**Course Project Report**

Michael D. Pineda

National University

CYB 333: Security Automation

Professor Sean Ayersman M S, CISSP

July 3, 2025

**Introduction**

The project, Automated Log Monitoring and Alerting System, is a scalable, robust, and secure implementation of an actual system log monitoring tool built using Python for real-time security relevant events on a system. It successfully addresses the challenge of detecting security-relevant events, like repeated failed logins or privilege escalations, by automating log pattern recognition and providing actionable alerts to security teams in a timely fashion. The implementation of cutting-edge monitoring, notification, and configuration management techniques within this tool successfully showcases reducing the load of manual monitoring and advanced security automation in line with industry and course standards.

**Development Process**

The overall methodology in working on the project would be characterized as incremental with some aspects of agile software development. In the planning phase, the initial project scope was defined, some of the research on the existing open-source solutions in the realm of log monitoring was completed, the requirements were gathered in regard to pattern detection, notifications, and cross-platform support, and a rough architecture was sketched with the core log file monitoring engine and modular pattern matching/alerting components. Design phase included specifications of high-level system components and their interactions. The following decisions were made Python as a language of choice for implementation due to readability, large active community, and libraries supporting both file I/O and system-level operations. The watchdog library was used to provide an efficient event-driven mechanism to eliminate the need for polling log files and react to their changes. For pattern detection, Python's built-in re module was used to allow for flexible regular expression matching, as well as to better match a broad set of security-relevant events. For notification, support for multiple alerting channels was used (file-based alerts and email notifications via smtplib). Configuration files were used for customization and deployment considerations.  
Implementation was an incremental approach, starting with the initial MVP and then gradually adding new features and improvements. Features were developed and tested in isolation before being merged into the main application. This helped in early bug detection, easier maintenance, and more fine-grained control over feature development. Code documentation was kept up to date during the implementation phase, and Git was used for version control to track changes and easily revert to previous versions if needed.  
GitHub Copilot and ChatGPT are some of the AI tools and techniques that were leveraged in some stages of the project. Copilot was used for generating boilerplate code and catching exception handling, and ChatGPT was used both in the research stage as well as a general debugging tool and for inspiration on alternative ways of implementing concepts. Code produced with AI assistance was then reviewed and adapted as necessary to meet the project requirements.  
Testing phase was done in both the Windows and Linux environments with some simulated log entries and attack scenarios to validate the detection accuracy and the reliability of the alerting mechanisms. Security measures in the work on the script included secure coding practices and error handling as well as a modular design principle. The final product is a secure, effective, and easily extensible solution.

**AI Tools Used**

AI tools were used at various points of the project, e.g., ChatGPT was used for gathering information on log file monitoring best practices/error handling and secure configuration of the notification system and for answering questions to clarify assumptions (file system event) and generating/refining code snippets (log file parsing/robust regex patterns). Prompts like "Python code for monitoring log file changes with watchdog" or "secure way to send email alerts in Python" were employed for contextually appropriate assistance/guidance to hasten the efficient implementation of those features.  
GitHub Copilot was used to accelerate the coding process and to provide autocomplete suggestions and context-aware recommendations, e.g., for exception handling, iterative development, and edge case management. For example, during the implementation of the email alerting module, Copilot offered code snippets that covered authentication errors and connection timeouts, later refined and adapted to improve security and reliability. The tools were used to generate test cases, debug complex logic, and validate adherence to security best practices (e.g., no hardcoded sensitive information).

**Challenges and Issues with the Project**

Automating the log monitoring and alerting functionalities involved several difficulties as well. I first faced difficulty in arriving at the log monitoring approach. This was because having a robust log monitoring mechanism required me to evaluate a set of criteria such as low resource footprint of the monitoring script, being able to capture real-time changes in the log file, parsing the unstructured log format, supporting variants in the log format. Therefore, it also involved experimenting with different python libraries and evaluating how libraries like watchdog and tailer could be used for continuous monitoring of the log files and reaching the best solution iteratively.  
Secure and reliable alerting was another thing that I struggled a lot in doing. The main problem here is not just to ensure reliable alerting but to think about ways on how I can manage the credentials required for SMTP securely without hard-coding them on the script. Network failures or interruptions and gracefully handling them such that it does not break the scripts was another problem that I had to address. Adding exception handling, retry logic and securing credentials using environment variables helped to solve the risks and concerns and script a reliable solution.  
Testing and validation was another domain that posed a lot of difficulties to me, particularly because of edge cases consideration. This was because testing the system behavior across different failure modes, corrupt log files, system crash, malformed log entries, etc. needed an exhaustive testing strategy that involved writing custom scripts as well as leveraging AI tools to generate numerous error conditions. The testing and refinement iterations that I did also helped me realize the corner cases that could emerge due to newer inputs with each iteration leading to some more additions in the parsing logic and error-handling.

**Learning Reflection**

By working on this project, I have improved my experience in almost all aspects of security automation. Automating an effective log monitoring and alerting solution from the ground up helped me immensely by providing real-world experience that was complementary to the theoretical learning the program provided. The primary lesson I have learned during the process is the need for a wider lens on the entire project life cycle of an actual security automation solution. Designing and iterative development of the system have given me a deeper appreciation for how a real-world automation problem would be approached in terms of identifying functional requirements, prototyping, coding, testing, debugging, and ultimately deploying a solution into a production environment. In doing so, I applied both theoretical knowledge and was able to uncover new ground on the particular nuances that do not come up in learning environments.  
The need for modularity and clarity of code in the project has taught me to think about potential future challenges and best practices in code management. Thinking about this project in terms of high-level functional components like log parsing, pattern matching, and alerting has not only improved my proficiency in the use of tools like GitHub for version control but also helps me keep track of changes and revert to older versions if needed, which is essential in real-world development and is a practice that is easy to take for granted while learning.  
I have learned to balance automation efficiency and security fundamentals. While automation is clearly more efficient and responsive, there are also new edge cases and challenges with no two instances of automation being exactly the same, which can also be new sources of vulnerabilities if left unchecked. This project also made me more fastidious and careful in terms of adherence to best practices and ensuring secure and careful design and implementation, in terms of protecting sensitive data, avoiding hard-coding credentials, using exception handling, etc., to ensure that you are not unintentionally making entire systems fail fail. Experimenting AI tools for code generation and suggestions and calibrating an appropriate use amount of speed vs thorough review prior to using any such code in this project was another challenge for working with this project. The ChatGPT and GitHub Copilot were great for speeding up and getting creative but also required some care in customization, in terms of security and specific functionality.  
As a professional, I have learned to have a growth mindset, stay in the game, and keep up to date. I believe that this project contributed to my professional growth. I have improved my skills, and thus, this has developed my competencies. The work I have done in this capstone project contributes to my professional growth by helping me to gain a fundamental understanding of how security automation projects are typically approached. This project has increased my confidence in my ability to both develop security automation and implement it, given me a mindset of life-long learning and adaptation to the rapidly evolving landscape, and taught me to pay close attention to a balance between innovation and more fundamental principles of security.

**Time Tracking**

Time Management for the project was definitely a learning curve, as well as a challenge. I had my initial time management plan in place at the beginning as a guide for myself, as well as a scheduled amount of time I wanted to dedicate to working on the project each week, but as is often said and then immediately proven untrue; it was easier said than done and I encountered more challenges than I originally expected. Building a comprehensive automated log monitoring and alerting solution required a good amount of trial and error to debug and work out edge cases, and the additional time I spent on the project to incorporate alerting capabilities, automate more and implement error handling often took longer than I expected. I also had several late nights and spent more time on weekends than I wanted to break down and debug and fix some unforeseen issues I was running into.  
Overall, from start to finish I probably spent about 25 to 30 hours total, but much of that time was simply troubleshooting and figuring out how to troubleshoot or solve problems that I would run into when I was trying to adapt more general code I found in examples to work for my particular security use case. I definitely learned to be flexible with my time, break tasks down into smaller steps, and be more resourceful when seeking additional help (AI included) when I was running into roadblocks. All in all, it was a little frustrating but it also helped me improve my time management skills, and it taught me patience and grit in building complex security automation tasks.

**Team Collaboration**

This was an individual project; however, I simulated best collaborative practices by maintaining regular commits, documenting all changes, and structuring code to be clear for reviewers.

**Screenshots and Code Demonstration**

I have attached screenshots of all the steps of the code execution process in the public GitHub repo. It can be seen from the attached screenshots that, to test the program, I have prepared my environment and generated events to be detected in the system logs.I have also set the target event for the script and created a rule in Suricata for that event to ensure that it will be detected correctly. I have also captured screenshots of the proper execution of the log monitoring script and detection of the target event and alerting to this event and the use of Jupyter Notebook for analysis and visualization of data or findings, inline comments and code blocks, and for the creation of documentation of development and testing processes using the Markdown feature of Jupyter Notebook.  
For each of the major scripts, including log\_monitor.py and additional alerting modules, I have provided detailed in-line comments on key parts of the logic and important functions. Step-by-step instructions are provided in the README.md file that will enable others to reproduce my setup and the screenshots I have provided using their own environment. Terminal outputs and appropriate graphical output (if applicable) are both captured in the provided screenshots to serve as proof of successful program execution and handling of error scenarios.

**Conclusion**

On reflection, I feel that this project allowed me to further solidify my knowledge and my understanding of how important automation is in today's cybersecurity landscape. Designing and building a fully functioning log monitoring and alerting system allowed me to directly apply some of the concepts and best practices that I have learned about security automation, while also giving me first-hand experience with the challenges and intricacies of creating working and reliable automation that can be used for practical cybersecurity scenarios. In particular, this experience has provided me with additional insights into the value of careful planning and iterative testing as well as the use of both traditional programming approaches and AI-based programming assistants to overcome these challenges.  
One last thing that I have come to truly understand and appreciate through this whole process is the value of good documentation, code hygiene, and general maintenance to any security automation endeavor. Thus, with an eye towards the future and the potential to build out and improve upon this system down the road, I see a couple areas for possible extension that are particularly interesting to me. This may include, for instance, the addition of more robust analytics and reporting capabilities, the integration of machine-learning based anomaly detection, or even the expansion of this system as a foundation for a more fully-featured security information and event management (SIEM) platform. In either case, I am thankful for the experience and skills that I have acquired from this project and look forward to applying them to future security automation endeavors.

**References**

ChatGPT. (2024). OpenAI language model [Large language model]. https://chat.openai.com

GitHub Copilot. (2024). Your AI pair programmer. https://github.com/features/copilot

Hussey, M. (2023). Python logging cookbook. Python Software Foundation. https://docs.python.org/3/howto/logging-cookbook.html

Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations (2nd ed.). IT Revolution Press.

Suehring, S. (2024). Learning DevSecOps: A practical guide to processes and tools. O'Reilly Media.

Python Software Foundation. (2024). Python documentation: logging. https://docs.python.org/3/library/logging.html

Youseff, L. (2021). Best practices for monitoring and alerting in cybersecurity. SANS Institute Whitepaper. https://www.sans.org/white-papers/best-practices-monitoring-alerting/

Pineda, M. (2025). CYB333-Midterm-Socket-PortScanner [Computer software]. GitHub. https://github.com/mpineda27/CYB333-Midterm-Socket-PortScanner