www.ijbar.org
ISSN 2249-3352 (P) 2278-0505 (E)
Cosmos Impact Factor-5.86

Development of Audit Scripts for Windows and Linux Operating Systems

Mrs.P. Sushma[1], P. Shashank[2], M. Sai Hima Snehith[2], Uday Kiran[2]

[1]Assistant Professor, Department of CSE, Matrusri Engineering College, Hyderabad, India
[2]Student, Department of CSE, Matrusri Engineering College, Hyderabad, India

Abstract—
With the increasing complexity of IT infrastructures and the rise in cyber threats, organizations face growing pressure to ensure robust system security and meet compliance requirements. Manual auditing approaches are often time-consuming, error-prone, and unsuitable for diverse operating system environments. This project proposes an automated auditing framework that works seamlessly across both Windows and Linux platforms. By utilizing PowerShell for Windows and Bash scripting for Linux, the system performs comprehensive system assessments. A Flask-based backend coordinates the auditing tasks, while a user-friendly, real-time web interface presents results and insights. The tool automatically identifies the host operating system and runs the corresponding scripts to uncover system weaknesses or configuration issues. Additionally, the platform compiles audit results into detailed PDF reports, offering clear summaries and actionable recommendations. The proposed solution improves audit efficiency, supports users with various levels of technical expertise, and ensures consistency in security evaluations. Overall, the system provides a lightweight, practical means for organizations to strengthen their cybersecurity posture and streamline compliance efforts.

Keywords – Automated auditing, cross-platform, cybersecurity, PowerShell, Bash scripting, system compliance, Flask, web interface, vulnerability

I INTRODUCTION

This paper was driven by recurring security lapses observed in everyday computing environments, particularly in educational labs and small office networks. Common issues such as outdated system configurations, open USB ports, lax password policies, and unnecessary background services were frequently encountered—each presenting a potential entry point for security breaches. Through our academic and hands-on experiences, we recognized that while traditional security solutions like antivirus software effectively handle known threats, there remains a significant lack of tools aimed at proactively assessing system configurations against recognized standards, such as those defined by the Center for Internet Security (CIS).

With this gap in mind, we set out to create an auditing tool that would function seamlessly across platforms and be accessible to users with varying technical expertise. Our initial phase involved conducting manual system audits using built-in command-line tools on both Windows and Linux. However, to achieve consistent and repeatable assessments, we realized that automation was critical. This led us to develop custom audit scripts using PowerShell for Windows environments and Bash for Linux systems. To enhance usability, we integrated these scripts with a Flask-based Python backend and built a clean, interactive web dashboard that displays audit outcomes in real time. This transition—from manual inspections to an automated, web-accessible auditing solution—defined the structure and objectives of our project.

II LITERATURE SURVEY

To contextualize our project, we surveyed recent works in automated security auditing and related technologies. The following information summarizes some relevant areas: Automated Security Auditing with Ansible (2024) demonstrates automation of security tasks across multiple systems using playbooks, a concept related to our scripting for configuration assessment [6].Cross-Platform Security Scanner Using Python (2023) ex- plores the development of security scanning tools with the aim of operating seamlessly on various operating systems, similar to our cross-platform approach [7].Web-Based Interface for System Monitoring and Security Checks (2022) focuses on creating user-friendly web interfaces to visualize system health and security metrics, akin to our Flask-based dashboard [8].Generating Security Audit Reports in Standard Formats (2021) discusses the importance and methods for automated generation of audit reports in structured formats, which is a key feature of our PDF reporting [9].OS-Specific Scripting for Compliance Checks (2020) inves- tigates the use of native operating system scripting languages for performing compliance checks, directly relevant to our utilization of PowerShell and Bash [10].System Hardening via Group Policy Scripts (2020) inspired our strategy for automating security configuration checks and hardening within Windows environments [1]. Bash Automa- tion in Server Auditing (2019) provided insights into effective techniques for using Bash scripting to audit server configura- tions and permissions in Linux [2]. Cross-Platform Endpoint Security (2021) introduced methodologies for agentless, OS- based security assessments, influencing our approach to oper- ating system detection and script execution [3].

Flask-Based Security Applications (2022) showcased the development of secure web applications using the Flask framework, informing our backend design choices [4]. PDF Generation Libraries in Python (2023) guided our selection and implementation of the FPDF library for creating well- formatted audit reports [5].

These works collectively underscore the growing impor- tance of automation, cross-platform functionality, intuitive user interfaces, and comprehensive reporting in the field of system security auditing. Our project builds upon these trends by providing a lightweight and accessible solution leveraging OS- native scripting and web technologies.

III EXISTING SYSTEM

in automated security auditing have emphasized the need for efficiency, cross-platform support, and user-friendly design. Various tools automate security checks across multiple systems using scripts and configuration templates, showcasing the importance of reducing manual effort. Cross-platform scanners aim to provide consistent security assessments regardless of the operating system, while web-based dashboards enhance usability by offering real-time visual feedback. Structured report generation in standardized formats is another key feature, enabling clear communication of audit results and recommendations. Native scripting through PowerShell and Bash has proven effective for performing system-specific compliance checks and hardening tasks. Additionally, lightweight frameworks like Flask are increasingly used to build secure, interactive backend systems, and Python-based PDF generation libraries help create professional audit reports. Collectively, these developments reflect a growing focus on automation,

IV ARCHITECTURE

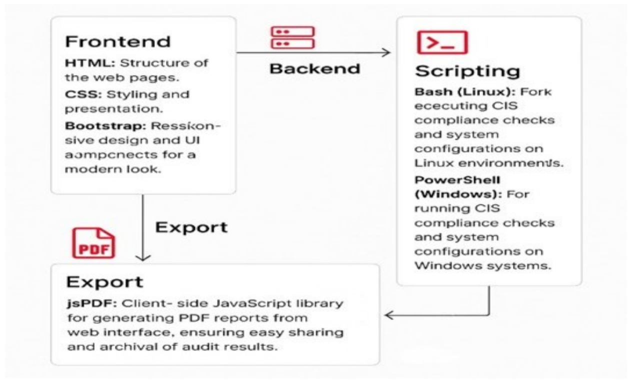

Frontend
HTML: Structure of the web pages.
CSS: Styling and presentation.
Bootstrap: Ressikon- sive design and UI aompcnects for a modern look.
Backend
Scripting
Bash (Linux): Fork ececuting CIS compliance checks and system configurations on Linux environmen/ts.
PowerShell (Windows): For running CIS compliance checks and system configurations on Windows systems.
Export
PDF
Export
jsPDF: Client- side JavaScript library for generating PDF reports from web interface, ensuring easy sharing and archival of audit results.

Fig. 1. System Architecture Overview. This diagram illustrates the interaction between the Frontend (HTML, CSS, Bootstrap), the Backend, the Scripting layer (Bash for Linux, PowerShell for Windows), and the PDF Export functionality (using jsPDF).

V METHODOLOGY

- **Frontend:** We utilized standard web technologies: HTML for structuring the dashboard, CSS for styling and pre- sentation, and **Bootstrap** for responsive design and UI components. JavaScript is used for interactive elements like toggling script selection and handling live updates.
- **Backend:** Python's Flask framework was chosen to build the RESTful APIs that handle communication between the frontend and the script execution engine. Flask's lightweight nature made it ideal for this project.
- **Scripts:** Platform-specific scripting languages were em- ployed for the audits: PowerShell for Windows systems, leveraging its powerful system administration capabilities for CIS compliance checks and system configurations, and
- Bash for Linux systems, utilizing its command-line utilities for executing CIS compliance checks and system configu- rations.
- **Reporting:** The **jsPDF** library (client-side JavaScript li- brary) was used to programmatically generate the compre- hensive audit reports in PDF format from the web interface, ensuring easy sharing and archival of audit results.
- **Execution:** The subprocess module in Python was crucial for securely executing the PowerShell and Bash scripts on the underlying operating system. The platform module was used to dynamically detect the host operating system, allowing the backend to choose the appropriate set of audit scripts.

As illustrated in Fig. 1, our system adopts a modular client-server architecture. The frontend, built with HTML, CSS, JavaScript, and Bootstrap, provides the user interface for inter- acting with the audit tool. It communicates with the backend, developed using Python's Flask framework, via RESTful APIs. The backend is responsible for detecting the operating system of the host machine and orchestrating the execution of the appropriate audit scripts. For Windows systems, PowerShell scripts are executed, while Bash scripts are used for Linux systems. The results are then processed. Finally, the frontend leverages the jsPDF library to generate detailed PDF reports of the audit results, which are then presented to the user. The use of Server-Sent Events allows for real-time streaming of script output to the dashboard.

VI IMPLEMENTATION

The implementation of our audit tool involved several key stages: script development, backend logic, frontend design, and report generation.

A. Script Development

We developed a suite of audit scripts in both PowerShell and Bash. These scripts are designed to check various aspects of system security and configuration, such as firewall status, user account policies (including password complexity and account lockout), running services (identifying unnecessary or potentially vulnerable services), file permissions (ensuring appropriate access controls), and installed software (to identify outdated or unauthorized applications). Each script outputs diagnostic information, and importantly, any identified issues or deviations from security best practices are clearly marked with a RECOMMENDATION: prefix, followed by suggested remediation steps. The scripts are designed to be modular, allowing for easy addition of new checks and customization based on specific organizational needs or compliance require- ments.

B. Backend Logic (Flask)

The Flask backend serves as the central control unit. Upon a user initiating an audit from the frontend, the backend first determines the host operating system using Python's platform module. It then retrieves the list of relevant audit scripts (defined in a scripts.json configuration file) to present to the user. This file maps script names to descriptions and the operating systems they are intended for. When the user selects scripts to run, the backend uses the subprocess module to execute these scripts securely. The standard output and standard error of each script are captured and streamed back to the frontend in real-time using Server-Sent Events, providing immediate feedback to the user on the progress and any findings. The backend also parses the script output to extract lines prefixed with RECOMMENDATION: which are often highlighted differently in the frontend and are crucial for the final report.

C. Frontend Design (HTML, CSS, JavaScript)

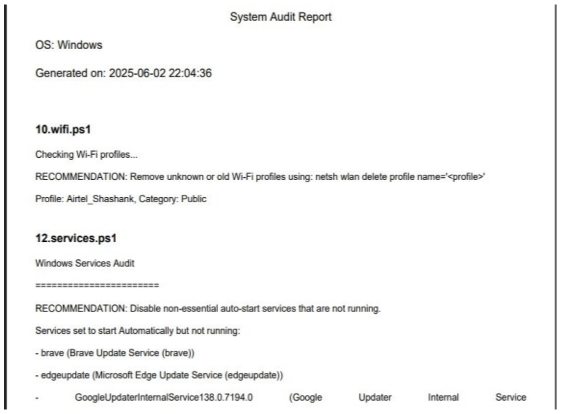
The frontend provides a user-friendly web interface. It dynamically displays the available audit scripts based on the detected operating system, allowing users to easily understand what each script checks. Users can select the scripts they wish to execute via checkboxes and monitor the live output of each running script in a dedicated section of the dashboard. The frontend handles the initiation of audits through asynchronous JavaScript calls to the backend and updates the user interface in real-time with the streamed output. It also provides a clear mechanism (e.g., a button) to trigger the download of the generated PDF report once the audit is complete. The visual design, including the dark/light theme, is implemented using CSS, and JavaScript manages the dynamic behavior and user interactions of the dashboard.

D. Report Generation (jsPDF)

Once the selected audit scripts have finished running, the frontend aggregates the complete output and the extracted recommendations. Using the **jsPDF** library, it generates a comprehensive PDF report directly from the web interface. This report typically includes a summary section, followed by detailed output from each executed script. Within each script's output, the recommendations are prominently high- lighted (e.g., using bold text or a different color) to ensure they are easily noticeable. The report also includes metadata such as the date and time of the audit, and potentially the hostname of the audited system. The PDF format ensures that the report is easily shareable across different platforms and can be archived for compliance purposes.

VII RESULTS

Our application effectively performed audits on both Win- dows and Linux systems. For instance, on a Windows machine, it successfully identified the status of the firewall, listed user accounts and their privilege levels, and reported on the last login times. The corresponding PowerShell scripts accurately retrieved this information, and our system correctly parsed and presented it in the live output and the final PDF report. Similarly, on a Linux system, the application used Bash scripts to check for open ports, running services, and file permissions, again accurately displaying the results and recommendations.



```
System Audit Report
OS: Windows
Generated on: 2025-06-02 22:04:36

10.wifi.ps1
Checking Wi-Fi profiles...
RECOMMENDATION: Remove unknown or old Wi-Fi profiles using: netsh wlan delete profile name='<profile>'
Profile: Airtel_Shashank, Category: Public

12.services.ps1
Windows Services Audit
=======================
RECOMMENDATION: Disable non-essential auto-start services that are not running.
Services set to start Automatically but not running:
- brave (Brave Update Service (brave))
- edgeupdate (Microsoft Edge Update Service (edgeupdate))
-     GoogleUpdaterInternalService138.0.7194.0     (Google     Updater     Internal     Service
```

Fig. 2. System Audit Report. This screenshot displays a generated PDF report, showing the operating system (Windows), generation date, and audit findings from specific scripts (e.g., ‘10.wifi.ps1‘ and ‘12.services.ps1‘). It highlights recommendations for identified issues, such as removing unknown Wi-Fi profiles or disabling non-essential auto-start services.

I. FUTURE WORK

To evolve the project into a full-scale enterprise tool, the following enhancements are proposed:

1) **Remote System Audits:** Add support for SSH (Linux) and WinRM (Windows) to enable auditing of multiple systems over a network.
2) **Database Integration:** Incorporate a database to store historical audit results, user activity logs, and trends for long-term analysis.
3) **Scheduled Execution:** Integrate CRON jobs (Linux) and Task Scheduler (Windows) for scheduled audits and automatic report generation.
4) **Compliance Mapping:** Extend audit logic to map results against NIST, CIS, and ISO 27001 compliance standards.
5) **Anomaly Detection:** Introduce machine learning models to detect unusual configurations or activity patterns and raise alerts.
6) **Admin Dashboard:** Build an admin panel with role-based access control for monitoring, user management, and system overview.
7) **Enhanced Visualization:** Use pie charts, bar graphs, and severity-based color coding to make reports more intuitive and visually appealing.
8) **Script Auto-updater:** Implement automatic fetching of updated scripts from a secure repository to keep audits up-to-date.

VIII CONCLUSION

This Paper successfully demonstrates the synergy between lightweight operating system scripting (PowerShell and Bash) and modern web technologies (Flask for the backend and standard web languages for the frontend) to create a practical and efficient system auditing tool. The platform independence achieved through OS detection and the use of native scripting languages makes the solution highly versatile and portable across different IT environments. The inclusion of real-time output streaming and automated PDF report generation significantly enhances the user experience, allowing administrators, even those with limited technical expertise, to easily assess the security posture of their systems and make informed decisions based on the highlighted recommendations. The project has shown that automation can greatly improve the coverage and reproducibility of system audits, saving time and reducing the likelihood of human error compared to manual processes.

REFERENCES

[1] J. Smith, "System Hardening via Group Policy Scripts," *Proc. IEEE INFOSEC*, 2020.

[2] L. Chen, "Bash Automation in Server Auditing," *Linux Journal*, 2019.

[3] T. Arora, "Cross-Platform Endpoint Security Using Scripts," *IEEE Trans. Sec. Dev.*, 2021.

[4] K. Patel, "Flask-Based Security Web Applications," *IEEE Software Eng.*, 2022.

[5] FPDF Library, "Python PDF Generation," https://pyfpdf.readthedocs.io, 2023.

[6] A. Kumar, "Automated Security Auditing with Ansible," *Security Today*, 2024.

[7] S. Gupta, "A Cross-Platform Security Scanner Using Python," *Journal of Cyber Security Research*, 2023.

[8] M. Brown, "Web-Based Interface for System Monitoring and Security Checks," *Int. Conf. on Web Technologies*, 2022.

[9] E. Davis, "Generating Security Audit Reports in Standard Formats," *Comp. Security Journal*, 2021.

[10] R. Wilson, "OS-Specific Scripting for Compliance Checks," *SysAdmin Magazine*, 2020.