CYB333 Project: Hybrid Intrusion Detection System with Honeypot Logging

Robert Kralle

National University

CYB333 Security Automation

Professor Todd D. Raines

10/26/2025

**1. Introduction**

The purpose of this project was to design and implement a hybrid cybersecurity automation system that combines a Python-based Honeypot with a Simple Intrusion Detection System (SIDS). I chose this project out of curiosity and a desire to challenge myself by doing something both practical and advanced. I could have chosen one or the other, but I wanted to combine them to maximize their effect and deepen my understanding of both tools.

My goal was to improve my Python skills and learn how automation ties into real-world cybersecurity functions. This project also gave me the opportunity to think like an analyst by examining logs and alerts that reflect potential attacker behavior.

The Hybrid IDS + Honeypot system autonomously detects, logs, and analyzes suspicious network activity while functioning as a controlled decoy to study the behavior, frequency, and intent of unauthorized connection attempts. It demonstrates how an automated defensive setup can detect intrusion attempts, analyze attacker behavior, and log critical data without human intervention.

**2. Development Process**

I began by creating a GitHub repository to manage version control and store all my code, screenshots, and documentation. I structured the environment with three main folders: /data for logs, /screenshots for visual proof, and /docs for my write-up.

The first stage was developing the Honeypot script, honeypot.py. I based its structure on the client-server setup I had previously built during my midterm project. It listens on multiple ports (8080, 8022, and 33060) and logs incoming connections into data/honeypot.log. This script records the timestamp, source IP, and port number of each connection, allowing me to see every access attempt in real time. Once the logs began populating and I saw the alerts trigger simultaneously, I knew I had built something functional.

Next, I wrote the Intrusion Detection System (sids.py), which monitors the honeypot log for patterns such as repeated connections from the same IP address within a short timeframe. When a threshold is exceeded, the IDS writes an alert into data/alerts.log. Both systems can run together, simulating how automated network monitoring would work in an actual environment.

I created a config.json file to store important settings such as file paths, alert thresholds, and ports. JSON made it easy to maintain readability while keeping the system flexible for future changes. I ran twelve test simulations to confirm that the IDS correctly triggered alerts based on honeypot activity.

There were no port conflicts or permission issues, but one major realization came when I could not activate my virtual environment. The issue turned out to be a simple one: I was running PowerShell as a non-administrator. Once I corrected that, the environment activated correctly, and everything else started falling into place.

Throughout the process, I followed a simplified agile-style workflow: write, test, document, and commit. Every improvement or bug fix was pushed to GitHub for version tracking. Each commit represented a complete milestone, which kept my progress organized and recoverable.

**3. Use of AI Tools**

AI tools were part of my workflow, but they were used for refinement and problem-solving rather than creation. ChatGPT helped me troubleshoot when I could not get the test triggers to behave as expected. After switching to CMD.exe and rerunning the simulated “attacks,” both systems worked exactly as intended. AI also helped clean up my documentation, especially my README file, which needed to be structured in Markdown format and easy to follow.

GitHub Copilot helped save time by filling in repetitive syntax patterns such as handling exceptions and print statements, but most of my code logic was built from scratch based on prior assignments. Grammarly helped me polish grammar and readability, ensuring the technical writing remained professional.

There were times when AI tools tried to change my writing style or suggest code rewrites, but I made sure that every piece of logic stayed mine. I understood and could explain each function line by line before finalizing it. AI acted purely as a reviewing tool, not a replacement for original thinking. When I encountered technical issues that I could not immediately solve, AI helped me pinpoint the exact cause of the issues faster.

I ensured all the logic and structure remained my own by independently testing and verifying each component. AI provided efficiency, but the analysis, problem-solving, and integration came from me.

**4. Project Challenges**

This project came with several technical challenges that improved both my patience and my ability to troubleshoot under pressure.

The first issue was PowerShell’s execution policy, which blocked me from activating my virtual environment. To fix the issue, I had to temporarily change the policy with the command Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass, which allowed scripts to run safely for that session.

I also experienced file path errors because the main project folder I created under “Desktop” did not populate correctly at first. Once I moved the project and re-synced the paths inside VS Code, everything compiled smoothly.

Another significant challenge was testing my alerts. I had to make sure the honeypot and IDS worked together without overwriting logs or missing connection data. Through trial and error, I fine-tuned the scripts until the IDS properly detected rapid connection attempts and logged alerts accurately.

Finally, I ran into issues with documentation files. My original README.md, .gitignore, and config.json had to be scrapped and rebuilt from scratch. This experience forced me to take organization seriously and taught me the importance of clean structure and consistent formatting.

During all this, I also learned how to connect VS Code to GitHub more efficiently. By the end, I could initialize, stage, commit, and push code in seconds rather than minutes. This saved a lot of time and frustration.

**5. Error Handling and Edge-Case Testing**

During the testing phase, I intentionally introduced several edge cases to ensure that both the Honeypot and IDS systems responded gracefully to unexpected input and runtime interruptions. For example, I simulated invalid port values and repeated connection resets to verify socket stability. The Honeypot was designed to handle these safely by catching exceptions and logging the connection attempts without crashing.

Additionally, I verified that file writing operations handled missing or locked log files properly by introducing controlled delays and reattempt logic. The IDS script was tested to confirm that thresholds triggered correctly even during bursts of malformed requests. This validation confirmed that both systems remained operational under error conditions and safely closed connections when errors were encountered.

**Code Block:**

[2025-10-12 14:33:21] Connection from 127.0.0.1:54012

[2025-10-12 14:33:22] Connection from 127.0.0.1:54013

[2025-10-12 14:33:23] Alert Triggered: 6 attempts from same IP within 10 seconds

[2025-10-12 14:33:23] IDS Notification Logged to data/alerts.log

**6. Reflection on Learning**

This project helped me strengthen several skills that directly apply to real-world cybersecurity work. I built on prior Python knowledge and learned how to automate log generation, real-time monitoring, and alerting. These are the same principles that drive SIEM systems and security automation frameworks in the field.

I also gained a deeper understanding of how automation can enhance visibility and efficiency within a security operations center. The project showed how a system can be built to detect abnormal activity, record it, and alert the operator, all without manual input. This kind of proactive defense approach is critical for modern cybersecurity.

I also learned how essential documentation is. It is not only for organization but also for ethical and safety purposes. Proper documentation ensures you can trace errors, track changes, and explain your system clearly to others. These behaviors also reflect discipline and accountability.

If I had to summarize this project to someone new in cybersecurity, I would tell them that automation is not about writing complicated code. It is about building systems that can observe, react, and make decisions without direct human input. Automation does not replace analysts; it empowers them. It handles repetitive, time-sensitive work so the human element can focus on analysis and decision-making. This project gave me firsthand experience of what that balance looks like.

**7. Scalability and Real-World Application**

From a scalability perspective, this hybrid Honeypot and IDS system could be expanded for enterprise or cloud-based use. By deploying the Honeypot across multiple subnets or virtual machines, it could collect intelligence from different network segments, while the IDS could aggregate and correlate data in a centralized logging server or SIEM environment such as Splunk or Elastic Stack. This would allow for automated detection of coordinated attacks, greater insight into adversary tactics, and integration with larger security automation workflows. The modular design of the scripts makes this system adaptable for real-world network defense scenarios with minimal modification.

**8. Time Management**

The total project time was around twenty-two hours spread across several days. I spent about six hours writing and debugging code, eight hours troubleshooting and testing, four hours on documentation, and four hours preparing the GitHub repository and README file.

Before development began, I spent three days researching and planning how I wanted the project to work. Once I started, I focused on building all three components, the honeypot, IDS, and config, before testing everything as a single integrated system.

I documented at the end, but I took notes, screenshots, and observations as I went, which made the final write-up much easier. This process kept me organized and allowed me to reflect on what worked and what did not.

My time management strategy was simple but effective. I worked in focused sessions and made a commit after every major success or fix. This method gave me a record of progress and made troubleshooting easier since I could always roll back to a working state.

**9. Conclusion**

The Hybrid IDS + Honeypot project achieved its goal of demonstrating how automation can detect, log, and analyze suspicious network activity. It effectively combined the passive monitoring of a honeypot with the analytical power of an IDS.

If given more time, I would expand the project with Geo-IP lookup to trace the location of connection attempts. This feature would allow the system to map potential attackers geographically, adding valuable intelligence to the alert process.

What I am most proud of is the knowledge gained from this project. Every part, from coding to debugging to documentation, played a role in my understanding of cybersecurity automation. The satisfaction came not from completing a class assignment, but from actually learning and applying concepts that will carry over into real-world scenarios.

This project gave me more confidence in my technical abilities and my approach to problem-solving. It also prepared me for future projects, including certification work such as the CompTIA Security+, where automation, detection, and analysis are essential topics.

Overall, this experience showed me that cybersecurity is not just about defense but about building intelligent systems that anticipate, detect, and respond to threats. I feel more capable now of contributing to that mission.

**10. Screenshots**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 1: Screenshot showing the project folder structure with organized directories (data, screenshots, and code files).

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 2: Screenshot showing the honeypot.py script successfully running and listening on multiple ports.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 3: Screenshot showing the test simulation with multiple rapid connection attempts sent through CMD.exe to the honeypot.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4: Screenshot showing the populated honeypot.log file recording connection attempts in real time.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 5: Screenshot showing the IDS (sids.py) actively running and monitoring the honeypot logs.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 6: Screenshot showing an IDS alert triggered by repeated connection attempts from the same IP address.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: Screenshot showing the alerts.log file successfully recording the generated alerts from the IDS.

A screen shot of a computer program

AI-generated content may be incorrect.

Figure 8: Screenshot showing the config.json file with paths, thresholds, and ports correctly configured.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 9: Screenshot showing the finalized README.md file properly formatted in Markdown and updated with full documentation.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: Screenshot showing the GitHub repository fully connected, organized, and displaying the final commits.

**11. Repository Link**

GitHub Repository:

<https://github.com/Horse700/Hybrid-IDS-Honeypot-Rkralle>

**10. Tools and Technologies Used**

* Python 3.12
* Visual Studio Code
* GitHub and Git Bash
* Windows PowerShell and CMD.exe
* ChatGPT (for troubleshooting and documentation support)
* GitHub Copilot (for syntax assistance)