Dustin Allen

CYB333

Professor Tukuafu

May 3, 2025

**Python Honeypot for Security Automation: Final Project Report**

This capstone project involves creating a Python honeypot that simulates an SSH service to detect and log unauthorized access attempts. The honeypot changed from a simple socket server to complex software, featuring real-time alerts, IP geolocation, and mail notification. The project investigates the potential and utility of security automation by designing the game to be modular, debugging it iteratively, and leveraging AI tools tactically. The process, difficulties, use of AI tools, and personal ideas obtained in this work are described.  
Automation in security is fundamental in the modern cybersecurity paradigm to have scalable, proactive threat detection. So, to make a more responsive, dynamic honeypot, I wanted to build upon the groundwork I laid with the midterm assignment. The project sets up an SSH trap on a fake port, records, and retaliates against hack attempts. Even though it may appear simple, this project represents classic automation concepts like real-time response, modularity, and evolvable design. The other and last tool logs the connections, does GeoIP lookups, plays a sound on trigger, and sends you an email. As the PSF highlights, “Python libraries provide a good infrastructure for writing automation scripts with (relatively) low level of overhead” (PSF, 2023).  
The project started life as a very small functional Python socket listener. This listener emulated an SSH banner to attract attackers,  and was on a non-standard port to keep things apart. I took a modular approach to development: test the base connection first, add logging on top of that, add alerts, and then bring in external stuff like GeoIP and SMTP for the emails. Tests involved Telnet-based intrusion attacks from the desktop and mobile.  
I hit the first roadblock immediately (it was already taken, so I used port 2323 instead). Once that was resolved, I looked at user experience and effectiveness– every new feature had to give a security analyst or administrator more actionable insight. For example, the alert played an acoustic sound when access attempts had been made passively,  while the GeoIP lookup enriched raw IP addresses with geographical information. According to GeoIP.io, “additional context is given through geolocation of the IP, an enrichment to the IP data itself which is sometimes critical to network activity” (GeoIP.io, 2024). Soft out-of-bound alerts were provided by email notifications, which mimic actual automated SOC (Security Operations Center) procedures. All this was done with the help of free, readily available Python libraries - a reminder of how ludicrously powerful open-source tooling can be, within the context of security.  
I leaned on AI tools a lot during development, most notably ChatGPT. I wrote code, troubleshooted errors, and worked on integration using triggers such as “write Python sockets honeypot” and “add GeoIP + email alert in Python code.” For instance, once, when I had an issue about port binding, which threw OSError : [WinError 10048], I wrote AI to solve the error and factor the code. It also forced me to document the code more completely with inline comments and keep things clean with how the methods are organized.

The main upside of AI was speed; something that might have taken hours of research, testing, and debugging could be condensed to a few queries and iterations. OpenAI says,  “AI Tools such as ChatGPT decrease development friction; they enable faster prototyping and increased creativity” (OpenAI, 2025). However, AI did not take the place of comprehension. I still had to validate outputs,  especially for outdated modules or security-relevant actions. AI was beneficial for breaking through blocks and faster prototyping, but having the final implementation always be done by humans was necessary.  
I faced various technical and practical challenges while working on the honeypot. The first was an issue of trying to bind to a port already in use. I ended up with another issue; the script was not running because of it . py. txt ─ there was a little mishap in naming the file. The generated request in this article is based on previous failing tests (GeoIP was not working by that time; this could likely be due to a lack of network permission or an API request misdesign), accomplishments, and mistakes.  
The highest priority among them was the email alert system. Dealing with SMTP and Gmail involved using security hoops such as app-specific passwords and SSL/TLS encryption. As Google Developers (2024) stated, “applications that access Gmail must use OAuth 2.0 or app-specific credentials for connecting to the service using basic authentication”. Debugging SMTP errors showed me how they deal with authentication and how to dispatch alert data securely. These obstacles served as avenues to develop technical skills further and think on the spot.  
What began as a quick thought experiment to simulate an SSH port became a helpful exercise in network monitoring, Python scripting, and real-time response planning. I now get what goes into building an intrusion detection tool from the ground up—not the coding,  the thinking. I have improved at exception handling, sockets, and using third-party libraries. More importantly, I have developed an attitude about modular design and automation workflows, which is critical to anyone working in cybersecurity today.  
This has taught me the value of gluing unassuming tools together to create something meaningful. The honeypot is not there to block attackers, but serves as a tripwire - a silent warning system that notifies, educates, and enables faster response. It made me realize how much even basic tools matter when designed intentionally. It also demonstrated that scripting and security are not mutually exclusive; they can be paired to produce innovative systems that do the legwork for analysts and defenders.  
Over 3 weeks, I worked on this project for about 28-32 hours. Here is the breakdown: PROTOTYPING, EXPERIMENTS, AND TESTING. I would focus on 2 hours of work in the evening, using Pomodoro to remain productive. I kept a journal to record what I had accomplished every time a bug was finally squished, and wrote commit messages for changes and features. This steady schedule prevented burnout and made me feel like I had moved forward, even when stuck on a particular feature. Looking back on it, I think the work rhythm and routine were probably the main factor in getting me to the end of the book without losing drive or desired direction.  
I have taken screenshots and recorded mock terminal sessions of the honeypot catching connection attempts to demonstrate its effectiveness. Console output shows GeoIP information, and multi-channel responses are done using sound notifications and confirmation via email. These resources can be found on GitHub and assist in the robustness and usability of the tool. This sort of evidence both demonstrates it works in practice and lays the groundwork for showing the tool to others, such as instructors or, eventually, employers.  
The modern honeypot is a lightweight IDS that illustrates practical uses of security automation. It takes fundamental ideas — like listening to traffic, analyzing IPs, and creating alerts — and makes them into something usable, scalable, and you learn from. Future ideas are kept in that file, such as longer-term plans for result storage in a database, support for other protocols, and dashboards one could create with Flask. The project allowed me to practice key cybersecurity capabilities and work with some automation while getting more comfortable leveraging Python for security operations. It has been an enriching yet exhausting step in my learning journey.

**References**  
GeoIP.io. (2024). IP geolocation API docs. <https://ipinfo.io/developers>  
Google Developers. (2024). Using the Gmail SMTP server. <https://developers.google.com/gmail/imap/imap-smtp>  
OpenAI. (2025). Prompting and coding ChatGPT. <https://chat.openai.com>  
Python Software Foundation (2023). Python documentation. <https://docs.python.org/3/>