Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies

Information Technologies Security

Semestral Project

Looking for vulnerabilities in large networks

Sližik Ján

Prof. Ing. Volodymyr Khylenko, Phd.

Date: 3.12.2024

Contents

Introduction

Network scanning is an important part of assessing information technology systems, as it helps identify security weaknesses, misconfigurations, and unpatched systems. Scanning large-scale networks presents significant challenges due to the increased number of devices and open ports. To effectively scan large networks, a solid understanding of network protocols and scanning tools is required. This project aims to address these challenges by first reviewing various scanning techniques that can be utilized. Based on this review, a custom Go network scanning tool is developed, showcasing some of these techniques in practice. Additionally, a simulated large network environment, consisting of multiple Linux servers, is created to allow for the comparison of different tools such as Nmap, Zmap, Masscan, and the custom script. The primary goal of this project is to compare these tools in terms of host discovery, service discovery, and port scanning, along with vulnerability detection within large network environments. This comparison will help identify the strengths and weaknesses of each tool and provide recommendations for improving scanning strategies. Other robust platforms, including Shodan, Axiom, and OpenVAS, are also covered.

Scanning Techniques

ICMP Probe

ICMP operates at the network level, making it fast and useful for determining if a host is reachable on a network, though it cannot identify open ports. ICMP Echo Request is sent to the target and Echo Reply is awaited. If a reply is received, the host is considered live. ICMP probes are efficient for quickly discovering live hosts across large IP ranges. However, they can easily be blocked by firewalls, leading to false negatives where live hosts are not detected.

TCP Scans

TCP operates at the transport level and is a connection-oriented protocol that uses a three-way handshake to establish a connection. This makes TCP scans slower for host detection compared to ICMP probes but much more reliable. Unlike ICMP, TCP scans can be used to discover open TCP ports. Common types of TCP scans include:

SYN Scan: This is the most efficient scanning method, as it doesn't fully establish a connection, which minimizes resource usage. A SYN packet is sent to the target port, and if the port is open, the target responds with a SYN-ACK, indicating the service is available.

Connect Scan: This scan establishes a full TCP connection by completing the three-way handshake. It is slower than SYN scanning but is useful when raw packet sending is restricted.

ACK Scan: The scanner sends an ACK packet to the target. If no reply is received, it indicates the port is likely filtered by a firewall. If the port is unfiltered, the target responds with an RST packet, showing that the ACK was received but the connection is reset. This scan doesn't reveal whether the port is open, only whether it is filtered or not.

Maimon Scan: Another type of scan that can help to determine if a port is filtered. A packet with the FIN and ACK flags is sent to the target. If the port is open or filtered, there will be no response. If the port is closed, the target sends an RST packet. This method can sometimes bypass firewalls that block SYN scans.

Window Scan: This scan checks the size of the TCP window in response packets to determine if a port is open, closed, or filtered. Different window sizes indicate the port's state, allowing Nmap to make an educated guess. It's useful when other scan types return unclear results.

UDP Scan

UDP operates at the transport level, but is a connection-less protocol, which makes scanning with it a lot more challenging than TCP, there is no handshake to indicate whether a port is open. A typical UDP scan involves sending a UDP packet to the target and waiting for a response. If no response is received, the port is either open or filtered. Due to the lack of response in many cases, UDP scanning tends to be slower and more resource intensive.

ARP Discovery

ARP operates at the link layer, responsible for physical addressing. This means ARP scans can only discover devices within the same subnet, even if they do not respond to ICMP pings or other network-level scans. The process involves sending an ARP request asking, "Who has this IP?" and devices with matching IP addresses respond with their MAC addresses. To simplify the process *arp -a* or *ip neigh* commands can be used to display the current ARP table, showing the IP addresses and corresponding MAC addresses of devices the local machine has communicated with. However, ARP scans are limited to the same broadcast domain and cannot scan devices on remote networks or across VLANs. VLANs create separate broadcast domains, so if you're on VLAN 10, you won't see devices on VLAN 20 unless routing is configured between them.

Banner Grabbing

This technique is used to gather information about network services running on a device by connecting to open ports and retrieving the initial message the service sends. It can expose critical information if not managed properly such as the software version, server type, and configuration details.

Evasion

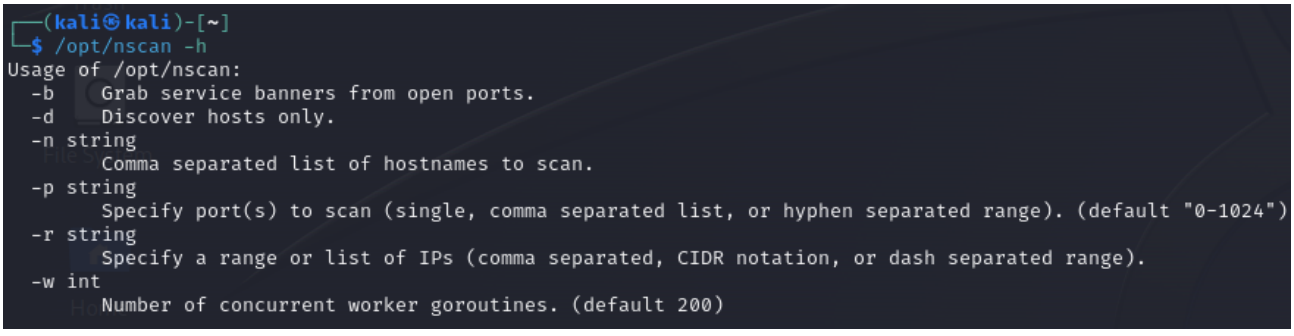
Although evading firewalls and network monitoring is not the primary focus of this semestral project, it's important to acknowledge that fast network scans can generate significant noise and potentially trigger denial-of-service conditions on weaker devices. Evasion techniques such as implementing timing and rate limiting, as well as scanning from multiple IP addresses, VPNs, or proxies, can help minimize detection and enhance reconnaissance efforts resulting in better results.

Custom Network Scanner

Nscan, was developed using Go for its speed and support for cross-platform compilation. My goal was to better my understanding of Go's concurrency model, specifically goroutines and channels, which allow the scanning of multiple IP addresses and ports in parallel without overwhelming system resources. Each scan is managed by a separate goroutine, and channels are utilized to collect the results.

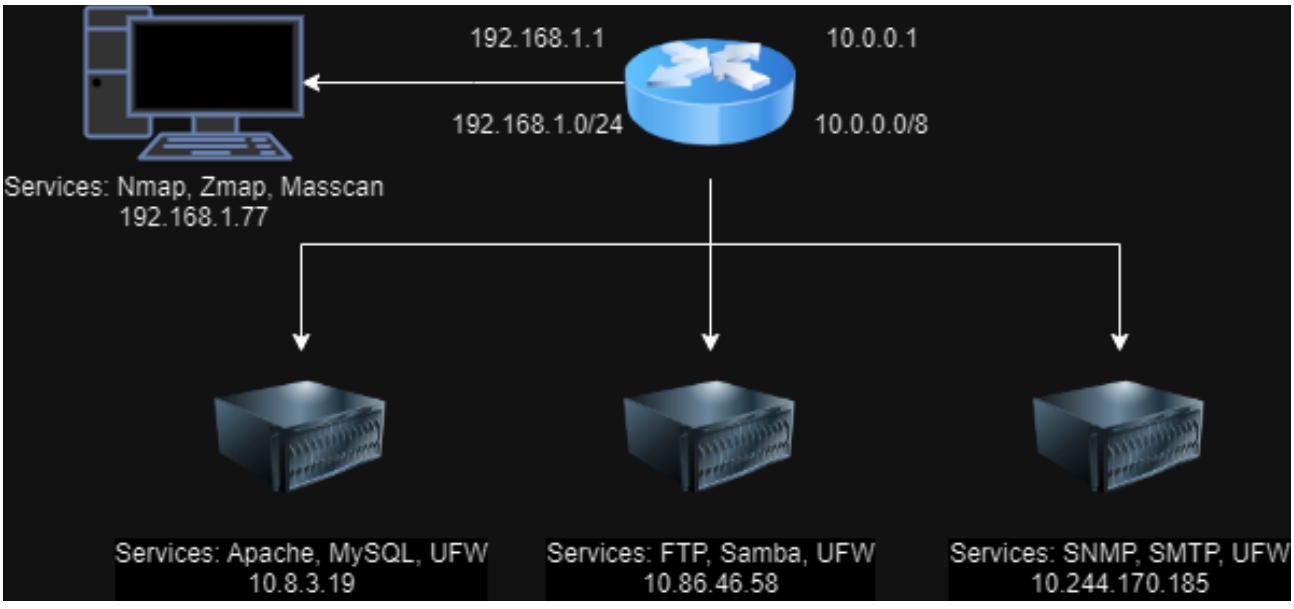
I started the project by developing an ICMP probe, which proved challenging initially due to the absence of native ping functionality in the Go main package. Luckily, I found the *prometheus-community/pro-bing* library, which adds this capability. Next, I implemented a DNS lookup option that enables users to input an address, resolving it to IPv4 addresses for port scanning. Once the available hosts are identified, a TCP SYN scan is performed. I also attempted to implement a UDP scan; however, due to the stateless nature of the protocol, the results included many false positives.

To improve the clarity of the output, I created a dictionary of potential services associated with specific ports. This allows the script to indicate that, for example, port 22 is open and likely running SSH. The script also includes an option to disable host discovery entirely and rely solely on provided IP addresses, CIDR notation, and hostnames provided. Additionally, users can specify port ranges or individual ports. Finally, I added a banner grabbing feature that connects to a specified port and reads data from the connection. This function captures data into a buffer and appends it to the banner slice, leading to interesting findings. By connecting to the service and reading its initial response, Nscan extracts valuable details, such as the service name and version.



Test Environment Setup

Modular virtual network topology was created using Vagrant and Ansible, allowing for deterministic deployment of virtual machines and services through a role-based approach. This method leverages roles that you can create yourself or you can import Ansible Galaxy roles, which simplify service deployment by providing reusable templates that can be easily configured with specific variable values. In this project, I learned how to use Ansible Galaxy roles to deploy services such as Apache and MySQL efficiently, while also configuring additional services through creating my own roles and intentionally misconfiguring some to test the scanning tools' effectiveness in identifying vulnerabilities. Although most businesses typically segment their networks into smaller CIDR ranges, I purposely chose to simulate a /8 network range, equivalent to 16,777,214 hosts. The IP addresses of the virtual Linux servers are statically assigned to them by the router. Initially, I wanted to have a virtual router in my environment, but it proved to be easy to overwhelm with scanning, so I decided to use a physical router with segmented subnets and static IP addresses for servers and computers instead. I initially used a TP-Link router, but it didn't support network segmentation, even after attempting to flash OpenWrt onto it. I then borrowed a MikroTik router from a friend, which turned out to be effective for this lab. The network is purposely segmented into the attacker subnet 192.168.1.0/24 and servers' subnet 10.0.0.0/8. The service ports are modular and configurable through global and host variable folders. Key services deployed include SSH, Apache, MySQL, FTP, Samba, SNMP, and SMTP. All servers are running a basic UFW firewall that filters all ports except those used by the active services. This setup facilitates easy adjustments to configurations and services, effectively challenging the scanning tools by incorporating common misconfigurations and outdated software versions.



Scanning Tools

Nmap

The most well-known network scanning tool that offers comprehensive service detection, port scanning, and host discovery capabilities. It supports all the scanning techniques mentioned above and provides detailed information about target services, operating systems, and version fingerprinting. Nmap also includes built-in vulnerability scanning modules and a powerful scripting engine, which enables the use of custom Lua scripts to automate tasks like brute-forcing, vulnerability detection, exploitation, firewall evasion, and custom network automation.

Zmap

Compared to Nmap, Zmap is optimized for high-speed scanning across large networks. With a one gigabit connection, it is said that it is possible to scan the whole Internet in about 45 minutes. Zmap specializes in rapid host discovery, making it ideal for large-scale network sweeps. However, its focus is primarily on identifying live hosts and open ports, one service at a time, lacking in-depth service detection and vulnerability analysis. Its efficiency is particularly useful when the objective is to quickly map out a large network, allowing an identification of potential targets for further examination. Among its many options, rate bandwidth cooldown and sender threads stand out the most, enabling the user to adjust the tool to their needs. Zmap supports TCP, UDP, and ICMP scans as well as DNS resolution.

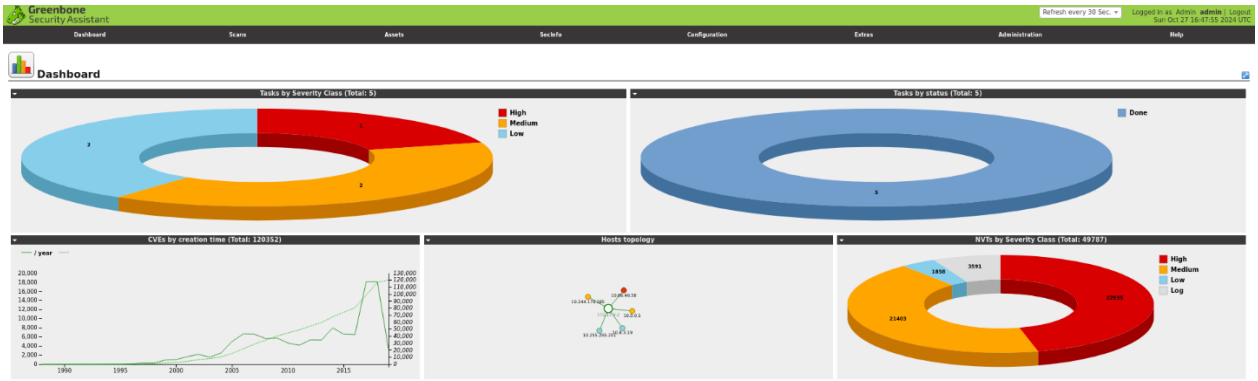
Masscan

The scanning tool designed for extreme performance, capable of scanning the entire IPv4 address space in just a few minutes. Like Zmap, it excels in host and port discovery but does not provide detailed service enumeration. Masscan is typically used when speed is the primary concern, and detailed service information can be collected later using tools like Nmap. Its ability to scan vast networks in record time makes it a valuable asset in scenarios where rapid data collection is essential. However, Masscan enables scanning based on multiple target ports and offers only SYN scanning, along with banner grabbing.

OpenVAS

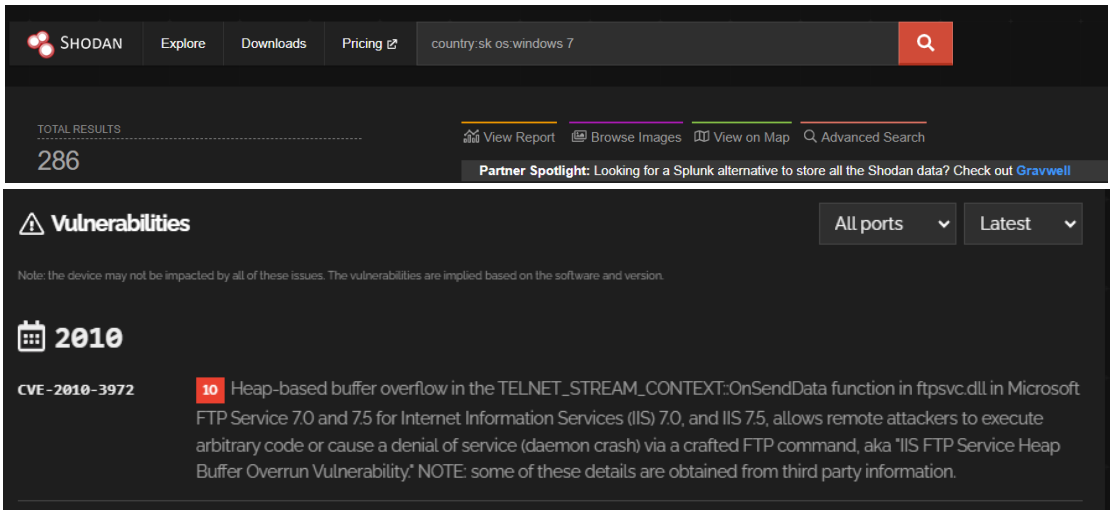
OpenVAS is a free, open-source alternative to the paid tool Nessus, integrating network discovery, port scanning, and vulnerability assessments all in one solution. Although it can be challenging to deploy, I found an older version inside Docker container for showcase. While the paid version includes additional rules, the free version remains powerful for identifying misconfigurations and outdated software. OpenVAS scans more slowly and is a lot more resource heavy than tools like

other scanning tools due to its thorough checks, and overhead, but provides detailed vulnerability reports and remediation advice. Users can also create tickets and schedule periodic scans.



Shodan

A good way to check what information about your services and devices is available on the internet is by using Shodan, a search engine for Internet-connected devices. Shodan allows users to search based on location, device type, service type, operating system, and more. While it has a paid version that offers enhanced API optimization and search capabilities, the free version still provides valuable insights. Shodan lists numerous vulnerabilities that could affect the services exposed to internet, making it an invaluable tool. Here is for example an alarmingly high number of Windows 7 devices connected to the internet, and an example of listing of vulnerabilities.



Axiom

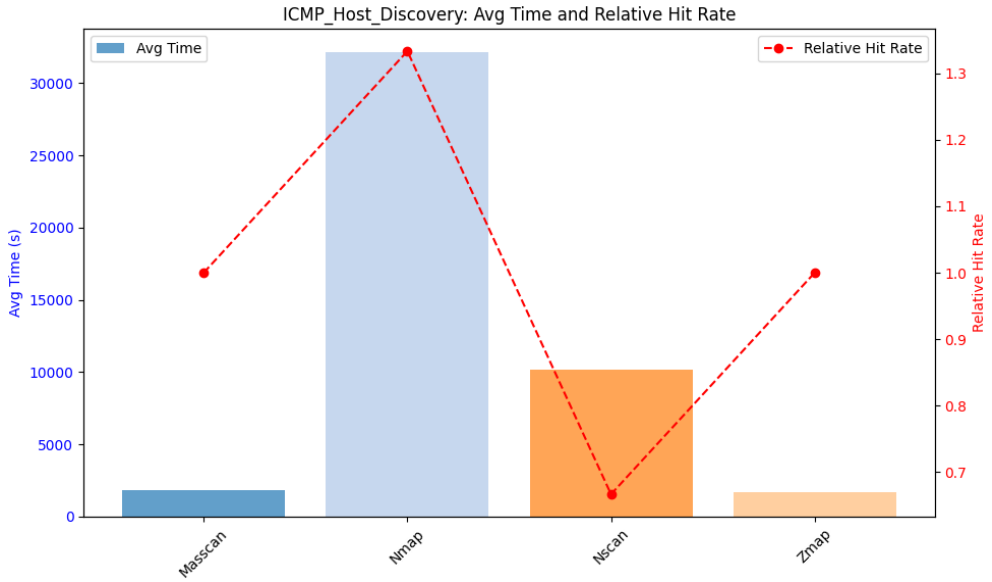
A tool worth mentioning even though it is out of scope of this semestral project. You can use it to distribute a scan or multiple types of scans across multiple virtual cloud providers. Essentially, you can create your own legal scanning botnet without fear of having your IP blacklisted, speeding up the process in the meantime. However, everything depends on how much money are you willing to pay to the cloud providers, and if are you willing to take the risk of getting banned from them.

Comparison

After building the lab environment and getting myself familiar with different tools I tried to find optimal combination of flags for each scenario, that I came up with: ICMP Host Discovery, Service Discovery, Fast Port Scan, Complex Port Scan. Based on trial and error I created test.sh script, that I ran multiple times, collected all the data, and crafted a Jupyter notebook that visualizes the collected data in graphs.

ICMP Host Discovery

The goal of this scan is essentially to find a needle in a haystack, given the vast number of potential hosts within a /8 subnet—up to 16,777,214 addresses. The objective is to send ICMP pings to each address and identify as many active hosts as possible. My observations indicate that while higher scan speeds, rates, and shorter timeouts can accelerate the process, they often reduce the reliability of results, as some responses may go undetected. Since I conducted these tests within my own home network, I set high rates to speed up completion. However, even with optimized settings, Nmap required 9.88 hours to finish. Nscan completed in about 2.73 hours, but its findings were less reliable due to a short dial timeout for pings. Masscan and Zmap performed best for host discovery; the entire scan finished in around 28 minutes, with Zmap providing a higher hit rate than Masscan, which I initially struggled to configure correctly.




```
masscan -i eth1 10.0.0.0/8 --ping --rate 10000
zmap -q -p 0 --probe-module=icmp_echoscan -i eth1 -r 10000 -G C4:AD:34:58:98:08 10.0.0.0/8
/opt/nscan -r 10.0.0.0/8 -d
nmap -sn -PE 10.0.0.0/8 -n -T5 --min-parallelism 100 --max-rtt-timeout 100ms --max-retries 1
```

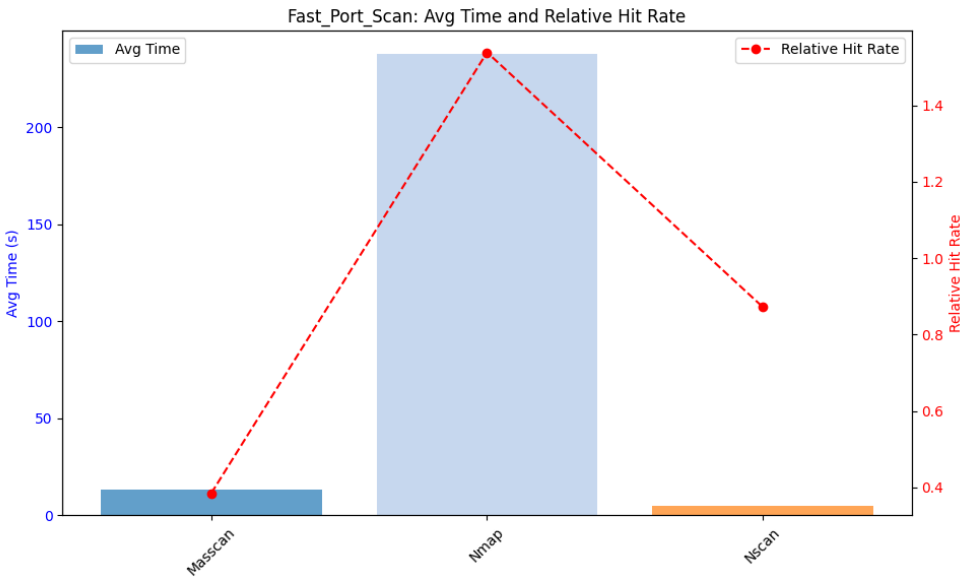
Service Discovery

After realizing that comparing Nmap and Nscan with Masscan and Zmap for scanning an entire /8 network wasn't meaningful, I focused on comparing Masscan and Zmap specifically for service discovery. Both tools were set to search for hosts with port 22 open, and I found their performance to be very similar, with each completing the scan in about 28 minutes. The main limitation with Zmap is its inability to scan multiple ports simultaneously, which makes it slightly less versatile than Masscan. However, Zmap is more reliable when scanning a limited number of ports or using ICMP. Adjusting parameters like --rate and --timeout also significantly impacts the effectiveness of each tool.

```
zmap -q -p 22 -i eth1 -G C4:AD:34:58:98:08 10.0.0.0/8 -r 10000
masscan -i eth1 10.0.0.0/8 -p22 --rate 10000
```

Fast Port Scan

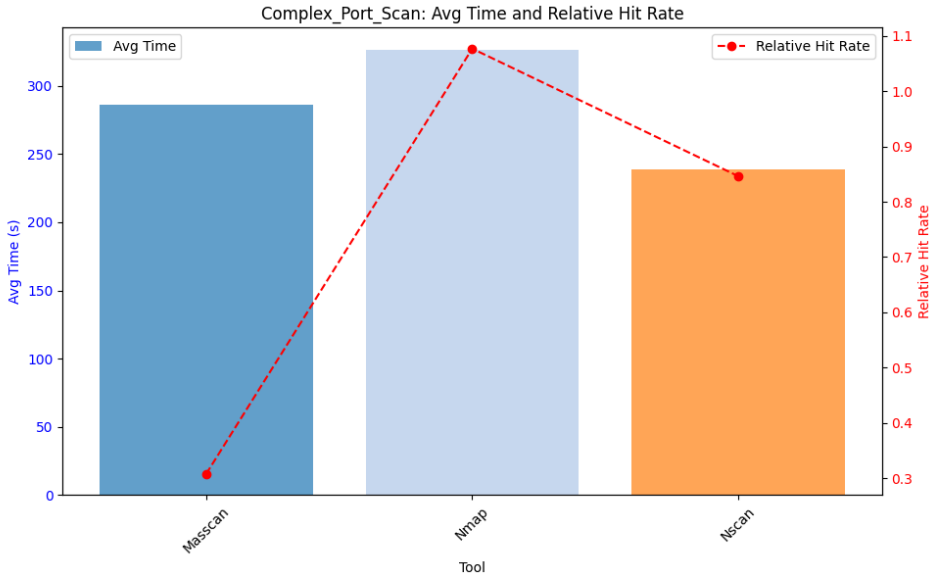
For scanning a specific set of six individual targets, I deliberately chose only Nmap, Masscan, and Nscan, as using Zmap would require running it 1,024 times to cover all basic ports effectively. I found that Nmap took the longest to complete up to 2.5 minutes, but provided the most accurate results, with a more comprehensive list of known ports than Nscan. However, Nscan was the fastest of the three tools finishing under 1 minute.




```
ips=(10.255.255.251 10.244.170.185 10.86.46.58 10.8.3.19 10.8.3.17 10.0.0.1)
nmap -sSV -Pn -n -T4 "${ips[@]}"
masscan "${ips[@]}" -p1-1024 -i eth1 --rate 10000
/opt/nscan -r "$coma_ips"
```

Complex Port Scan

The goal of this complex port scan was to test all 65,535 ports using Masscan, Nmap, and Nscan, rather than limiting the scan to known ports. Masscan demonstrated again that without careful rate adjustment, its high speed can lead to missed results, making it better suited for scanning a select few ports across a large network. Nmap proved to be the most thorough, completing the scan in 6 minutes while also capturing detailed information, including banner grabbing. My tool, Nscan, was the fastest, finishing in 4 minutes, though it was less reliable and captured only a few banners in comparison.




```
ips=(10.255.255.251 10.244.170.185 10.86.46.58 10.8.3.19 10.8.3.17 10.0.0.1)
nmap -sSV -p- -Pn -n -T4 "${ips[@]}"
masscan "${ips[@]}" --banners -p1-65535 -i eth1 --rate 10000
/opt/nscan -r "$coma_ips" -p 0-65535 -b
```

Output Options

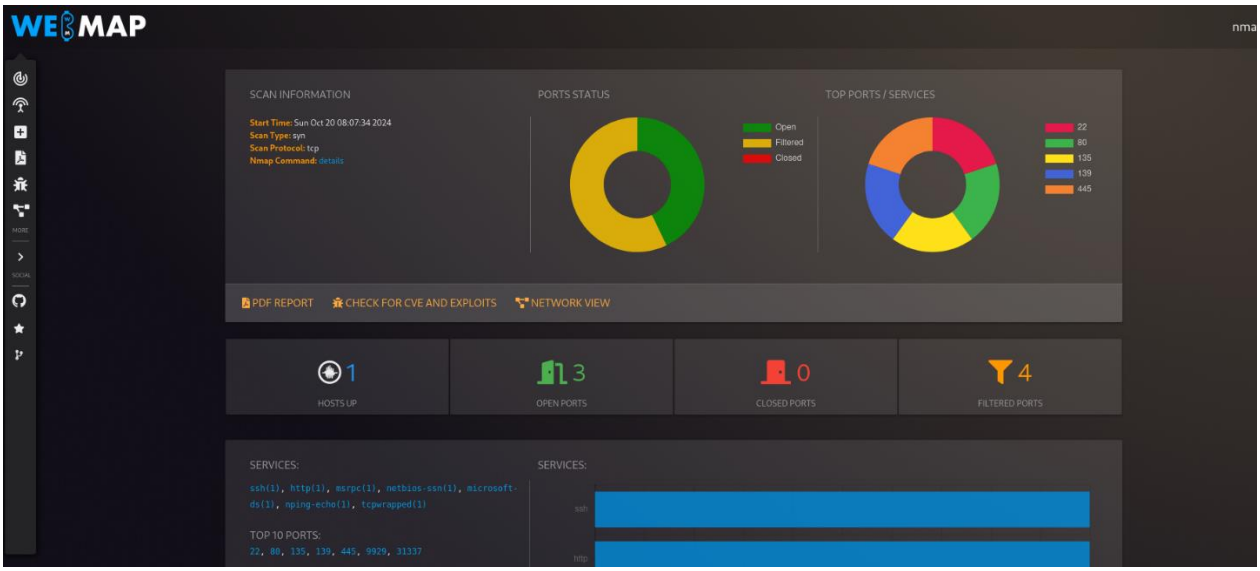
Nmap offers various output formats tailored to different needs: Normal provides human-readable text, while XML and in newer versions JSON deliver machine-readable data for easy parsing in code. The Grepable format simplifies output for scripting, and Script Kiddie serves as a humorous take on the results. Zmap primarily outputs in CSV, which is structured for straightforward analysis, and supports sending results to Redis for real-time processing. It can also import results directly into MongoDB and provides a Binary format for conversion into other types. Masscan, like Nmap, produces XML and JSON outputs for machine readability, offers a Grepable format for streamlined output, and includes a Binary option for flexible transformations. Nscan can only output to stdout.

Testing Summary

During testing I explored various configurations for each scanning tool across multiple scenarios. Nmap excels in detailed service detection and has a powerful scripting engine, making it ideal for comprehensive scans, though it's slower than other tools. Zmap is optimized for rapid scanning across large networks, sacrificing detail for speed, and is suited for large-scale ICMP sweeps or single service discovery. Masscan is designed for extreme performance, capable of scanning the entire IPv4 space quickly, but it requires precise target ports and only supports TCP SYN/ACK, limiting its ability to detect UDP ports, perform DNS lookups, or ICMP probing. It is not for scanning individual hosts. My Nscan proved to be competitive with other tools, but it needs further development to include as many options as they do, such as rate limiting, input/output options, and even simple features like blacklists, which are crucial for conducting scans of large networks. Overall, these tests underscored the importance of choosing a correct tool, and of tuning parameters like rate and timeout for each tool to optimize performance and reliability.

Example Scanning Pipeline

As a proof of concept for simple scanning, I proposed a pipeline based on the findings of my tests. I developed a script that performs an ICMP scan of the entire given range using Zmap. The output is saved as a .csv file, which is then fed into a comprehensive Nmap scan configured to scan all TCP, and optionally all UDP ports. The results of this scan are saved in an .xml file, allowing for processing with either a custom tool or an open-source tool like WebMap, which I set up inside Docker for this demonstration.



Conclusion

In this semester project, I explored various scanning techniques and developed a simplified version of my own network scanner. I analyzed other similar tools and designed a virtual testing environment. Despite facing challenges with the router configuration, I successfully made Zmap and Masscan work. I conducted multiple tests, and based on them created a simple example of your own scanning pipeline. I demonstrated how Shodan can be utilized to identify exposed services and their vulnerabilities. Additionally, I explained how one could create a legal botnet using Axiom for scanning and how to leverage tools like OpenVAS for periodic assessments.

Overall, my analysis indicates that Nmap is preferable for detailed examination, while Zmap and Masscan are better suited for rapid host discovery. This comparative analysis highlights the strengths and weaknesses of each tool, assisting in selecting the appropriate scanner based on their specific requirements and the context of their network assessments.

Sources - from October 2024

[1] Lothos612. *Shodan*. https://github.com/lothos612/shodan

[2] Prometheus Community. *Pro-bing*. https://github.com/prometheus-community/pro-bing

[3] Pry0cc. *Axiom*. https://github.com/pry0cc/axiom

[4] Hackers Arise. *ZMap for scanning the internet: Scan the entire internet in 45 minutes*. https://www.hackers-arise.com/post/zmap-for-scanning-the-internet-scan-the-entire-internet-in-45-minutes

[5] ZMap Project. *Zmap*. https://github.com/zmap/zmap

[6] Haax. *Masscan cheat sheet*. https://cheatsheet.haax.fr/network/port-scanning/masscan_cheatsheet/

[7] Robert David Graham. M*asscan*. https://github.com/robertdavidgraham/masscan

[8] Steele, T., Patten, C., & Kottmann, D. (2020). *Black Hat Go: Go Programming for Hackers and Pentesters*. No Starch Press. ISBN-13: 9781593278656.

[9] Hack The Box Academy. *Module 19*. https://academy.hackthebox.com/module/19

[10] TryHackMe. *OpenVAS room*. https://tryhackme.com/r/room/openvas

[11] Cisco. *CyberOps Associate*.

[12] Nmap Project. *Nmap*. https://nmap.org/

[13] Rana, S. *WebMap*. GitHub. https://github.com/SabyasachiRana/WebMap