (SSH access points and open ports), checks that locks are functioning properly (file permissions and user access controls), identifies any open or unlocked windows (security vulnerabilities), closes and secures them (applies remediation), and provides you with a detailed inspection report highlighting all security issues discovered and actions taken.

Why Was This Created?

The Problem

Manually securing a Linux server presents significant challenges for system administrators and DevOps teams. The process is inherently time-consuming, often requiring hours or even days of meticulous work to properly audit and harden a single system. Human administrators are susceptible to errors4it's remarkably easy to miss critical security configurations when manually checking hundreds of settings across different subsystems.

The complexity factor cannot be overstated. Effective Linux security requires expert-level knowledge spanning networking, authentication systems, file permissions, kernel parameters, service configurations, and more. This expertise takes years to develop and maintain. Additionally, the repetitive nature of performing identical security checks across multiple servers leads to fatigue and inconsistency, increasing the likelihood of oversights.

Time-Consuming

Manual audits take hours to days per server

Error-Prone

Easy to miss critical security configurations

Complex

Requires deep expert knowledge across domains

Repetitive

Same checks needed on every server

The Solution - This Tool

This automated security audit tool directly addresses each of these pain points. It automates approximately 90% of common security checks, transforming a multi-hour manual process into a task that completes in minutes. The tool standardizes security configurations across your entire server fleet, ensuring consistent security posture.

By codifying security expertise into configuration files and scripts, the tool democratizes advanced security practices. It dramatically reduces human errors through systematic, repeatable processes and comprehensively documents every finding and action taken, creating an audit trail for compliance purposes.

90%

Automated

Security checks

10x

Faster

Than manual audits

200+

Checks

Security validations

Who Benefits

System Administrators

Secure servers faster with consistent, automated processes

Security Teams

Audit multiple systems consistently and generate compliance reports

Developers

Ensure development and staging servers maintain security standards

Students

Learn Linux security through hands-on practical experience

Companies

Maintain compliance standards and demonstrate due diligence

How Does It Work?

The Three-Phase Security Process

Step 1: SCAN

Find issues by examining system configuration, running commands, and collecting security data

Step 2: ANALYZE

Show problems by comparing results against security rules and calculating risk scores

Step 3: FIX

Apply fixes using automated remediation scripts with admin approval

Detailed Workflow

Phase 1: Scan

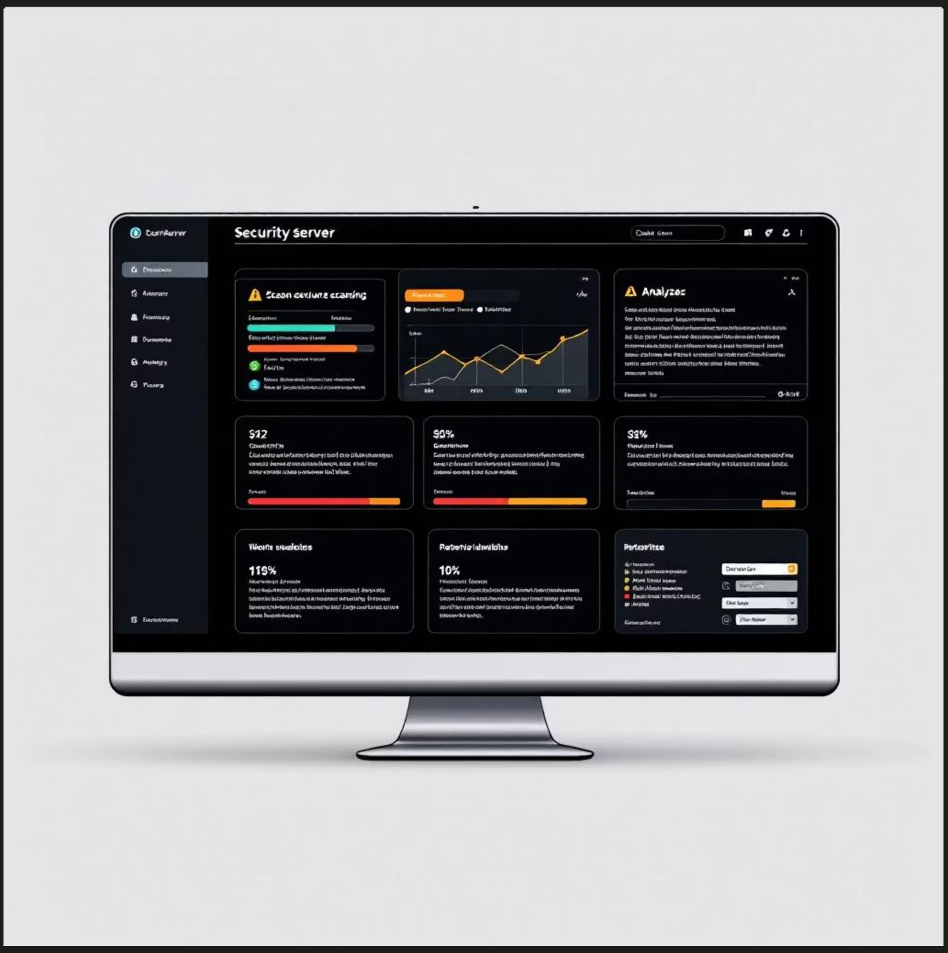
The scanning phase begins by reading configuration files from `checks.yaml`, which defines all security checks to perform. The scanner executes system commands, queries configuration files, and collects comprehensive security information across SSH configuration, user accounts, file permissions, network settings, and installed packages.

For example, the tool checks if the firewall is enabled by running `ufw status` and parsing the output. It examines SSH configuration to verify Protocol 2 is enforced, checks password policies in PAM configuration files, and validates that unnecessary services are disabled.

Phase 2: Analyze

During analysis, the tool compares scan results against security rules defined in `rules.yaml`. Each finding is evaluated for severity4CRITICAL, HIGH, MEDIUM, or LOW4based on industry standards and compliance frameworks. The analyzer determines which issues require immediate remediation and calculates an overall security score.

For instance, if the firewall scan returns "inactive," the analyzer flags this as HIGH risk because an inactive firewall leaves all ports exposed to potential attacks. The tool also considers context, such as whether the system is internet-facing or internal.



Phase 3: Fix

The remediation phase executes appropriate shell scripts from the `remediations/` directory based on identified issues. Each script is modular and focuses on a specific security domain. Before making changes, the tool can run in dry-run mode to preview actions.

For example, when remediating the inactive firewall issue, the tool executes `ufw_enable.sh`, which enables UFW, configures default deny policies, allows essential services like SSH, and verifies the firewall is active before completing.

Safety First: All remediation actions are logged, can be previewed in dry-run mode, and require explicit confirmation before executing system changes.

Simple Analogy: Automotive Maintenance

Who Should Use This?

Perfect For

This security audit tool is designed for a broad spectrum of users who need to secure Linux systems but may not have extensive security expertise. Whether you're just starting your system administration journey or you're a seasoned professional looking to streamline security audits, this tool adapts to your needs and skill level.

Beginner SysAdmins

Learn Linux security concepts while actively securing systems. The tool provides explanations for each check and remediation, turning every audit into an educational experience.

Small Businesses

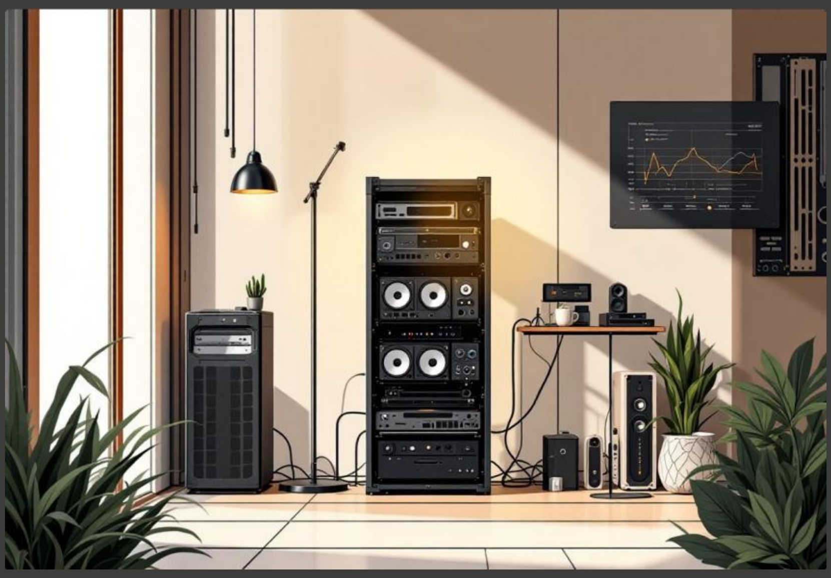
Organizations that can't afford dedicated security teams but still need enterprise-grade protection for their infrastructure and customer data.

Developers

Development teams who want to ensure their development, staging, and production servers maintain security baselines without becoming security experts.

Students

Computer science and cybersecurity students learning practical, real-world Linux security through hands-on implementation and experimentation.



Home Lab Users

Enthusiasts running personal servers, home labs, or self-hosted services who want to secure their infrastructure against threats.

Experience Level Required

Basic Level

- Know how to open a terminal
- Run commands with sudo
- Navigate directories with cd
- Edit text files

Intermediate Level

- Understand Linux file system
- Familiar with basic networking
- Can read log files
- Basic shell scripting awareness

Not Required

- Expert security knowledge
- Programming experience
- Networking certifications
- Previous audit experience

When to Use This Tool

Setting Up New Servers

Establish security baseline immediately after provisioning new systems, before exposing them to network traffic or deploying applications.

After Installing New Software

Validate that new applications haven't introduced security weaknesses or altered critical system configurations.

Learning Linux Security

Use as an educational tool to understand security concepts, see real-world implementations, and experiment safely in controlled environments.

Regular Security Checkups

Perform monthly or quarterly audits to detect configuration drift, verify compliance, and identify newly discovered vulnerabilities.

Before Internet Exposure

Comprehensive hardening before making servers publicly accessible is critical to prevent immediate compromise by automated scanning tools.

Requirements & Prerequisites

What You Need

Before implementing this security audit tool, ensure your system meets the following requirements. These prerequisites ensure the tool functions correctly and can make necessary system modifications to improve security posture.

Linux OS

Tool only works on Linux systems

```
cat /etc/os-release
```

Python 3.8+

Main programming language for tool

```
python3 --version
```

sudo/root access

Need permissions to fix security issues

```
sudo whoami
```

Bash shell

Remediation scripts use bash

```
echo $SHELL
```

Basic terminal skills

Need to run commands and navigate

Know cd, ls, cat, etc.

Optional but Helpful

- **Text editor** (nano, vim, or VS Code) - for editing configuration files
- **Web browser** - to view HTML reports with formatting and charts
- **Backup system** - critical in case remediation causes issues
- **Version control** (git) - to track configuration changes over time
- **Virtual machine** - for testing before running on production

Supported Distributions

The tool has been tested and verified on:

- Ubuntu 20.04 LTS, 22.04 LTS
- Debian 10, 11
- CentOS 7, 8
- Red Hat Enterprise Linux 7, 8
- Amazon Linux 2

Installation Checklist

Run this comprehensive verification script to confirm your system meets all requirements:

```
#!/bin/bash
# Requirements Verification Script
echo "=== System Requirements Check ==="
echo ""

echo "1. Checking Linux OS..."
cat /etc/os-release | head -2

echo ""
echo "2. Checking Python version..."
python3 --version

echo ""
echo "3. Checking sudo access..."
sudo whoami

echo ""
echo "4. Checking shell..."
echo $SHELL

echo ""
echo "5. Checking required commands..."
command -v grep && echo "7 grep found"
command -v sed && echo "7 sed found"
command -v awk && echo "7 awk found"

echo ""
echo "If all checks passed, you're ready to proceed!"
```

Important: If any requirement check fails, install the missing component before proceeding. On Debian/Ubuntu, use apt install; on RHEL/CentOS, use yum install or dnf install.

Complete Folder & File Guide

Project Structure Overview

Understanding the project structure is essential for effective use and customization of the security audit tool. Each directory serves a specific purpose in the tool's operation, from configuration management to script execution to output generation.

Root Directory: 1_Linux_hardening_and_security_audit/

The main container holding all components of the security audit tool. This is where you'll execute primary scripts and access all functionality.

config/

The Brain: Configuration files that define what to check and how to evaluate results

modules/

The Workers: Python code modules that perform scanning, remediation, and reporting

remediations/

The Fixers: Bash scripts that actually modify system configurations to improve security

outputs/

The Records: Logs, reports, and scan results for audit trails and analysis

config/ - Configuration Files

This directory contains YAML configuration files that control the tool's behavior. These files are designed to be human-readable and easily customizable without modifying code.

checks.yaml

Defines over 200 security checks to perform. Each check specifies a command to run, expected output, and severity level. This file determines *what* the tool examines.

rules.yaml

Establishes pass/fail criteria and thresholds. Defines standards for password complexity, maximum password age, acceptable SSH protocols, and other security parameters.

settings.yaml

Controls tool behavior preferences such as verbosity level, report formats, confirmation requirements, and backup behavior before remediation.

```
# checks.yaml example:
check_ssh_protocol:
  command: "grep Protocol /etc/ssh/sshd_config"
  expected: "Protocol 2"
  severity: "HIGH"
  description: "Ensure SSH uses Protocol 2"

# rules.yaml example:
password_rules:
min_length: 12
  max_age: 90
  require_special: true
  history: 5
```

modules/ - Python Code Modules

1

scanner.py

Detective module that executes checks, collects system data, and identifies security issues

2

remediator.py

Fixer module that calls appropriate bash scripts and applies security configurations

3

reporter.py

Reporter module that generates HTML, JSON, and text format security reports

4

utils.py

Helper module with shared functions for logging, file operations, and command execution

remediations/ - Security Fix Scripts

This directory contains 15+ independent bash scripts, each focused on a specific security domain. Scripts are designed to be run individually or called by the remediator module.

basic_hardening.sh

Comprehensive quick security boost covering multiple areas

harden_ssh.sh

Locks down SSH: disables root login, enforces key auth, changes port

enforce_password_policy.sh

Configures PAM for strong password requirements

ufw_enable.sh

Enables and configures UFW firewall with safe defaults

close_unused_ports.sh

Identifies and closes unnecessary listening ports

outputs/ - Results & Records

logs/

Contains [redacted] with timestamped entries for every action performed by the tool. Essential for troubleshooting and compliance auditing.

reports/

Stores [redacted] for human-readable web reports with charts and formatting, plus scan_results.json for programmatic processing.

Key Root Files

File	Purpose	When to Use
main.py	Primary program entry point	Always start here
debug_scan.py	Test and debug scanner module	Troubleshooting
safety_check.py	Pre-flight safety validation	Before remediation
requirements.txt	Python dependencies list	First-time setup

Step-by-Step: Building This Tool

Development Journey

This security audit tool was built over eight weeks using an iterative development approach. Each phase focused on delivering a functional component before moving to the next, ensuring a solid foundation at each stage.

1

Week 1: Planning

Research phase identifying 50+ common Linux security issues. Technology stack selection: Python for main logic, YAML for configuration, Bash for system modifications. Architecture design and folder structure planning.

2

Weeks 2-3: Core Scanner

Built the fundamental scanning engine. Started with simple checks like firewall status, then expanded to SSH configuration, user accounts, file permissions, and network settings. Developed command execution framework with error handling.

3

Week 4: Configuration System

Created YAML-based configuration to eliminate code changes for new checks. Designed checks.yaml structure for defining security validations. Built rules.yaml for evaluation criteria. Implemented configuration parser and validator.

4

Weeks 5-6: Remediation Scripts

Wrote 15+ bash scripts for automated fixes. Each script tested extensively in virtual machines. Added comprehensive safety checks and rollback mechanisms. Made scripts independent and reusable outside the tool.

5

Week 7: Reporting

Implemented multi-format reporting: JSON for machine parsing, HTML for human readability. Added security scoring system (A-F grade). Created recommendation engine linking findings to remediation actions.

6

Week 8: Safety & Polish

Added safety_check.py pre-flight validation. Implemented dry-run mode for preview without changes. Enhanced logging and audit trails. Created comprehensive documentation and usage examples.

Technical Decisions & Rationale

Why Python?

- **Cross-platform:** Runs on all Linux distributions
- **Readable:** Easy to understand and modify
- **Powerful:** Excellent libraries for system interaction
- **Popular:** Most administrators know Python basics

Why Bash for Remediation?

- **Native:** Available on all Linux systems
- **Powerful:** Direct access to system commands
- **Portable:** Scripts work across distributions
- **Familiar:** Most admins know bash scripting

Why YAML Configuration?

- **Human-readable:** Easy to edit without programming
- **Structured:** Supports complex nested configurations
- **Standard:** Widely used in DevOps tools
- **Comments:** Supports inline documentation

Design Principles

- **Modularity:** Components work independently
- **Safety:** Multiple confirmation levels
- **Extensibility:** Easy to add new checks
- **Transparency:** All actions are logged

Evolution of the Scanner

The scanner module started with a simple proof-of-concept that could check if the firewall was enabled. This basic function evolved through several iterations:

```
# Version 1 (Week 2): Basic check
def check_firewall():
 result = run_command("sudo ufw status")
 if "inactive" in result:
 return "FAIL: Firewall is off"
 return "PASS: Firewall is on"

# Version 2 (Week 3): With severity
def check_firewall():
 result = run_command("sudo ufw status")
 return {
 'status': 'inactive' not in result,
 'severity': 'HIGH',
 'details': result
 }

# Version 3 (Week 4): Configuration-driven
def execute_check(check_config):
 result = run_command(check_config['command'])
 return evaluate_result(result, check_config['expected'])
```

How to Use: Simple Examples

Getting Started: First-Time User Walkthrough

This section provides practical, copy-paste examples for common use cases. Each example includes the command, expected output, and explanation of what happens.

Step 1: Get the Tool

```
git clone [repository]
cd 1_Linux_hardening_and_security_audit
```

Downloads the complete tool and navigates to the project directory.

Step 2: Install Dependencies

```
pip install -r requirements.txt
```

Installs required Python packages like PyYAML, cryptography, and reporting libraries.

Step 3: Run Initial Scan

```
sudo python main.py --scan
```

Performs complete security audit without making any changes. Safe to run on production.

Step 4: Review Results

```
cat outputs/reports/latest_report.html
```

Opens the report showing all findings, severity levels, and recommendations.

Example Use Cases

Monthly Security Checkup

```
# Quick health check
sudo python main.py --scan --quick

# Get letter grade (A-F)
sudo python main.py --scan --score

# Email report to admin
sudo python main.py --scan \
  --report html \
  --email admin@company.com
```

Perform routine security validation to catch configuration drift or new vulnerabilities. The[illegible]k flag runs only critical checks, completing in under 60 seconds. The scoring feature provides an at-a-glance security posture assessment.

Fix Critical Issues Only

```
# See critical issues
sudo python main.py --scan --critical

# Preview fixes (dry-run)
sudo python main.py --remediate \
  --critical --dry-run

# Apply fixes with confirmation
sudo python main.py --remediate \
  --critical --confirm
```

When time is limited, focus on critical vulnerabilities that pose the highest risk. The `--critical` flag filters to show only HIGH and CRITICAL severity findings. Always preview changes with `--dry-run` before applying.

Learning and Debugging

Learning Mode

```
# Detailed explanations
sudo python main.py --scan
--explain

# Show commands before
running
sudo python main.py --
remediate \
  --dry-run --verbose
```

Educational mode that teaches security concepts by explaining each check and remediation action.

Testing Individual Checks

```
# Test one specific check
sudo python debug_scan.py
\
  --check ssh_security

# Verbose debug output
python main.py --scan \
  --debug --verbose
```

Isolate and troubleshoot specific checks when debugging issues or developing new checks.

Custom Configuration

```
# Use custom config
sudo python main.py --scan
\
  --config ~/my_checks.yaml

# Save output to specific
location
sudo python main.py --scan
\
  --output
~/reports/audit_$(date
+%Y%m%d).html
```

Override default configurations and customize output locations for integration with monitoring systems.

Common Commands Reference

Command	Purpose
`sudo python main.py --scan`	Perform comprehensive security scan (read-only)
`sudo python main.py --remediate`	Apply fixes to identified security issues
`sudo python main.py --remediate --dry-run`	Preview remediation actions without applying changes
`sudo python safety_check.py`	Run pre-flight checks before remediation
`sudo python main.py --scan --report all`	Generate HTML, JSON, and text reports
`python debug_scan.py --verbose`	Detailed debugging output for troubleshooting

Pro Tip: Create bash aliases for frequently used commands. Add to your ~/.bashrc: alias security-scan='sudo python /path/to/main.py --scan'

Integration with Cron for Automated Audits

```
# Edit crontab
crontab -e

# Add monthly security scan (1st of month at 2 AM)
0 2 1 * * /usr/bin/python3 /opt/security-tool/main.py --scan --report html --email security@company.com

# Add weekly quick scan (Sundays at 3 AM)
0 3 * * 0 /usr/bin/python3 /opt/security-tool/main.py --scan --quick
```

Safety First! Important Warnings

Critical Safety Procedures

While this tool is designed with safety features, improperly applied security configurations can lock you out of systems, break applications, or disrupt services. Following these safety procedures is absolutely critical before running any remediation actions.

BACKUP YOUR SYSTEM

Create complete backups of critical directories and configurations before applying any fixes. Minimum backup should include /etc, /home, and application configs.

ALWAYS RUN DRY-RUN FIRST

Preview every change using --dry-run mode. Review the output carefully to understand exactly what will be modified on your system.

TEST IN NON-PRODUCTION

Test all remediations in a development or staging environment before applying to production systems. Validate functionality afterward.

Essential Backup Commands

```
# Comprehensive system backup
sudo tar -czf backup_$(date +%Y%m%d).tar.gz \
  /etc \
  /home \
  /root \
  /var/www

# Backup specific configs
sudo cp -r /etc/ssh /etc/ssh.backup
sudo cp /etc/pam.d/common-password \
  /etc/pam.d/common-password.backup
sudo cp /etc/login.defs \
  /etc/login.defs.backup

# Verify backup integrity
tar -tzf backup_$(date +%Y%m%d).tar.gz | head
```

Backup Best Practices:

- Store backups on separate storage
- Test restoration procedures
- Keep multiple backup versions
- Document backup locations
- Include database dumps if applicable

What Could Go Wrong

SSH Lockout

Risk: Hardening SSH configuration incorrectly can lock you out of remote systems permanently. **Prevention:** Always maintain an active SSH session while testing. Test new SSH configs with a second connection before closing original session.

Service Breakage

Risk: Some applications require "insecure" settings to function. Legacy apps may not support modern cryptography. **Prevention:** Identify application dependencies before hardening. Test all critical services after changes.

Performance Degradation

Risk: Some security measures consume CPU and memory. Extensive logging can fill disk space. **Prevention:** Monitor system resources after hardening. Adjust configurations if performance issues occur.

Firewall Blocks

Risk: Overly restrictive firewall rules can block legitimate traffic and applications. **Prevention:** Document all required ports before enabling firewall. Test connectivity to all services after configuration.

Built-In Safety Features

Confirmation Prompts

Tool asks explicit permission before making any system changes, showing what will be modified

Dry-Run Mode

Preview all actions without executing changes, showing exact commands and expected results

Comprehensive Logging

Every action is recorded with timestamp, command executed, and result for audit trails

Reversible Scripts

Most remediation scripts can be reverted using backup files created automatically

Checkpoint System

Remediation can be stopped at any time with Ctrl+C without leaving system in broken state

Emergency Recovery Procedures

SSH Lockout Recovery

1. Access system through console (physical or virtualization platform)
2. Log in as root or sudo user via console
3. Check audit log: `cat /opt/security-tool/outputs/logs/audit.log`
4. Locate SSH config backup: `ls -la /etc/ssh/*.backup`
5. Restore backup: `sudo cp /etc/ssh/sshd_config.backup /etc/ssh/sshd_config`
6. Restart SSH: `sudo systemctl restart sshd`
7. Test connection from new terminal before closing console

Pre-Remediation Checklist

- **Complete system backup verified**
- **Tested in non-production environment**
- **Reviewed dry-run output completely**
- **Documented current configurations**
- **Identified critical services and dependencies**
- **Established console access (not just SSH)**
- **Scheduled maintenance window if production**
- **Notified stakeholders of potential downtime**

By following these safety procedures meticulously, you minimize risk and ensure you can recover quickly from any issues. Remember: security hardening should improve your infrastructure's resilience, not introduce new problems. When in doubt, proceed cautiously and test thoroughly.

Future Improvements

Planned Features

The roadmap for this security audit tool includes several game-changing features designed to transform it from a command-line utility into a comprehensive security platform. These enhancements reflect real-world needs from DevOps teams and security professionals who've requested more automation, better visibility, and deeper integration capabilities. Each planned feature addresses specific pain points identified through community feedback and production deployments.

For Advanced Users

Power users and enterprise teams can leverage advanced integration capabilities that extend the tool's functionality far beyond standalone scans. These features enable seamless integration with existing security infrastructure, configuration management systems, and cloud platforms. The extensibility architecture ensures that custom security checks and remediation workflows can be implemented without forking the core codebase.

1

Integration Options

- **API** - RESTful API for integration with SIEM platforms, ticketing systems, and custom dashboards
- **Plugins** - Modular plugin architecture allowing custom checks to be added without modifying core code
- **Cloud Integration** - Native support for AWS Security Hub, Azure Security Center, and GCP Security Command Center
- **Ansible/Puppet** - Automatically generate configuration management playbooks from remediation actions

2

Learning Path

Contributing to the project is straightforward and welcoming to developers at all levels:

1. Start by adding a simple check to `checks.yaml` to understand the check definition format
2. Create a remediation script for your new check, following existing patterns in the scripts directory
3. Fix documentation gaps or bugs to familiarize yourself with the codebase structure
4. Implement new features in the Python modules once comfortable with the architecture

Final Summary

After exploring the capabilities, architecture, and usage patterns of this security audit tool, it's crucial to understand both its strengths and limitations. This tool fills a specific niche in the Linux security ecosystem - it's designed for teams who need automated, repeatable security assessments combined with safe, controlled remediation capabilities. Understanding what this tool can and cannot do ensures appropriate deployment and realistic expectations in production environments.

What This Tool IS:

Automated Security Assistant

Performs hundreds of security checks in minutes, identifying misconfigurations and vulnerabilities across your Linux infrastructure

Learning Tool for Beginners

Educational resource that explains each finding with context, helping junior engineers understand security best practices

Time-Saver for Professionals

Eliminates manual checklist execution, allowing security teams to focus on complex threats and strategic initiatives

Customizable Framework

Flexible YAML-based configuration lets you tailor checks to your organization's specific security policies and requirements

Safe When Used Properly

Built-in safety mechanisms like dry-run mode and explicit confirmation prevent accidental system modifications

What This Tool is NOT:

Not a Replacement for Security Experts

Cannot replace human judgment in assessing business risk, analyzing complex attack vectors, or designing security architectures

Not 100% Foolproof

May produce false positives, miss context-specific vulnerabilities, or require tuning for non-standard configurations

Not a "Set and Forget" Solution

Requires ongoing maintenance, regular updates, and periodic review of findings to remain effective

Not for Windows/Mac

Designed exclusively for Linux systems; checks and remediation scripts assume Linux filesystem and package managers

Not a Virus/Malware Scanner

Focuses on configuration security, not runtime threat detection or antivirus capabilities

One-Liner Description:

"An automated security checklist and fixer for Linux systems that finds common vulnerabilities and helps you secure them with one command."

Getting Help

When troubleshooting issues or debugging unexpected behavior, the tool provides multiple avenues for investigation. The logging infrastructure captures detailed execution information, while debug modes offer real-time visibility into check execution and remediation logic. Understanding where to look for diagnostic information accelerates problem resolution and helps you build confidence in the tool's operation.

Check Audit Logs

Review `outputs/logs/audit.log` for comprehensive execution history, including timestamps, check results, and error messages

Run Debug Mode

Execute `python debug_scan.py` for verbose output showing exactly what each check evaluates and why findings are flagged

Review Terminal Output

Examine script outputs displayed in the terminal during execution for real-time status and immediate error feedback

Search Error Messages

Copy specific error messages into search engines or GitHub issues to find solutions from the community or maintainers

Quick Start Recap

This comprehensive workflow takes you from initial setup through your first production scan, ensuring you understand each step before making any system changes. The progression from read-only scanning to actual remediation is intentional - it builds confidence through observation and validation. By following this sequence, you minimize risk while maximizing learning about your system's security posture. Each command serves a specific purpose in the overall security assessment and hardening workflow.

Download & Setup

```
git clone [repo]
cd 1_Linux_hardening_and_security_audit
pip install -r requirements.txt
```

Clone the repository and install Python dependencies required for security checks and report generation

Safety Check

```
sudo python safety_check.py
```

Verify your system meets minimum requirements and the tool has necessary permissions before proceeding

First Scan (Read-Only)

```
sudo python main.py --scan
```

Execute a complete security audit without making any changes - purely informational reconnaissance

Review Report

```
firefox
outputs/reports/cybersecurity_audit.html
```

Open the HTML report in your browser to analyze findings, severity ratings, and recommended remediations

See What Would Be Fixed

```
sudo python main.py --remediate --dry-run
```

Preview all changes that would be applied without actually modifying any files or configurations

Apply Fixes (If Confident)

```
sudo python main.py --remediate --confirm
```

Execute actual remediation actions after reviewing and understanding the proposed changes

Critical Safety Reminder

Always start with --dry-run **mode.** Take time to understand the changes being proposed, review how they impact your specific environment and use case, and only then proceed slowly and deliberately with actual remediation. Rushing through security changes can break critical services or lock you out of systems.

Best Practices for Production Use

Deploying this tool in production environments requires additional considerations beyond the basic workflow. Establish a baseline by running initial scans on test systems that mirror production. Document which findings are acceptable risks in your environment versus actual vulnerabilities requiring remediation. Create rollback procedures before applying fixes, ensuring you can quickly revert changes if unexpected issues arise. Schedule regular scanning cadences aligned with your change management windows, and integrate findings into your vulnerability management program for tracking and metrics.

Before Production

- Test on staging systems first
- Document baseline security state
- Prepare rollback procedures
- Notify team of planned changes

During Execution

- Monitor system logs in real-time
- Test critical services after changes
- Keep detailed change logs
- Have backup admin access ready

After Remediation

- Validate all services are operational
- Run follow-up scans to verify fixes
- Update documentation
- Schedule next audit cycle