Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment Version 4.1

Tom Davis
March 16, 2005

SANS Network Security 2004 Annual Conference
Las Vegas, Nevada

Table of Contents

1

2 Executive Summary

As I'm certain you are aware, your University's computer systems house a considerable amount of data that are protected by federal and state regulations, such as the Family Educational Rights and Privacy Act (FERPA), the Health Insurance Portability and Accountability Act (HIPAA), and California's Senate Bill (SB) 1386 (also referred to as the Database Protection Act). These same systems also host institutional and individual intellectual property and research data. Any unauthorized disclosure of data in these categories can have a serious impact on your institution, such as loss of federal funding, criminal sanctions, negative media coverage, reduced enrollment, and an inability to obtain future research grants.

During this security review of your network, I analyzed three sets of logs from April 20th, 21st, and 22nd, 2004. These logs were generated by a network-based Intrusion Detection System (IDS); an IDS allows "real-time reporting of suspicious and malicious system and network activity"[1]. Based on my analysis of these IDS logs, I submit the following three strategic recommendations: establish a Virtual Private Network (VPN) service, enhance network perimeter defense measures, and evaluate additional security software. Tactical recommendations will be made throughout the latter sections of this document.

2.1 Establish a Virtual Private Network Service

In its simplest form, a Virtual Private Network (VPN) is a service that allows geographically dispersed computer systems to connect to one another in a secure fashion. The most common implementation consists of a VPN server that is housed on the University network and compatible VPN clients that reside on desktop and laptop computer systems. These desktop and laptop systems can be directly attached to the University network, or they can be attached to the Internet from remote locations such as hotel rooms, faculty members' homes, and public access wireless kiosks. When a user of a remote client establishes a connection to the VPN server, all of the network communications between the two computer systems are encrypted.

The increasing mobility of faculty and staff computer systems increases the risk that the University's sensitive institutional data will be exposed. By deploying a VPN service and requiring its use, it will afford the data improved security while they are being transmitted across the Internet. A VPN service can also be used to improve the security of the wireless networking infrastructure if you require all wireless network traffic be transmitted through the VPN server. And, just as importantly, a VPN service will allow for a stricter network perimeter defense policy as described below.

2.2 Enhance Network Perimeter Defense Measures

All local University network traffic sent to and from the Internet must pass through networking devices called routers; thus, routers can act as a single choke point where security policies can be enforced before the traffic enters or exits your network. The

[1] Beale et al, p. 2

most common security measure performed by routers is via access control lists (ACL's). These ACL's, also known as filters, can discard network traffic based on criteria such as source or destination IP address, network protocol, and port.

Many applications rely on specific network protocols and ports being allowed through routers in order to function properly. For example, Internet browsing normally occurs through the TCP protocol on port number 80. If we were to use a router ACL to discard outbound port 80 traffic, users would not be able to use their Internet browsers to browse the web. Other less used or more dangerous applications, however, should be discarded by router ACL's.

While these filters have the potential to impact the use of legitimate applications, their impact can be eliminated if users first establish a VPN connection to a VPN server. The VPN connection will "tunnel" all underlying application traffic on its own network port(s), thus bypassing the router filters.

The IDS logs that I reviewed clearly indicate that very few router ACL's are in place on your University's network. This lack of filtering exposes almost all of your computer systems to thousands of malicious attacks per day. Therefore, it is my recommendation that your use of router ACL's be increased significantly. Specific filtering recommendations are provided later in this document.

2.3 Evaluate Additional Security Software

It is critical that a computer's operating system and applications be regularly updated to address any security vulnerabilities that have been discovered in them. It is also important to ensure that these same computers have adequate virus protection. This is especially true in a University environment where (in your case) thousands of student-owned machines are connected to the network; students would much rather spend their money on things other than anti-virus software.

Considering the compromised systems that I observed in the IDS logs, there are a number of unpatched and unprotected systems on your network. I recommend that site licenses be purchased for both patch management and anti-virus products. Patch management software will make the task of deploying important security patches to computer systems on your network much easier. And, by offering free or low-cost anti-virus software to your faculty, staff, and students, you dramatically increase your chances of having adequately protected machines at your University.

2.4 Closing Remarks

Recent high profile computer security incidents at George Mason University, the University of Texas at Austin, and the University of California at Berkeley (among several others) clearly illustrate the importance of computer security in institutions of higher education. It is imperative that you act now to reduce the chance of your University being added to that list. If not, it's only a matter of time.

3 Detailed Analysis

3.1 Logs Analyzed

The following logs were downloaded and analyzed from <http://isc.sans.org/logs/>

Table 1: Log Files Analyzed

Alerts	Scans	Out-Of-Spec
alert.040420.gz	scans.040420.gz	oos_report_040416.gz
alert.040421.gz	scans.040421.gz	oos_report_040417.gz
alert.040422.gz	scans.040422.gz	oos_report_040418.gz

Notice the date stamp used in the Out Of Specification (OOS) file names. I had originally downloaded the OOS files that had the same date stamps as the alert and scan logs. Upon inspection, however, I noticed that the logs that were contained within the downloaded OOS files were not from the correct days. I subsequently downloaded the OOS files listed above since those contained entries from the correct days. To make it easier for my scripts to process these logs, I took the liberty of renaming the OOS files to reflect the actual logs that they contained.

To protect the identity of the institution from which the logs originated, alert and oos log entries were modified such that the beginning two octets of their local IP addresses are MY.NET. Because MY.NET. does not conform to a valid IP address, it is not processed correctly by tools such as SnortSnarf. I substituted 10.64. in place of MY.NET. in the alert and oos log files.

However, the scan file entries were not sanitized with MY.NET. Based on the practical submissions of others, it was clear that the 130.85.0.0/16 addresses in the scan files were the same ones that were obfuscated in the alert and oos files. Therefore, I changed all occurrences of 130.85. IP addresses in the scan files with the same 10.64. IP octets I used in the alert and oos logs.

3.2 Relational Analysis

3.2.1 Network Topology

It is my opinion that the University has deployed its Snort sensors behind their perimeter router (i.e., on the non-Internet facing side). The sensors are either obtaining their traffic from one, perhaps multiple, spanning feeds from their campus switches or are placed between the perimeter router and the campus router. The flow of incoming traffic would then be Internet -> Perimeter Router -> Campus Router -> Campus Switches -> Building Switches -> Data Jacks.

One interesting observation is that there were no alert or scan log entries that had both a source and destination IP address from within the University's address space. Either the Snort rules are all written in such a way as to not record those alerts/scans or the Snort sensors just aren't seeing that traffic. It could also be that the Snort sensors are being fed from a network tap positioned between the perimeter router and the campus router as mentioned above.

3.3 Alerts Analysis

3.3.1 Summary of Alerts

There were 551,968 entries in the three alert files analyzed covering 51 distinct alert classifications. 464,618 (or 84.2%) of those were portscan entries, with the remaining 87,350 (15.8%) being more traditional alerts (see Table 2 below). The data reflected in the portscan alerts will be analyzed in the "Scan Analysis" section on page 19; therefore, they will not be discussed further in this section.

Table 2: Alerts Summary

Alert	# of Alerts
10.64.30.4 activity	30866
High port 65535 tcp - possible Red Worm – traffic	19276
EXPLOIT x86 NOOP	11184
10.64.30.3 activity	9275
SMB Name Wildcard	5985
Tiny Fragments - Possible Hostile Activity	4392
RFB - Possible WinVNC - 010708-1	2360
Null scan!	1730
NMAP TCP ping!	627
Possible trojan server activity	374
SUNRPC highport access!	243
DDOS shaft client to handler	145
High port 65535 udp - possible Red Worm - traffic	106
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	98
TCP SMTP Source Port traffic	97
TCP SRC and DST outside network	77
Incomplete Packet Fragments Discarded	67
FTP passwd attempt	60
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	57
ICMP SRC and DST outside network	43
SMB C access	40
TFTP - Internal UDP connection to external tftp server	31
External RPC call	23
EXPLOIT x86 setgid 0	18

[UMBC NIDS IRC Alert] Possible drone command detected.	17
TFTP - External TCP connection to internal tftp server	16
EXPLOIT x86 setuid 0	16
FTP DoS ftpd globbing	15
NIMDA - Attempt to execute cmd from campus host	15
DDOS mstream client to handler [CVE] [arachNIDS]	13
EXPLOIT NTPDX buffer overflow	9
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	9
[UMBC NIDS] External MiMail alert	8
[UMBC NIDS] Internal MiMail alert	8
IRC evil – running XDCC	7
TFTP - Internal TCP connection to external tftp server	6
Attempted Sun RPC high port access	6
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	6
DDOS mstream handler to client [CVE]	4
EXPLOIT x86 stealth noop	4
connect to 515 from inside	4
SYN-FIN scan!	3
Probable NMAP fingerprint attempt	2
NETBIOS NT NULL session [BUGTRAQ] [CVE] [arachNIDS]	2
External FTP to HelpDesk 10.64.70.49	1
TFTP - External UDP connection to internal tftp server	1
Back Orifice	1
External FTP to HelpDesk 10.64.70.50	1
Traffic from port 53 to port 123	1
External FTP to HelpDesk 10.64.53.29	1

3.3.2 Three Alerts: In-Depth Analysis

3.3.2.1 Alert 1: High port 65535 tcp – possible Red Worm - traffic

Description:

The Red worm, or more commonly known as the Adore worm, "scans the Internet checking Linux hosts to determine whether they are vulnerable to any of the following well-known exploits: LPRng, rpc-statd, wu-ftpd and BIND"[2]. Initially discovered in April 2001, this worm mails out key system information (e.g., IP address, account information, etc) from the compromised host in an attempt to inform the attacker(s) that another machine has fallen victim to their malware. In addition, it contacts a web server to download additional malware, and uses port 65535/tcp for its network interactions. And, true to its worm classification, it attempts to compromise other

[2] Fearnow and Stearnes.

machines by exploiting the same vulnerabilities.

Reason Selected:

The alert was selected because patches for the vulnerabilities exploited by the worm have been available for well over three years prior to the alerts being generated. It would be apparent that adequate system administration is not being performed on hosts that succumb to this worm. In addition, it was chosen because this particular alert, if written as I presume, is prone to a high rate of false positives. The latter aspect is extremely important in that bogus alerts cloud the intrusion analyst's perspective of the network as they have a tendency to draw the analyst's attention away from more critical alerts.

Generated By:

Snort generated these alerts using "Fast Alert" mode. Since I was not given the actual rule that triggered these alerts, I can only assume that it is only checking for a source or destination port of 65535. The alerts were of the form:

```
04/20-13:54:19.309575  [**] High port 65535 tcp - possible Red Worm - traffic [**] 10.64.24.34:80 -
> 66.194.21.200:65535
04/20-13:54:19.349648  [**] High port 65535 tcp - possible Red Worm - traffic [**]
66.194.21.200:65535 -> 10.64.24.34:80
```

Probability Source Address Spoofed:

In order to compromise the system and attempt to exploit other systems, the worm must use TCP. Therefore, the likelihood that the source addresses are spoofed in legitimate alerts is extremely low.

Attack Mechanisms:

As mentioned earlier, the Adore worm attempts to exploit vulnerabilities in LPRng (port 515), rpc-statd (port 111), wu-ftpd (port 21) and BIND (port 53). No end user interaction is required for this worm to be successful as it attacks unpatched and otherwise unprotected services that are listening on the University network.

Correlations:

This particular worm has been in the wild for almost four years; therefore, it has been well documented.

Evidence of Active Targeting:

Snort alerts that are written to trigger only on source or destination port, as this one has been, are prone to false positives. When establishing a network connection, client machines select an indiscriminate source port to bind to. If the client so happened to choose a source port of 65535, it would generate an alert with this particular rule. So, it is necessary that the analyst perform additional verification steps to determine whether the machine indicated in the alert logs has in fact fallen victim to the Adore worm.

For example, how can we be certain the following alerts for 10.64.60.38 aren't the result of it being a web server (port 80) and 211.2.199.82 randomly selecting 65535 as

a source port?

```
04/21-04:23:05.985525 [**] High port 65535 tcp - possible Red Worm - traffic [**]
211.2.199.82:65535 -> 10.64.60.38:80

04/21-04:23:05.985535 [**] High port 65535 tcp - possible Red Worm - traffic [**] 10.64.60.38:80 -
> 211.2.199.82:65535
```

Since the external host apparently initiated this connection to our internal host, and there is no additional alert or scan information incriminating our internal host, I'm left to conclude that the above two alerts are false positives.

Severity:

Criticality

I am assigning a criticality value of 4 for this particular worm since it targets important services, particularly FTP and BIND. FTP in that the FTP servers traditionally house sensitive institutional data, and BIND in that DNS attacks are very dangerous. If one of the University's DNS servers were compromised it would subject the network to DNS spoofing attacks.

Lethality

If successful, the worm will grant the malware root-level access to the system. Therefore, I'm assigning a lethality value of 5.

System Countermeasures

Given the high number of false positives, this analyst is led to assume that most Linux systems have been patched against these particular vulnerabilities. Therefore, I am assigning a system countermeasure value of 3.

Network Countermeasures

Based on my analysis of the scan logs, only port 515 (LPRng) appears to be blocked at the University's edge routers with access control lists (ACL's). All of the other service ports exploited by Adore (i.e., 21, 53, and 111) are not being filtered as I observed many external IP addresses scanning for those ports on our local machines. Therefore, I am assigning a network countermeasures value of 1.

Calculated Severity Value

$(4 + 5) – (3 + 1) = 5$

3.3.2.2 Alert 2: [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC

Description:

Sdbot is a worm that compromises Microsoft Windows systems. The original worm has been highly modified resulting in numerous variants. It compromises systems through a variety of methods: behaving as a Trojan horse which entices a user to initiate it; exploiting vulnerabilities in Microsoft and DameWare software; exploiting vulnerabilities in other malware, or attacking Microsoft Windows shares. Once installed on a system Sdbot acts as an IRC client and will join a channel on a specific IRC server. Thereafter, the Sdbot client can be controlled via this IRC channel where it

can be instructed to wage Denial of Service (DoS) attacks upon other targets on the network and to execute other commands on the compromised host.

Table 3: Sources triggering this attack signature

Source IP	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
10.64.84.186	25	25	1	1
10.64.17.45	14	14	5	5
10.64.112.193	8	8	2	2
10.64.153.195	3	3	2	2
10.64.80.119	2	2	1	1
10.64.112.189	2	2	1	1
10.64.69.210	1	1	1	1
10.64.69.155	1	1	1	1
10.64.43.10	1	1	1	1

Reason Selected:

Sdbot variants can be extremely damaging, not only to the hosts they infect but the network infrastructure itself. If several Sdbot hosts on the University network were instructed to initiate a DoS attack at the same time, they would severely hinder and perhaps deny legitimate network traffic. In addition, Sdbot's key logging traits can be a danger if it is able to obtain user passwords, credit card numbers, and other personally identifiable data. And, certain variants have been known to disable security software such as anti-virus programs, thus opening up the machine to a variety of other attack vectors.

Generated By:

Snort generated these alerts using "Fast Alert" mode. I was not given the actual rule that triggered these alerts, but I estimate that the rule looked for a specific command sent from the IRC server to the Sdbot IRC client instructing it to begin an ICMP, TCP, or UDP DoS attack. In addition, all of the alerts I reviewed had a destination port of 7000, so the alert could have been further tuned to only look at port 7000 traffic. The alerts were of the form:

```
04/20-13:02:43.206211 [**] [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to
IRC [**] 10.64.17.45:1029 -> 164.15.194.17:7000
04/20-13:02:43.915276 [**] [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to
IRC [**] 10.64.17.45:1031 -> 131.234.100.43:7000
```

Probability Source Address Spoofed:

The IRC communications between the Sdbot client and the controlling IRC server are done over TCP which requires an established connection. Therefore, the probability that the source addresses are spoofed in these alerts is extremely low. However,

Sdbot will spoof the source address for traffic it generates when performing a DoS attack.

Attack Mechanisms:

As mentioned above and described in Zone Lab's Virus Information Center, this multi-faceted worm uses a variety of attack mechanisms in its attempt to spread through the Internet. It can be disguised as a legitimate application (i.e., act as a Trojan horse) and distributed to users through normal file distribution methods (e.g., peer-to-peer applications, email, etc.). It can spread itself by brute forcing Microsoft Windows shares. It can exploit vulnerabilities in Microsoft software (see Microsoft Security Bulletins MS01-059, MS03-007, MS03-039, and MS04-011) and DameWare (see DameWare bulletin #2). And, it even exploits vulnerabilities in other malware, such as Bagle, MyDoom, and OptixPro.

Correlations:

Zone Labs' Virus Information Center had by far the best written description I was able to find regarding the attack methods used by Sdbot, though Zone Labs had the worm classified as Rbot (McAfee's alias was [W32/]Sdbot.worm.gen.g). Walter Clarie also analyzed similar alerts in his GCIA practical.

Evidence of Active Targeting:

It is clear from the alert and scan logs that the machines listed in Table 3 above are indeed compromised. For example, 10.64.112.189 was responsible for 713,606 entries in the scan logs and had scanned 530,124 distinct destination IP addresses. Of those scan entries, all but 27 were scanning for destination ports of 135 – most likely looking for other Windows machines vulnerable to the RPC vulnerabilities addressed by the Microsoft Bulletins mentioned in the Attack Mechanisms section above. 10.64.17.45 accounted for 1,179.229 scan log entries across 142,832 target hosts, and it scanned for almost all of the ports normally indicative of an Sdbot infected host. See Table 4 below for a break down of those scan entries.

Table 4: 10.64.17.45 scan entries analyzed

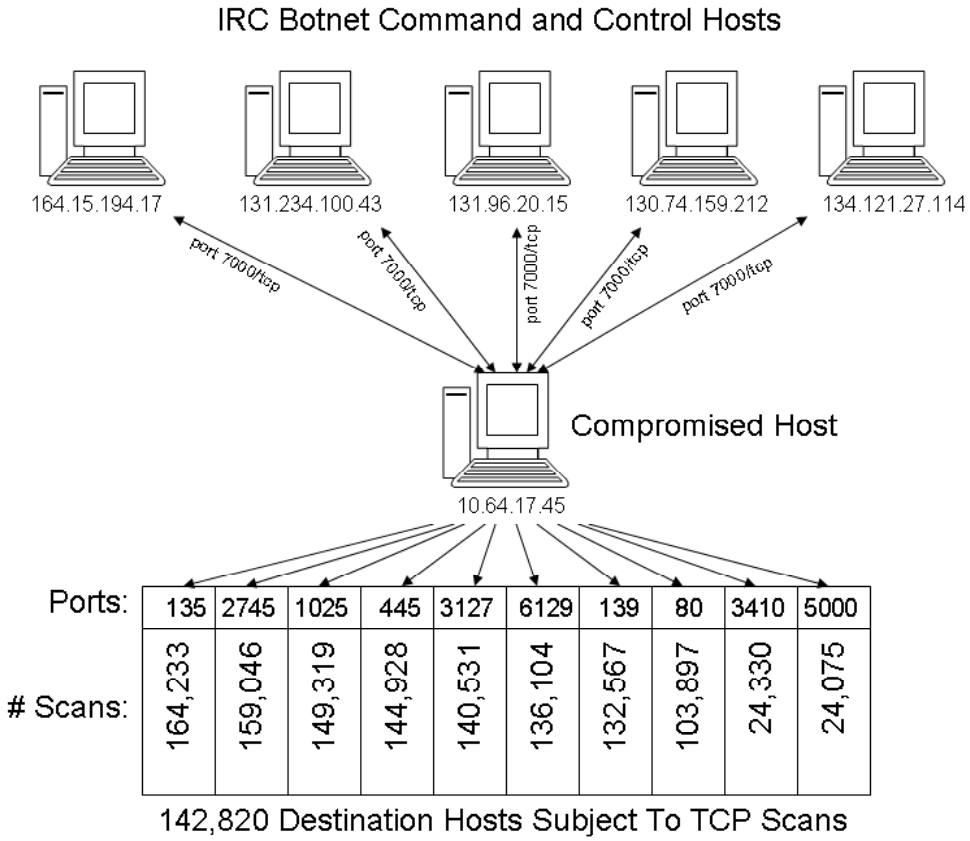
Port	Port Description	# Scan Entries
135	Microsoft RPC	164,233
2745	Bagle	159,046
1025	Microsoft RPC	149,319
445	Microsoft Network Share	144,928
3127	MyDoom	140,531
6129	DameWare	136,104
139	Microsoft Network Share	132,567

80	Web (http)	103,897
3410	Backdoor.OptixPro	24,330
5000	Universal Plug-n-Play	24,075

Link Graph:

The link graph in Figure 1 below clearly reflects the Sdbot compromise of 10.64.17.45. It has joined IRC channels on five different IRC servers and is being remotely controlled by the attacker(s) through those channels. It also reflects the magnitude of scanning that this University host has performed in a relatively short period of time (three days).

Figure 1: 10.64.17.45 Link Graph



I also analyzed the scan logs looking for external IP addresses that had scanned for ports used by Sdbot and found an alarming number of external hosts attacking those ports. See Table 5 below for a break down of those scan entries.

Table 5: External IP scans of internal ports

Port	Port Description	# Scan Entries
6129	DameWare	90,980
2745	Bagle	9,610
135	Microsoft RPC	7,982
445	Microsoft Network Share	1,237
139	Microsoft Network Share	648
3127	MyDoom	183

Let's take a look at a portion of the scan log entries for an external IP address that was scanning for Sdbot hosts on our network.

```
Apr 20 20:49:13 218.155.217.136:2150 -> 10.64.29.155:2745 SYN ******S*
Apr 20 20:49:13 218.155.217.136:2155 -> 10.64.29.155:6129 SYN ******S*
Apr 20 20:49:13 218.155.217.136:2157 -> 10.64.29.155:80 SYN ******S*
Apr 20 20:49:13 218.155.217.136:1523 -> 10.64.60.37:2745 SYN ******S*
Apr 20 20:49:13 218.155.217.136:1528 -> 10.64.60.37:6129 SYN ******S*
Apr 20 20:49:13 218.155.217.136:1530 -> 10.64.60.37:80 SYN ******S*
```

We can easily see that this host is scanning machines on the University's network for Sdbot hosts. Notice that the source port numbers are sequential, but have gaps between them. This indicates that it is also scanning other machines that are most likely not on the University's network.

Take special note of the TCP flags at the end of each line; only the SYN flag is set on each of these scan entries. The University could easily block incoming initial TCP connections (i.e., those with only the SYN flag set) to the ports listed in Table 5 above via router ACL's. And, by making use of Cisco's tcp-initial ACL keyword (consult your router documentation), it will only block incoming connections that have only the SYN bit set. Doing this allows us to slow down the entry of the worm into the University's network. I use the phrase "slow down" as this ACL by itself does not protect the University completely from the worm; for example, an Sdbot compromised laptop that is attached directly to the University's network will not be subject to the router ACL and will attempt to compromise other internal hosts. This type of block has the added benefit in that it does not discard legitimate TCP connections where the network stack has chosen one of those ports randomly as the client port. In that case, the packets would either have both the SYN and ACK flags or just the ACK flag set.

Severity:

Criticality

Without knowing the purpose of each of the Windows machines compromised by this worm, it is difficult to assign a criticality for the individual machines. If it were a staff workstation with institutional data, it would warrant a higher

criticality ranking than a departmental test machine. However, because of Sdbot's capability to perform DoS attacks, its impact on the network could be very dramatic. Therefore, I'm assigning a criticality score of 4.

Lethality

Given the number of vulnerabilities exploited by Sdbot and the fact that it results in a system level compromise of the attacked hosts, I'm assigning a lethality value of 5.

System Countermeasures

These hosts were obviously compromised, thus appropriate system administration practices were not being adhered to. I am assigning a system countermeasures value of 1.

Network Countermeasures

Traffic to the ports exploited by Sdbot is allowed to enter the network, thus I am assigning a network countermeasures value of 1.

Calculated Severity Value

(4 + 5) – (1 + 1) = 7

3.3.2.3 Alert 3: SMB C Access

Description:

Server Message Block (SMB) is a protocol used by Windows hosts to share documents and printers. This alert indicates that a non-University host has attempted a connection to a Windows host residing on the University network using SMB. Furthermore, the attempt has been made to connect to the "C" share which is normally the location of the operating system installation files.

Reason Selected:

The alert could indicate that someone external is attempting to brute force username and password combinations to gain access to the University's host or already has a valid set of authentication credentials for the University host.. Or, a faculty or staff member could be accessing the host from home (e.g., via cable modem or DSL) to access the files stored on a computer at work. The latter is dangerous in that SMB provides no protection for the data that is transmitted across the network. If the employee is accessing sensitive institutional data from home using this method, the data is not encrypted while it is transmitted across the Internet.

In addition, the mere fact that this alert was triggered by external sources further exemplifies the need for stricter perimeter network protection. External access to port 139, among others, should be filtered by the University's border router(s).

Generated By:

Snort generated these alerts using "Fast Alert" mode. Although I'm certain that there have been revisions to the actual rule that triggered the alerts I analyzed, the following current rule is to detect similar activity.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB C$ share
```

```
access"; flow:established,to_server; content:"|00|"; offset:0; depth:1;
content:"|FF|SMBu"; distance:3; within:5; byte_test:1,!&,128,6,relative;
content:"C|24 00|"; nocase; distance:33; content:!"IPC|24 00|"; nocase;
distance:-5; within:5; classtype:protocol-command-decode; sid:533; rev:13;)
```

Probability Source Address Spoofed:

The SMB protocol uses TCP which requires an established connection. Therefore, the probability that the source addresses are spoofed in these alerts is extremely low.

Attack Mechanisms:

This type of attack is successful if the attacker is able to authenticate to the SMB share using brute-forced (i.e., attempts at password guessing), non-existent (i.e., no password), or otherwise obtained (e.g., social engineered) authentication credentials.

Correlations:

Alex Wood analyzed this detect in his GCIA practical entitled “Intrusion Detection: Visualizing Attacks in IDS Data”.

Evidence of Active Targeting:

Let’s take a look at the alerts generated by the external host 82.129.132.122:

```
04/22-09:15:53.295629  [**] SMB C access [**] 82.129.132.122:2837 -> 10.64.190.93:139
04/22-09:15:56.542169  [**] SMB C access [**] 82.129.132.122:2838 -> 10.64.190.95:139
04/22-09:16:23.300853  [**] SMB C access [**] 82.129.132.122:2837 -> 10.64.190.93:139
04/22-09:16:32.506553  [**] SMB C access [**] 82.129.132.122:2839 -> 10.64.190.97:139
04/22-09:16:55.959450  [**] SMB C access [**] 82.129.132.122:2838 -> 10.64.190.95:139
```

And, let’s also review the scan logs generated by that same host:

```
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.87:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.88:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.89:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.94:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.97:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.98:137 UDP
Apr 22 09:15:50 82.129.132.122:1030 -> 10.64.190.99:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.103:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.105:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.106:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.108:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.109:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.111:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.113:137 UDP
Apr 22 09:15:51 82.129.132.122:2837 -> 10.64.190.93:139 SYN ******S*
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.114:137 UDP
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.115:137 UDP
Apr 22 09:15:51 82.129.132.122:2838 -> 10.64.190.95:139 SYN ******S*
Apr 22 09:15:51 82.129.132.122:1030 -> 10.64.190.118:137 UDP
```

This external host is clearly scanning the University’s network looking for NETBIOS shares (port 137/udp) and then coming back later (as indicated by the alert logs) to attempt to connect to those shares.

Severity:

Criticality

Again, without knowing the purpose of each of the Windows machines implicated by these alerts, it is difficult to assign a criticality for the individual machines. If it were a staff workstation with institutional data, it would warrant a higher criticality ranking than a departmental test machine. Therefore, I'm assigning a criticality score of 4.

Lethality

The lethality associated with each alert depends on whether the alert was being generated by a connection to the hosts from an authorized University user or by an attacker. I'm assigning a lethality value of 4.

System Countermeasures

There is no clear indication from the logs that all of the machines were compromised. Therefore, I am assigning a system countermeasures value of 3.

Network Countermeasures

External sourced traffic to the ports used by SMB is allowed to enter the network, thus I am assigning a network countermeasures value of 1.

Calculated Severity Value

$(4 + 4) – (3 + 1) = 4$

3.4 Out-of-Spec Analysis

There were 2,972 out-of-spec log entries for the three days that I analyzed. The top five source and destination IP's can be found in the tables below.

Table 6: Top 5 OOS Source IP's

Source IP	# Alerts	Destinations Involved
68.54.84.49	931	10.64.6.7
66.225.198.20	133	10.64.12.6
128.59.22.253	89	10.64.71.246
195.38.115.167	88	10.64.43.5
141.152.34.103	82	10.64.12.6

Table 7: Top 5 OOS Destination IP's

Destination IP	# Alerts	Originating sources
10.64.6.7	1001	(7 source IPs)
10.64.12.6	944	(135 source IPs)
10.64.24.44	137	(28 source IPs)
10.64.71.246	126	(213.186.36.219, 128.59.22.253
10.64.5.67	117	(5 source IPs)

And, the following table summarizes the TCP flags reflected in these logs.

Table 8: OOS TCP Flags

TCP Flags	# Alerts
12****S*	2,878
********	48
12***R**	24
****P***	11
12UAPRSF	1
12U**RS*	1
12*A**S*	1
12**PR*F	1
12****SF	1
1*U*P*SF	1
1**APRSF	1
1***P*SF	1
*2U*P*SF	1

The characters listed in the TCP flags column represent the individual options that are set within a given TCP communication: "1" and "2" indicate that the two 'reserved" flags are set; U (URG) the urgent flag is set; A (ACK) the acknowledgement flag is set; P (PSH) the push flag is set; R (RST) the reset flag is set; S (SYN) the synchronize flag is set; and, F (FIN) the finish flag is set. An asterisk indicates that a given flag is not

set.

Let's focus for the moment on the two rows that have neither of the "reserved" flags set (i.e., the ******** and ****P*** rows). The former is referred to as a null scan because no TCP flags have been set. It is most often used in reconnaissance scanning to determine the remote machine's operating system as different TCP/IP implementations will respond to this invalid flag combination in different ways. The latter is obviously anomalous given that it is contained in the out-of-spec logs, but it is unclear what its intent was. They all originated from the same IP address (148.63.216.97) and were destined for the same University IP address (10.64.70.254). It is common for routers to be assigned .254 addresses, so this may have been an attempt to fingerprint our router in some fashion. The source IP address did not show up in the alert logs, but the same entries found in the OOS logs are reproduced in the scan logs.

The 'reserved" bits in the TCP flags, as the name implies, were initially reserved for future expansion; any TCP packet that had those two flags set were viewed as out-of-spec. With the introduction of Explicit Congestion Notification (ECN), those two flags are now being used for legitimate purposes. The remaining rows in Table 8 have at least one of the ECN-related flags set, and that could be the reason they are being flagged as out-of-spec. However, those with SYN and FIN both set are definitely out-of-spec as well. The largest majority of the out-of-spec logs have a TCP flag set of 12****S* which most likely indicates a Linux host as the Linux kernel was one of the first to implement support for ECN.

3.5 Scan Analysis

The majority of the scan log analysis can be found throughout the other sections of this document. However, I would like to mention one specific finding.

In order to understand what, if any, ports were being blocked by the University's border router(s), I reviewed the scan logs for all entries that had an external (i.e., non-University owned) source IP address and an internal (i.e., University owned) IP address. I then extracted all of the destination ports and reviewed that list. Any ports not found in that list are possibly being blocked by border router ACL's. Please note that the analysis assumes that the IDS system is inside the University's network. And, it's also important to note that more accurate results could be obtained by reviewing more than just three days worth of scan logs.

It is clear from this analysis that very few of the most dangerous ports (e.g., Microsoft networking, Sun RPC) are being blocked at the border router(s). The only notable port blocks found were 1434/udp (to protect Microsoft SQL Servers against Slammer-like compromises) and 161/udp and 162/udp (SNMP traffic). Again, a more accurate view could be attained by examining the scan log files for additional days.

3.6 Network Statistics

3.6.1 Top Talkers

To get a grasp of the top talkers on the University network, we need to look at both the alert and scan log files. The tables below reflect the top five internal and external machines found in those log files.

3.6.1.1 Top Alert Sources

Table 9: Internal & External Alert Sources

Internal IP	# alerts	External IP	# alerts
10.64.30.4	30,866	134.192.42.11	21,803
10.64.30.3	9,277	68.55.155.26	3,733
10.64.43.8	3,433	131.92.177.18	3,248
10.64.43.13	2,102	64.12.24.34	3,076
10.64.69.232	1,522	64.12.24.35	2,334

Internal Hosts: 10.64.30.4 and 10.64.30.3 are both on this list because of Snort rules "10.64.30.4 activity" and "10.64.30.3 activity" respectively. Those rules appear to alert on all traffic destined to/from those specific IP addresses. 10.64.43.8 and 10.64.43.13 are both on the list because of the "High port 65535 tcp - possible Red Worm – traffic" alert. In fact, that is the only alert recorded for both IP addresses. There are no scan logs to implicate 10.64.43.8 in additional suspect activity. However, 10.64.43.13 has been an active scanner and appears to be compromised with an Sdbot variant. Both warrant further investigation. 10.64.69.232 is another "High port 65535 tcp - possible Red Worm – traffic" alert generator, and there are several TCP and UDP scan entries for it in the scan logs as well. It too warrants further investigation.

External Hosts: 134.192.42.11 is on the list because of "10.64.30.4 activity". Since it has triggered this alert almost seven times more than any other external host, it does appear to be suspicious. 68.55.155.26 is also on the list because of "10.64.30.4 activity". Neither of them have generated any scan entries, however. 131.92.177.18 has generated the most "10.64.30.3 activity" alerts of any host. Again, this warrants further investigation. 64.12.24.34 and 64.12.24.35 both generated a large number of "High port 65535 tcp - possible Red Worm – traffic". Neither generated scan alerts.

3.6.1.2 Top Scan Sources

Table 10: Internal & External Scan Sources

Internal IP	# scans	External IP	# scans
10.64.1.3	2,536,694	213.180.193.68	36,433
10.64.17.45	1,179,178	220.197.192.39	25,926
10.64.1.4	747,956	64.136.199.197	20,487
10.64.112.189	713,579	80.191.163.12	19,073
10.64.81.39	694,880	207.3.145.130	18,380

Internal Hosts: 10.64.1.3 and 10.64.1.4 are clearly the University's DNS servers based on the DNS traffic (port 53) that is evident in the logs. The Snort sensors should be configured to exclude these from the scan entries. 10.64.17.45 and 10.64.112.189 are compromised by an Sdbot variant as they have generated a large quantity of scan logs in its attempt to exploit other machines (numerous port 135, 2745, 1025, 445, 3127, 6129, 139, 80, 3410, and 5000 entries). The scan entries associated with 10.64.81.39 were almost exclusively scans for TCP port 135. This host is most likely looking for Windows machines that are vulnerable to the RPC vulnerabilities mentioned elsewhere in this document.

External Hosts: 213.180.193.68 scanned a wide range of ports against just two internal hosts (10.64.97.65 and 10.64.97.159). Because of this targeted scan, it should be investigated. 220.197.192.39 is compromised with an Sdbot variant has scanned 5,903 distinct internal IP's. 64.136.199.197 scanned for port 80 and 443 (HTTP and HTTPS) against 11,182 distinct internal IP's, most likely searching for vulnerable Apache or Microsoft IIS web servers. 80.191.163.12 scanned exclusively for port 6129 (DameWare) against 13,220 distinct IP's, searching for vulnerable DameWare installations. And, finally, 207.3.145.130 is scanning for vulnerable FTP and TELNET services on ports 21 and 23.

3.6.2 Top Five Targeted Services or Ports

To generate the list of the top five targeted services, I parsed the scan logs looking for scan entries that had an external (i.e., non-University owned) source IP address that was scanning ports on internal (i.e., University owned) IP addresses.

Table 11: Top Five Targeted Ports By External Sources

Destination Port	# of Scans	Service/Application
443	99,446	HTTPS
6129	90,980	DameWare
80	59,603	HTTP
4899	55,722	radmin?
20168	41,014	Lovegate Worm

3.6.3 Most Suspicious External Source Addresses

I selected the following as most suspicious external source addresses as they did not show up prominently in the alerts, but were obviously attempting to connect to several University systems.

3.6.3.1 213.180.193.68

This IP address initiated 36,433 scans covering a wide range of TCP ports against two University hosts (10.64.97.65 and 10.64.97.159). This is most likely an intelligence gathering effort to prepare for a more concerted attack. I provide the ARIN information for the party responsible for this IP address:

```
Search results for: 213.180.193.68

  OrgName:    RIPE Network Coordination Centre
  OrgID:      RIPE
  Address:    P.O. Box 10096
  City:       Amsterdam
  StateProv:
  PostalCode: 1001EB
  Country:    NL

  ReferralServer: whois://whois.ripe.net:43

  NetRange:   213.0.0.0 - 213.255.255.255
  CIDR:       213.0.0.0/8
  NetName:    RIPE-213
  NetHandle:  NET-213-0-0-0-1
  Parent:
  NetType:    Allocated to RIPE NCC
```

```
NameServer: NS-PRI.RIPE.NET
NameServer: NS3.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: AUTH00.NS.UU.NET
NameServer: SEC1.APNIC.NET
NameServer: SEC3.APNIC.NET
NameServer: TINNIE.ARIN.NET
Comment:    These addresses have been further assigned to users in
Comment:    the RIPE NCC region. Contact information can be found in
Comment:    the RIPE database at http://www.ripe.net/whois
RegDate:
Updated:    2004-03-16
```

3.6.3.2 207.3.145.130

This host scanned three specific ports (port 21, port 23, and port 111) against 11,027 distinct IP's. Scans against specific ports are much more concerning as the attacker performing the scan is most likely actively attempting to exploit the services (i.e., FTP, TELNET, and sunrpc). Whereas a scan of all ports on a machine is most likely just attempting to determine what services are listening; though, this sort of scan almost always leads to the former.

```
OrgName:    Savvis
OrgID:      SAVVI-3
Address:    3300 Regency Parkway
City:       Cary
StateProv:  NC
PostalCode: 27511
Country:    US

NetRange:   207.2.128.0 - 207.3.255.255
CIDR:       207.2.128.0/17, 207.3.0.0/16
NetName:    SAVVIS
NetHandle:  NET-207-2-128-0-1
Parent:     NET-207-0-0-0-0
NetType:    Direct Allocation
NameServer: NS01.SAVVIS.NET
NameServer: NS02.SAVVIS.NET
NameServer: NS03.SAVVIS.NET
NameServer: NS04.SAVVIS.NET
NameServer: NS05.SAVVIS.NET
Comment:
RegDate:    1995-12-13
Updated:    2004-11-17

TechHandle: IA3-ORG-ARIN
TechName:   Cable and Wireless US
TechPhone:  +1-800-977-4662
TechEmail:  ip@clp.cw.net

OrgAbuseHandle: ABUSE11-ARIN
OrgAbuseName:   Abuse
OrgAbusePhone:  +1-877-393-7878
OrgAbuseEmail:  abuse@savvis.net
```

```
OrgNOCHandle: NOC99-ARIN
OrgNOCName:   Network Operations Center
OrgNOCPhone:  +1-800-213-5127
OrgNOCEmail:  ipnoc@savvis.net

OrgTechHandle: UIAA-ARIN
OrgTechName:   US IP Address Administration
OrgTechPhone:  +1-888-638-6771
OrgTechEmail:  ipadmin@savvis.net
```

3.6.3.3 206.222.14.209

This external machine scanned 11,957 distinct IP's for TCP port 20168. Most likely the attacker was looking for the backdoor of a Lovegate compromised host.

```
Search results for: 206.222.14.209

  OrgName:    eNET Inc.
  OrgID:      ENET
  Address:    3000 East Dublin Granville Rd.
  City:       Columbus
  StateProv:  OH
  PostalCode: 43231
  Country:    US

  NetRange:   206.222.0.0 - 206.222.31.255
  CIDR:       206.222.0.0/19
  NetName:    EE3-DOM
  NetHandle:  NET-206-222-0-0-1
  Parent:     NET-206-0-0-0-0
  NetType:    Direct Allocation
  NameServer: DNS2.EE.NET
  NameServer: DNS3.EE.NET
  Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
  RegDate:    1996-04-10
  Updated:    1996-04-10

  TechHandle: KK326-ARIN
  TechName:   Kitsmiller, Kelly
  TechPhone:  +1-614-794-5971
  TechEmail:  dns@ee.net
```

3.6.4 Most Suspicious Internal Source Addresses

I selected the following as most suspicious internal source addresses as they did not show up prominently in the alerts, but were obviously being mischievous.

3.6.4.1 10.64.81.39

This University host generated 694,880 scan entries, of which 693,977 (or 99.9%) were of TCP destination port 135 against a wide range of hosts. As mentioned above in Top Scan Sources, this host is probably looking for Windows machines that are vulnerable to the RPC vulnerabilities mentioned elsewhere in this document. This host should be investigated.

3.6.4.2 10.64.53.225

This machine generated 122,802 scan entries but did not generate a single alert. These scans were almost exclusively UDP scans on a variety of ports and IP addresses. However, there was no distinguishable pattern for the scans. This host should be investigated.

3.6.4.3 10.64.97.90

This host generated 104,166 scan entries but did not generate a single alert. These scans were almost exclusively TCP scans of port 80, and encompassed 63,096 distinct destination IP's. This host could be a web crawler or it could be scanning for Apache and/or Microsoft IIS vulnerabilities. Again, this host should be investigated.

3.7 Correlations

Correlations from other sources are referenced in the appropriate sections of this document and can also be found in the List of References at the end of this document.

3.8 Recommendations

I respectfully submit the following recommendations:

- A VPN service should be established. As mentioned in the Executive Summary section on page 3, such a service will have multiple benefits; it will afford the data improved security while they are being transmitted across the Internet, it can be used to improve the security of the wireless networking infrastructure if all wireless network traffic is required to be routed through the VPN server, and it will allow for a stricter network perimeter defense policy.
- The network perimeter is extremely porous. Much stricter router access control lists (ACL's) should be implemented on the University's border routers to block commonly abused ports and protocols. Ports 135, 137, 138, 139, and 445 (Microsoft related), 111 (Sun RPC), 1433 and 1434 (Microsoft SQL Server), and 3306 (MySQL) are the obvious first choices. The University should also consider blocking the standard IRC server ports for inbound connections (e.g., ports 6000, 6660-6669, 7000, 7001, and 8888) to prevent the most prevalent botnet command and control malware from being served on its network.
- A University-wide license should be purchased for anti-virus software and offered to its students, faculty, and staff at a minimal cost.
- Centralized patch management software should be purchased and offered to the University's IT administrators. This will make the process of distributing and deploying patches to systems much more efficient, thus improving the overall security of the network.
- Spy-ware and ad-ware software should be purchased and offered to the University's students, faculty, and staff at a minimal cost.
- The University should increase its use of RFC 1918 (i.e. private IP) address space.
- The Snort rules should be rewritten to be less reliant on port-based rules, which are prone to false positives.
- The Snort rules should be tuned to lessen the alerts generated, thus focused on the more important alerts. For example:
    - 10.64.30.4 and 10.64.30.3 alert activity clutters up alerts
    - 10.64.1.3 and 10.64.1.4 are clearly DNS servers and the Snort sensors should exclude them from scan logs. It is common practice to exclude DNS servers from port scan detection in Snort sensors because they generate so many entries in the scan logs.

4 Analysis Process

4.1 Analysis Platform

The analysis was performed with Gentoo Linux using a 2.4 kernel. The system included 1 GB of RAM, 10GB of disk space, and a 1.6 GHz processor.

4.2 Tools and Techniques

Distillation of the massive amount of log data was made easier by SnortSnarf, snort_stat, scripts from Les Gordon and Ricky Smith's GCIA practicals, assorted unix commands, a shell script I developed called format-gcialogs.sh that pulls all of the others together, and another shell script I developed called snacs.sh that extracts useful bits of information from the scan logs given an IP address.

4.2.1 Description of Tools

SnortSnarf is "a Perl program to take files of alerts from snort, and produce HTML output intended for diagnostic inspection and tracking down problems"[3]. After SnortSnarf processes an alert file, it generates a report that contains the total number of occurrences for each alert as well as the number of distinct source and destination IP addresses for each alert. This HTML-based summary report makes it possible to drill down into the more detailed aspects of the report to get a different view of the data.

snort_stat is Perl program designed to "generate statistical data from every day snort log file"[4]. snort_stat was developed by Yen-Ming Chen and gives us yet another view, albeit more verbose, of the alert data.

format_alerts and sum_alerts are Perl programs provided to the community by Les Gordon in his GCIA practical submission. format_alerts processes a snort alert file, splits each log entry into logical comma separated values (CSV), and generates two output files; one contains the portscan-related alert entries, the other contains the remaining alerts. These remaining alerts can then be fed into sum_alerts to generate several views of the alert data, such as a list of the IP addresses connected to by internal hosts and the list of ports connected to by external hosts.

parse-oos and parsescan are Perl programs provided to the community by Ricky Smith in his GCIA practical submission. parse-oos will process the out-of-specification log entries and put them in a format that is easily processed by SnortSnarf. This makes the interpretation of the data much easier. parsescan parses a scan file and generates two output files containing IP addresses and count pairs; one contains the list of IP addresses used as scanning sources and a count representing the number of times it was found in the input file, the other contains the same information but for IP addresses that were the target of the scan attempts (i.e., the destination IP address

[3] SnortSnarf home page.
[4] snort_stat script comment.

from the entries).

format-gcialogs.sh is a shell script that I developed that extracts and organizes the log files, processes them using the scripts mentioned above, and creates reports that summarize the data to assist in the analysis process. Complete documentation and source code for this script can be found in the appendices.

snacs.sh is a shell script that I developed that extracts useful information from the scan logs given a specific IP address. It displays information such as: the number of scan entries found for the IP; the number of TCP and UDP scan entries for the IP; the total number of distinct destination IP addresses scanned by the IP; the total number of distinct destination TCP and UDP ports scanned by the IP; the number of scans against each distinct destination IP address; and, the number of scans against each distinct TCP and UDP port. Complete documentation and source code for this script can be found in the appendices.

4.2.2 Process Details

One of the first tasks involved in performing the analysis was to simply view the log files provided with the `less` command. This makes it is easy to understand the format of and data contained within the files. I also reviewed several practicals that were submitted by others to obtain some insight into the tools that others have written or used in support of their certification efforts. Next, it was just a matter of putting the tools to use on the log files. If you follow the order in which the functions in format-gcialogs.sh are called, you will have a good idea of the steps that I used during the process. The narrative on format-gcialogs.sh in the appendices also describes the process in detail.

I wrote snacs.sh in an effort to stream-line the process of identifying the Most Suspicious External Source Addresses and the Most Suspicious Internal Source Addresses above. It is especially helpful to compare the scan entries for the individual IP's against the alerts for those same IP's. If you find a host that has several scan entries but no (or few) alert entries, it might be worth further investigation. A possible future enhancement to snacs.sh would be to have it read the IP addresses provided by Ricky Smith's parsescan program, and then generate a comma separated value file containing the values generated by snacs.sh for each IP address.

4.2.3 Problems Encountered

During the analysis process, I encountered the following problems with either the tools or the log files:

- To protect the identity of the institution from which the logs originated, alert and oos log entries have been modified such that the beginning two octets of their local IP addresses are MY.NET. Because MY.NET. does not conform to a valid IP address, it is not processed correctly by tools such as SnortSnarf. This was solved by replacing MY.NET. with a valid IP address octet that did not appear in any of the log files.

- The scan files were not sanitized with MY.NET. Based on the practical submissions of others, it was clear that the 130.85.0.0/16 addresses in the scan files were the same ones that were obfuscated in the alert and oos files. I changed all occurrences of 130.85. IP addresses in the scan files with the same valid IP used in the alert and oos logs.
- The alert files were extremely large because they contained portscan entries in addition to the standard alerts. This made processing the alerts through SnortSnarf difficult. Since the data represented by these portscan entries are also reflected in the scan files, they were removed from the alert files before the alerts were processed.
- The alert files contained a number of entries that that were not formatted correctly. I found this out when I ran an initial test of SnortSnarf against just one of the three alert files as SnortSnarf generated several error messages. After reviewing several of these lines, it was obvious that these alerts had somehow been corrupted when they were added to the file. If they were generated by a Snort sensor writing out in ASCII format (as opposed to a script that processes a unified binary version of the alerts after the fact), then this could be one possible reason for the corruption. Converting alerts to ASCII, especially on a sensor that resides on a busy network (which would be the case on a university network), is extremely taxing on the system. Another possible explanation might be that there were multiple Snort instances running on one machine, all logging to the same alert file. This could generate file locking issues with them all competing for the file. format-gcialogs.sh will generate a report on these mal-formed alert entries so they can be identified and fixed.
- Ricky Smith's parse-oos.pl and parsescan.pl scripts will not accept fully qualified filenames as input. format-gcialogs.sh had to be designed to deal with that.
- The version of parsescan.pl in Ricky Smith's published practical had a slight error in one statement. Specifically, the first "open RESULTS" should open $resultsfile, not $srcresultsfile.
- The site hosting the Snort sensor that generated the alerts had a custom rule that had a comma in its description, and the commas interfered with Les' format_alerts program. format-gcialogs.sh was designed to remove those commas from the alert logs before they were given to format_alerts.

5 Appendices

5.1 format-gcialogs.sh Overview

format-gcialogs.sh was developed mainly as a self-documenting archive of the steps I took to extract important information from the massive amount of log files provided for this practical. However, it turned into a script that I believe will assist other intrusion analysts in processing these same logs.

5.1.1 Crash Course

This section describes just those steps required to get the script to work. Anyone performing the analysis on a Unix-like machine should have no problem using the script as it makes use of simple shell code. It does, however, call programs that have been written in Perl, so you'll need a recent version of Perl installed as well.

1. download log files from <http://isc.sans.org/logs/> and place them in /var/log/gcia
2. download format-gcialogs.sh and place it in /usr/local/bin (or another directory that is included in your PATH)
3. modify GCIALOGS to reflect the directory in which you placed the downloaded log files
4. modify MYNETREPLACEMENT variable in format-gcialogs.sh to reflect the first two IP address octets (e.g., 10.64.) you'd like to be substituted for MY.NET. in the log files
5. review ACTUALNET variable in format-gcialogs.sh to ensure that it still matches the "real" first two octets of the IP address (the IP addresses in the scan logs I processed were not modified to begin with MY.NET.)
6. modify MYDATExx variables in format-gcialogs.sh to match the dates contained in the logs you downloaded from <http://isc.sans.org/logs/>
7. modified SNORTRULES* variables to reflect where your Snort configuration files are located
8. install Perl (if necessary)
9. download programs used by this script (i.e., SnortSnarf, snort_stat, format_alerts, sum_alerts, parse-oos, and parsescan) and place them in /usr/local/bin (or another directory that is included in your PATH)
10. change "2002-" in format_alerts.pl to match the year of your log files
11. change $HOME_NET variable in sum_alerts.pl to match the value you used for MYNETREPLACEMENT in format-gcialogs.sh
12. change the first "open RESULTS" line of code in parsescan.pl to open $resultsfile, not $srcresultsfile
13. initiate "`format-gcialogs -A`"
14. fix mal-formed alerts (not as critical to fix the mal-formed scans) as instructed by script output
15. initiate "`format-gcialogs -B`"
16. consult extracts and reports as instructed by script output

5.1.2 Narrative

This section will describe the various functions that are used within format-gcialogs.sh for the "-A" and "-B" command line switches. A word of warning, though; this script performs several modifications on the logs that you have downloaded. It might be in your best interest to archive the pristine logs you have downloaded before initiating this script.

5.1.2.1 format-gcialogs -A

OrganizeLogs will gunzip each of the logs you downloaded and place them in appropriately named sub-directories (i.e., alerts, oos, and scans) based on their file names. These sub-directories will be created off of the directory that contained the .gz version of the logs (by default, that location is /var/log/gcia). If it processes a file whose name it does not recognize, and thus can't make a determination which sub-directory to move it to, it will generate an error message and continue to process the remaining files.

FixIPs will adjust IP addresses in the alert, oos, and scan logs so the analysis programs used against them can interpret the data correctly. IP addresses within the alert and oos files have been sanitized by replacing the first two octets with MY.NET.. FixIPs will change those to a valid IP address. This replacement octet pair is configurable within the script (by default it uses 10.64.). In addition, the scan logs I chose to download did not have their IP addresses sanitized. Based on the practical submissions of others, it was clear that 130.85.0.0/16 IP addresses in the scan logs were the same ones that were obfuscated in the alert and oos files. Therefore, FixIPs will change all occurrences of 130.85. IP addresses in the scan files with 10.64. so that they match the alert and oos files.

FindBagLogs will search through the alert and scan logs looking for entries that appear to be mal-formed. It does this by examining the start of each entry to ensure that it matches the dates of the logs we should be analyzing. The scan logs have so many entries that ignoring the mal-formed ones will not have a dramatic impact on your analysis results. However, the alert logs should be corrected if possible before proceeding on with the "-B" flag to format-gcialogs. You will find these erroneous log entries in the "malformed" sub-directory off of the scans and alerts directory.

5.1.2.2 format-gcialogs -B

RemoveScansFromAlerts removes portscan entries from the alert logs. This drastically reduces the size of the alert log files and allows them to be processed much more efficiently. The data reflected in the discarded portscan entries can be found in the scan logs themselves.

ReportOnAlerts processes the alert logs with SnortSnarf and snort_stat. The resulting

reports can be found in the reports/snarf and reports/stat sub-directories off of the alerts directory.

LesGordon uses the programs provided by Les Gordon in his practical to process the alert logs. The extracts generated by his programs give us yet another view of the alerts to use in our analysis process. The first step performed in this routine is to remove any commas found in the alert log entries. The site hosting the Snort sensor that generated these logs had a custom rule that had a comma in its description, and these commas interfere with Les' format_alerts program. format_alerts manipulates the logs into a format that is understood by sum_alerts, and then sum_alerts is used to generate reports based off of the data. These reports can be found in the reports/les sub-directory off of the alerts directory.

ReportOnOOS uses Ricky Smith's parse-oos program to format the out-of-spec log files into a format that can be understood by SnortSnarf, and then SnortSnarf is used to generate reports based off of the data. These reports can be found in the reports/snarf sub-directory off of the oos directory.

ReportOnScans uses Ricky Smith's parsescan to parse the scan file and generates two output files containing IP addresses and count pairs. Results are stored directly in the scans directory. Additional unix commands are used to extract entries where the destination IP address is on the network we are analyzing and the source IP address is external. This allows us to determine which destination TCP and UDP destination ports are being accessed by those not on our network. Results are stored in the reports sub-directory off of the scans directory.

5.2 format-gcialogs.sh Source

For brevity, only the initial portion of the script is included below. You can find the entire script here:

<https://itso.iu.edu/staff/tdavis/format-gcialogs.sh>

```
#!/bin/sh
#
# Catch-all script to manipulate and report on data in GCIA logs.
#
# Tom Davis, 01/14/2005
#
# Based on work by:
#  Chris Baker
#    - http://www.giac.org/practical/Chris_Baker_GCIA.zip
#    * idea of placing logs into aptly named directories
#
# Calls scripts developed by:
#  Silicon Defense (http://www.silicondefense.com/)
#    * SnortSnarf
#      - http://www.snort.org/dl/contrib/data_analysis/snortsnarf/
#  Yen-Ming Chen
#    * snort_stat.pl
#      - http://www.snort.org/dl/contrib/data_analysis/snort_stat.pl
#  Les Gordon
#    * format_alerts.pl
#      - http://forum.sans.org/discus/messages/78/7466.html?1055304069
#      NOTE: format_alerts.pl has 2002- hard coded in the script, so you'll
#            want to modify that to match the year of the log records
#            you are analyzing.
#      INFO: commas contained within the alert records will cause
#            format_alerts to have issues because it parses the alert logs
#            using comma as a delimeter.  we "tr" the commas within
#            format-gcialogs.sh (this script) to remove commas from the
#            input file to resolve this issue
#    * sum_alerts.pl
#      - http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc
#      NOTE: You'll want to change $HOME_NET within this script to match
the
#            value you use below for MYNETREPLACEMENT
#  Ricky Smith
#    * parse-oos.pl
#      - http://www.giac.org/practical/GCIA/Ricky_Smith_GCIA.pdf
#    * parsescan.pl
#      - http://www.giac.org/practical/GCIA/Ricky_Smith_GCIA.pdf
#      NOTE: You'll have to modify parsescan.pl as there was a slight error
#            in the version included in Ricky's practical.  Specifically,
#            the first "open RESULTS" should open $resultsfile, not
#            $srcresultsfile
#    INFO: Neither of the above scripts can take fully qualified path names
#          as input.  this script (format-gcialogs.sh) has been coded to
#          compensate for that issue.
#
# NOTE: Place all of the above scripts in /usr/local/bin (or another
```

```
#       directory that is included in your PATH).
#

#
# NOTE: Change the following to match your environment!
#
# location of the GCIA log files you downloaded
GCIALOGS=/var/log/gcia
# make sure this doesn't already appear in the log files
MYNETREPLACEMENT=10.64.
# scan logs weren't sanitized with MY.NET., so actual IP needs to be
# provided here so we can change those to MYNETREPLACEMENT as well
ACTUALNET=130.85.
# list of dates for GCIA log files you've chosen to use
# future enhancement could pull these dates from the actual
# log file names, but I'm too lazy to add it now!  note that
# I'm really only processing three log files, but I put the
# fourth date in here because the log file rotation for
# the third day was performed slightly after midnight and I
# didn't want to lose any relevant logs.
MYDATE1=04/20
MYDATE2=04/21
MYDATE3=04/22
MYDATE4=04/23
MYDATE1b="Apr 20"
MYDATE2b="Apr 21"
MYDATE3b="Apr 22"
MYDATE4b="Apr 23"
# Snort file location (used by SnortSnarf)
SNORTRULESFILE=/etc/snort/snort.conf
SNORTRULESDIR=/etc/snort
.
.
.
```

5.3 snacs.sh Source

You can also download the entire script here:

<https://itso.iu.edu/staff/tdavis/snacs.sh>

```
#!/bin/sh
#
# Generate useful info on scans.
#
# Tom Davis, 03/03/2005
#

#
# NOTE: Change the following to match your environment!
#
# location of the GCIA log files you downloaded
GCIALOGS=/var/log/gcia

# General use variables.
SCANS=$GCIALOGS/scans
SCANSBAD=$SCANS/malformed
SCANSCOMBINEDFILEBASE=scans.combined
SCANSCOMBINEDFILE=$SCANS/$SCANSCOMBINEDFILEBASE
SCANSREPORTS=$SCANS/reports
SCANSRPTINTDESTFILE=$SCANSREPORTS/scans-extsrc-intdest
MAXPORT=50
MAXIP=50
TMPFILE1=/tmp/snacs1
TMPFILE2=/tmp/snacs2

#
# Main shell function
#
main ()
{
  echo " "
  echo "======= Generating reports for scan entries with source IP:
$IPADDRESS"
  echo " "
  REGEXPIPADDRESS=`echo $IPADDRESS | sed -e "s/\./\\\\\./g"`

  # grab all entries that have our IP address, exclude UDP entries,
  # exclude all entries where our IP is in the destination, and
  # only keep the IP:port destination information (removes TCP flags)
  egrep "$REGEXPIPADDRESS:" $SCANSCOMBINEDFILE | egrep -v "UDP" | cut -f2 -
d ">" | sed 's/^ //g' | egrep -v "$REGEXPIPADDRESS" | cut -f1 -d "S" | sed
's/ $//g' > $TMPFILE1
  COUNT=`cat $TMPFILE1 | wc -l`
  echo "*** TCP scan statistics: $COUNT total log entries ***"
  echo " "

  cat $TMPFILE1 | cut -f2 -d ":" | sort > $TMPFILE2
  COUNT=`cat $TMPFILE2 | uniq | wc -l`
```

```
  echo "Destination port frequency counts for $COUNT distinct ports (TCP
scans):"
  if [ $COUNT -gt $MAXPORT ]
  then
    echo "  NOTE: Number of ports is greater than $MAXPORT, listing is
abbreviated"
  fi
  echo " "

  echo "# scans|port"
  echo "---------------"
  if [ $COUNT -gt $MAXPORT ]
  then
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1 | head -n $MAXPORT
  else
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1
  fi
  echo " "

  cat $TMPFILE1 | cut -f1 -d ":" | sort > $TMPFILE2
  COUNT=`cat $TMPFILE2 | uniq | wc -l`

  echo "Destination IP address frequency counts for $COUNT distinct IPs
(TCP scans):"
  if [ $COUNT -gt $MAXIP ]
  then
    echo "  NOTE: Number of IPs is greater than $MAXIP, listing is
abbreviated"
  fi
  echo " "

  echo "  # ips|ip"
  echo "-----------------------"
  if [ $COUNT -gt $MAXIP ]
  then
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1 | head -n $MAXIP
  else
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1
  fi
  echo " "

  # grab all entries that have our IP address, only grab UDP entries,
  # exclude all entries where our IP is in the destination, and
  # only keep the IP:port destination information (removes UDP info)
  egrep "$REGEXPIPADDRESS:" $SCANSCOMBINEDFILE | egrep "UDP" | cut -f2 -d
">" | sed 's/^ //g' | egrep -v "$REGEXPIPADDRESS" | cut -f1 -d "U" | sed
's/ $//g' > $TMPFILE1
  COUNT=`cat $TMPFILE1 | wc -l`
  echo "*** UDP scan statistics: $COUNT total log entries ***"
  echo " "

  cat $TMPFILE1 | cut -f2 -d ":" | sort > $TMPFILE2
  COUNT=`cat $TMPFILE2 | uniq | wc -l`

  echo "Destination port frequency counts for $COUNT distinct ports (UDP
scans):"
```

```
  if [ $COUNT -gt $MAXPORT ]
  then
    echo "  NOTE: Number of ports is greater than $MAXPORT, listing is
abbreviated"
  fi
  echo " "

  echo "# scans|port"
  echo "---------------"
  if [ $COUNT -gt $MAXPORT ]
  then
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1 | head -n $MAXPORT
  else
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1
  fi
  echo " "

  cat $TMPFILE1 | cut -f1 -d ":" | sort > $TMPFILE2
  COUNT=`cat $TMPFILE2 | uniq | wc -l`

  echo "Destination IP address frequency counts for $COUNT distinct IPs
(UDP scans):"
  if [ $COUNT -gt $MAXIP ]
  then
    echo "  NOTE: Number of IPs is greater than $MAXIP, listing is
abbreviated"
  fi
  echo " "

  echo "  # ips|ip"
  echo "-----------------------"
  if [ $COUNT -gt $MAXIP ]
  then
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1 | head -n $MAXIP
  else
    cat $TMPFILE2 | uniq -c | sort -n -r -k 1,1
  fi
  echo " "

  return
}

#
# Provide a little help on the script's use.
#
DisplayUsage()
{
cat << EOF

Usage: $0 -h -i ip_address

-h Display help
-i generate reports based on IP address

Examples:
$0 -h
$0 -i
```

```
EOF

  return
}  # end of DisplayUsage()

#
# Initialize our working environment and call the main function.
#

# Process command line arguments
while getopts i: useropts
do
  case "${useropts}" in
     h) DisplayUsage
        exit ;;
     i) IPADDRESS=${OPTARG} ;;
    \?) DisplayUsage
        exit ;;
  esac
done

# Call the main function
main
```

6 List of References

Baker, Chris GIAC Certified Intrusion Analyst (GCIA) Practical Assignment
<http://www.giac.org/practical/Chris_Baker_GCIA.zip>

Beale, Jay, et al. Snort 2.1 Intrusion Detection 2nd ed. Rockland: Syngress, 2004

Clarie, Walter GIAC Certified Intrusion Analyst (GCIA) Practical Assignment
<http://www.giac.org/certified_professionals/practicals/gcia/0758.php>

DameWare Security Bulletins & Advisories
<http://www.dameware.com/support/security/bulletin.asp?ID=SB2>

Fearnow, Matt, and William Stearnes, Adore Worm
<http://www.sans.org/y2k/adore.htm>

Fearnow, Matt, and William Stearnes, Lion Worm
<http://www.sans.org/y2k/lion.htm>

Gordon, Les GIAC Certified Intrusion Analyst (GCIA) Practical Assignment
<http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc>

Hayes, Brian GIAC Certified Intrusion Analyst (GCIA) Practical Assignment
<http://www.giac.org/certified_professionals/practicals/gcia/0770.php>

McAfee's Security Headquarters, [W32/]Sdbot.worm.gen.g
<http://vil.nai.com/vil/content/v_100454.htm>

Microsoft Security Bulletins
<http://www.microsoft.com/technet/security/bulletin/MS01-059.mspx>
<http://www.microsoft.com/technet/security/bulletin/MS03-007.mspx>
<http://www.microsoft.com/technet/security/bulletin/MS03-039.mspx>
<http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx>

Ramakrishnan, K. and S. Floyd, A Proposal to add Explicit Congestion Notification (ECN) to IP
<http://www.ietf.org/rfc/rfc2481.txt>

Rekhter, Y., et al., Address Allocation for Private Internets
<http://www.ietf.org/rfc/rfc1918.txt>

Smith, Ricky GIAC Certified Intrusion Analyst (GCIA) Practical Assignment
<http://www.giac.org/practical/GCIA/Ricky_Smith_GCIA.pdf>

SnortSnarf, <http://www.snort.org/dl/contrib/data_analysis/snortsnarf/>

snort_stat, <http://www.snort.org/dl/contrib/data_analysis/snort_stat.pl>

State of California. BILL NUMBER: SB 1386
<http://info.sen.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html>

Stearnes, William Ramen Worm
<http://www.sans.org/y2k/ramen.htm>

United States. Department of Education. Family Educational Rights and Privacy Act (FERPA)
<http://www.ed.gov/policy/gen/guid/fpco/ferpa/index.html>

Wood, Alex Intrusion Detection: Visualizing Attacks in IDS Data
<http://www.giac.org/certified_professionals/practicals/gcia/0619.php)

Zone Labs Virus Information Center, Rbot.T (aka Sdbot)
<http://vic.zonelabs.com/tmpl/body/CA/virusDetails.jsp?VId=39437>