Bryan Leigh, Sam Lezon

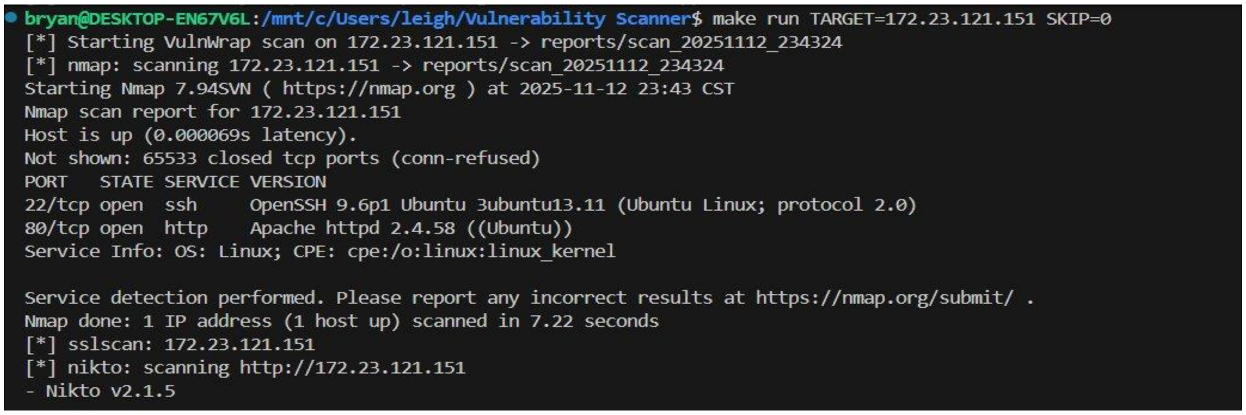
12/4/25

IT 359

Vulnerability Scanner (Bash)

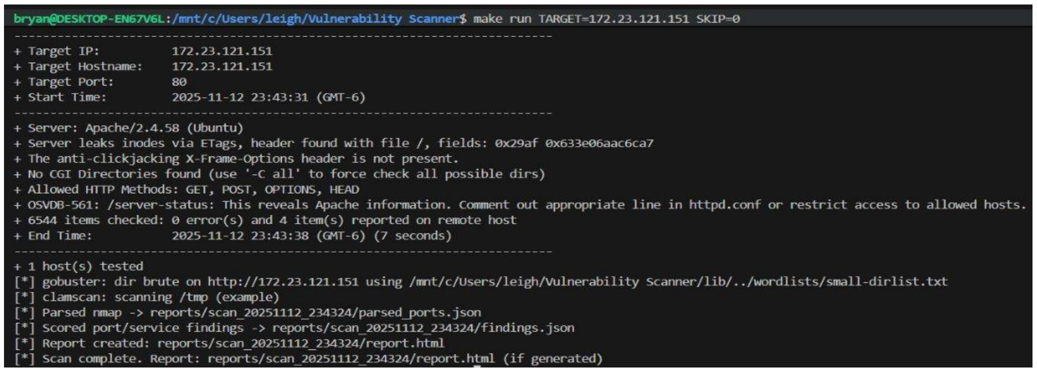
Introduction & Purpose

Our project is a vulnerability scanner that is made using a Bash-based script. It is designed to automate and simplify the process of running multiple security tools. Instead of typing each scanner, saving outputs, this scanner puts it all together into a single workflow. With one command in the command line, it can perform discovery, scanning, analysis, and give you an HTML report. Below are a couple of pictures of what it looks like when it is running.



```
bryan@DESKTOP-EN67V6L:/mnt/c/Users/leigh/Vulnerability Scanner$ make run TARGET=172.23.121.151 SKIP=0
 [*] Starting VulnWrap scan on 172.23.121.151 -> reports/scan_20251112_234324
 [*] nmap: scanning 172.23.121.151 -> reports/scan_20251112_234324
 Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-11-12 23:43 CST
 Nmap scan report for 172.23.121.151
 Host is up (0.000069s latency).
 Not shown: 65533 closed tcp ports (conn-refused)
 PORT   STATE SERVICE VERSION
 22/tcp open  ssh     OpenSSH 9.6p1 Ubuntu 3ubuntu13.11 (Ubuntu Linux; protocol 2.0)
 80/tcp open  http    Apache httpd 2.4.58 ((Ubuntu))
 Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

 Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
 Nmap done: 1 IP address (1 host up) scanned in 7.22 seconds
 [*] sslscan: 172.23.121.151
 [*] nikto: scanning http://172.23.121.151
 - Nikto v2.1.5
```

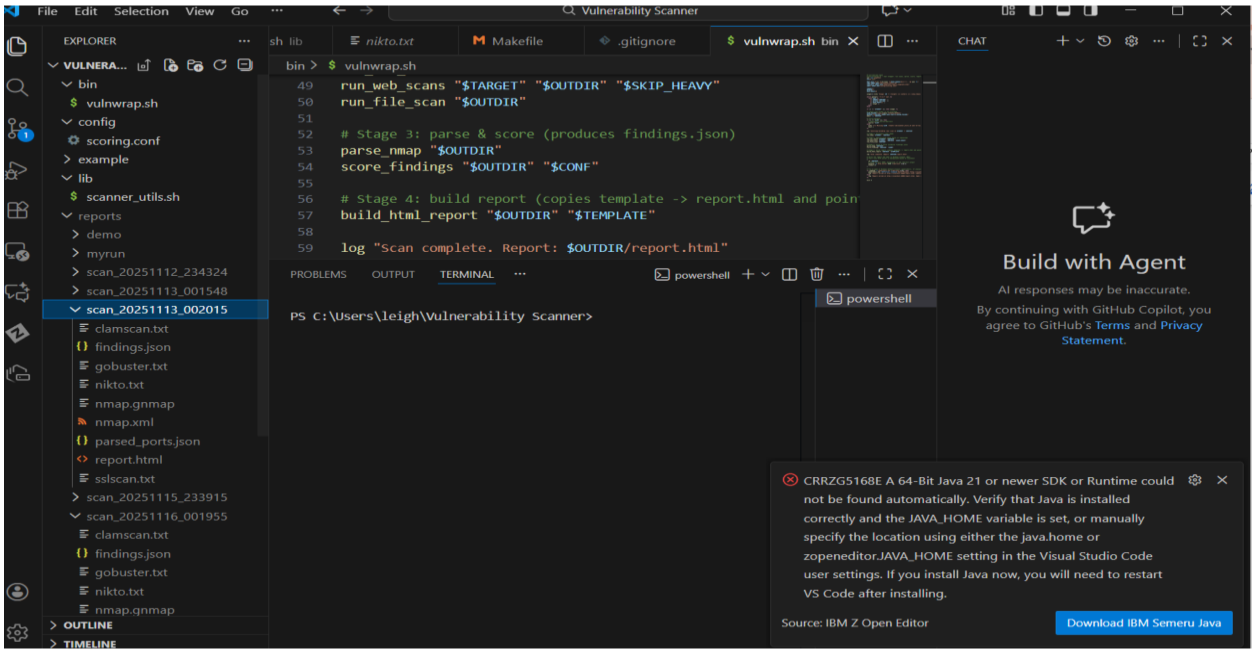


```
bryan@DESKTOP-EN67V6L:/mnt/c/Users/leigh/Vulnerability Scanner$ make run TARGET=172.23.121.151 SKIP=0
---------------------------------------------------------------------------
+ Target IP:          172.23.121.151
+ Target Hostname:    172.23.121.151
+ Target Port:        80
+ Start Time:         2025-11-12 23:43:31 (GMT-6)
---------------------------------------------------------------------------
+ Server: Apache/2.4.58 (Ubuntu)
+ Server leaks inodes via ETags, header found with file /, fields: 0x29af 0x633e06aac6ca7
+ The anti-clickjacking X-Frame-Options header is not present.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Allowed HTTP Methods: GET, POST, OPTIONS, HEAD
+ OSVDB-561: /server-status: This reveals Apache information. Comment out appropriate line in httpd.conf or restrict access to allowed hosts.
+ 6544 items checked: 0 error(s) and 4 item(s) reported on remote host
+ End Time:           2025-11-12 23:43:38 (GMT-6) (7 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
[*] gobuster: dir brute on http://172.23.121.151 using /mnt/c/Users/leigh/Vulnerability Scanner/lib/../wordlists/small-dirlist.txt
[*] clamscan: scanning /tmp (example)
[*] Parsed nmap -> reports/scan_20251112_234324/parsed_ports.json
[*] Scored port/service findings -> reports/scan_20251112_234324/findings.json
[*] Report created: reports/scan_20251112_234324/report.html
[*] Scan complete. Report: reports/scan_20251112_234324/report.html (if generated)
```

The primary purpose of the tool is to streamline vulnerability assessment for small environments or lab networks. It runs a set of commonly used tools such as Nmap for port and service discovery, optional web scanners like Nikto and Gobuster, and file or malware checks when available, and then normalizes their outputs into a simple findings file with risk scores. These scores are visualized in a report so that a user can quickly see which services or areas might be at higher risk. The main goals of the tool is to reduce the time and effort of running multiple tools, give consistent results, and make it easier to perform a structural scan with basic knowledge. This makes the tool useful for students, entry-level analysts, or anyone who wants a simple way to run repeatable scans against their own systems or lab machines.

Technical Implementation

This project is implemented as a Bash-based wrapper that ties together several existing security tools into a single, consistent workflow. The code is split into multiple Bash scripts and supporting files, so it feels more like a small real tool rather than just one long script as shown in the picture below. The main pieces include a primary wrapper script, a utilities script with the scanning functions, a configuration file for scoring, a simple wordlist for directory brute forcing, an HTML template for reporting, and a reports folder where each scan's results are saved.



```
run_web_scans "$TARGET" "$OUTDIR" "$SKIP_HEAVY"
run_file_scan "$OUTDIR"

# Stage 3: parse & score (produces findings.json)
parse_nmap "$OUTDIR"
score_findings "$OUTDIR" "$CONF"

# Stage 4: build report (copies template -> report.html and poin
build_html_report "$OUTDIR" "$TEMPLATE"

log "Scan complete. Report: $OUTDIR/report.html"
```

```
PS C:\Users\leigh\Vulnerability Scanner>
```

Build with Agent

AI responses may be inaccurate.

By continuing with GitHub Copilot, you agree to GitHub's Terms and Privacy Statement.

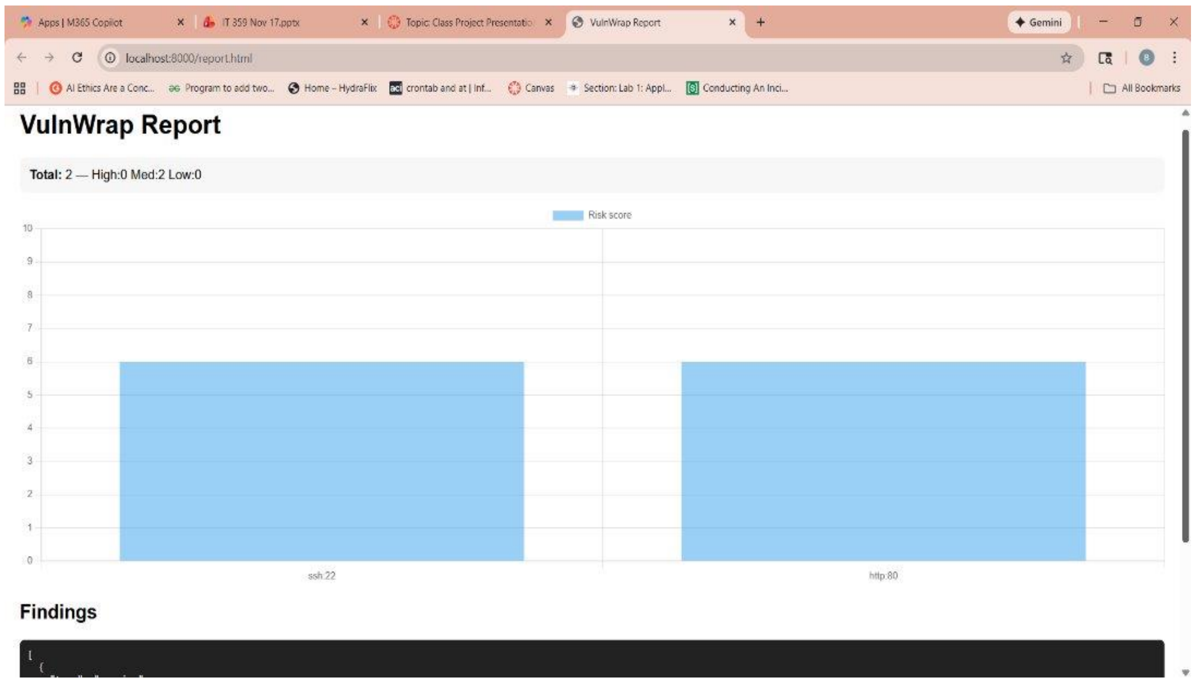
CRRZG5168E A 64-Bit Java 21 or newer SDK or Runtime could not be found automatically. Verify that Java is installed correctly and the JAVA_HOME variable is set, or manually specify the location using either the java.home or zopeneditor.JAVA_HOME setting in the Visual Studio Code user settings. If you install Java now, you will need to restart VS Code after installing.

Source: IBM Z Open Editor

The main wrapper script is responsible for running the entire scan. It reads the target IP address, sets up a timestamped output directory inside the reports folder, and then calls a series of helper functions in order. These functions perform tasks such as running Nmap for port and service discovery, checking SSL/TLS settings when possible, optionally running web scans like Nikto and Gobuster, and running an example file. The wrapper also calls functions that parse the output and build the final report. It uses standard Bash safety practices like set -euo pipefail and checks whether tools are installed before trying to run them, so the script can still complete even if not every tool is available on the system.

Most of the actual scanning logic lives in a separate utilities script. In that file, functions are defined to run each tool and handle its output. Nmap is used to scan the target for open ports and services, and saves its results in formats that are easy to parse. Another function reads Nmap’s greppable output and, with the help of the jq tool, converts the port and service information into a JSON file. A scoring function then takes that JSON and assigns simple numeric risk scores to each service, based on a small configuration file that maps service names (like SSH or HTTP) to scores. The scored results are written into a findings.json file. If tools like Nikto or Gobuster produce results, those can also be reflected by adding additional findings with their own scores.

The final step is report generation. The project uses an HTML template file that contains a placeholder for the findings path. The wrapper copies this template into the report folder, replaces the placeholder with findings.json, and produces a report.html file for that scan. When opened in a browser as shown in the picture below, this report can load the JSON data and show the findings in a simple, readable format, including a basic visualization of the scores. Together, this technical setup demonstrates how Bash, existing security tools (Nmap, Nikto, Gobuster, etc.), and a bit of JSON and HTML can be combined to create a lightweight vulnerability scanning workflow that is easier to run and interpret than using each tool by itself.



VulnWrap Report

Total: 2 — High:0 Med:2 Low:0

Findings

```
[
  {
```

Justification & Analysis

Running multiple vulnerability tools by hand is slow, repetitive, and easy to mess up. Each scanner produces its own output format, so beginners and small teams can struggle to understand what matters most. VulnWrap is justified to simplify that process: it automates common tools into a single workflow and combines their results into one report.

Using Bash makes sense because it's already available in most security labs and is well-suited for chaining command-line tools like Nmap, Nikto, and Gobuster. Instead of reinventing scanning logic, the script focuses on orchestration, parsing, and basic risk scoring. Converting raw scan data into a simple score per service helps highlight which findings are potentially more important, rather than dumping pages of raw output.

Overall, VulnWrap is valuable as a lightweight, realistic learning tool. It demonstrates how different types of checks network, web, and file scanning can be tied together in a repeatable way, and it provides a foundation that could be extended with more tools or a more advanced scoring model in the future.

Conclusion

This project shows how a simple Bash wrapper can turn several separate security tools into one coordinated vulnerability scanning workflow. Instead of manually running Nmap, web scanners, and other utilities one by one, VulnWrap automates the process, organizes the outputs, and generates a single HTML report with basic risk scoring. This makes it easier to quickly see which services or findings might deserve more attention.

While VulnWrap is not meant to replace full enterprise vulnerability management platforms, it demonstrates important ideas on a smaller scale: automation, repeatability, and prioritization. It also highlights how common open-source tools can be combined effectively using scripting rather than building everything from scratch. In the future, this project could be extended with more scanners, richer analysis, or additional visualizations, but even in its current form it provides a practical, educational example of how to streamline vulnerability assessment in a lab environment.