

Introduction

In this lab, a .pcap file was provided in which the tools Wireshark, Snort, and NetworkMiner can be used to perform intrusion detection on a LAN. Along with this two different events logs are provided in which event viewers and scripts can be run to analyze the files.

Objective

The objective of this lab is to detect any sort of intrusion in accordance with the basic intrusion detection framework on the local network depicted in the packet capture file, along with identifying the type and extent of the attack with TTP information included.

Report

For this lab it is wise to start analysis with a general view of the system, much like the general practice of intrusion detection and investigation, which in this case the system is presented to us in a packet capture file. A general idea of traffic can best be surmised by analyzing basic capture statistics, such as session length and number of packets sent in this time. Provided in figure 1 is this basic information as presented by the statistics panel in Wireshark. The elapsed time of the capture is about four and a quarter minutes, and with a total of 2364 packets, meaning that about 9 packets are sent per second on average. This infers a moderate internal network, supported by the fact that the total amount of bytes is 1921245, which when divided by the time the capture file was taken over comes to about 7623, and further comes to 813 when divided by 9.3, where 813 is average packet size in bytes (*Ethernet statistics*). With the OS of the host of the capture being Win7SP1 and with the fact that there are few endpoints in the capture, we see that this rate of packets is abnormal given what is known, so DDoSing or some other attack that uses rapid segment transfer is a first point of investigation. As seen in figure 1, this information takes averages over all types of packets, so to get an even clearer image of normal and expected traffic for this network, we must investigate the types of protocols shown in the packet capture.

Time

First packet:

2017-07-08 14:09:32

Last packet:

2017-07-08 14:13:45

Elapsed:

00:04:12

Capture

Hardware:

Unknown

OS:

64-bit Windows 7 Service Pack 1, build 7601

Application:

Dumpcap 1.10.3 (SVN Rev 53022 from /trunk-1.10)

Interfaces

Interface

\Device\NPF_{AA48AEC9-CEF5-43EC-B3F1-25D52CAE7321}

Dropped packets

Unknown

Capture filter

none

Link type

Ethernet

Packet size limit (snaplen)

65535 bytes

Statistics

Measurement

Packets

Time span, s

Average pps

Average packet size, B

Bytes

Average bytes/s

Average bits/s

Captured

2364

252.934

9.3

813

1921245

7595

60 k

Displayed

2364 (100.0%)

252.934

9.3

813

1921245 (100.0%)

7595

60 k

Marked

—

—

—

—

0

—

—

Figure 1 – Capture file statistics

Wireshark allows sorting by protocol, so when analyzing the network the presence of certain protocols informs the structure of the network. The presence of ARP, DNS, LLMNR, TCP, UDP, DHCP, NBNS, and BROWSER protocols infers web connections through a router and the presence of a networked host (*Types of network protocols explained with functions*). What is interesting is that DNS, LLMNR, and NBNS in essence perform the same functions, but a device might use one out of the three depending on fallback in resolving hostnames, telling us that host resolution is a main priority in the network configurations (Miller, 2021). ESP infers a tunnel or VPN connection, ICMP infers pinging commands and network device management, IGMP infers multicasting, showing that the network requires outside commands and remote means of managing and connecting to the network (*What is IGMP? | internet group management protocol | cloudflare*). DCERPC, SPOOLSS, SRVSVS, SMB infers access to shared files and printer devices on the network with the capability for remote command and code execution (Goikhman, *SMB::DCERPC*). SSDP allows for the discovery of new devices, and in conjunction with the remote execution protocols, printer device protocols, and network management protocols, we see that this network has automated processes for adding and using printers and other such devices across the network along with remote management of the network and devices (*SSDP, Wireshark*). There is one more protocol listed, being RK512, however all 3 of these packets are malformed in some way, as seen in figure 2. Even so, RK512 is a protocol that is used to communicate with PLC automats of Seimans Sematic, and with a combination of old legacy and modern protocols, so with just the information found just from the presence of these protocols, an IT network that is directly related to an OT network dealing in some way with printers (*Flexi soft – RK512 - sick germany*). However, to gain context as to the protocols' use, the overall connections and dynamic of the network depicted in the capture must be investigated.

160	8.116791	192.168.134.129	192.168.134.132	RK512	1514 Continuous Data[Malformed Packet]
243	8.118010	192.168.134.129	192.168.134.132	RK512	1514 Unknown
381	8.119347	192.168.134.129	192.168.134.132	RK512	1514 Continuous Data
31	7.873039	192.168.134.129	192.168.134.132	SMB	154 Negotiate Protocol Request
32	7.873401	192.168.134.132	192.168.134.129	SMB	155 Negotiate Protocol Response
34	7.875834	192.168.134.129	192.168.134.132	SMB	247 Session Setup AndX Request, NTLMSSP_NEGOTIATE
35	7.876218	192.168.134.132	192.168.134.129	SMB	417 Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_MORE_PROCESSING_REQUIRED

> Frame 243: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF...	0000	00 0c 29 7d f3 78 00 0c	29 3f 20 31 08 00 45 00	..}}x.. }? 1:E
> Ethernet II, Src: VMware_3f:20:31 (00:0c:29:3f:20:31), Dst: VMware_7d:f3:78 (00:0c:29:7d:f3:78)	0010	05 dc 76 af 40 00 40 06	30 16 c0 a8 86 81 c0 a8	..v@ 0
> Internet Protocol Version 4, Src: 192.168.134.129, Dst: 192.168.134.132	0020	86 84 7a e2 04 49 60 81	46 b1 39 08 a0 a5 50 10	..t...F 9...p.
> Transmission Control Protocol, Src Port: 29922, Dst Port: 1241, Seq: 147949, Ack: 1, Len: 1460	0030	39 08 0b 9a 00 00 33 c7	33 ce 8b 74 24 1c 8b fa	9.....3: 3:~\$...
> [72 Reassembled TCP Segments (104934 bytes): #160(1446), #161(1460), #162(1460), #164(1460), #165(1460)]	0040	23 54 24 48 f7 d7 23 7c	24 50 8b de 23 74 24 4c	#TSH-# SP-~#tSL
> SICK RK512	0050	33 fa f7 d3 23 5c 24 54	33 de 03 c7 13 cb 03 04	3~#U\$T 3~.....
Reply Header: 0x00000000	0060	24 1c 01 00 00 8b 74 24	30 13 8c 24 20 01 00 00	\$.....t\$ 0..\$...
Data Block Type: Continuous Data (0)	0070	03 44 24 38 8b fe 13 4c	24 3c 05 b1 96 16 3b 81	..0\$8...L \$c.....
Continuous Data	0080	d1 fe b1 de 80 01 44 24	20 89 4c 24 14 89 44 240\$..L\$..0\$
> [Malformed Packet: RK512]	0090	10 11 4c 24 24 8b 4c 24	34 8b c6 8b d9 0f ac df	..L\$S..L\$ 4.....
> SICK RK512	00a0	1c c1 eb 1c 1e 04 0b c3	33 db 8b 0f 0f ad f5~3.....
Reply Header: 0x1cc3cccc	00b0	1e 0b dd 33 c3 8b 5c 24	30 33 d2 0b d7 c1 e6 1e3~L\$ 03.....
Data Block Type: Unknown (52428)	00c0	8b f9 c1 ef 02 0b fe 8b	e9 0f ad dd 19 33 d7 c1~3.....
Data (8 bytes)	00d0	e3 19 8b f1 c1 ee 07 0b	f3 33 ff 0b fd 8c 6c 24~3.....L\$
	00e0	44 33 c7 8b 7c 24 30 33	d6 8b 74 24 40 33 e9 23	03~L\$03 ~t\$03~#
	00f0	6c 24 2c 8b de 33 df 23	5c 24 2b 23 f7 8b 7c 24	L\$~3~# L\$03~#
	0100	44 23 f9 8b 4c 24 60 33	de 33 ef 03 d3 13 c5 03	0e~L\$T 3~3.....
	0110	54 24 10 13 44 24 14 83	c1 60 89 54 24 38 89 44	T\$~0\$~ ..~T\$0-D
	0120	24 3c e8 17 ac ff ff 89	84 24 24 01 00 00 8b 74	\$c.....~\$5...t
	0130	24 24 8b de 89 54 24 28	01 00 00 8b 54 24 20 33	\$S~\$C ..~T\$ 3
	0140	c9 8b c6 8b fa 0f ad fb	17 0b cb c1 e7 17 33 db~3.....
	0150	c1 e8 09 0b c7 8b ea 0f	ac f5 12 0b dd 8b 6c 24~L\$
	0160	24 24 33 c3 c1 ee 12 8b	fa c1 e7 0e 0b fe 33 c7 33	\$3~.....~3~3
	0170	ff 8b da 0f ac eb 0e 0b	fb 33 c7 c1 ed 0e 8b f2~3.....
	0180	c1 e6 12 0b f5 8b 6c 24	18 33 ce 8b 74 24 24 23~L\$ ~3~t\$5#
	0190	ea 8b fa 8b 54 24 1c 23	d6 f7 d7 23 7c 24 48 8bT\$~# ~# SH~

Figure 2: malformed RK512 packet

Looking at the I/O graph on Wireshark gives the first indicator that there is anomalous traffic. There is a large spike of TCP traffic at second 8 as seen in figure 3, where many TCP segments are sent with the max size of an ethernet frame, being 1514 bytes. The vast majority are Ack statements sent from the IP 192.168.134.129 to IP 192.168.134.132 with the destination port designation of 1241, meaning that the .129 host is communicating over Nessus to the .132 host with many full tcp packets, possibly scanning for vulnerabilities as is the function of Nessus. As Nessus sends tons of ACK packets with payloads (sending scripts, requests, etc.) where it tries to overwhelm the server with small, rapid probes to enumerate vulnerabilities quickly, we can further prove that Nessus is being used to scan for vulnerabilities on the .132 machine (Mallick, 2025). In the midst of these packet spikes are the rk512 packets, but seeing that, as seen in figure 3, the message that the PDUs were assembled in packet 243, we find that one of the rk512 packets being 243 is likely the cause of errors in the reassembly of the tcp flags, where Wireshark is unable to identify specifics of the stream and relays an error for continuous data. This is the case for all three packets, so it is likely that there is no OT factor on the network, or at least, this aspect is the cause of the scanning we see in the traffic spikes in the packet capture.

231	8.118006	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=134809 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
232	8.118007	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=136269 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
233	8.118007	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=137729 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
234	8.118007	192.168.134.132	192.168.134.129	TCP	60 1241 → 29922 [ACK] Seq=1 Ack=123129 Win=19144 Len=0
235	8.118007	192.168.134.132	192.168.134.129	TCP	60 [TCP Window Update] 1241 → 29922 [ACK] Seq=1 Ack=123129 Win=42340 Len=0
236	8.118008	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=139189 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
237	8.118008	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=140649 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
238	8.118008	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=142109 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
239	8.118009	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=143569 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
240	8.118009	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=145029 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
241	8.118010	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=146489 Ack=1 Win=14600 Len=1460 [TCP PDU reassembled in 243]
242	8.118010	192.168.134.132	192.168.134.129	TCP	60 1241 → 29922 [ACK] Seq=1 Ack=139189 Win=26280 Len=0
243	8.118010	192.168.134.129	192.168.134.132	RK512	1514 Unknown
244	8.118010	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=149409 Ack=1 Win=14600 Len=1460
245	8.118011	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=150869 Ack=1 Win=14600 Len=1460
246	8.118011	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=152329 Ack=1 Win=14600 Len=1460
247	8.118012	192.168.134.129	192.168.134.132	TCP	1514 29922 → 1241 [ACK] Seq=153789 Ack=1 Win=14600 Len=1460

There are three main spikes depicted in the graph, and the second spike follows the same rule of thumb and the first discussed earlier, but the last spike, as seen in figure 4, relates to a series of RST flags. This section involves the .129 host sending a syn flag, and the .132 host rejecting and resetting the connection, and with each pair of packets the port numbers increase. This is also indicative of portscanning, as the .129 host is searching for over various ports to

establish a connection with SYN, but the .132 host rejects the connections. While in terms of time this takes place about 4 minutes after the first and largest spike, this is not unusual in in-depth scans, as Nessus allows for comprehensive searches that use multiple tools. Right before this spike was a series of ICMP requests and responses, so one the ping phase of the Nessus scan was completed the program tried to directly connect over common ports that are left open such as port 21 and port 80, which is FTP and HTTP respectively. It seems that before the program sought vulnerabilities, but now it is seeking open ports that the host may be able to be exploited by.

1970	230.295572	192.168.134.129	192.168.134.132	TCP	60 1000 → 44464 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198986 TSecr=0 WS=128
1971	230.295621	192.168.134.132	192.168.134.129	TCP	60 1000 → 44464 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1972	230.295935	192.168.134.129	192.168.134.132	TCP	74 41479 → 1099 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198986 TSecr=0 WS=128
1973	230.296021	192.168.134.132	192.168.134.129	TCP	60 1099 → 41479 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1974	230.296332	192.168.134.129	192.168.134.132	TCP	74 50795 → 1100 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198986 TSecr=0 WS=128
1975	230.296374	192.168.134.132	192.168.134.129	TCP	60 1100 → 50795 [Seq, ACK] Seq=1 Ack=1 Win=0 Len=0
1976	230.296635	192.168.134.129	192.168.134.132	TCP	74 43773 → 1433 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198986 TSecr=0 WS=128
1977	230.296694	192.168.134.132	192.168.134.129	TCP	60 1433 → 43773 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1978	230.297060	192.168.134.129	192.168.134.132	TCP	74 58793 → 921 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198986 TSecr=0 WS=128
1979	230.297117	192.168.134.132	192.168.134.129	TCP	60 921 → 58793 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1980	230.297427	192.168.134.129	192.168.134.132	TCP	74 32859 → 912 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198987 TSecr=0 WS=128
1981	230.297463	192.168.134.132	192.168.134.129	TCP	60 912 → 32859 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1982	230.297807	192.168.134.129	192.168.134.132	TCP	74 43057 → 510 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198987 TSecr=0 WS=128
1983	230.297843	192.168.134.132	192.168.134.129	TCP	60 910 → 43057 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
1984	230.298285	192.168.134.129	192.168.134.132	TCP	74 60928 → 902 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM TSval=1198987 TSecr=0 WS=128
1985	230.298323	192.168.134.132	192.168.134.129	TCP	60 902 → 60928 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 4: Third traffic spike

When providing the capture file to snort, many alerts are generated, and as seen in figure 5 all the errors relate to small rapid TCP segments with 1 alert being for a suspected portscan. This correlates the suspicions found in Wireshark, where, as seen in figure 6, we see that the .129 host is trying to log into various SMB services through default credentials and requests. What is noteworthy is that there are some alerts generated by the .132 host communicating through SMB to the .129 host, and this means that the .132 host is actively responding to the .129 host, enough to generate alerts, showing that the scanning efforts are returning results for the client. The DCERPC, SPOOLSS, and SRVSVS protocols are also part of the request and response pattern to the vulnerability scanning at the beginning of the packet capture, and as they appear first, it is likely that the .132 machine is a printer and is known as such on the network, as otherwise Nessus would perform regular scans against the OS first (Mallick, 2025). Some of the SPOOLSS packets are malformed, meaning that there is likely some other supplementary program that sits between the .126 and .132 machine that allows for the crafting and sending packets during the Nessus scan (*What is the reason for malformed packet error ?*). Looking more into the printing protocols, as seen in figure 6, we see 15 rejections and 1 acceptance by DCERPC, which is alarming and would surely be desirable information to the user of the .129 host, but as Nessus once started runs to completion, there are no more developments or packets relating to a possible DCERPC connection. This acceptance comes after many login attempts and many full TCP packets, so some form of buffer overflow over DCERPC is likely to allow unauthorized acceptance into the SMB service and DCERPC will provide a c prompt as to its service of remote code execution over the \pipe\spoolss address. To better grasp the architecture as a whole, all the various hosts needs to be mapped as to their relation with the 192.168.134 subnet to see if there are printers that could explain this traffic and if the local network is a standard IT network,

as depending on these answers we can figure that most of the traffic and anomalies are a result of the vulnerability scanning.

```

07/08-15:09:40.731553 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.132:445 -> 192.168.134.129:41254
07/08-15:09:45.771675 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.129:29922 -> 192.168.134.132:1241
07/08-15:10:07.690853 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.129:29922 -> 192.168.134.132:1241
07/08-15:10:20.198281 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.129:29922 -> 192.168.134.132:1241
07/08-15:10:56.824496 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.129:51865 -> 192.168.134.132:1000
07/08-15:13:22.663281 [**] [122:1:1] (portscan) TCP Portscan [**] [Classification: Attempted Information Leak] [Priority: 2] {PROTO:255} 192.168.134.129 -> 192.168.134.132
07/08-15:13:26.506722 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.132:445 -> 192.168.134.129:57567
07/08-15:13:26.508439 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.129:57567 -> 192.168.134.132:445
07/08-15:13:26.528204 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.132:445 -> 192.168.134.129:57567
07/08-15:13:26.530763 [**] [129:12:2] Consecutive TCP small segments exceeding threshold [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.134.132:445 -> 192.168.134.129:57567

```

Figure 5: Snort alerts

No.	Time	Source	Destination	Protocol	Length	Info
42	7.901581	192.168.134.129	192.168.134.132	SMB	161	NT Create AndX Request, Path: \SRVSVC
43	7.901704	192.168.134.132	192.168.134.129	SMB	105	NT Create AndX Response, FID: 0x0000, Error: STATUS_ACCESS_DENIED
44	7.904453	192.168.134.129	192.168.134.132	SMB	162	NT Create AndX Request, FID: 0x400a, Path: \BROWSER
45	7.904829	192.168.134.132	192.168.134.129	SMB	205	NT Create AndX Response, FID: 0x400a
46	7.912909	192.168.134.129	192.168.134.132	DCERPC	733	Bind: call_id: 0, Fragment: Single, 13 context items: 0e042bc0-cab3-517d-523f-7cbe71a19dd5 V5.1 (32bit NDR), ce50d6c7-a5f7-9555-4431-671e8d522f90 V1
47	7.913008	192.168.134.132	192.168.134.129	SMB	117	Write AndX Response, FID: 0x400a, 600 bytes
48	7.914816	192.168.134.129	192.168.134.132	SMB	129	Read AndX Request, FID: 0x400a, 467 bytes at offset 562
49	7.914893	192.168.134.132	192.168.134.129	DCERPC	486	Bind_ack: call_id: 0, Fragment: Single, max_send: 4280 max_recv: 4280, 13 results: Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection, Provider rejection
50	7.918113	192.168.134.129	192.168.134.132	SMB	149	Write AndX Request, FID: 0x400a, 16 bytes at offset 702
51	7.918198	192.168.134.132	192.168.134.129	SMB	117	Write AndX Response, FID: 0x400a, 16 bytes
52	7.919972	192.168.134.129	192.168.134.132	SRVSVC	201	NETPRNAMECANONICALIZE request[Long frame (60 bytes)]
53	7.920052	192.168.134.132	192.168.134.129	SMB	117	Write AndX Response, FID: 0x400a, 68 bytes
54	7.921760	192.168.134.129	192.168.134.132	SMB	129	Read AndX Request, FID: 0x400a, 447 bytes at offset 482
55	7.921867	192.168.134.132	192.168.134.129	DCERPC	162	Fault: call_id: 0, Fragment: Single, Ctx: 12, status: nca_s_fault_ndr
56	7.924788	192.168.134.129	192.168.134.132	SMB	162	NT Create AndX Request, FID: 0x400b, Path: \BROWSER
57	7.925109	192.168.134.132	192.168.134.129	SMB	205	NT Create AndX Response, FID: 0x400b
58	7.930307	192.168.134.129	192.168.134.132	SMB	376	Write AndX Request, FID: 0x400b, 243 bytes at offset 45

Frag Length: 356 Auth Length: 0 Call ID: 0 Max Xmit Frag: 4280 Max Recv Frag: 4280 Assoc Group: 0x00006780 Scndry Addr len: 14 Scndry Addr: \PIPE\browser Num results: 13 > Ctx Item[1]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[2]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[3]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[4]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[5]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[6]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[7]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[8]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[9]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[10]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[11]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[12]: Provider rejection, PHIO (Implicit Ar) > Ctx Item[13]: Acceptance, 32bit NDR	0030 fa b1 77 92 00 00 01 01 08 0a 00 00 d8 ae 00 11SMB..... 0040 72 5f 00 00 01 a0 ff 53 4d 42 2e 00 00 00 00 98 0050 01 28 00 00 00 00 00 00 00 00 00 00 00 02 08 0060 2e 0f 01 10 f1 5c 0c ff 00 00 00 00 00 00 00 0070 00 64 01 3c 00 00 00 00 00 00 00 00 00 00 65 0080 01 00 05 00 0c 03 10 00 00 00 00 00 00 00 00 0090 00 00 b8 10 b8 10 80 67 00 00 0e 00 5c 50 49 50g...VIP 00a0 45 5c 62 72 6f 77 73 65 72 00 0d 00 00 02 00E...brow... 00b0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00c0 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 00d0 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00e0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00f0 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 0100 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 0110 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0120 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 0130 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 0140 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0150 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 0160 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 0170 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0180 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 0190 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 01a0 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01b0 00 00 00 00 00 00 02 00 01 00 00 00 00 00 00 01c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01d0 00 00 04 5d 00 0a 0b 1c c9 11 9f a8 00 00 2b 10
---	---

Figure 6: SMB service scanning and login attempts

To find out more about the hosts, we can use Network Miner, which can also help identify the structure of the network such as the router and the related servers. As seen in figure 7, we see that the .129 machine is a Linux machine with Ettercap and that the .132 host is a windows machine and a printer, explaining the presence of the SMB printer services, and that there are 3 other devices on the subnet. The presence of Ettercap on the machine explains how the attacker might have been crafting malformed packets as Ettercap allows packet sniffing and the changing of traffic, so this was likely another tool alongside Nessus that allowed for sniffing all hosted on the Ubuntu OS. Looking at the other devices, as seen in figure 8, the .1 host is a workstation, a server, an NT workstation, and is available to become a browser if needed, showing that the printer device has services and capabilities beyond printing provided by the

Microsoft OS (*Browserprotocol*). The .2 device is a DNS/NBNS resolver and router, and the .158 is another workstation involved in IGMP traffic. One reason we must look outside the two hosts investigated earlier is that we see .2 pinging .158 generating a destination unreachable alert, which warrants further investigation to the other hosts on the network. The .2 device also has the same message returned when communicating with the .132 host, and all of this is likely due to the amount of traffic on the network exceeding normal operational levels, to because the .132 is being communicated with the rapid segments, it is likely designated as unreachable as it cannot respond to the ping, showing that the attack can be seen from indicators on the network as a whole. The .158 device is a workstation that likely utilizes the printer, but due to the scans it is also unable to receive traffic, but to find more information the .129 host should be investigated in how it first interacted with the network and how it utilizes the scans.

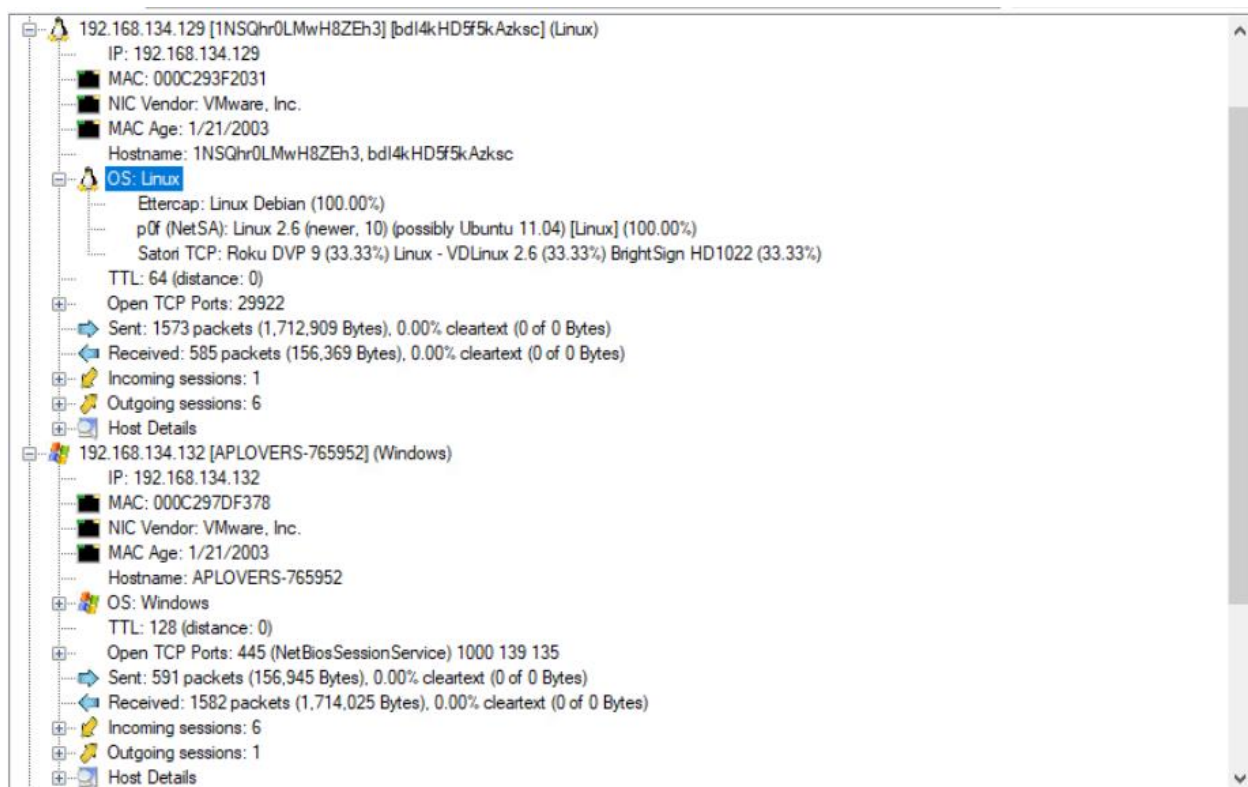


Figure 7: NetworkMiner host information

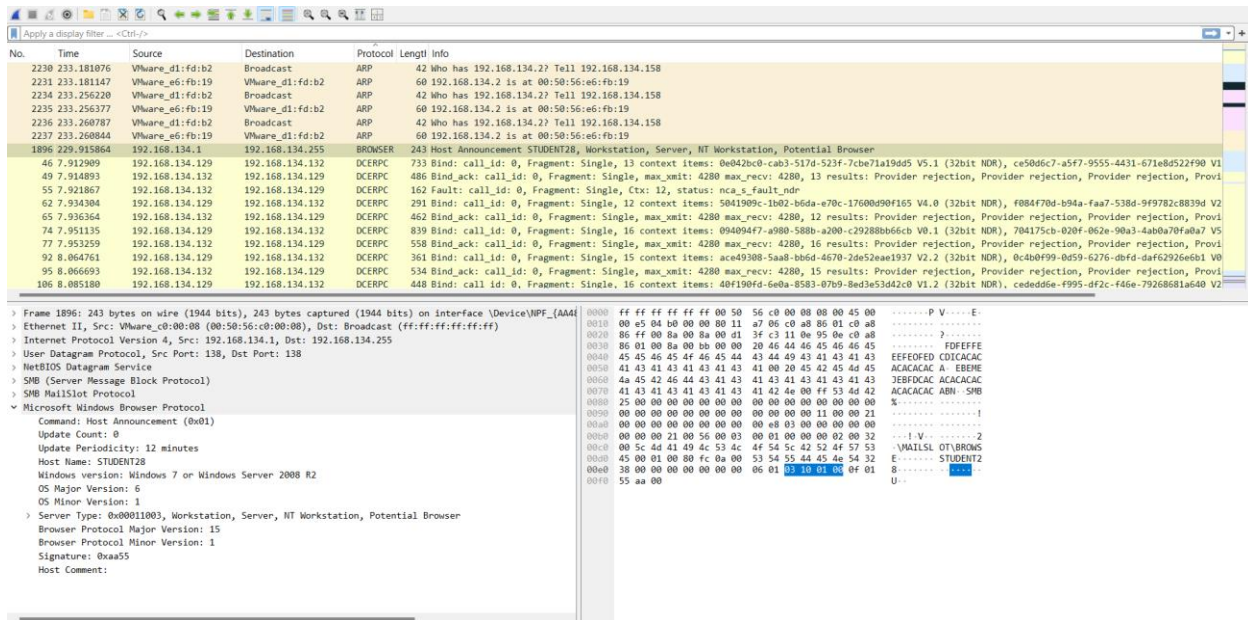
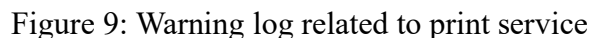


Figure 8: .132 BROWSER announcement

To better grasp the scanning efforts of the .129 host, we can look at the beginning and end of the scanning efforts to see how the host began communication and if the host exploited any vulnerability after the scans. We see that the connection started through a TCP handshake over port 445, or SMB. As far as what can be ascertained by the packet capture, the .129 host was already on the LAN and configured as a regular device. This gives the possibility that vulnerability scans were allowed and condoned by the system manager and owners of the devices, as vulnerability scans can be used for defense just as much as offence. The final sent packet by .129 is over the Nessus port, so with all the information gathered by the pcap, it cannot be ascertained whether the depicted vulnerability scan is being used for malicious purposes, so using the other two sources of information is paramount to investigating the context of the scan. As was found earlier DCERPC allowed a connection but was not followed up on, so it is possible an exploit or a patch was installed, but to figure which is the case the security and application logs provided can provide more insight.

Looking at the first log file being an application log in .evt format proves difficult, but using the date identified in snort we can start our search at the logs generated the day of the attack. When searching through the logs we find a few errors and warnings, but the dates do not match the snort alerts, so it is likely the logs are not synced with the NTP protocol (*NTP - how does it work?* 2022). Looking for SPOOLSS and printer keywords (which is done so because of the attempted enumerations being the first point of scanning) yields logs that the spooler service was stopped and started, and when we look at logs with the source “print”, as seen in figure 9, we find that the message that “this event is not installed on your local computer or the installation is corrupted”. Looking up the DLLs listed in the log as seen in the figure, we see that the cause of this message is likely that the printer driver or print component is missing,



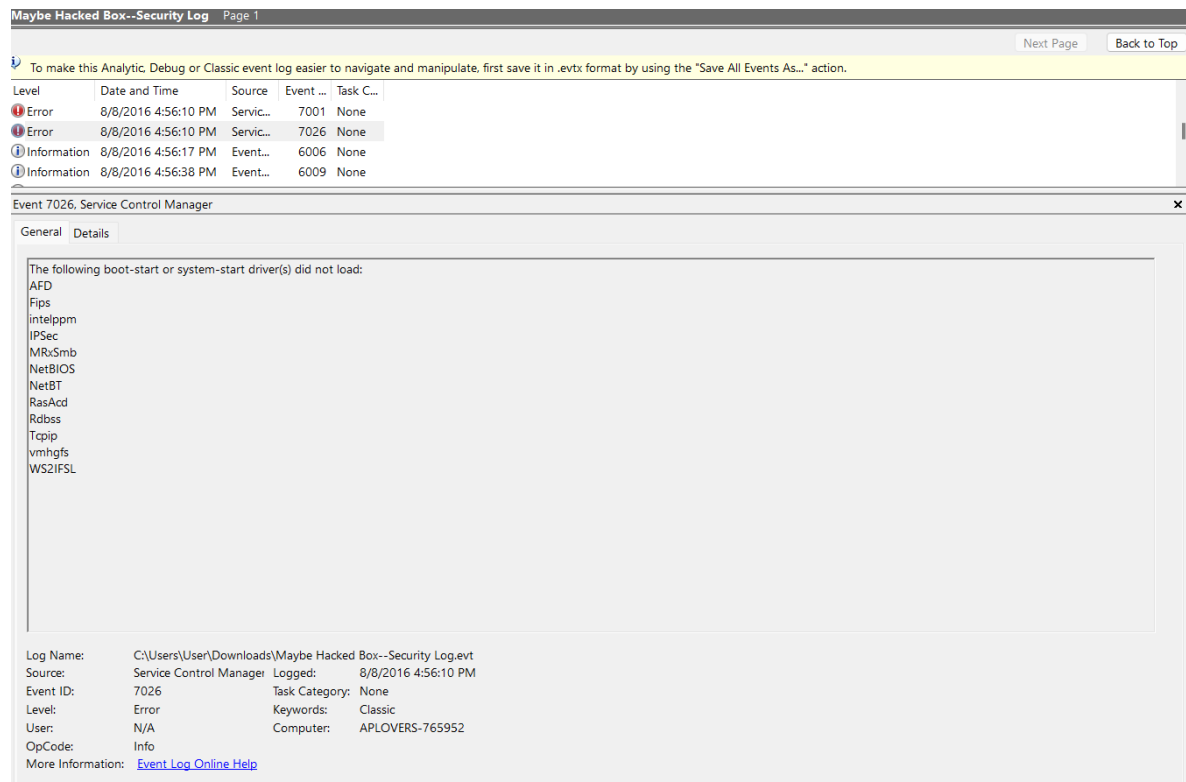


Figure 10: Security services error

In the application log, as seen in figure 11, we find a log when searching for the keyword print that relays information that ThinPrint registered a WMI provider (TPVCGProv) into WMI under Root\ThinPrint, and that either the provider is broken, uninstalled incorrectly, or corrupted (*WMI providers - win32 apps*). The reason for the generated warning is that when Windows tried to query something from that WMI namespace (maybe during boot, or printing), it threw an error (Decker, 2023). Considering that in the application logs these are the only warnings in the file and that the source is WinMgmt, we can figure that with this warning the printer's system monitoring tools will fail (like Performance Monitor and Task Scheduler) along with admin scripts using WMI and remote management via tools like PowerShell, SCCM, or WBEM (*WINMGMT - win32 apps*). The fact that all the logs relating to the printer services are incomplete or corrupted in some way in the event viewer and that they are warnings, it is likely that the printer is compromised on a kernel level, and that the errors generated in the security log are a result of this tampering. In the last few logs in the application file we see that WebFldrs XP has been set up and that the windows update client database was shut down by unexpected means in a dirty shutdown, so while not conclusive, it gives credence to the idea that the attacker is using the printer as a network driver for files and that the updates meant for the printer are not being installed through interruptions. This means that the attacker likely has deep control of the printer and that through programs and the prevention of updates that may remove access or these programs, the printer can be used to manage files on the network and allow lateral movement across the network.

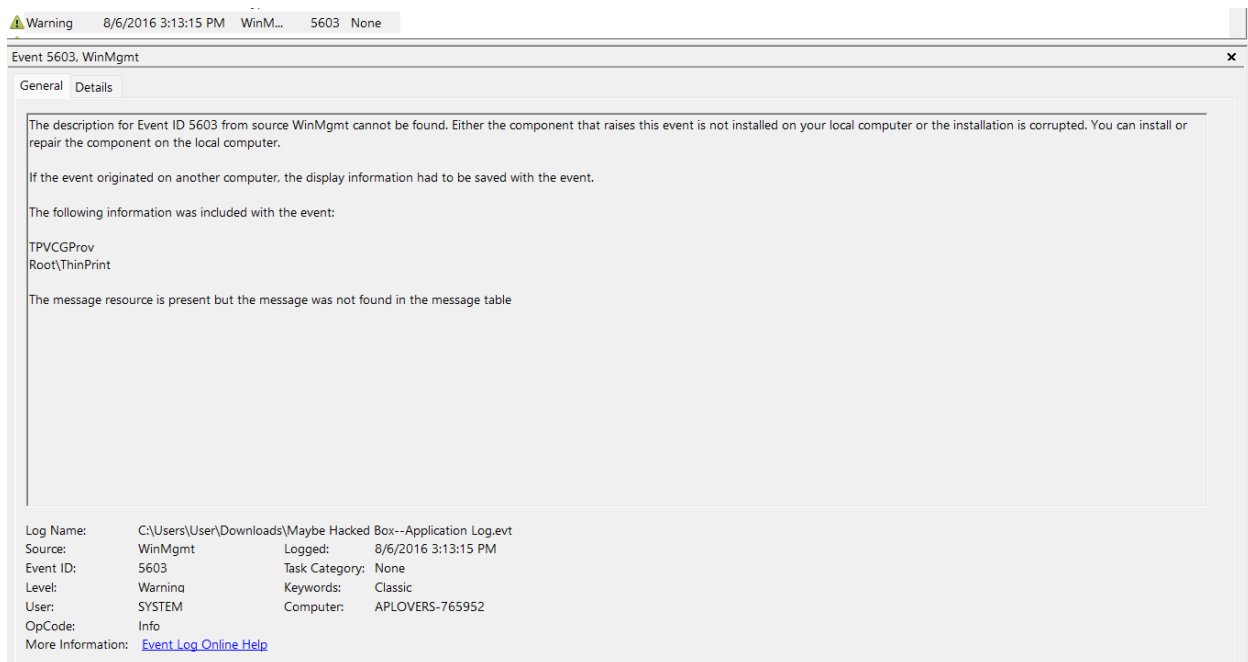


Figure 11: ThinPrint service warning

To conclude a story by summing up the found artifacts, we find the presence of scanning and enumeration by the .129 IP by looking at traffic spikes that include TCP segments relating to the Nessus vulnerability scanner program port and TCP segments relating to attempted connection to common ports. Other scanning efforts can be seen in the SPOOLSS requests and ICMP pings, and along with this we see a connection accepted message in a DCERPC message. Looking at the devices we see a Linux machine with Ettercap trying to connect to a printer running a windows OS and SMB services as to give printer functionality on the network. Looking at the host logs of APLOVER, the printer, we find in the security file that warnings were generated for rapid TCP segments, likely resulting in the RST flags we see in the PCAP, and that there are warnings for security services that were not able to start up. This gives evidence to compromise through the printer services and looking at the application log we see the setting up of WebFldr and the shutdown of the update client. With all of this, we see two stages of an attack, where first comes scanning and then enumeration. Printers are a host such as any other device on the network, and Ettercap and Nessus were able to find a way to exploit insecure printing protocol or configurations, so in this the attacker was able to move laterally through the network, using the printer's SMB and connection statuses to create a moving point for files and himself throughout the network.

Looking at the tactics used by the attacker, we see the tactic T1595.002 being used being "Active Scanning: Vulnerability Scanning" as defined by MITRE (*Active scanning: Vulnerability scanning*). While not able to ascertain a tactic across all the columns of an attack, as seen in figure 12, we can make a diagram that highlights the key points of the adversary, where at which point in could be mapped to threat intelligence data. The most important aspects

of this attack are the scanning and the defense avoidance, being T1562.001 or “Impair Defenses: Disable or Modify Tools”, where, as seen in the security logs, the logging system was stopped along with TCP/IP and IPSec (*Impair defenses: Disable or modify tools*). This modeling is imperative to the investigative process and incorporates intelligence to both better system response and system defense. As seen in figure 13, we can also map out the attack according to the steps to a cyber-attack, where the portion of the attacker is supplanted by the analysis done with the MITRE framework in the tactics, techniques, and procedures mapping.

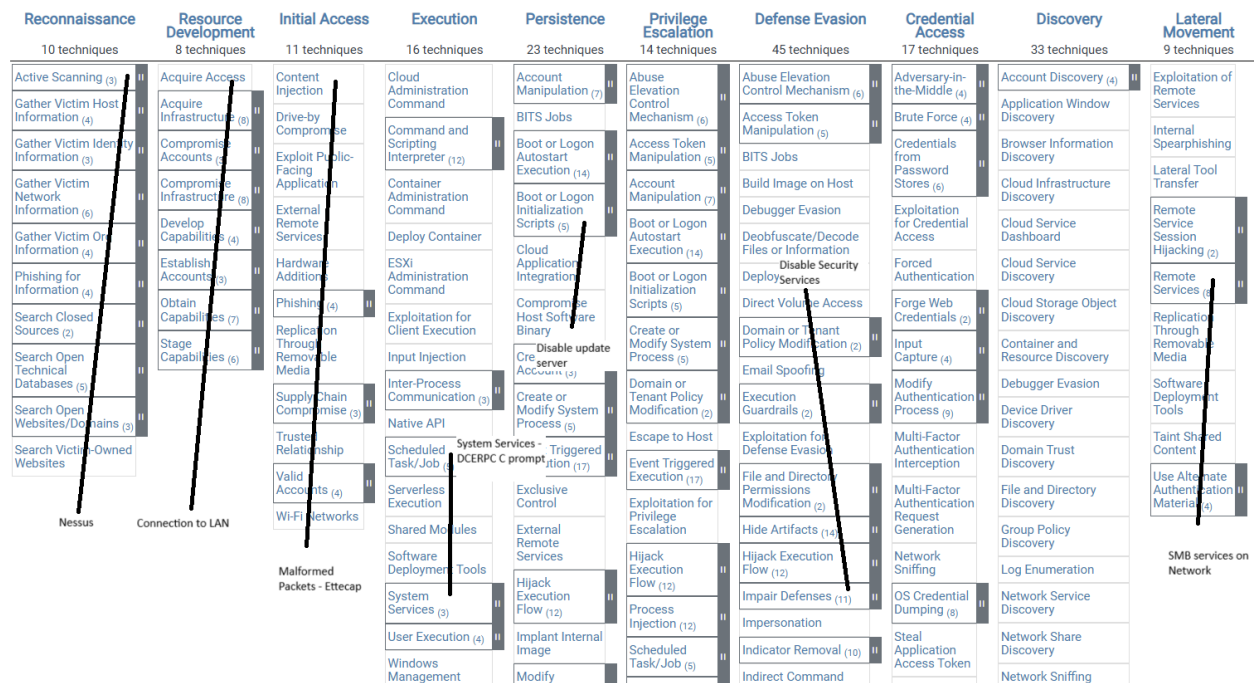


Figure 12 – MITRE attack techniques

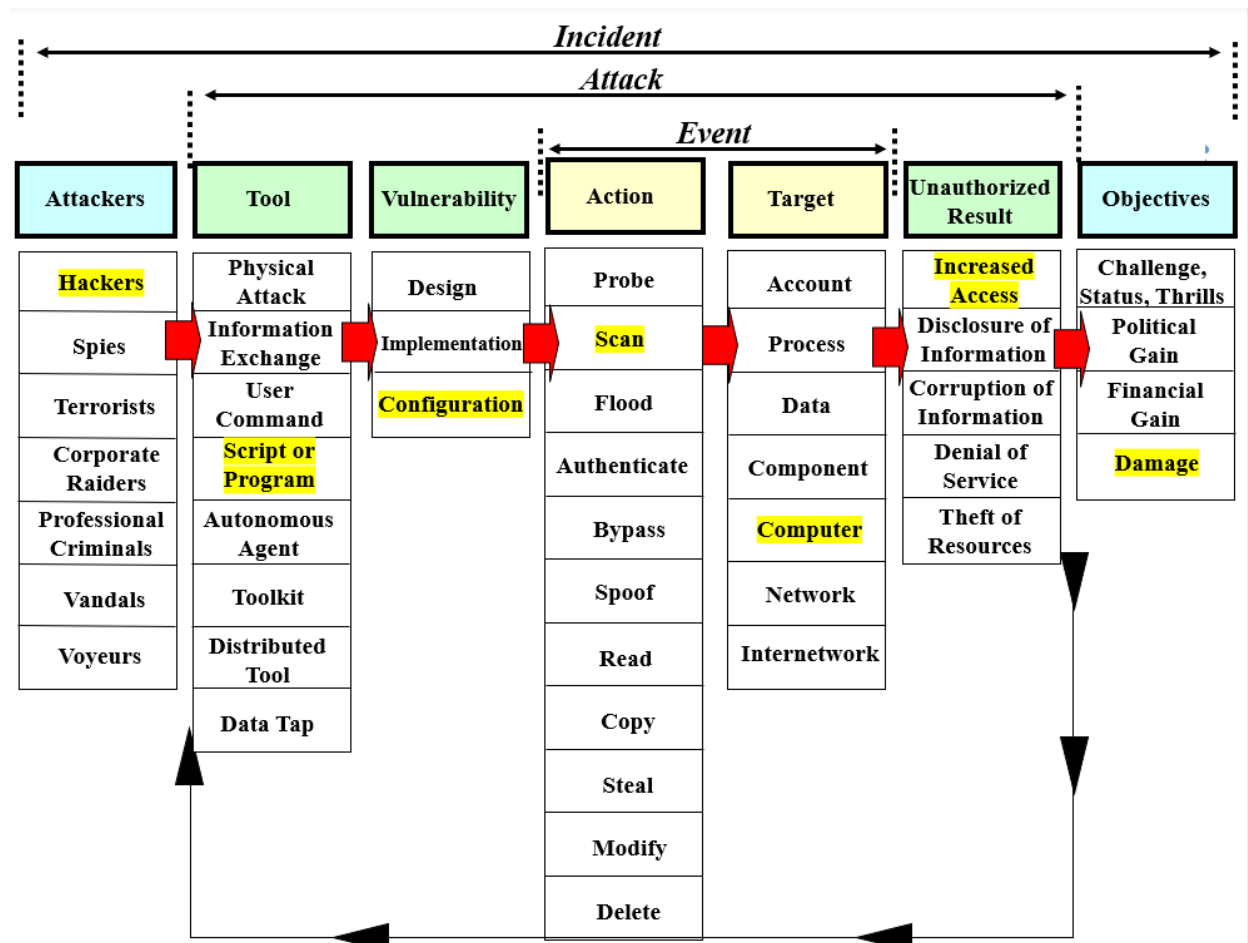


Figure 13 – Attack mapping

Conclusion

In this lab along with a Wireshark pcap was provided, event logs originating from the host system were subject to investigation, and understanding how to extract information from logs in this format is a cornerstone baseline for security analysts. Network logs are often read in combination, but because windows systems provide native logging solutions, many times these sources of information are always available, however this is known to attackers, where an often target and MITRE method is to disrupt and remove the logging process so that one may move undetected. Despite this, in this lab we see that scanning cannot be hidden, at least not if not done so retroactively, and traffic like this is the first sign of a malicious actor, and in this user and intruder profiling can be done to disable connections or perform other means of prevention against an attack when these scanning patterns arise. It also highlights the need for test documentation, as if a vulnerability scan is permitted by system managers, documentation and clear parameters and permissions must be set so that the network can be managed accordingly with the upmost security.

Bibliography

Active scanning: Vulnerability scanning. Active Scanning: Vulnerability Scanning, Sub-technique T1595.002 - Enterprise | MITRE ATT&CK®. (n.d.).

<https://attack.mitre.org/techniques/T1595/002/>

Browserprotocol. BrowserProtocol - Wireshark Wiki. (n.d.).

<https://wiki.wireshark.org/BrowserProtocol>

Decker, E. (2023, May 15). *How to view Windows 10 crash logs and error logs.* TechCult.

<https://techcult.com/how-to-view-windows-10-crash-logs-and-error-logs/>

Ethernet statistics. The Packet Company. (n.d.). <https://packet.company/ethernet-statistics>

Flexi soft – RK512 - sick germany. (n.d.-a).

https://cdn.sick.com/media/docs/2/02/302/online_help_flexi_soft_rk512_en_im0048302.pdf

GmbH, T. (n.d.). *TPPRN.DLL download and fix TPPRN.DLL missing error: Fileinspect.* File Inspect Library. <https://www.fileinspect.com/fileinfo/tpprn-dll/>

Goikhman, M. (n.d.). *SMB::DCERPC.* MetaCPAN. <https://metacpan.org/pod/SMB::DCERPC>

Impair defenses: Disable or modify tools. Impair Defenses: Disable or Modify Tools, Sub-technique T1562.001 - Enterprise | MITRE ATT&CK®. (n.d.).

<https://attack.mitre.org/techniques/T1562/001/>

Mallick, C. (2025, March 10). *Why nessus scanner is the Cybersecurity Solution You Want - Spiceworks.* Spiceworks Inc. <https://www.spiceworks.com/it-security/data-security/articles/what-is-nessus-scanner/>

Miller, M. (2021, March 22). *Vulnerability walkthrough - NBNS and LLMNR spoofing.* Triaxiom Security. <https://www.triaxiomsecurity.com/vulnerability-walkthrough-nbns-and-llmnr-spoofing/#:~:text=What%20are%20NBNS%20and%20LLMNR%3F%20Both%20NetBIOS%20Name,through%20the%20organizational%20DNS%20%28Domain%20Name%20Server%29%20server.>

NTP - how does it work?. NTP. (2022, June 27). <https://www.ntp.org/ntpfaq/ntp-s-algo/>

SSDP. SSDP - Wireshark Wiki. (n.d.). <https://wiki.wireshark.org/SSDP>

Types of network protocols explained with functions. ComputerNetworkingNotes. (n.d.).

<https://www.computernetworkingnotes.com/networking-tutorials/types-of-network-protocols-explained-with-functions.html>

What is IGMP? | internet group management protocol | cloudflare. (n.d.-b).

<https://www.cloudflare.com/learning/network-layer/what-is-igmp/>

What is the reason for malformed packet error ?. Wireshark Q&A. (n.d.). <https://osqa-ask.wireshark.org/questions/52190/what-is-the-reason-for-malformed-packet-error/>

WINMGMT - win32 apps. Win32 apps | Microsoft Learn. (n.d.-a). <https://learn.microsoft.com/en-us/windows/win32/wmisdk/winmgmt>

WMI providers - win32 apps. Win32 apps | Microsoft Learn. (n.d.-b). <https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-providers>