

**PRIVCHECK – Privileged Command Analysis Tool**

**Course:** Theory Of Automata

**Submitted To:** Sir Hassam Shakil

**Group Members:**

|  |  |
| --- | --- |
| **Names** | **Reg Id** |
| Muhammad Jahanzaib | 63843 |
| Abdul Sarim Khan | 62527 |
| Kashish Aneet | 62607 |

**REMARKS: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Abstract

This report presents a Deterministic Finite Automata (DFA)-based system designed to detect high-privileged system commands within script files, offering users actionable insights into potentially sensitive or risky operations. Built as part of a Theory of Automata course, the system utilizes regular expressions modeled as DFA to perform efficient and accurate pattern matching on shell, batch, and script code. Instead of detecting traditional injection vulnerabilities, the tool focuses on identifying commands such as sudo, chmod 777, net user, and rm -rf, which require elevated privileges and can significantly impact system integrity if misused. Integrated with a Streamlit-powered web interface, users can upload or paste scripts for real-time command classification. The results are visualized through an interactive Plotly gauge chart, alongside detailed feedback about each detected command's function, associated risks, and recommended usage guidelines. Lottie animations further enhance user engagement, making the tool approachable for both developers and learners. This project highlights the practical use of automata theory in the field of cybersecurity, specifically in the detection and contextual analysis of privilege-sensitive commands.

# Introduction

In modern computing environments, system scripts play a vital role in managing user accounts, configuring services, and automating administrative tasks. However, these scripts often include high-privileged commands that, if misused or left unchecked, can pose serious security risks.

Commands such as sudo, chmod 777, net user, and rm -rf can alter system permissions, escalate privileges, or delete critical data — making them highly sensitive from a cybersecurity standpoint.

Unlike typical injection vulnerabilities such as SQL Injection (SQLi) or Cross-Site Scripting (XSS), high-privileged command misuse stems not from external inputs, but from the intentional or unintentional inclusion of risky operations in internal scripts. These commands may not be inherently malicious but can be dangerous when executed without proper authorization or context. As organizations and developers rely increasingly on automation, the need to audit script contents for sensitive command usage has become critical. Deterministic Finite Automata (DFA), a core concept in automata theory, offers a powerful mechanism for efficient pattern matching and token recognition. By modeling known high-privileged command patterns as regular expressions and converting them into DFA structures, we can enable fast, deterministic scanning of scripts to flag and interpret potentially risky instructions.

This project applies DFA principles to build an automated command detection engine focused on identifying high-privilege commands within uploaded or pasted scripts. The system supports common formats such as .txt, .sh, .ps1, and .bat, and delivers real-time analysis through an interactive web interface built using Streamlit. Results are visualized using a severity gauge chart and detailed command match reports, helping users understand the purpose, risk, and safe handling of each identified command. By bridging theoretical automata concepts with practical cybersecurity concerns, this project serves as a valuable educational and operational tool. It not only highlights risky system-level operations but also provides informed guidance on responsible scripting practices, thereby enhancing script transparency and system safety.

# Literature Review

Automata theory, as comprehensively explained by Hopcroft, Motwani, and Ullman (2006), provides a robust theoretical foundation for pattern recognition through Deterministic Finite Automata (DFA). DFAs operate by deterministically transitioning between a finite number of states based on input symbols, enabling efficient, linear-time analysis of input strings. This makes them especially suitable for tasks such as lexical analysis and command pattern detection, where performance and reliability are essential.

In the domain of cybersecurity, most existing tools target specific vulnerabilities. For example, utilities like SQLMap focus on detecting and exploiting SQL Injection vulnerabilities, while tools like XSS Auditor or browser-based filters address client-side injection issues. However, fewer tools focus on **script-level auditing for high-privileged command usage**—commands which are not inherently malicious, but can severely compromise a system if executed carelessly or without proper authorization. These include actions such as unrestricted file permission changes, privilege escalation, user creation, or service configuration. Security literature emphasizes the importance of analyzing **static scripts** to catch potential abuse points before execution. The Sprinto Cybersecurity Report Guide (2025) highlights the critical role of context-aware, interpretable security reporting and emphasizes the value of tools that are both actionable and user-friendly. This insight supports the integration of security detection into educational and administrative workflows, such as script review and configuration audits.

Unlike tools that focus on external threats, this project brings attention to **internal risk factors** embedded within administrative and deployment scripts. It combines DFA-based pattern matching with an accessible user interface, enabling automated detection of high-privilege commands across shell, batch, and script files. Furthermore, leveraging modern development frameworks such as Streamlit and data visualization libraries like Plotly helps ensure a smooth and interactive user experience. By uniquely focusing on **privilege-sensitive command detection**, this project addresses a gap in existing security tooling, offering a lightweight yet powerful educational solution that blends theoretical automata concepts with practical cybersecurity insight.

# Methodology

## Pattern Matching Using DFA

The core of the privileged command detection system is built upon the conversion of regular expressions representing high-privilege operations into Deterministic Finite Automata (DFA) for efficient pattern matching. The process involves these key steps:

* **Definition of Regular Expressions:** For each command category Privilege Escalation, User Management, File Permissions, System Configuration, and Destructive Operations—comprehensive regular expressions are defined to capture privileged command signatures. These patterns identify operations like 𝑠𝑢𝑑𝑜, 𝑐ℎ𝑚𝑜𝑑 777, 𝑢𝑠𝑒𝑟𝑎𝑑𝑑, and 𝑟𝑚 − 𝑟𝑓 that require elevated privileges.
* **Conversion to Non-deterministic Finite Automata (NFA):** Using Thompson's construction, each regular expression is transformed into an NFA. This handles command variations like:
  + Different argument formats (𝑐ℎ𝑚𝑜𝑑 777 vs 𝑐ℎ𝑚𝑜𝑑 𝑢 + 𝑥) o Command aliases (𝑟𝑚 − 𝑟𝑓 vs 𝑑𝑒𝑙 /𝑠 /𝑞) o Platform-specific syntax (Linux vs Windows)
* **Transformation to DFA:** NFAs are converted to equivalent DFAs via subset construction, enabling deterministic processing of scripts. This optimizes scanning by:
  + Eliminating transition ambiguity o Enabling single-pass scanning
  + Ensuring O(n) time complexity relative to script length

Before scanning, scripts undergo preprocessing where comments are removed using regex patterns like 𝑟′//? $|/\?$′ . This reduces false positives by excluding nonexecutable text. The DFA then processes the cleaned content sequentially, matching command patterns regardless of malicious intent—focusing instead on privilege requirements and potential risks. This DFA-based approach provides deterministic, linear-time scanning suitable for both interactive analysis (small scripts) and batch processing (scripts up to 200MB). The resulting matches are categorized by risk level and contextualized for user understanding.

## DFA Diagram

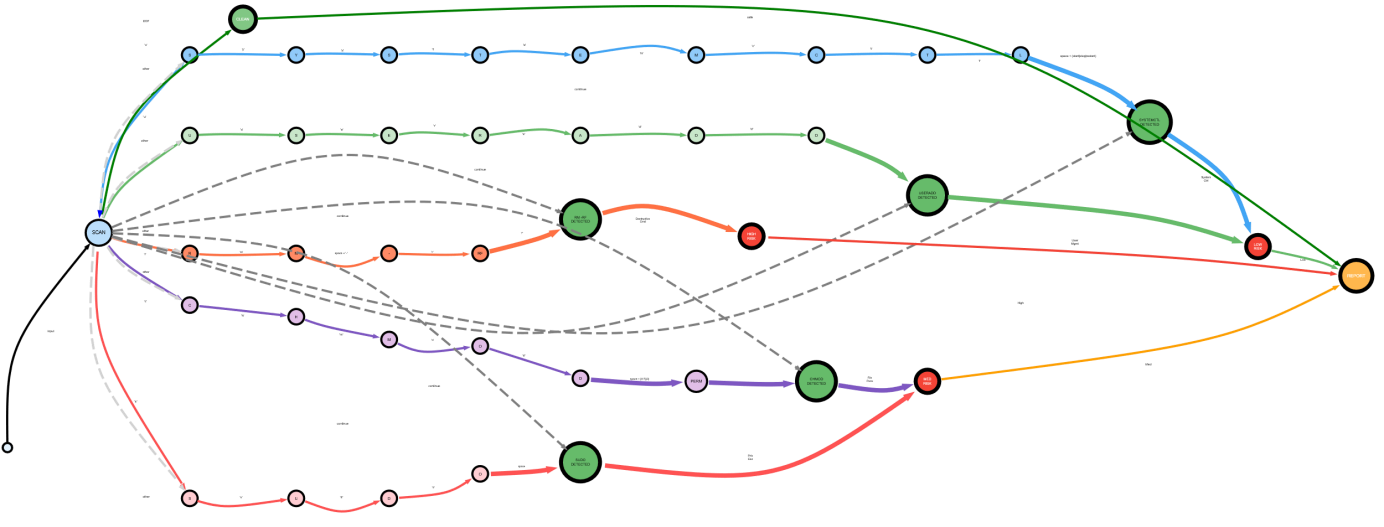


Figure - Complete Flow DFA of System

ip route show



Figure - Simplified DFA for Detecting <ip route show> (Network)

## Privileged Commands Patterns

The detection patterns are categorized based on command functionality and risk level as follows: **a. Privilege Escalation**

Patterns identify commands that elevate user privileges:

* 𝑟"\𝑏𝑠𝑢𝑑𝑜\𝑏" - Temporary admin rights
* 𝑟"\𝑏𝑠𝑢\𝑠 + 𝑟𝑜𝑜𝑡\𝑏" - Switch to root account
* 𝑟"\𝑏𝑑𝑜𝑎𝑠\𝑏" - Alternative privilege escalation

**Example Command**: 𝑠𝑢𝑑𝑜 𝑟𝑚 /𝑠𝑦𝑠𝑡𝑒𝑚/𝑙𝑜𝑔𝑠

1. **User Management**

Patterns detect account modification operations:

* + 𝑟"\𝑏𝑢𝑠𝑒𝑟𝑎𝑑𝑑\𝑏" - Create new users
  + 𝑟"\𝑏𝑛𝑒𝑡\𝑠 + 𝑢𝑠𝑒𝑟\𝑏" - Windows user management • 𝑟"\𝑏𝑝𝑎𝑠𝑠𝑤𝑑\𝑏" - Password changes

**Example Command**: 𝑢𝑠𝑒𝑟𝑎𝑑𝑑 − −𝑠𝑦𝑠𝑡𝑒𝑚 𝑎𝑑𝑚𝑖𝑛

1. **File Permission Modification**

Patterns capture file/directory permission changes:

* + 𝑟"\𝑏𝑐ℎ𝑚𝑜𝑑\𝑠 + [0 − 7]{3,4}\𝑏" - Numeric permission changes
  + 𝑟"\𝑏𝑖𝑐𝑎𝑐𝑙𝑠\𝑏" - Windows ACL modifications
  + 𝑟"\𝑏𝑠𝑒𝑡𝑓𝑎𝑐𝑙\𝑏" - Advanced Linux permissions **Example Command**: 𝑐ℎ𝑚𝑜𝑑 777 /𝑣𝑎𝑟/𝑤𝑤𝑤

1. **System Configuration**

Patterns detect service/OS configuration changes:

* + 𝑟"\𝑏𝑠𝑦𝑠𝑡𝑒𝑚𝑐𝑡𝑙\𝑠 + (𝑒𝑛𝑎𝑏𝑙𝑒|𝑠𝑡𝑜𝑝)\𝑏" - Service management
  + 𝑟"\𝑏𝑟𝑒𝑔\𝑠 + 𝑎𝑑𝑑\𝑏" - Windows registry edits
  + 𝑟"\𝑏𝑐ℎ𝑘𝑐𝑜𝑛𝑓𝑖𝑔\𝑏" - Service autostart configuration **Example Command**: 𝑠𝑦𝑠𝑡𝑒𝑚𝑐𝑡𝑙 𝑑𝑖𝑠𝑎𝑏𝑙𝑒 𝑓𝑖𝑟𝑒𝑤𝑎𝑙𝑙

1. **Destructive Operations**

Patterns identify high-risk data operations:

* + 𝑟"\𝑏𝑟𝑚\𝑠 𝑟𝑓\𝑠  - Forceful recursive deletion
  + 𝑟"\𝑏𝑓𝑜𝑟𝑚𝑎𝑡\𝑠  - Disk formatting
  + 𝑟"\𝑏𝑑𝑑\𝑠  - Raw disk operations

**Example Command**: 𝑟𝑚  𝑟𝑓 /𝑡𝑚𝑝/

1. **Information Access**

Patterns detect credential/system info access:

* + 𝑟"\𝑏𝑐𝑎𝑡\𝑠 +/𝑒𝑡𝑐/𝑠ℎ𝑎𝑑𝑜𝑤\𝑏" - Password hash access
  + 𝑟"\𝑏𝑠𝑢𝑑𝑜\𝑠 + −𝑙\𝑏" - List sudo privileges

**Example Command**: 𝑠𝑢𝑑𝑜 𝑐𝑎𝑡 /𝑟𝑜𝑜𝑡/.𝑠𝑠ℎ/𝑖𝑑\_𝑟𝑠𝑎

1. **Network Administration**

Patterns identify network configuration changes:

* + 𝑟"\𝑏𝑖𝑝𝑡𝑎𝑏𝑙𝑒𝑠\𝑏" - Firewall rule modification
  + 𝑟"\𝑏𝑟𝑜𝑢𝑡𝑒\𝑏" - Network routing changes

**Example Command**: 𝑖𝑝𝑡𝑎𝑏𝑙𝑒𝑠 − 𝐹 (𝑓𝑙𝑢𝑠ℎ 𝑓𝑖𝑟𝑒𝑤𝑎𝑙𝑙 𝑟𝑢𝑙𝑒𝑠)

All patterns are modularly stored in patterns.py with clear categorization, facilitating updates and maintenance. Example pattern structure:

|  |
| --- |
| FILE\_PERMISSION\_PATTERNS = [  r"\bchmod\s+[0-7]{3,4}\b", r"\bchown\b", r"\battrib\b"  ] |

**Pattern Design Principles:**

* 1. **Platform Agnostic**: Cover both Linux (chmod) and Windows (attrib)
  2. **Argument Awareness**: Match dangerous arguments (rm -rf)
  3. **Whitespace Tolerance**: Account for formatting variations
  4. **Case Insensitivity**: Detect SUDO and sudo equally

## Risk Scoring

Detected privileged commands are assigned risk scores reflecting their potential system impact. The overall risk level for a script is determined by the highest severity command detected. The scoring scheme is as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Command**  **Category** | **Risk Level** | **Score** | **Description** |
| Destructive  Commands | Critical | 3 | Can cause immediate system damage or data loss (e.g., rm -rf /, format C:) |
| Privilege Escalation | Critical | 3 | Grants administrative/root access  (e.g., sudo su, sudo bash) |
| File Permission  Changes | High | 2 | Modifies file/directory permissions  (e.g., chmod 777, icacls . /grant  Everyone:F) |
| System Configuration | High | 2 | Alters system services or OS settings (e.g., systemctl disable firewall, reg add) |
| User Management | High | 2 | Creates/modifies user accounts or groups (e.g., useradd, net user admin /add) |
| Sensitive  Information  Access | Medium | 1 | Accesses credentials or security configurations (e.g., cat /etc/shadow, sudo -l) |
| Network Administration | Medium | 1 | Modifies network/firewall configurations (e.g., iptables -  F, route del default) |
| No Privileged Commands | Clean | 0 | No elevated-privilege operations detected |

***Table*** *1****- Risk Severity Classification***

## Detailed Threat Table

|  |  |  |
| --- | --- | --- |
| **Token Type** | **Example Lexemes**  **(Keywords)** | **Description** |
| PRIV\_ESCAL ATION\_CMD | sudo, su root, runas,  setuid, doas | Commands used to escalate privileges or execute with elevated permissions. |
| USER\_MGMT  \_CMD | adduser, useradd, usermod, net user, passwd, groupadd,  dsadd, net localgroup | Commands related to user and group management in Unix and Windows environments. |
| FILE\_PERMIS  SION\_CMD | chmod 777, chmod 755, chown, icacls, attrib, setfacl,  takeown | Commands that modify file or directory permissions and ownership. |
| SYSTEM\_CO  NFIG\_CMD | systemctl enable,  systemctl start, sc config, regedit, powercfg, services.msc | Commands for managing system services,  configurations, and Windows registry. |
| DESTRUCTIV  E\_CMD | rm -rf /, del /s /q, mkfs, format, shutdown, reboot, dd if=, mv /system | Commands that are potentially harmful, capable of causing data loss or system crashes. |
| INFO\_GATHE  RING\_CMD | cat /etc/shadow, cat  /etc/passwd, type  \*.pem, whoami  /priv, sudo -l | Commands that gather information about system configurations or security credentials. |
| NETWORK\_C  MD | iptables, route,  netstat, ifconfig, ip route, netsh | Network administration commands used to configure or inspect networking. |
| WHITESPACE | spaces, tabs,  newlines | Delimiters that separate tokens, typically ignored during token analysis. |
| COMMENT | #, //, /\* ... \*/ | Comments in scripts, ignored during lexical analysis but used for code documentation. |
| STRING\_LITE  RAL | "password", "C:\\Windows\\Syst em32", 'admin' | String literals representing passwords, file paths, or other arguments in commands. |
| NUMERIC\_LI TERAL | 777, 1001, 0 | Numeric values used for permissions, configurations, or other numerical inputs. |
| UNKNOWN | Any unrecognized or malformed input | Default category for any invalid or undefined token during lexical analysis. |

***Table*** *2****- Threat Classification***

## Web Application

The detection engine is integrated into a Streamlit-based web interface that offers the following features:

* **File Uploader:** Supports uploading script files with extensions .txt, .js, .sql, and .sh, with a size limit of up to 200MB.
* **Text Input:** Allows users to paste script content directly for immediate analysis.
* **Threat Meter**: Utilizes a Plotly gauge chart displaying severity on a scale from 0 to 3, with color coding—green for clean, yellow for medium, orange for high, and red for critical threats.
* **Results Display:** Presents matched patterns in a tabular or expandable format, grouped by vulnerability type for clarity.
* **User Interface Enhancements:** Incorporates Lottie animations (e.g., shield animations) and custom CSS styling to improve user engagement and provide a modern, responsive experience.

# Implementation

## Detector Module (detector.py)

The detector module performs the following operations:

* **Cleaning:** Removes comments and non-code content using regular expressions (𝑒. 𝑔. , #.∗ ,//.∗,/∗ ∗/).
* **Pattern Matching:** Applies regex-based matching logic for multiple privileged command categories (𝑒. 𝑔. , 𝑠𝑢𝑑𝑜, 𝑐ℎ𝑚𝑜𝑑, 𝑟𝑚 − 𝑟𝑓, 𝑒𝑡𝑐. ).
* **Severity Scoring:** Classifies findings into severity levels (0 to 3) and maps them to tiers such as *Low*, *Medium*, *High*, or *Critical* risk.

## Pattern Definitions (patterns.py)

Contains categorized regular expressions for detecting privileged command usage. Examples include:

* **Privilege Escalation:** 𝑟"\𝑏𝑠𝑢𝑑𝑜\𝑏|\𝑏𝑠𝑢\𝑠",𝑒𝑡𝑐.
* **File Permission**: 𝑟"𝑐ℎ𝑚𝑜𝑑\𝑠 +\𝑑{3}",𝑟"𝑐ℎ𝑜𝑤𝑛\𝑠 +\𝑤 + "
* **Destructive Command:** 𝑟"𝑟𝑚\𝑠 + −𝑟𝑓\𝑠 +\/"
* **User Management:** 𝑟"𝑢𝑠𝑒𝑟𝑎𝑑𝑑\𝑠 +\𝑤 + ", 𝑟"𝑝𝑎𝑠𝑠𝑤𝑑\𝑠 +\𝑤 + "

Each category is used to identify elevated-privilege or risky operations within the script.

## Web Interface (app.py)

Implemented using **Streamlit**, the app features:

* **Animated Header:** Custom Lottie animation with glowing title and dark theme styling.
* **Script Input:** Allows either file upload (.txt, .sh, .js, .sql) or direct pasting of scripts.
* **Threat Visualization:** Interactive **Plotly gauge** to show privilege risk level with colorcoded severity.
* **Detection Results:**
* Privileged commands are listed with contextual explanations.
* Severity and risk tier are summarized with styling and feedback.
* Displays a “✅ Clean” message with confetti when no threats are found.

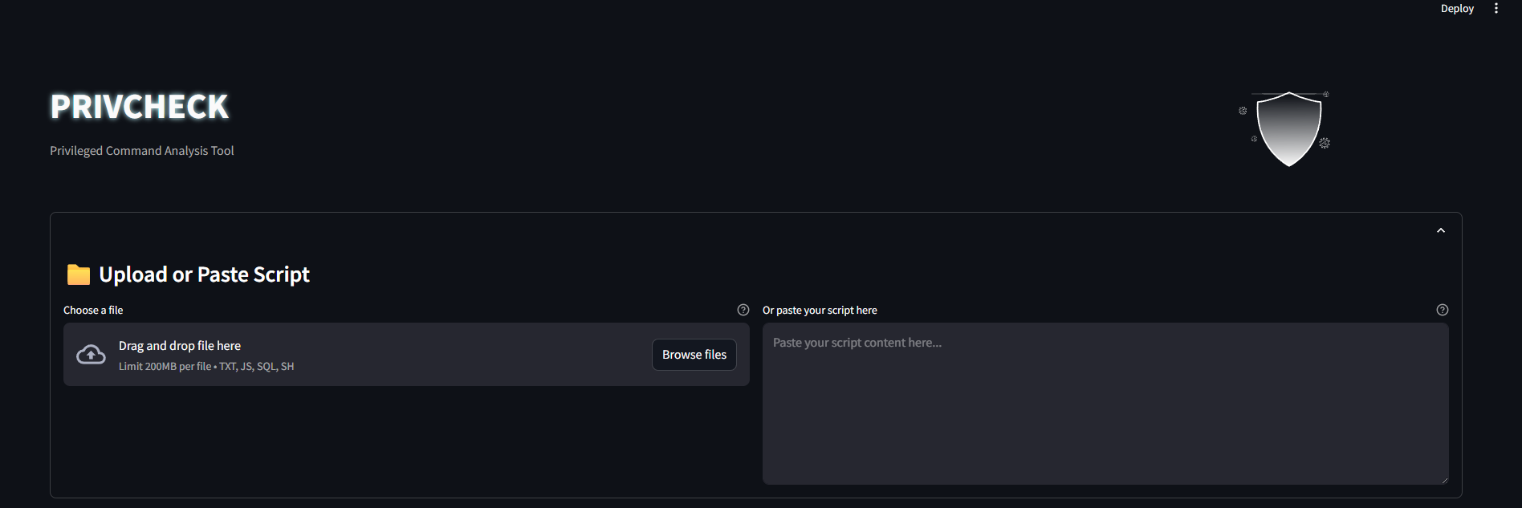


Figure - Privcheck Dashboard

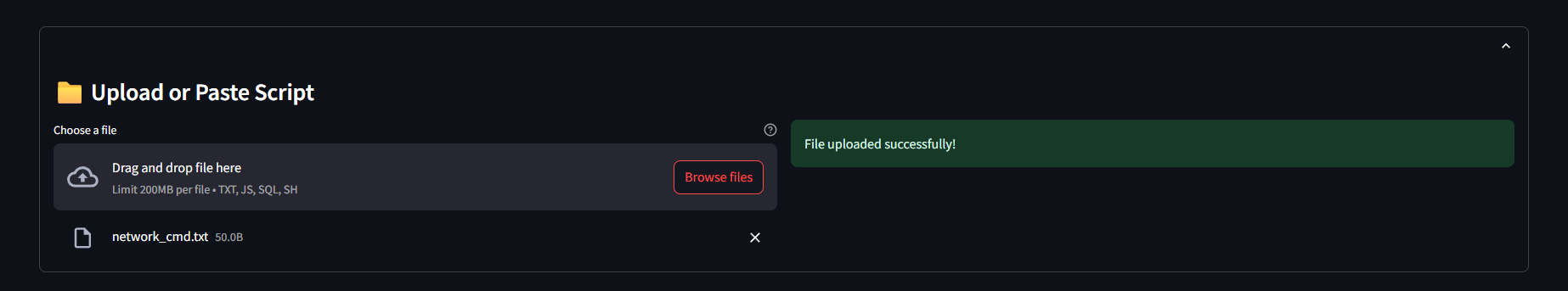


Figure - File Uploader & Script Pasting

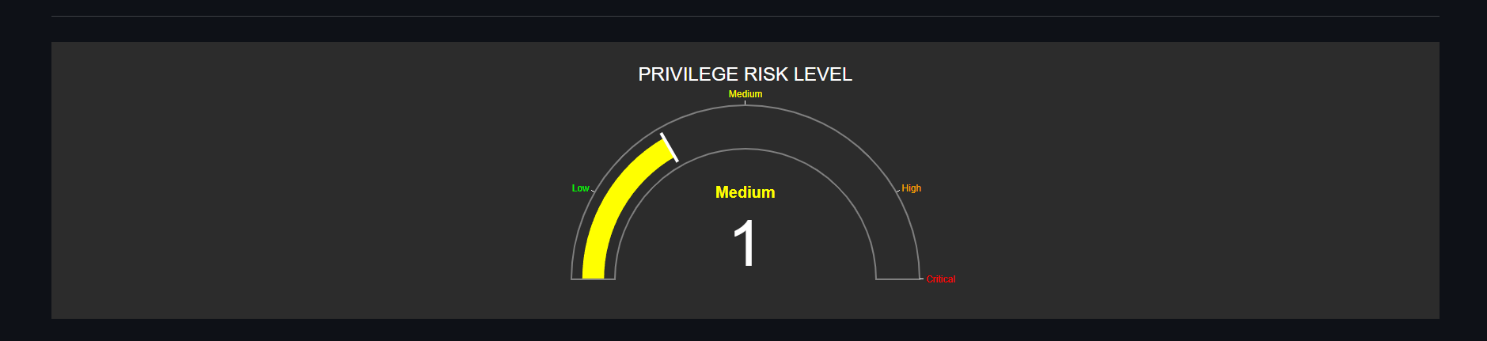


Figure - Threat Meter



Figure - Results

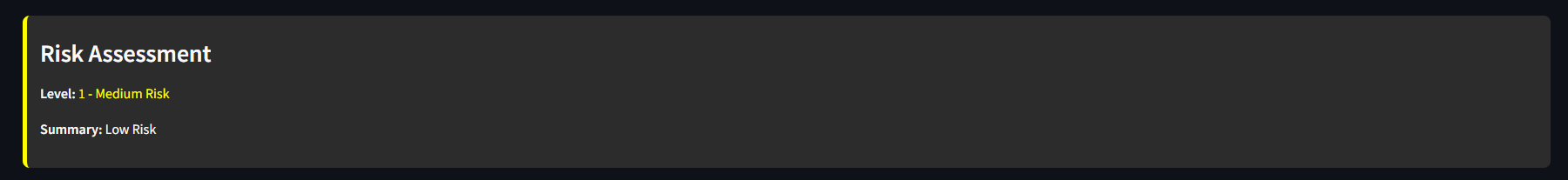


Figure - Assessment Overview

## Additional Assets

The project includes several supplementary assets that support both the functionality and testing of the vulnerability detection system:

* **shield\_animation.json:** A Lottie animation file used to enhance the visual appeal of the web interface header, providing an engaging and professional user experience.
* **Threats Folder:** Contains sample .txt script files used for testing and validating the detection engine. These text files include various scripts with known vulnerability patterns, enabling thorough evaluation of the system’s detection capabilities.

## Modules

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

# Results and Discussion

## Testing Scenarios

The Privilege Risk Detection System was thoroughly tested using various types of script inputs to evaluate its effectiveness in identifying high-risk, elevated-privilege commands across scripting environments such as **Shell**, **Bash**, and **Python**. The following scenarios summarize its detection capabilities:

* **Shell Scripts:** Scripts containing critical operations like sudo, rm -rf /, or systemctl stop were analyzed. The system correctly flagged these as **Critical** risks due to their potential to cause system damage or unauthorized control.
* **Privilege Misuse Scripts:** Files with commands such as chmod 777 file.txt or chown user:group /etc/passwd were tested. These were detected with **High** to **Critical** severity levels, indicating improper permission changes or access control violations.
* **User and Service Management Scripts:** Scripts using useradd, passwd, or systemctl to manage users and services were assessed. These were classified as **Medium** severity, depending on context, reflecting administrative-level operations.
* **Clean Scripts:** Scripts without any elevated-privilege or sensitive command patterns were tested to ensure accuracy. These were appropriately categorized as **Low Risk**, with no false positives, demonstrating the tool's reliability in normal scenarios.

These scenarios confirm the system’s **robust capability to detect and classify elevatedprivilege operations**, validating its practical utility in privilege risk assessment and secure script evaluation.

## Performance

|  |  |  |
| --- | --- | --- |
| **Script Size** | **Average Time (s)** | **Notes** |
| 1 KB | 0.3 | Small scripts, minimal overhead |
| 10 MB | 2.5 | Typical web application scripts |
| 100 MB | 12.8 | Large scripts, DFA remains efficient |

*Table 3- Processing Times for Script Analysis*

## Limitations

While the privilege risk detection system is effective in identifying elevated-risk commands, it also presents several limitations:

* **False Positives:** The pattern-based matching may sometimes flag benign administrative commands (𝑒. 𝑔. , 𝑠𝑢𝑑𝑜 𝑎𝑝𝑡 𝑢𝑝𝑑𝑎𝑡𝑒) as threats when used in safe contexts. Manual review is still required to differentiate legitimate operations from potential misuse.
* **Limited Context Awareness:** The detection logic evaluates command presence without fully understanding their execution context. For example, 𝑐ℎ𝑚𝑜𝑑 777 might be necessary in specific deployment scripts but is always flagged due to its inherent risk.
* **Pattern Maintenance:** The system depends on a fixed set of regular expressions to identify risky command patterns. As scripting practices evolve or new high-risk utilities emerge, the pattern database needs periodic updates to ensure continued accuracy.
* **Static Analysis Only:** The analysis is purely static, inspecting script content without evaluating actual execution or runtime conditions. As a result, it may overlook risks arising from dynamically constructed or obfuscated commands that are only revealed during execution.

# Conclusion

This project demonstrates the effectiveness of rule-based static analysis in identifying high-risk, privileged command usage within scripts, providing a practical and accessible tool for privilege risk assessment. By scanning script content for predefined elevated-privilege patterns—such as 𝑠𝑢𝑑𝑜 , 𝑐ℎ𝑚𝑜𝑑 777 , or 𝑟𝑚 − 𝑟𝑓 —the system empowers developers, administrators, and cybersecurity learners to evaluate potential misuse of system-level operations quickly and reliably. The integration of this detection engine into a modern **Streamlit-based interface** enhances usability by offering real-time analysis, intuitive visual feedback (like animated risk gauges), and actionable summaries. This seamless blend of foundational cybersecurity heuristics with user-friendly design bridges the gap between technical knowledge and practical security awareness, making the tool both educational and operationally valuable.

# References

1. J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston, MA, USA: Pearson Education, 2006.
2. M. Sipser, *Introduction to the Theory of Computation*, 3rd ed. Boston, MA, USA: Cengage Learning, 2012.
3. B. Chess and J. West, *Secure Programming with Static Analysis*. Boston, MA, USA: Addison-Wesley, 2007.
4. National Institute of Standards and Technology, “Security and Privacy Controls for

Information Systems and Organizations,” NIST Special Publication 800-53, Rev. 5, Sep. 2025. [Online]. Available: [https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final.](https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final) [Accessed: Jun. 18, 2025].

1. Streamlit, “Streamlit Documentation,” 2025. [Online]. Available: [https://docs.streamlit.io/.](https://docs.streamlit.io/) [Accessed: Jun. 18, 2025].
2. Plotly, “Plotly Python Documentation,” 2025. [Online]. Available: [https://plotly.com/python/.](https://plotly.com/python/) [Accessed: Jun. 18, 2025].
3. LottieFiles, “LottieFiles Animation Platform,” 2025. [Online]. Available: [https://lottiefiles.com/.](https://lottiefiles.com/) [Accessed: Jun. 18, 2025].