Final Report

Ariel Mitchell

Ife Akindipe

12/4/25

Team Rocket Network Automation Project Report

Introduction & Purpose ("The What")

Our project focuses on automating network reconnaissance and vulnerability detection. It uses the outputs from the commands nmap, netcat and masscan to gather detailed information about hosts, open ports, and services running on a network. Instead of manually reviewing long and complex scan outputs, our software automatically parses and organises the data into a clear, readable output. It also includes an alerting feature that notifies users when a suspicious or unexpected open port is detected. The ultimate goal of this project is to reduce manual work, highlight key security findings, and allow users to focus on identifying and understanding potential vulnerabilities instead of searching for them.

After parsing, storing, and checking for policy violations, the system produces its main output in the form of a human readable Markdown report. This report organizes findings by host, presenting each IP address followed by the open ports and detected services associated with that machine. Product names and version details are included whenever available. Markdown was chosen specifically because it is simple to read in plain text and can also be rendered cleanly on platforms such as GitHub or converted to PDF if needed. The system also creates a separate Markdown document that summarizes any unexpected ports it detected during the scan. This structure ensures that both normal reconnaissance results and potential security concerns are clearly laid out for the user.

Lastly we also added a vulnerability correlation engine. This part of the system behaves like a simplified version of a commercial vulnerability scanner. It loads a set of vulnerability signatures

from a YAML file. Each signature describes a known issue along with the specific ports, services, products, and version ranges affected. Once these signatures are loaded, the system compares them to the stored scan results. Whenever it finds a match, meaning the version of a service on a host appears to fall within the range of a known vulnerability, the tool generates a correlation entry. These entries are later formatted into a vulnerability risk report presented in Markdown. Each entry explains the vulnerability, its severity, the affected host, and a clear explanation of why the tool flagged it.

Technical Implementation ("The How")

The technical implementation of our Automated Recon Output Parsing and Alerting System is built around a modular Python framework that behaves much like a simplified version of a professional reconnaissance and vulnerability analysis pipeline. At a high level, the system takes raw scan data, primarily from Nmap, and turns it into something structured, searchable, and ultimately useful for identifying potential risks on a network. Everything begins with the ingestion layer. Because Nmap can export its results in XML format, and XML is far easier to parse reliably than raw text, the system uses Python's built in XML parsing library to walk through the host and port elements in the scan file. For each host it encounters, the script extracts details such as the IP address, any hostname associated with the machine, and each of the open ports discovered during the scan. For every open port, the script saves information such as the protocol, service name, detected product, and version number. All of this is immediately normalized into a consistent internal format. This step is extremely important because tools like Nmap and Masscan do not always present information in the same way, so having a predictable structure ensures the rest of the system does not have to deal with any formatting quirks.

After normalization, the next step is storing the information in a SQLite database. SQLite is ideal for a project like this because it is reliable, lightweight, and requires no server installation or configuration. Each scan result is saved in a dedicated findings table that holds fields for the host, port, service, and any product or version data discovered. This gives the system a stable and persistent record of what was found during a scan. It also provides a central source of truth for all later stages of the pipeline. Reports, alerts, and vulnerability correlation features all read directly from this database instead of trying to reparse the XML file. This separation between parsing and analysis ensures that improvements to one part of the project do not break the others and makes it easy to extend the system later on.

After normalization, the next step is storing the information in a SQLite database. SQLite is ideal for a project like this because it is reliable, lightweight, and requires no server installation or configuration. Each scan result is saved in a dedicated findings table that holds fields for the host, port, service, and any product or version data discovered. This gives the system a stable and persistent record of what was found during a scan. It also provides a central source of truth for all later stages of the pipeline. Reports, alerts, and vulnerability correlation features all read directly from this database instead of trying to reparse the XML file. This separation between parsing and analysis ensures that improvements to one part of the project do not break the others and makes it easy to extend the system later on.All of these components operate together under the control of the orchestrator script. The orchestrator is essentially the project's command center. It takes command line input from the user, such as the location of the scan file or the output directory, and then runs the entire pipeline in the correct sequence. First it parses the scan file, then writes the results to the database, then checks for unexpected ports, then generates reports, and finally performs vulnerability correlation if the user has provided the necessary signature file. The orchestrator ensures consistency between runs and also makes the system easy to expand. New features can be added without rewriting the entire workflow.

Justification & Analysis ("The Why")

Because this project was created as students learning how real security tools operate, we focused on building something that teaches and demonstrates core concepts.Our goal was not to create the most advanced scanning tool, but to understand how raw technical data becomes meaningful, organized, and actionable. The structure of the system therefore grew from decisions based on clarity, learning value, and realistic security practices.

One of the earliest decisions we made was to use structured XML input from Nmap instead of relying on simple text parsing. XML was chosen not because it is the easiest format, but because it offered consistency. As students trying to build a tool that could be expanded later, we needed predictability in the data. It prevents us from having to write fragile text-parsing code and allows us to focus on understanding the meaning of the data rather than wrestling with formatting problems.

Similarly, we chose to normalize all scan results into our own consistent internal format. The reason behind this was the desire to decouple our analysis logic from Nmap itself. Even though we only used Nmap in this version of the project, designing the system around normalized data gives the project room to grow.

We decided to also store results in SQLite because we wanted something persistent because a scan is not very useful if the data disappears the moment the program ends. SQLite allowed us to keep structured history, and supported future ideas like comparing multiple scans. Yet it remained simple enough to set up without the overhead of a full database server.

Finally, the decision to build an orchestrator instead of a series of disconnected scripts came from the need to design a complete workflow. A real security tool is not a loose collection of functions. We wanted the project to show that we understood how to automate a multi-stage process from start to finish: parsing, storing, checking, reporting, and correlating. The

orchestrator exists because we wanted to demonstrate control, structure, and reliability. It allowed us to treat the project as a full system rather than a set of isolated components.

In the end, we chose approaches that reinforced good engineering habits, supported clear communication, and mirrored real cybersecurity practices. Those really helped shape the tool alongside the technical requirements themselves.

Conclusion

Our project aimed to reduce manual work, highlight key findings, and allow users to focus on identifying potential vulnerabilities rather than searching for them. We did this by using the outputs from commands such as nmap, netcat and masscan to gather important information about the system. Our project organises all key data into a clear, readable report. Overall, our project goal was to save the user time and energy by giving them a tool that they can input large scan files into, and get back the key details from those scanning tools.