Samuel Leonard

Candidate Name: Sam Leonard  
Candidate number: 4059  
Centre Number: 56400

Vulnerability Scanner

NEA

# Contents

ANALYSIS:

Identification and Background to the Problem ……………………………………………………………………………….. 2.

Analysis of problem ………………………………………………………………………………………………………………………… 5.

Numbered List of Objectives (also called Success Criteria, the end user requirements) ………………………

Description of current system or existing solutions ………………………….…………………………………………………

Prospective Users ……………………………………………………………………………………………………………………………....

Data Dictionary ………………..…………………………………………………………………………………………………………………

Data Flow Diagram ………………..……………………………………………………………………………………………………………

Data Sources ……………………………………………………………………………………………………………………………………….

Description of Solution Details, Networking ……………………………………………………………………………………….

Data Volumes ………………………….………………………………………………………………………………………………………….

Test Strategy ………….……………………………………………………………………………………………………………………………

DESIGN:

Overall System Design (High Level Overview) (Core) ………………………………………………………………………….

Design of User Interfaces HCI (Core) ………………………………………………………………………………………………….

Database Structure (ERD, Normalisation) (Core, if relevant) ………………………………………………………………

System Algorithms (Flowcharts) (Core) ………………………………………………………………………………………………

Input data Validation (Core) ………………………………………………………………………………………………………………

Proposed Algorithms for complex structures (flow charts or Pseudo Code) ………………………………………

Design Data Dictionary (Core) ……………………………………………………………………………………………………………

TECHNICAL SOLUTION:

Program Listing ………………………………………………………………………………………………………………………………….

Comments (Core) ………………………………………………………………………………………………………………………………

Overview to direct the examiner to areas of complexity and explain design evidence ………………………

TESTING:

Test Plan ……………………………………………………………………………………………………………………………………………

Test Table ………………………………………………………………………………………………………………………………………….

EVALUATION:

Reflection on final outcome ………………………………………………………………………………………………………………

Evaluation against objectives, end user feedback ……………………………………………………………………………..

Potential Improvements ……………………………………………………………………………………………………………………

APPENDICES:

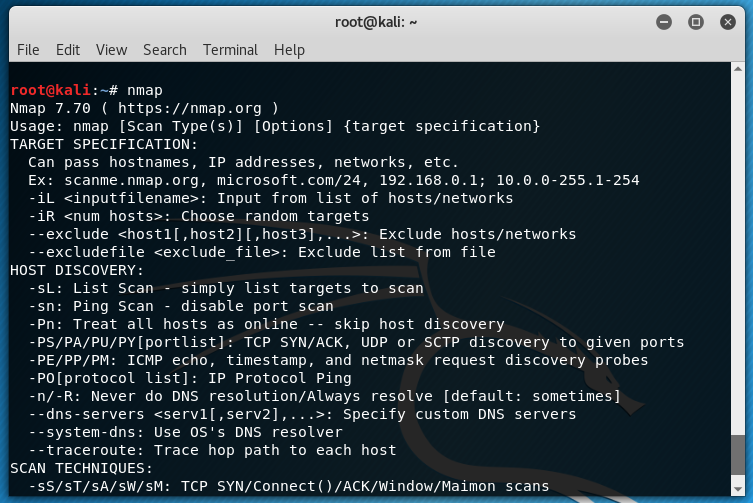
# Analysis

## Identification and Background to the Problem

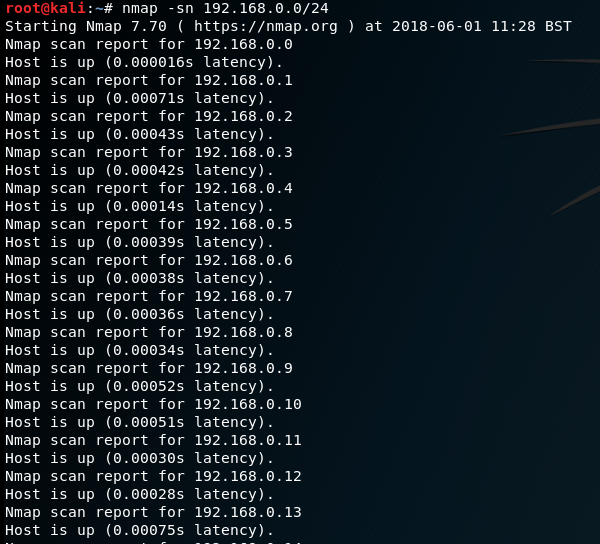
My NEA is a vulnerability scanner which scans a supplied ip/subnet range for open ports, and from there tries to detect what services are running on the ports, are the ports filtered etc. From there it would attempt to identify the operating system version as well as the version of the services running on any open ports.

### Existing Tools:

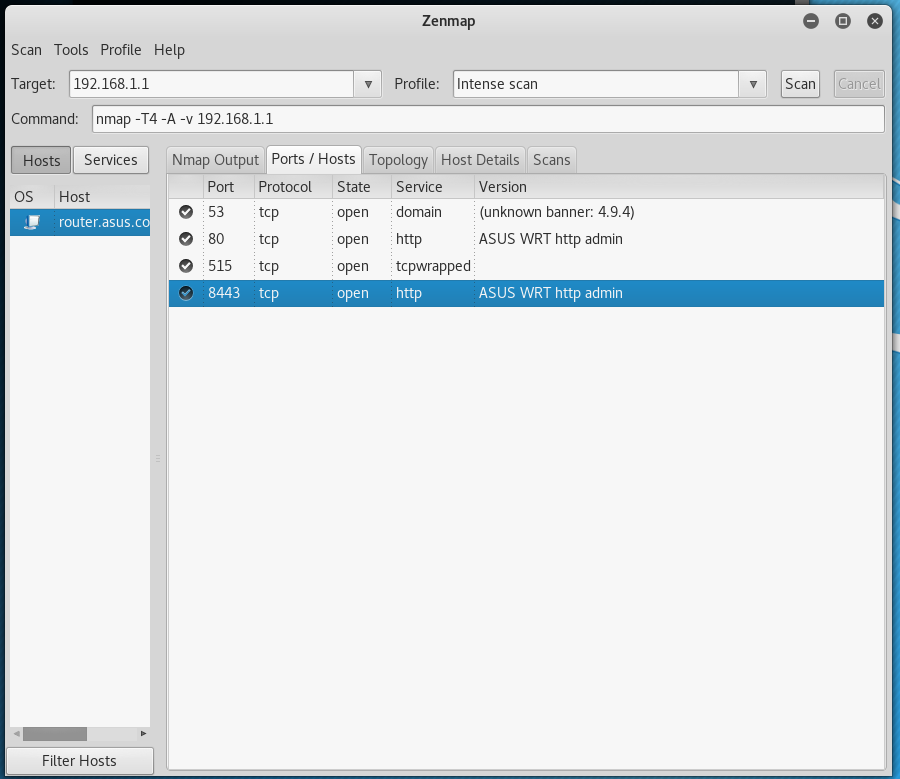
nmap is currently one of the best tools for doing network mapping/enumeration, it is a well established product that is used by many people and has a huge number of options.

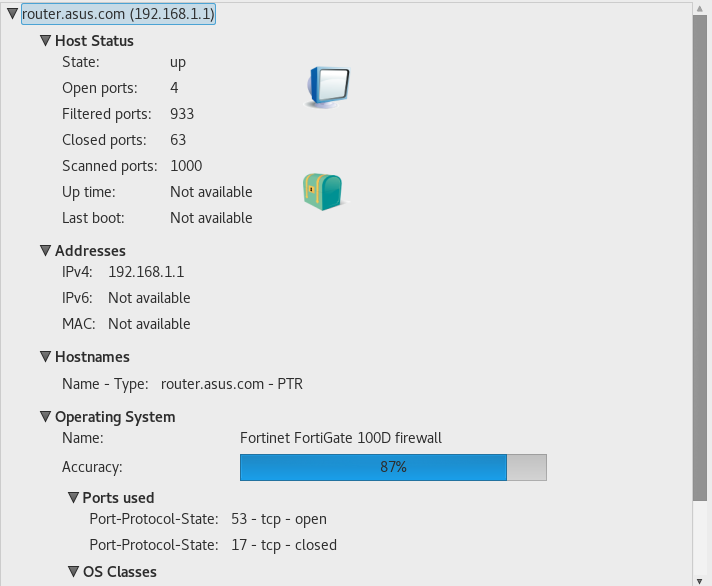


Nmap can be used to scan a subnet with the following command: nmap -pn 192.168.0.0/24



nmap also has a gui version called zenmap which contains several features to make life easier for someone who is new to the tool, such as the preset modes including: intensive scan, ping scan, slow comprehensive scan and quick scan. Zenmap presents the information in a nice format that is easy to understand. The two images below are a zenmap intensive scan of an asus router.





### Adaptations for my Solution.

My Solution will include integration with the exploit-db database among other tools to check whether a service that is running on a port is vulnerable to any known exploits and whether there are any patches available for the service.

## Analysis of problem

The commands

The user will interact with the program via either a terminal interface (BASH/CMD) or with a python shell. On Linux the program should be placed in a directory which is contained by the PATH variable so that the user can run the program without providing the absolute directory to the program; if the user doesn’t do this they will have to provide absolute directory e.g. ~/scripts/mapper.py –pn 192.168.1.0/24 instead of mapper –pn 192.168.1.0/24.

From there you could use the exploit-db API to see if there were any publicly disclosed vulnerabilities for the system/services and also whether they were any patches out.

It would then display this information back to the user. To achieve this, I will need to implement algorithms which can translate IP address / subnet form ranges into a list of IP addresses, I will also need to learn about how network protocols interact on a low level so that I can identify what services are running on a port, as well as this the integrated exploit-db search will require me to learn how I can scrape information from a webpage and/or interact with a public API, this will also require me to handle the JSON/XML data returned by the API.

I have decided that I will use the same syntax as nmap for passing data to my program. This is so that users of nmap will be familiar with the basic features of my program and will have an easier time transitioning across.

Nmap 5.51 ( [http://nmap.org](http://nmap.org/) )

usage: nmap [Scan **Type**(s)] [Options] {target specification}

TARGET SPECIFICATION:

Can pass hostnames, IP addresses, networks, etc.

Ex: scanme.nmap.org, 192.168.0.1; 10.0.0-255.1-254

-iL <inputfilename>: Input from list of hosts/networks

-iR <num hosts>: Choose random targets

--exclude <host1[,host2][,host3],...>: Exclude hosts/networks

--excludefile <exclude\_file>: Exclude list from file

HOST DISCOVERY:

-sL: List Scan - simply list targets to scan

-sn: Ping Scan - disable port scan

-Pn: Treat all hosts as online -- skip host discovery

-PS/PA/PU/PY[portlist]: TCP SYN/ACK, UDP or SCTP discovery to given ports

-PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes

-PO[protocol list]: IP Protocol Ping

-n/-R: Never do DNS resolution/Always resolve [default: sometimes]

--dns-servers <serv1[,serv2],...>: Specify custom DNS servers

--system-dns: Use OS's DNS resolver

--traceroute: Trace hop path to each host

SCAN TECHNIQUES:

-sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/Maimon scans

-sU: UDP Scan

-sN/sF/sX: TCP Null, FIN, and Xmas scans

--scanflags <flags>: Customize TCP scan flags

-sI <zombie host[:probeport]>: Idle scan

-sY/sZ: SCTP INIT/COOKIE-ECHO scans

-sO: IP protocol scan

-b <FTP relay host>: FTP bounce scan

PORT SPECIFICATION AND SCAN ORDER:

-p <port ranges>: Only scan specified ports

Ex: -p22; -p1-65535; -p U:53,111,137,T:21-25,80,139,8080,S:9

-F: Fast mode - Scan fewer ports than the default scan

-r: Scan ports consecutively - don't randomize

--top-ports <number>: Scan <number> most common ports

--port-ratio <ratio>: Scan ports more common than <ratio>

SERVICE/VERSION DETECTION:

-sV: Probe open ports to determine service/version info

--version-intensity <level>: Set from 0 (light) to 9 (try all probes)

--version-light: Limit to most likely probes (intensity 2)

--version-all: Try every single probe (intensity 9)

--version-trace: Show detailed version scan activity (for debugging)

SCRIPT SCAN:

-sC: equivalent to --script=default

--script=<Lua scripts>: <Lua scripts> is a comma separated list of

directories, script-files or script-categories

--script-args=<n1=v1,[n2=v2,...]>: provide arguments to scripts

--script-trace: Show all data sent and received

--script-updatedb: Update the script database.

OS DETECTION:

-O: Enable OS detection

--osscan-limit: Limit OS detection to promising targets

--osscan-guess: Guess OS more aggressively

TIMING AND PERFORMANCE:

Options which take <time> are in seconds, or append 'ms' (milliseconds),

's' (seconds), 'm' (minutes), or 'h' (hours) to the value (e.g. 30m).

-T<0-5>: Set timing template (higher is faster)

--min-hostgroup/max-hostgroup <size>: Parallel host scan group sizes

--min-parallelism/max-parallelism <numprobes>: Probe parallelization

--min-rtt-timeout/max-rtt-timeout/initial-rtt-timeout <time>: Specifies

probe round trip time.

--max-retries <tries>: Caps number of port scan probe retransmissions.

--host-timeout <time>: Give up on target after this long

--scan-delay/--max-scan-delay <time>: Adjust delay between probes

--min-rate <number>: Send packets no slower than <number> per second

--max-rate <number>: Send packets no faster than <number> per second

FIREWALL/IDS EVASION AND SPOOFING:

-f; --mtu <val>: fragment packets (optionally w/given MTU)

-D <decoy1,decoy2[,ME],...>: Cloak a scan with decoys

-S <IP\_Address>: Spoof source address

-e <iface>: Use specified interface

-g/--source-port <portnum>: Use given port number

--data-length <num>: Append random data to sent packets

--ip-options <options>: Send packets with specified ip options

--ttl <val>: Set IP time-to-live field

--spoof-mac <mac address/prefix/vendor name>: Spoof your MAC address

--badsum: Send packets with a bogus TCP/UDP/SCTP checksum

OUTPUT:

-oN/-oX/-oS/-oG <file>: Output scan in normal, XML, s|<rIpt kIddi3,

and Grepable format, respectively, to the given filename.

-oA <basename>: Output in the three major formats at once

-v: Increase verbosity level (use -vv or more for greater effect)

-d: Increase debugging level (use -dd or more for greater effect)

--reason: Display the reason a port is in a particular state

--open: Only show open (or possibly open) ports

--packet-trace: Show all packets sent and received

--iflist: Print host interfaces and routes (for debugging)

--log-errors: Log errors/warnings to the normal-format output file

--append-output: Append to rather than clobber specified output files

--resume <filename>: Resume an aborted scan

--stylesheet <path/URL>: XSL stylesheet to transform XML output to HTML

--webxml: Reference stylesheet from Nmap.Org for more portable XML

--no-stylesheet: Prevent associating of XSL stylesheet w/XML output

MISC:

-6: Enable IPv6 scanning

-A: Enable OS detection, version detection, script scanning, and traceroute

--datadir <dirname>: Specify custom Nmap data file location

--send-eth/--send-ip: Send using raw ethernet frames or IP packets

--privileged: Assume that the user is fully privileged

--unprivileged: Assume the user lacks raw socket privileges

-V: Print version number

-h: Print this help summary page.

Above you can see the man page[1] (man pages are a system for storing the manuals for all of the software on a system). NMAP implements an enormous number of flags and as such it would be unrealistic for me to implement all of them in my program, instead I will implement the most common/useful commands[2] as well as some others which nmap itself doesn’t have.

### List of commands I will implement along with their syntax.

* Ping scan (-sn)

<EXPLAIN HOW OS DETECTION WILL BE VERY HARD>

Success requirements:

1. Scan an IP address for open TCP ports and determine the state of each port, i.e. open, filtered or closed.
2. Scan an IP address for open UDP ports and determine the state of each port, i.e. open, filtered or closed.
3. Attempt to identify any services running on ports, i.e Secure Shell(SSH) on port 22
4. Run basic operating system detection (Windows / Linux / Solaris etc. differentiation)
5. Return all vulnerable services as per exploit-db
6. Check if a server is vulnerable to the ssl heartbleed bug etc.

Output a JSON file with the data in afterwards so that the scan results can loaded

Success requirements:

1. Return a list of the open ports both User Datagram Protocol(UDP) and Transport Control Protocol(TCP) on a subnet
2. Attempt to identify the services that are running on the open ports, such as detecting Secure Shell(SSH) on port 22, HyperText Transfer Protocol(HTTP) on port 80 or HyperText Transfer Protocol Secure(HTTPS) on port 443.
3. Attempt to implement basic operating system detection (Windows / Linux / Solaris etc. differentiation)
4. Once services have been detected attempt to identify which version of the services are running
5. Once the versions of the services have been identified interface with exploit-db(an online vulnerability database) to see if any of the running services are vulnerable to any known vulnerabilities as well as whether there are patches out for the services.
6. If a secure webserver is running attempt to detect whether it is vulnerable to the ssl heartbleed bug by sending malformed SSL heartbeats.

# Bibliography

1. <https://linux.die.net/man/1/nmap>
2. <https://bencane.com/2013/02/25/10-nmap-commands-every-sysadmin-should-know/>