# Lecture 5: Scanning & Enumeration

BKACAD's Security Training

# Table of Content

# Table of Content

# Wireshark & Protocols

## Wireshark – Network sniffer

Used for learning network protocols, analyzing network traffic, and debugging network services.

Uses Libpcap (on Linux) or Winpcap (on Windows) libraries in order to capture packets from the network.

```
kali@kali:~$ sudo wireshark
```

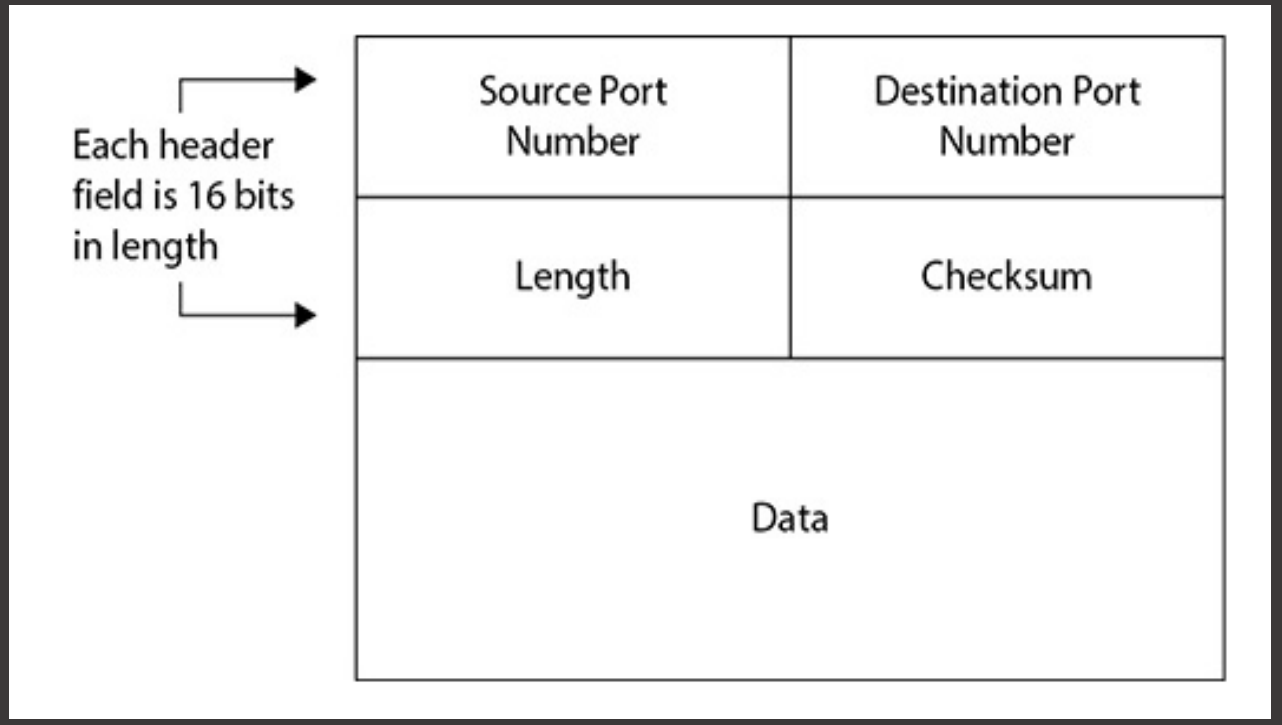
# Wireshark & Protocols

## Connectionless Communication – UDP

At the Transport layer, connectionless communication is accomplished with UDP. UDP is a low-overhead, simple, and fast transport protocol. Examples of protocols using UDP are: TFTP, DNS (for lookups), and DHCP.

# Wireshark & Protocols

## UDP Structure



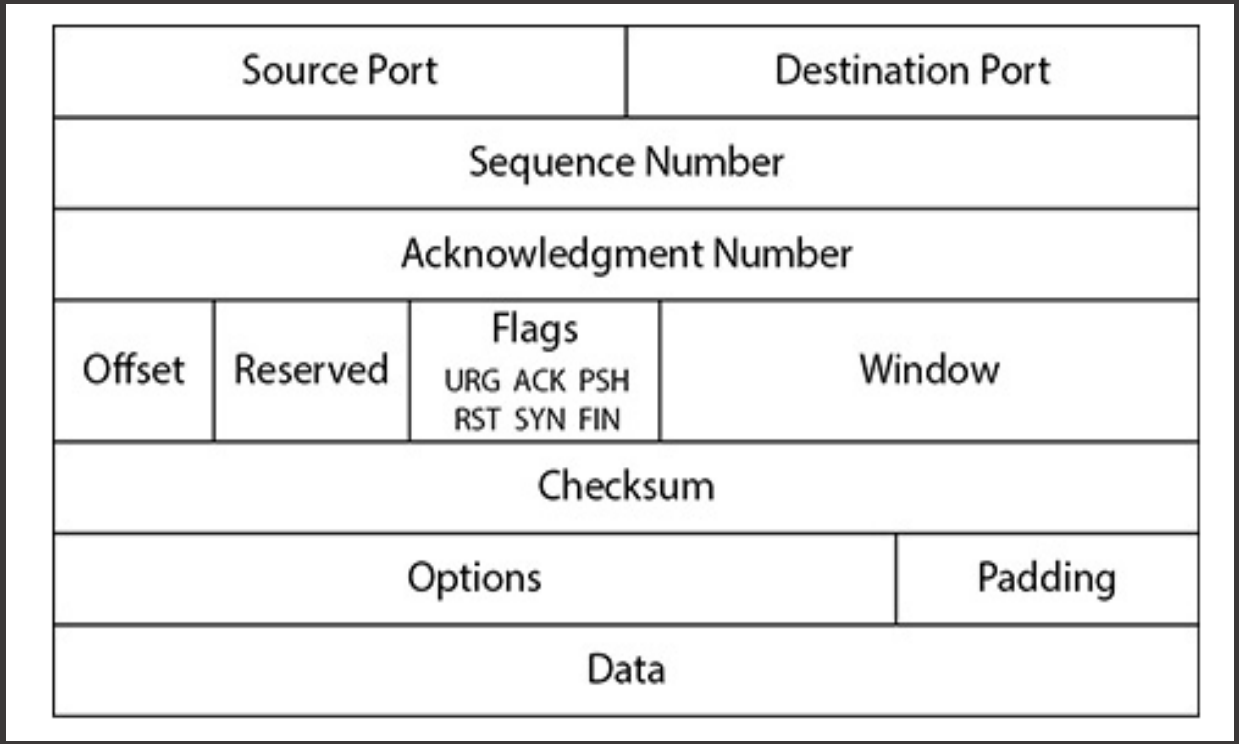
# Wireshark & Protocols

## Connection-Oriented Communication – TCP

It requires a lot more overhead and if oftentimes a lot slower than connectionless communication, is a much more orderly form of data exchange and makes a lot more sense for transporting large files or communicating across network boundaries. Senders will reach out to recipients, before data is ever even sent, to find out whether they're available and whether they'd be willing to set up a data channel. Once the data exchange begins, the two systems continue to talk with one another, making sure flow control is accomplished, so the recipient isn't overwhelmed and can find a nice way to ask for retransmissions in case something gets lost along the way.

# Wireshark & Protocols

## TCP Structure



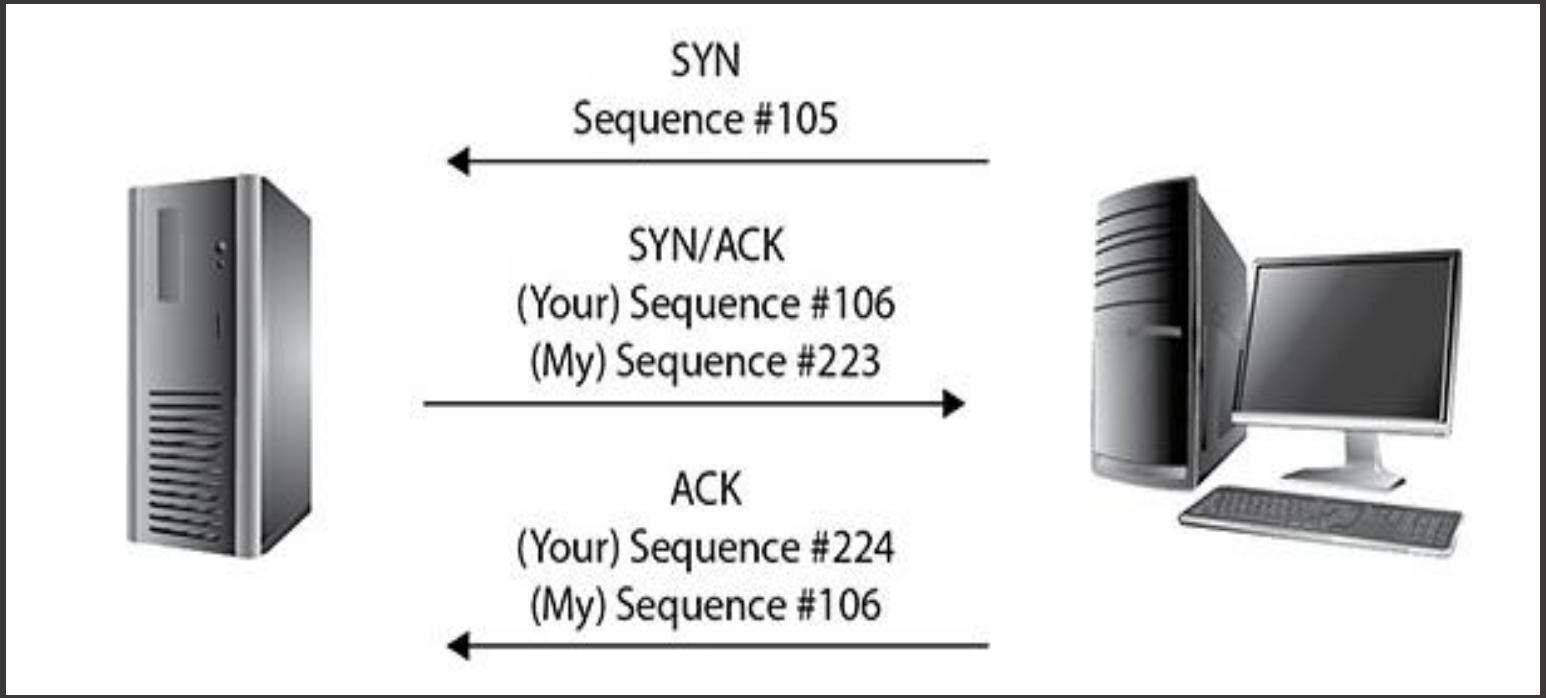
# Wireshark & Protocols

## TCP flags

1. SYN (Synchronize): This flag is set during initial communication establishment. It indicates negotiation of parameters and sequence numbers.
2. ACK (Acknowledgment): This flag is set as an acknowledgment to SYN flags. This flag is set on all segments after the initial SYN flag.
3. RST (Reset): This flag forces a termination of communications (in both directions).
4. FIN (Finish): This flag signifies an ordered close to communications.
5. PSH (Push): This flag forces the delivery of data without concern for any buffering. In other words, the receiving device need not wait for the buffer to fill up before processing the data.
6. URG (Urgent): When this flag is set, it indicates the data inside is being sent out of band. Cancelling a message mid-stream is one example.

# Wireshark & Protocols

## Three-ways handshake



# Wireshark & Protocols

## Port Numbering

Internet Assigned Numbers Authority (IANA) maintains something called the Service Name and Transport Protocol Port Number Registry, which is the official list for all port number reservations.

The port numbers range from 0 to 65535 and are split into three different groups:

- Well-known ports: 0–1023
- Registered ports: 1024–49151
- Dynamic ports: 49152–65535

# Wireshark & Protocols

## Port Numbering – Common port numbers

| Port Number | Protocol | Transport Protocol | Port Number | Protocol | Transport Protocol |
|---|---|---|---|---|---|
| 20/21 | FTP | TCP | 110 | POP3 | TCP |
| 22 | SSH | TCP | 135 | RPC | TCP |
| 23 | Telnet | TCP | 137–139 | NetBIOS | TCP and UDP |
| 25 | SMTP | TCP | 143 | IMAP | TCP |
| 53 | DNS | TCP and UDP | 161/162 | SNMP | UDP |
| 67 | DHCP | UDP | 389 | LDAP | TCP and UDP |
| 69 | TFTP | UDP | 443 | HTTPS | TCP |
| 80 | HTTP | TCP | 445 | SMB | TCP |

# Wireshark & Protocols

## Exercise

1. Open 3 captured packets in Wireshark.


2. Read and understand the output. Where is the three-way handshake happening? Where is Seq numbers?
3. Follow the TCP stream to read the login attempt.

# Table of Content

# Check for Live Systems

## ICMP

Checking for live systems is the first step. The simplest and easiest way to do this is to take advantage of a protocol that's buried in the stack of every TCP/IP-enabled device on the planet - Internet Control Message Protocol (ICMP).

# Check for Live Systems

## ICMP message types

| ICMP Message Type | Description and Important Codes |
|---|---|
| 0: Echo Reply | Answer to a Type 8 Echo Request |
| 3: Destination Unreachable | Error message indicating the host or network cannot be reached. The codes follow:<br>**0**—Destination network unreachable<br>**1**—Destination host unreachable<br>**6**—Network unknown<br>**7**—Host unknown<br>**9**—Network administratively prohibited<br>**10**—Host administratively prohibited<br>**13**—Communication administratively prohibited |
| 4: Source Quench | A congestion control message |
| 5: Redirect | Sent when there are two or more gateways available for the sender to use and the best route available to the destination is not the configured default gateway. The codes follow:<br>**0**—Redirect datagram for the network<br>**1**—Redirect datagram for the host |
| 8: Echo Request | A ping message, requesting an Echo reply |
| 11: Time Exceeded | The packet took too long to be routed to the destination (code 0 is TTL expired) |

# Check for Live Systems

## ICMP Scanning

Ping scan involves sending ICMP ECHO requests (Type 8) to a host.

If the host is alive, it will return an ICMP ECHO reply (Type 0).

This scan is useful for locating active devices or determining if the ICMP is passing through a firewall.

# Check for Live Systems

## ICMP Scanning – Ping Sweep

```
kali@kali:~$ nmap -sn 192.168.1.1-254

kali@kali:~$ nmap -sn 192.168.1.1-254 -oG ping-sweep.txt

kali@kali:~$ grep Up ping-sweep.txt | cut -d " " -f 2
```

# Wireshark & Protocols

## Exercise

1. Download Metasploitable 2 from below link:
   https://sourceforge.net/projects/metasploitable/
2. Setup Kali-Linux & Metasploitable 2 to one NIC like: NAT.
3. Perform “Ping Sweep” with `nmap` to determine Metasploitable’s IP address.

# Table of Content

# Port Scanning

## Stealth / SYN Scanning

SYN scanning is a TCP port scanning method that involves sending SYN packets to various ports on a target machine without completing a TCP handshake. If a TCP port is open, a SYN-ACK should be sent back from the target machine, informing us that the port is open. At this point, the port scanner does not bother to send the final ACK to complete the three-way handshake.

```
kali@kali:~$ sudo nmap -sS 192.168.1.3
```

# Port Scanning

## TCP Connect Scanning

Nmap TCP connect scan makes use of the Berkeley sockets API to perform the three-way handshake, it does not require elevated privileges.

Because Nmap has to wait for the connection to complete before the API will return the status of the connection, a connect scan takes much longer to complete than a SYN scan.

```
kali@kali:~$ nmap -sT 192.168.1.3
```

# Port Scanning

## UDP Connect Scanning

When performing a UDP scan, Nmap will use a combination of two different methods to determine if a port is open or closed. For most ports, it will use the standard “ICMP port unreachable” method described earlier by sending an empty packet to a given port. However, for common ports, such as port 161, which is used by SNMP, it will send a protocol-specific SNMP packet in an attempt to get a response from an application bound to that port. To perform a UDP scan, the `-sU` option is used and `sudo` is required to access raw sockets.

```
kali@kali:~$ sudo nmap -sU 192.168.1.3
```

# Port Scanning

## Port Scanning + Ping Sweep

To save time and network resources, we can also scan multiple IPs, probing for a short list of common ports. For example, let's conduct a TCP connect scan for the top twenty TCP ports with the `--top-ports` option and enable OS version detection, script scanning, and traceroute with `-A`

```
kali@kali:~$ nmap -sT -A --top-ports=20 192.168.1.1-254 -oG
top-port-sweep.txt
```

# Port Scanning

## Exercise

1. Perform TCP scan, SYN can & UDP scan with `nmap` versus Metasploitable 2 machine and then analyze traffic with `wireshark`
2. Write a simple Port Scan again common port like: 80, 22, 21, 23, 443 with Python socket module.

Hint: `import socket, dir(socket), List, socket.socket(), connect_ex(),…`

# Port Scanning

## Exercise – Walkthrough

GNU nano 3.2 portScan.py

```
#!/usr/bin/python

'''
TCP Port scanner with socket
'''

import sys
import socket

def scanner(target, port):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(5)
        if s.connect_ex((target, port)) == 0:
                print "Port %d open" %(port)
        else:
                pass
        s.close()

def main():
        try:
                target = raw_input("Input your target: ")
                list_port = [80, 21, 22, 23, 443, 110]
                for port in list_port:
                        scanner(target, port)
        except socket.error:
                print "Network error!"
        except KeyboardInterrupt:
                print "You pressed Ctrl+C"
                sys.exit()


if __name__ == "__main__":
        main()
```

# Table of Content

# Banner Grabbing

## OS Fingerprinting

Nmap has a built-in feature called OS fingerprinting, which can be enabled with the `-O` option.

This is possible because operating systems often have slightly different implementations of the TCP/IP stack (such as varying default TTL values and TCP window sizes) and these slight variances create a fingerprint that Nmap can often identify.

```
kali@kali:~$ nmap -O 192.168.1.3
```

# Banner Grabbing

## Banner Grabbing/Service Enumeration

Identify services running on specific ports by inspecting service banners `-sV` and running various OS and service enumeration scripts `-A` against the target.

Keep in mind that banners can be modified by system administrators. As such, these can be intentionally set to fake service names in order to mislead a potential attacker.

```
kali@kali:~$ nmap -A -sV 192.168.1.3
```

# Banner Grabbing

## Nmap Scripting Engine (NSE)

Use the Nmap Scripting Engine (NSE) to launch user-created scripts in order to automate various scanning tasks.

NSE scripts are located in the `/usr/share/nmap/`scripts directory.

```
kali@kali:~$ nmap --script-help script_name

kali@kali:~$ nmap 192.168.1.3 --script=script_name
```

# Banner Grabbing

## Exercise

1. Perform Banner Grabbing, OS finger-printing for Metasploitable 2
2. Try to use NSE script named `http-headers.nse` again Metasploitable 2
3. Write a simple Banner Grabbing script again common port like: 80, 22, 21 with Python socket module.

Hint: `import socket, dir(socket), List, socket.socket(), connect(),…`

# Banner Grabbing

## Exercise – Walkthrough

```
#!/usr/bin/python

'''
Banner Grabbing ports with socket
'''

import sys
import socket

def scanner(target, port):
        try:
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s.connect((target, port))
                if port == 80:
                        s.sendall("GET HTTP/1.1 \r \n")
                else:
                        s.sendall("Hello \r \n")
                print s.recv(1024)
                s.close()
        except socket.error:
                pass
```

```
                else:
                        s.sendall("Hello \r \n")
                print s.recv(1024)
                s.close()
        except socket.error:
                pass

def main():
        try:
                target = raw_input("Input your target: ")
                list_port = [80, 21, 22, 23, 443]
                for port in list_port:
                        scanner(target, port)
        except KeyboardInterrupt:
                print "You pressed Ctrl+C"
                sys.exit()


if __name__ == "__main__":
        main()
```

# Table of Content

# Enumeration

## SMTP Enumeration Users

```
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect(("192.168.1.137", 25))
>>> s.send("VRFY root \r\n")
>>> s.recv(1024)
'220 metasploitable.localdomain ESMTP Postfix (Ubuntu)\r\n252 2.0.0 root\r\n'
>>> s.send("VRFY root2 \r\n")
>>> s.recv(1024)
'550 5.1.1 <root2>: Recipient address rejected: User unknown in local recipient table\r\n'
```

# Enumeration

## Exercise

1. Write a simple SMTP Enumeration script
2. Improve Python code to automate the process of username discovery using a text file with usernames as input.

Hint: `import socket, dir(socket), 25, socket.socket(), connect(),…`

# Banner Grabbing

## Exercise – Walkthrough

```
GNU nano 3.2                                    smtpEnum.py

#!/usr/bin/python

'''
SMTP enumeration with Python socket
'''

import sys
import socket

def enum(target, user):
        try:
                s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s.connect((target, 25))
                s.recv(1024)
                s.sendall("VRFY " + user + " \r\n")
                result = s.recv(2048)
                if "252" in result:
                        print result
                else:
                        pass
                s.close()
        except socket.error:
                pass

def main():
        try:
                target = raw_input("Input your target: ")
```

```
GNU nano 3.2

root
Alice
msfadmin
Bob
nooby
```

```
root@kali:~# python smtpEnum.py
Input your target: 192.168.1.137
252 2.0.0 root

252 2.0.0 msfadmin
```

```
def main():
        try:
                target = raw_input("Input your target: ")
                data = open("usernames.txt").read()
                users = data.split("\n")
                for user in users:
                        enum(target, user)
        except KeyboardInterrupt:
                print "You pressed Ctrl+C"
                sys.exit()


if __name__ == "__main__":
        main()
```

# Enumeration

## NSF Enumeration

Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984. It allows a user on a client computer to access files over a computer network as if they were on locally-mounted storage.

Both Portmapper and RPCbind run on TCP port 111. RPCbind maps RPC services to the ports on which they listen. RPC processes notify rpcbind when they start, registering the ports they are listening on and the RPC program numbers they expect to serve.

The client system then contacts rpcbind on the server with a particular RPC program number. The rpcbind service redirects the client to the proper port number (often TCP port 2049).

```
kali@kali:~$ nmap -v -p 111 192.168.1.1-254
```

# Enumeration

## NSF Enumeration

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/nfs*
kali@kali:~$ nmap -p 111 --script nfs* 192.168.1.3
PORT    STATE SERVICE
111/tcp open  rpcbind
| nfs-showmount:
|_  / *
MAC Address: 00:0C:29:5A:3F:6A (VMware)

kali@kali:~$ sudo apt install nfs-common
kali@kali:~$ mkdir /home/test
kali@kali:~$ mount 192.168.1.3:/ /home/test/
```

# Enumeration

## Exercise

1. Retry NFS Enumeration Process on Metasploitable 2

# Enumeration

## SMB Enumeration

Windows supports file and printer sharing traffic by using the Server Message Block (SMB) protocol directly hosted on TCP.

The NetBIOS service listens on TCP port 139 as well as several UDP ports. It should be noted that SMB (TCP port 445) and NetBIOS are two separate protocols. NetBIOS is an independent session layer protocol and service that allows computers on a local network to communicate with each other. While modern implementations of SMB can work without NetBIOS, NetBIOS over TCP (NBT) is required for backward compatibility and is often enabled together. For this reason, the enumeration of these two services often goes hand-in-hand.

https://support.microsoft.com/en-us/help/204279/direct-hosting-of-smb-over-tcp-ip

```
kali@kali:~$  nmap -v -p 139,445 -oG smb.txt 192.168.1.1-254
```

# Enumeration

## Nmap SMB NSE Scripts

Nmap contains many useful NSE scripts that can be used to discover and enumerate SMB services. These scripts can be found in the `/usr/share/nmap/scripts` directory.

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/smb*
kali@kali:~$ nmap -v -p 139, 445 --script=smb-os-discovery
10.11.1.227
```

# Enumeration

## SNMP

SNMP is based on UDP, a simple, stateless protocol, and is therefore susceptible to IP spoofing and replay attacks. In addition, the commonly used SNMP protocols 1, 2, and 2c offer no traffic encryption, meaning that SNMP information and credentials can be easily intercepted over a local network. Traditional SNMP protocols also have weak authentication schemes and are commonly left configured with default public and private community strings.

The SNMP Management Information Base (MIB) is a database containing information usually related to network management. The database is organized like a tree, where branches represent different organizations or network functions. The leaves of the tree (final endpoints) correspond to specific variable values that can then be accessed, and probed, by an external user. The IBM Knowledge Center contains a wealth of information about the MIB tree.

# Enumeration

## The SNMP MIB Tree

The following MIB values correspond to specific Microsoft Windows SNMP parameters and contains much more than network-based information.

| Value | Description |
|---|---|
| 1.3.6.1.2.1.25.1.6.0 | System Processes |
| 1.3.6.1.2.1.25.4.2.1.2 | Running Programs |
| 1.3.6.1.2.1.25.4.2.1.4 | Processes Path |
| 1.3.6.1.2.1.25.2.3.1.4 | Storage Units |
| 1.3.6.1.2.1.25.6.3.1.2 | Software Name |
| 1.3.6.1.4.1.77.1.2.25 | User Accounts |
| 1.3.6.1.2.1.6.13.1.3 | TCP Local Ports |

# Enumeration

## SNMP Enumeration

To scan for open SNMP ports, we can run nmap as shown in the example that follows. The `-sU` option is used to perform UDP scanning and the `--open` option is used to limit the output to only display open ports.

```
kali@kali:~$ sudo nmap -sU --open -p 161 192.168.1.1-254 -oG open-snmp.txt
```

# Enumeration

## Windows SNMP Enumeration

Using some of the MIB values provided, we can attempt to enumerate their corresponding values. This command enumerates the entire MIB tree using the `-c` option to specify the community string as "public", and `-v` to specify the SNMP version number as well as the `-t 10` to increase the timeout period to 10 seconds.

```
kali@kali:~$ snmpwalk -c public -v1 -t 10 192.168.1.4
kali@kali:~$ snmpwalk -c public -v1 192.168.1.4 1.3.6.1.4.1.77.1.2.25
```

# Table of Content

# TBD

## TBD

TBD

```
kali@kali:~$ TBD
```