Linux Privilege Escalation Toolkit - Project Documentation

Author: Om Fulsundar
Version: 1.0

Contents :

Abstract :

The Linux Privilege Escalation Toolkit is a modular, detection-focused utility designed to automate enumeration of common privilege escalation vectors on Linux systems. It inspects file permissions, SUID/SGID binaries, cron jobs, systemd services, sudo rules, and kernel version information to identify misconfigurations and risky artifacts that could enable privilege escalation. The toolkit emphasizes safety and auditability: it performs read-only checks, produces structured JSON logs per module, and generates a consolidated human-readable report to support triage and remediation. This document describes the toolkit's goals, architecture, implementation details, testing methodology, evaluation, limitations, and a prioritized roadmap for future enhancements. The toolkit is intended for security practitioners, system administrators, and educators who require a repeatable, auditable method to surface potential privilege escalation issues without executing exploits or altering system state.

Keywords :

Linux, privilege escalation, SUID, SGID, permissions audit, cron analysis, systemd, sudo, kernel CVE, security auditing, detection toolkit

---

1 Introduction :

Privilege escalation is a central concern in post-compromise activity and security assessments. Misconfigured file permissions, improperly set SUID/SGID bits, writable cron scripts, and outdated kernels are frequent sources of privilege escalation opportunities. Manual discovery of these vectors is time consuming and inconsistent across environments. The Linux Privilege Escalation Toolkit was developed to automate safe enumeration of these vectors, producing structured outputs that facilitate triage and remediation without performing any exploitative actions.

This document provides a comprehensive overview of the project: the problem it addresses, the design and responsibilities of each module, implementation details, testing and observed outputs, evaluation of strengths and limitations, and a roadmap for future work.

---

2 Related Work and Motivation :

Security practitioners commonly rely on a mix of manual checks, ad-hoc scripts, and community resources to discover privilege escalation vectors. Public knowledge bases and curated lists of exploitable binaries provide practical guidance, while many automated tools either produce noisy output or attempt exploitation.

The motivation for this toolkit is to provide:

- **Modularity** so checks can be extended or replaced independently.
- **Safety** by restricting the tool to detection only and avoiding exploit execution.
- **Structured output** to support automated processing and integration into audit workflows.
- **Practicality** through use of standard Linux utilities and Python for portability and maintainability.

This approach aims to reduce the time required for initial enumeration while producing artifacts suitable for reporting and remediation planning.

---

3 Problem Statement and Objectives :

Problem: Manual privilege escalation discovery is inefficient and error prone. Automated tools that attempt exploitation can be unsafe in production environments and may not produce structured artifacts for audits.

Objectives:

- Automate detection of common privilege escalation vectors on Linux systems.
- Produce machine-readable logs and a consolidated human-readable report.
- Keep the toolkit modular and extensible.
- Ensure the toolkit performs read-only checks and avoids executing exploits.

---

4 Design and Architecture :

4.1 High-Level Pipeline

1. **Entry point (main.py)**: initializes the environment and orchestrates module execution.
2. **Module execution**: each module runs independently and writes a JSON output to report/output/.
3. **Report consolidation (report.py)**: reads module outputs, normalizes severity, and produces final_report.txt plus a terminal summary.
4. **Artifacts**: per-module JSON logs and the consolidated text report are stored for review.

4.2 Module Responsibilities

- **suid_scan.py** — Enumerates SUID/SGID binaries and flags unusual or risky entries.
- **perms_scan.py** — Detects world-writable files, root-owned writable paths, and misconfigured directories.
- **services_scan.py** — Inspects systemd unit files and sudo configuration for misconfigurations that could lead to escalation.
- **cron_scan.py** — Aggregates system and user cron entries and checks referenced scripts for insecure permissions.
- **kernel_scan.py** — Captures kernel version and compares it against a local CVE mapping to flag outdated or vulnerable kernels.

Each module is designed to:

- Run safe, read-only checks where possible.
- Output a JSON file with summary and findings fields.
- Include metadata such as timestamp, module name, and execution status.

4.3 Data and Output Format

- **Per-module JSON**: includes summary (counts and severity distribution) and findings (list of objects with path, description, severity, and evidence).
- **Consolidated report**: human-readable text with a terminal summary and final_report.txt saved in report/output/.

- **Directory layout (actual)**:

```
linux_prevesc/
├── .gitignore
├── main.py
├── report.py
│
├── modules/
│   ├── suid_scan.py
│   ├── perms_scan.py
│   ├── services_scan.py
│   ├── cron_scan.py
│   └── kernel_scan.py
│
└── report/
    └── output/
        ├── suid_scan_output.json
        ├── perms_scan_output.json
        ├── services_scan_output.json
        ├── cron_scan_output.json
        ├── kernel_scan_output.json
        └── final_report.txt
```

---

5 Implementation Details :

5.1 Environment and Dependencies

- **Language:** Python 3.x
- **Utilities invoked:** find, ls, systemctl, getcap, sudo -l, crontab, uname, grep, awk, sed (used via subprocess where appropriate).
- The toolkit is designed to run on standard Linux distributions and requires only common system utilities and Python standard libraries.

5.2 Module Behavior and Detection Logic

- **SUID/SGID detection:** Uses find to locate files with permission bits matching SUID/SGID and annotates each candidate with owner, permissions, and file type. Optionally filters by known safe lists.
- **Permissions scan:** Walks configured directories (e.g., /etc, /var, /usr/local) to find root-owned files or directories that are world-writable; records exact permission bits and ownership.
- **Services scan:** Reads systemd unit files and checks ExecStart paths for writable directories or scripts; parses sudo -l output to detect NOPASSWD or overly permissive rules.

- **Cron scan:** Aggregates cron entries from /etc/cron.*, /etc/crontab, /var/spool/cron/, and user crontabs; checks referenced scripts for write permissions by non-root users.
- **Kernel scan:** Captures uname -r and compares against a local kernel_cves.json mapping; flags matches with severity based on CVE criticality.

5.3 Report Generation

report.py:

- Reads all JSON outputs from report/output/.
- Normalizes severity levels (High, Medium, Low) using a consistent rule set.
- Produces a terminal summary (color coded) and writes final_report.txt with sections per module, counts, and representative findings.
- Ensures the output directory exists and is writable before writing artifacts.

---

6 Testing and Results :

6.1 Test Plan and Environment

Testing was performed on representative environments:

- **Parrot OS / Kali**: useful for richer default findings.
- **Ubuntu Server**: used as a clean baseline.
- **Docker containers**: for isolated, repeatable tests.

Tests verified:

- Each module produces a JSON file even when no findings exist.
- report.py consolidates outputs and produces a readable final report.
- The toolkit does not modify system state or execute exploits.

6.2 Observed Outputs and Artifacts

The toolkit reliably produces:

- Per-module JSON logs in report/output/.
- A consolidated final_report.txt summarizing findings and suggested mitigations.
- Terminal output showing a color-coded summary.

A separate screenshots file is included in the repository that demonstrates the tool running, the terminal report, the output directory contents, and sample JSON outputs.

---

7 Evaluation and Limitations :

Strengths

- **Modular design** simplifies extension and maintenance.
- **Structured outputs** support automated processing and audit workflows.
- **Safety by design** avoids exploit execution and reduces risk during audits.

Limitations

- **Detection scope** is limited to implemented checks and may miss environment-specific vectors until modules are extended.
- **Kernel CVE mapping** is static unless updated and does not query live CVE feeds by default.
- **Local execution only**: the toolkit is designed to run on the target host and does not include remote scanning capabilities.
- **False positives/negatives**: automated checks should be validated manually; some benign configurations may be flagged and some subtle misconfigurations may be missed.

---

8 Future Work and Roadmap :

Planned enhancements to increase coverage and utility:

- **Broaden enumeration paths** to include additional directories and user home paths for SUID/SGID and permission checks.
- **Live CVE integration** to query authoritative feeds and keep kernel vulnerability data current.
- **Improved sudo parsing** to robustly handle /etc/sudoers.d/ and multi-user contexts.
- **Interactive report UI** (CLI or lightweight web interface) to filter and explore findings.
- **Optional safe PoC references**: link to GTFOBins or remediation guidance without executing code.
- **Remote scanning mode**: add an SSH-based agent for controlled remote enumeration.

---

9 Conclusion :

The Linux Privilege Escalation Toolkit provides a practical, modular framework for detecting common privilege escalation vectors on Linux systems. By focusing on safe enumeration and producing structured outputs, the toolkit supports both automated workflows and manual triage. It is particularly useful in assessment scenarios where exploitation is not permitted or desirable. While not exhaustive, the toolkit establishes a solid foundation for automated enumeration and reporting. Future development will prioritize expanding detection coverage, integrating live vulnerability data, improving parsing accuracy, and adding user-friendly reporting interfaces to make findings easier to prioritize and remediate.

References :

- GTFOBins public knowledge base.
- Linux system documentation for systemd, cron, and file permissions.
- Public CVE databases for kernel vulnerability awareness.

Screenshots: A separate screenshots file is included in the repository showing the toolkit running, terminal output, the report/output/ directory contents, and sample JSON outputs. Refer to that file for visual artifacts.
