Global Information Assurance Certification Paper

Fun with Intrusion Detection
SANS GIAC
GCIA
v. 3.3

Samuel C. Adams

23 Jun 2003

Tuning a Signature Based IDS

Different IDS Technologies

A signature based IDS specializes in alerting on known vulnerabilities using rules that are custom made or provided by the IDS vendor. One big drawback with a signature based IDS is that you either have to know what you want to look at or be prepared to look at what someone else thinks is important. This may or may not include all the significant and worthwhile traffic on your network and could include a great deal that isn't significant or worthwhile. Custom configuration and tuning of signatures is critical to effective use of this type of IDS. Most IDSs on the market today are primarily signature based. Some examples would be Snort, NFR, Real Secure and Cisco IDS.

Protocol anomaly detection involves monitoring traffic that differs from what is specified in the RFCs defining a protocol. By modeling the correct use of a protocol and defining when changes in the state of a connection should take place – a protocol anomaly based IDS can inform an analyst of protocol misuse. The advantage to this approach is that malicious activity can be detected without a pre-generated signature. Two disadvantages are that there are a number of TCP/IP stacks that do not conform to protocol specifications and there a number of exploits that do. With the potential for false positives and false negatives, it is unlikely that a protocol anomaly based IDS will be effective without tuning. Examples of IDSs that provide protocol anomaly detection would be Symantec's ManHunt and Snort. It is worth noting that both of these tools provide signature based intrusion detection as well.

Statistical anomaly detection involves compiling a pattern of network usage and then looking for deviations in that pattern. The anomaly detection engine must monitor the network for a period of time to compile usage statistics. Once these statistics have been compiled, the IDS is able to use them to detect deviations from the normal pattern of traffic on the network. These statistics can be particularly useful for detecting port scans – even the low and slow variety – and denial of service attacks. One problem with statistical based systems is that malicious traffic can become normal traffic if it is seen often enough. Also, the traffic pattern for an organization can change frequently and may cause a need for frequent updates to the normal traffic baseline. Plus, an analyst must become familiar with normal traffic patterns in order to be able to use this type of IDS effectively. One example of a statistical anomaly based IDS is Spade – a Snort preprocessor.

A packet logger provides an important backup to the IDS. Many alerts generated by an IDS do not provide enough information to accurately diagnose what happened to cause the alert or how dangerous it is. A system that stores all the data that passed by on the network and that can be queried in the event of an IDS alert could be a valuable tool in

quickly determining what is going on and how dangerous the traffic might be. Most IDS tools can be used as packet loggers. Snort and Shadow are open source examples. Due to high data rates, this is a difficult proposition even on a small network. Yet, it's worth thinking about given the value in having extra data available for researching inscrutable activity.

Importance of Tuning

An organization's primary IDS is likely to rely on signatures. In order to be effective, signatures must be kept up to date and unnecessary signatures must be eliminated. The remaining signatures must be analyzed and modified so they will generate as few alerts as possible while still highlighting what an analyst needs to see. Two reasons for signature tuning are to reduce false positives and to improve sensor performance. If funds are limited, signature tuning can be a key factor in ensuring an IDS is able to process all the traffic flowing over the network. One important point is that not all IDSs allow the capability to inspect and alter signatures. If the analyst has no visibility into the signature it becomes more difficult to discover false positives and impossible to adjust the signature to make it more effective.

Most IDSs ship with some signatures that aren't appropriate for every organization. Some of these can be eliminated and others can be modified to make them more useful. As an example, Snort 1.9.1 arrived with over 1500 signatures. Snort's architecture allows for faster processing by using chain headers (a list of common packet attributes such as IP address or source/destination port) and chain options (detection modifier options such as TCP flags or payload content). If the criteria for the chain header are not met, then there is no need to search the chain options. This architecture can allow Snort to absorb a large signature set without a significant performance penalty. However, many IDSs do not work this way and even Snort can be bogged down with an excessive number of signatures.

In addition to performance issues there are finite analysis resources that can be applied to IDS alerts. There is always a limit to how long an analyst can stare at a computer screen and be able to interpret alerts intelligently. Large numbers of false positives can wear an analyst down and make her complacent. Tuning and reducing signatures will allow an analyst to focus on what's important and ensure that significant alerts are not overlooked.

Tuning Philosophy

The first thing to determine is the priorities for intrusion detection. Many organizations make a significant investment of time and resources in intrusion detection without determining what they expect to get out of it. Most intrusion detection analysts must make sacrifices due to time and resource constraints. Obviously the least important

things should be sacrificed first. What follows is a list of intrusion detection priorities. This may not be suitable for all organizations but hopefully works as a starting point.

1. Detecting system compromise as quickly as possible. It's impossible to ensure confidentiality, availability or integrity on systems that we no longer control. Ensuring that an analyst is quickly informed of system compromise should be the first goal of IDS configuration. An examination of analysis resources and the traffic passing through an IDS should allow us to determine whether other goals must be sacrificed in pursuit of this one. The next two goals are important primarily because they contribute to this one.

2. Improving system security. Occasionally during the course of intrusion detection analysis will show security flaws in protected systems. If attackers are looking for something via a scan or even a single connection – we need to know what they found. Is the system vulnerable remote exploit? Did the attacker find what they were looking for? Is the system giving out information it shouldn't be? Even if a site performs network assessments and penetration testing – how often is this done? Networks change constantly and sometimes security doesn't keep pace with the changes. Ideally an analyst should be able to quickly scan or direct the scanning of systems that may be vulnerable to compromise or are just giving out too much information.

3. Improving analysis capability. There is always room for improvement in any organization. Can signatures be improved or new signatures added to more effectively pinpoint attackers or eliminate false positives? Or should some signatures be removed because they have proven to be more trouble than they are worth? Are there other tools out there that could help us secure out network more effectively? Maybe there are scripts or programs that could be written to help view and correlate logs. Keeping up to date on the latest vulnerabilities is also important. Even if an IDS vendor is providing new signatures, the analyst still needs to be informed about the vulnerabilities she is trying to detect. During the course of analysis – the need for improvements of some kind will generally become apparent. Prioritizing and pursuing these improvements is critical to successful intrusion detection.

4. Detecting scans. Who is looking at our network? Do we see trends in the sources of attack? Are they looking for something specific? Are we generating statistics, graphs and charts? Are these statistics useful – can we respond to them in a meaningful way? Are we contacting the sources of scans? Perhaps the sources sites have compromised systems and would appreciate some notification. Or perhaps the source is an ISP we can complain to – and they can pull connectivity from the source of the scan. In some organizations this has a way of becoming top priority because it's visible to management and leads them to believe

resources are being well spent. It is the lowest priority here because while charts and graphs documenting scans can be interesting to look at, they do not have a great impact on the security of our network. This is not to say that looking at scans is not important, merely that if something needs to be sacrificed due to resource constraints, documenting scans should be the first to go.

Once the priorities of intrusion detection have been decided, we can move to prioritizing signatures. Most of the signatures below are from the stable rule set of April 5, 2003 from http://www.snort.org

Criteria:
The following criteria can be used to choose the most important signatures.

- Unique or rarely seen content match strings. If possible, these should include content modifiers such as depth, offset, distance and within.
- Alerts on traffic originating from $HOME_NET that might indicate a system compromise. This would include signatures designed to generate alerts based on internal system compromise. It might also include well defined attack responses. It would not include signatures that could be frequently triggered by normal traffic.
- Alerts which contain a number of different criteria (i.e. port, content and a modifier) and refer to very high risk network activity
- Alerts that suggest the organization is being specifically targeted – not just part of a catch all scan.

Signatures to keep:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"WEB-IIS MDAC Content-Type
overflow attempt"; flow:to_server,established; uricontent:"/msadcs.dll";
content:"Content-Type\:"; content:!"|0A|"; within:50; reference:cve,CAN-2002-1142;
reference:url,www.foundstone.com/knowledge/randd-advisories-display.html?id=337;
classtype:web-application-attack; sid:1970; rev:1;)
```

This signature incorporates uricontent, content and within rules. It's also only triggered for incoming traffic on $HTTP_PORTS. There is a publicly available exploit for this signature that has been around for some time. What's interesting about the way this is written is that if a new vulnerability involving this library emerges, this signature will most likely detect exploits for that as well. By looking for large content strings the signature is designed to detect a buffer overflow sent to MDAC – not any particular exploit. This technique has been used with some success in other signatures as well.

Another useful signature:

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 25 (msg:"Virus - Possible FIZZER email and
p2p worm"; content:"AHMAZQByAHYAYwAuAGUAeABl";
reference:url,securityresponse.symantec.com/avcenter/venc/data/w32.hllw.fizzer@mm.html
; reference:url,www.europe.f-secure.com/v-descs/fizzer.shtml; sid:4005;
classtype:trojan-activity; rev:1;)
```

This is a home grown signature. It is a good signature because the content match is unlikely to occur for something other than FIZZER and an alert is only triggered when a protected system is infected. Unfortunately, this signature will only work for this specific virus. If a new strain or something new comes out, this signature will not be useful.

Other examples:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 32770: (msg:"RPC rstatd query"; content:"|00
00 00 00 00 00 00 02 00 01 86 A1|";offset:5;
reference:arachnids,9;classtype:attempted-recon; sid:592; rev:3;)

alert udp $EXTERNAL_NET any -> $DNS_SERVERS 53 (msg:"DNS named version attempt";
content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase; offset: 12;
reference:nessus,10028; reference:arachnids,278; classtype:attempted-recon; sid:1616;
rev:4;)
```

While these signatures may not detect vulnerabilities as serious as the previous examples they are unlikely to be triggered by legitimate activity. The DNS signature is particularly well written (as are all the DNS signatures included in this snort rule set), combining multiple content and offset modifiers. This may have been simplified by the predictable layout of DNS packets. Alerts from these signatures would strongly suggest the organization had been specifically targeted.

Signatures covering individual exploits and shell code might also be included. These tend to cause minimal false positives. Removing signatures for nonexistent systems could be worthwhile as well. If the site has no cold fusion web servers – perhaps those signatures are not needed. This may only be practical for organizations where there is a close relationship between the people doing the intrusion detection and the people managing the network. While intrusion detection is far more effective if this relationship is close – this is not always feasible in large organizations or when intrusion detection is outsourced.

Elimination of signatures:
Many times it's easier to determine what signatures can be eliminated than it is to determine which should be kept. Here are some that generated a lot of noise for little return. Hopefully this will help in locating other signatures like this with similar characteristics.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET !80 (msg:"P2P GNUTella GET";
flow:to_server,established; content:"GET "; offset:0; depth:4; classtype:misc-
activity; sid:1432;  rev:3;)
```

"GET" occurs way too often in protocols other than GNUTella (particularly http) for this to be effective. Plus, there are lots of other signatures that do a good job of detecting GNUTella and produce far fewer false positives.

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT javascript URL
host spoofing attempt"; flow:to_client,established; content:"javascript\://"; nocase;
classtype:attempted-user; reference:bugtraq,5293; sid:1841; rev:2;)
```

This is a signature that generated a huge number of false positives (javascript:// is a pretty common string) for a vulnerability in older versions of Mozilla and Netscape that is unlikely to occur in an environment where most users use Windows workstations. Perhaps the signature could be improved with an additional content rule to look for "cookie".

```
alert tcp any 110 -> any any (msg:"Virus - Possible MyRomeo Worm"; content: "I Love
You"; sid:726; classtype:misc-activity; rev:3;)
```

The "I Love You" virus peaked several years ago and "I Love You" is a common string to see in email messages. If you really wanted a signature for the "I Love You" virus it would be much more effective to look for something unique in the virus payload.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 5032 (msg:"BACKDOOR NetMetro File List";
flow:to_server,established; content:"|2D 2D|"; reference:arachnids,79; sid:159;
classtype:misc-activity; rev:4;)
```

This signature came from the backdoor.rules file. A URL that may be of assistance in determining what backdoor signatures are worth keeping is http://www.dshield.org/port_report.php. Plug ports into this form to obtain a chart detailing how popular the port has been for backdoors and what's been running on it lately.

The signature for NetMetro has a commonly occurring content string used to detect an old (1999), rarely seen vulnerability. One thing to note is that many organizations avoided using the stream4 reassembly preprocessor with Snort 1.9 because of its vulnerabilities. The NetMetro signature, as well as many others, would undoubtedly be more useful with stream4 enabled.

Exclusions:

In many cases it is advisable to keep signatures but set up pass rules for certain hosts or networks. To do this, you may need to use the –o option, which tells snort to look at pass rules before alert and log rules. Below are some signatures that might require pass rules (variables would need to be defined first of course):

```
pass udp $SNMP_TRAP_SRC_IP any -> $HOME_NET 162 (msg:"SNMP trap udp";
reference:cve,CAN-2002-0012; reference:cve,CAN-2002-0013; sid:1419; rev:2; classtype:
attempted-recon;)
```

Many organizations use snmp traps to obtain updates on the health of their network devices. While snmp is not a secure protocol, it is so ubiquitous that sometimes its use is unavoidable. Pass rules can be used to ensure that attackers attempting to obtain

unauthorized information will not be lost in all the legitimate activity.

```
pass udp $DNS_UDP_ZT_IP any <> $DNS_SERVERS 53 (msg:"DNS zone transfer UDP"; content: "|00 00 FC|"; offset:14;
reference:cve,CAN-1999-0532; reference:arachnids,212; classtype:attempted-recon; sid:1948; rev:1;)
```

It's a good idea, particularly in a large organization, to allow zone transfers from at least one off site DNS server. While this is not a rule we would want to eliminate altogether, pass rules for legitimate transactions will help ensure that analysts do not become desensitized to this alert.

```
pass tcp $NET_BIOS_SRC any -> $NET_BIOS_DEST 139 (msg:"NETBIOS NT NULL session";
flow:to_server,established; content: "|00 00 00 00 57 00 69 00 6E 00 64 00 6F 00 77 00
73 00 20 00 4E 00 54 00 20 00 31 00 33 00 38 00 31|"; reference:bugtraq,1163;
reference:cve,CVE-2000-0347; reference:arachnids,204; classtype:attempted-recon;
sid:530; rev:7;)
```

While it is usually not good policy to allow Netbios traffic outside an organization, sometimes it cannot be avoided. This may be so that remote users can access file shares or different parts of the organization can share information efficiently (if not securely). Pass rules like this will ensure that unauthorized traffic doesn't get lost in all the legitimate Netbios alerts.

Changes:

In some cases it may be advisable to change or add new scan rules. The current scan rules look like this:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 1080 (msg:"SCAN SOCKS Proxy attempt";
flags:S; reference:url,help.undernet.org/proxyscan/; classtype:attempted-recon;
sid:615; rev:3;)
```

This is good for detecting someone knocking on the door but in most cases what we really want to know is – did the attacker find what he was looking for? Was the scan successful anywhere? Is there something to fix/improve?

```
alert tcp $HOME_NET 1080 -> $EXTERNAL_NET any (msg:"SCAN SOCKS Proxy response";
flags:SA; reference:url,help.undernet.org/proxyscan/; classtype:attempted-recon;
sid:???; rev:1;)
```

This will tell us if any systems on our protected network responded to a scan. Did we want them to respond? Are there supposed to be systems there? This may require a few pass rules for legitimate systems depending on the architecture of the protected network. It may be that a site will want to have both types of scan signatures – or, if resources are scarce, the first could be eliminated.

Something similar to this could be done with virus signatures. For example, for the klez virus, something like this might be useful:

```
alert tcp $HOME_NET 25 -> $EXTERNAL_NET any (msg:"VIRUS Klez Outgoing";
flow:to_server,established; dsize:>120; content:"MIME"; content:"VGhpcyBwcm9";
classtype:misc-activity; sid:1800; rev:2;)
```

This signature will let an analyst know if one of his protected systems is infected without distracting him with alerts resulting from traffic from other organizations.

Conclusion:

The most important thing to take away from reading this paper is the value of setting goals and priorities for intrusion detection. These goals and priorities then need to be made meaningful by configuring the IDS and applying man-hours to most effectively achieve them. An initial investment of time and possibly consulting resources to set priorities and configure an IDS properly will help create analysts as opposed to a bunch of people sitting around looking at computers.

References:

Arach-NIDS. “Trojan-Active-NetMetro”.
http://www.whitehats.com/info/IDS79

Das, Kumar. “Protocol Anomaly Detection for Network-based Intrusion Detection”, SANS InfoSec Reading Room. 15 Jan 02
http://www.sans.org/rr/intrusion/anomaly.php

Dshield Reports and Database Summaries
http://www.dshield.org/reports.php

Farshchi, Jamil. “Statistical Based approach to Intrusion Detection”.
http://www.sans.org/resources/idfaq/statistic_ids.php

Foundstone Labs. “Remote Buffer Overflow in Microsoft MDAC and Internet Explorer.”
http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/advisories_template.htm%3Findexid%3D4

Naval Surface Warfare Center. “NWSC Shadow Index”.
http://www.nswc.navy.mil/ISSEC/CID/

Northcutt, Stephen and Novak, Judy. Network Intrusion Detection. 3rd Ed, New Riders, Indianapolis IA, September 2002.

Roesch, Martin. “Snort – Lightweight Intrusion Detection For Networks”
http://www.snort.org/docs/lisapaper.txt

Roesch, Martin and Green, Chris. “Snort Users Manual”, Sourcefire Inc, 2003
http://www.snort.org/docs/writing_rules/chap2.html#tth_chAp2
http://www.snort.org/docs/writing_rules/chap1.html#tth_sEc1.4

SANS Intrusion Detection FAQ
Liston, Kevin. “Can you explain analysis and anomaly detection?”
http://www.sans.org/resources/idfaq/anomaly_detection.php

SecurityFocus. “Mozilla JavaScript URL Host Spoofing Arbitrary Cookie Access Vulnerability”.
http://www.securityfocus.com/bid/5293

Part 2 – Network Detects

Network Detect #1
Analysis of Code Red I

1. Source of Trace
http://www.incidents.org/logs/raw/2002.10.17
Although the filename is 2002.10.17 the time stamp on the packets shows a date of 17 Nov 2002. Assuming that all the traffic in this file has been detected by the same IDS, we can conclude that the site being protected owns the 170.129.0.0/16 network – or at least a block of addresses of similar size that has been obfuscated to the 170.129.0.0/16 network. There do not appear to be any responses to the scans and exploits attempted in this file. This could indicate that they failed or were filtered out by the perimeter defenses or it could mean that we do not have access to that traffic.

Looking at the MAC addresses for the hosts in this file – the same two reappear over and over again. Both addresses correspond to two different ranges owned by Cisco Systems. Most likely what this means is that the sensor is between two different types of Cisco devices. This could mean the sensor is between the organization's perimeter router and its ISP's router. Or it could mean the sensor is inside the organization's perimeter router and there is some other Cisco device (such as a Pix Firewall) between the sensor and the organization's network. The first scenario seems more likely because of the wide range of destination ports we can see in the raw log file – ports that would most likely be filtered by a perimeter router.

An attempt at passive fingerprinting of the source using p0f was unsuccessful. While it has a window size of 32120 characteristic of the Linux 2.0/2.2 TCP/IP stack, the TTL is 240 suggesting an original TTL of 255 – which suggests a different stack. Since the IP ID field and header checksum (which is incorrect) are both 0 – it seems likely that they – and perhaps other parts of the header as well - have been modified.

2. Generated by:
Snort 1.9.1 (Build 231), using the rule set included with the Linux snort-1.9.1-1snort.i386.rpm obtained from www.snort.org running on Redhat 8.0. The command used to generate the alerts was:

```
snort -r 2002.10.17 -c /etc/snort/snort.conf -l log -A console -q -k none -d
```

Note the -k none to disable checksum verification. While most packets in the raw file had valid IP header checksums and invalid TCP checksums, this packet had an invalid IP header checksum. The traffic generated the following snort alert:

```
[**] WEB-IIS ISAPI .ida attempt [**]
11/16-23:36:34.706507 204.106.15.146:1058 -> 170.129.50.3:80
```

```
TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:1504
***AP*** Seq: 0x902295C2  Ack: 0x3BCAD6EB  Win: 0x7D78  TcpLen: 20
```

which was generated by this signature:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS ISAPI .ida
attempt"; flow:to_server,established; uricontent:".ida?"; nocase;
reference:arachnids,552; classtype:web-application-attack; reference:bugtraq,1065;
reference:cve,CAN-2000-0071; sid:1243; rev:8;)
```

Ethereal dump of relevant packet payload:
Used Ethereal version 0.9.11 from RPMs ethereal-0.9.11-1.80.0 and ethereal-gnome-0.9.11-1.80.0 also running on RedHat 8.0.

```
47 45 54 20 2F 64 65 66 61 75 6C 74 2E 69 64 61    GET /default.ida
3F 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    ?NNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E 4E    NNNNNNNNNNNNNNNN
4E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    N...............
C3 03 00 00 00 78 00 FA 20 25 75 39 30 39 30 25    .....x.. %u9090%
75 36 38 35 38 25 75 63 62 64 33 25 75 37 38 30    u6858%ucbd3%u780
31 25 75 39 30 39 30 25 75 36 38 35 38 25 75 63    1%u9090%u6858%uc
62 64 33 25 75 37 38 30 31 25 75 39 30 39 30 25    bd3%u7801%u9090%
75 39 30 39 30 25 75 38 31 39 30 25 75 30 30 63    u9090%u8190%u00c
33 25 75 30 30 30 33 25 75 38 62 30 30 25 75 35    3%u0003%u8b00%u5
33 31 62 25 75 35 33 66 66 25 75 30 30 37 38 25    31b%u53ff%u0078%
75 30 30 30 30 25 75 30 30 3D 61 20 20 48 54 54    u0000%u00=a  HTT
50 2F 31 2E 30 0D 0A 43 6F 6E 74 65 6E 74 2D 74    P/1.0..Content-t
79 70 65 3A 20 74 65 78 74 2F 78 6D 6C 0A 48 4F    ype: text/xml.HO
53 54 3A 77 77 77 2E 77 6F 72 6D 2E 63 6F 6D 0A    ST:www.worm.com.
```

3. Probability the source address was spoofed:
It seems unlikely the source address was spoofed. Although the initial SYN connection is unavailable – there was most likely a TCP 3-way handshake, otherwise there would be no hope of the destination system accepting this packet. Although it might be possible to execute a buffer overflow using only an initial SYN packet – since that's not what this is, a response from the destination would have been required for this exploit attempt to have a chance of success.

4.Description of attack:
This appears to be a buffer overflow attempt on TCP port 80. The specific means used to exploit the Index Server vulnerability is characteristic of the Code Red worm. Note

that the signature is characteristic of the original Code Red worm – not Code Red II. The easiest way to spot the difference is to note the list of N characters as padding in the payload. Code Red II is characterized by a list of X characters used as padding. Another characteristic of Code Red (as well as Code Red II) is that it only propagates from the 1st to the 19th of the month. Since the timestamp on the packet is Nov 17 this also suggests the Code Red worm.

The snort signature triggered on the presence of .ida? in a URL (uricontent) going from an external client to an internal web server. While this is a fairly simple signature that tends to generate lots of false positives – in this case it led us to the attack.

Code Red is categorized as CVE-2001-0500.

5.Attack Mechanism:

A buffer overflow occurs when a program allocates a block of memory (i.e. 128 bytes) and then puts more data into the block (or buffer) than it was designed to hold (i.e. 129 bytes). The way most system architectures are designed, there is nothing to differentiate executable code from data. This means that when the when the size of a block of data is larger than the block of memory it is stored in, the data that overflows can modify the flow of instruction execution and cause code of the attacker's choice to be executed. This architecture design leaves it up to the programmer to ensure that data placed into allocated blocks of memory is properly checked to make sure it doesn't spill out of those blocks. Sadly, some programmers fail to properly check data before stuffing it into memory buffers. This failure is what leads to buffer overflows. One of the most popular ways to exploit a buffer overflow vulnerability is through improperly checked input data.

.ida files contain what Microsoft refers to as Internet Data Administrator scripts. An unchecked buffer exists in a component of the Index Server (idq.dll) that can be exploited to gain System level access. Because of script mappings associating .idq and .ida files with idq.dll these files can be used to exploit the vulnerability. The Code Red worm uses the script mapping between the default.ida script and idq.dll to take advantage of the unchecked buffer in idq.dll and attempts to exploit it by passing it more data than it was designed to hold.

6.Correlations

[1] Danyliw, Roman. CERT Advisory CA-2001-19 "Code Red Worm Exploiting Buffer Overflow in IIS Indexing Service DLL". Jul 19, 2001.
http://www.cert.org/advisories/CA-2001-19.html
A short and sweet guide to Code Red. This has a packet capture of what Code Red looks like and talks about what it looks like when successful.

[2] Danyliw, Roman. CERT Incident Note IN-2001-09. "Code Red II". August 6, 2001.

http://www.cert.org/incident_notes/IN-2001-09.html
A short and sweet guide to Code Red II. This also has a packet capture and allowed me to see the difference between the two.

[3] eEye Digital Security. ".ida Code Red Worm". Jul 17, 2001.
http://www.eeye.com/html/Research/Advisories/AL20010717.html
The definitive guide to Code Red by the folks that discovered it. This gives the intimate details of Code Red and talks about how and when it spreads.

[4] IEEE Organizationally Unique Identifier Listing
http://standards.ieee.org/regauth/oui/oui.txt
Provides a reference for MAC address allocation.

[5] Litchfield, David. Exploiting Windows NT 4 Buffer Overruns (A Case Study: RASMAN.EXE)
http://www.atstake.com/research/reports/wprasbuf.html
This article provides an easy to understand explaination of what a buffer overflow is and how it works.

[6] Microsoft Security Bulletin "Unchecked Buffer in Index Server ISAPI Extention Could Enable Web Server Compromise". Jun 18, 2001.
http://www.microsoft.com/technet/security/bulletin/MS01-033.asp
This article gave me all the little details of the buffer overflow. It talks about what idq.dll is, what's wrong with it and the other files that relate to it.

[7] Young, Paul. GCIA Practical. Jan 2003.
http://www.giac.org/practical/GCIA/Paul_Young_GCIA.pdf
Paul has a good talk on Code Red in the first section of his paper. He is also very thorough in his network detects and helped me figure out what to talk about.

[8] Zelewski, Michael; Stearns, William. p0f - Passive OS Fingerprinting Tool
http://www.stearns.org/p0f/
A handy tool for analyzing libpcap formatted data to find operating systems.

IP Address: `204.106.15.146`
HostName: `zylomed3-colo.host.net`
Whois:

```
Doe Spun, Inc.
DOESPU
3500 NW Boca Raton Blvd., Suite 902
Florida
```

Dshield reports:

```
No reports for 204.106.15.146.
```

Given the high probability that parts of the header including the source IP were obfuscated after the alert was logged, it is not surprising that there are no Dshield reports on the IP.

7. Evidence of Active Targeting

Code Red spreads itself using a sequence of more or less randomly chosen IP addresses[1]. This argues against the hypothesis that the destination address was specifically targeted even though there is no evidence of previous reconnaissance. This attack could also have been generated by someone using Code Red to actively target a web server found through Google or a scan from another host or one that isn't in the log file.

According to ARIN the 170.129.0.0/16 network belongs to Standard Microsystems Corporation. – and DNS records indicate that 170.129.50.3 is www.smsc.com. Since Code Red is more or less random it seems odd that the attack would go straight to the web server without fishing around for other targets first. This makes it seem likely that the site was actively targeted.

8. Severity:

The severity of the attack will be calculated with the following formula:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Each value will be ranked on a scale from 1 (lowest) to 5 (highest).

Criticality - The criticality of a web server depends a good deal on the purpose and role of the server in the organization. It also depends how much the reputation of the organization would suffer if the server was compromised. For example, the compromise of the web server for an organization that specializes in Internet Security would be much more damaging than the compromise of a web server for an organization that sells coffee. Since this server appears to be the primary means for the organization to sell its products, keeping it operational is of the utmost importance. That calls for a **5**.

Lethality – This is a buffer overflow that would lead to system level access if successful – that calls for a **5**.

System Countermeasures – There was no response from this server to any of the

[1] eEye Digital Security

attacks in the log file. The server never tries to compromise any other hosts using Code Red – also suggesting the compromise was unsuccessful. And, considering its importance to the organization it seems likely that they would take measures to protect it. However, this is a publicly accessible web server and a number of attackers seem to be aware of it. This calls for a **4**.

Network Countermeasures – All attack attempts against this network appear to have failed. Not only that, it doesn't appear as though any of them were even responded to. It's possible that servers were compromised but this data is just unavailable. However, it seems more likely – since there is some outgoing data available from this network – that no attacks were successful. Furthermore the organization owning the server does still appear to be in business and continuing to sell their products. This suggests that at least some attention has been paid to network security. Still, we can't be sure how much. Since the organization provides a fairly sophisticated public web server – it exposes itself to some degree of risk. This calls for a **4**.

(5 + 5) – (4 + 4) = 2

9. Defensive Recommendations

Assuming we have all the available data, defensive measures appear to have been adequate. It's difficult to tell what's in place from what we have, however there are measures that can be taken against Code Red.

- Ensure all appropriate security patches have been applied to the web server
- Unless absolutely necessary, ensure the web server cannot initiate outbound connections
- Isolate the web server in a DMZ so that if it is compromised an attacker will be unable to affect other systems
- Attempt to choose software where vulnerabilities are not often found and/or where code is written and reviewed with security in mind.

10. Multiple choice test question:

How can buffer overflows vulnerabilities be eliminated?

a. A default deny firewall access policy
b. Shutting down unnecessary system services
c. Encrypting packets to ensure data is not compromised
d. Careful programming practices and code auditing

Answer: d

A buffer overflow is one of the most commonly used methods of compromising a remote system. It's important to know that even if a network is secure and all patches are up to date – vulnerabilities may still exist. An analyst needs to be aware of this threat and take what measures she can to counter it.

Questions:
I posted this analysis on Sunday 1 June 2003 and Johnny Calhoun promptly answered (thanks Johnny).

My words:
```
> Network Countermeasures - All attack attempts against this
> network appear to have failed. Not only that, it doesn't appear > of
them were even responded to.
```

Johnny's comment:
```
I am curious to know how you can draw upon this conclusion when only
packets that match a Snort Signature are logged.
```

Answer:
This is a good point. I must admit that I misread the assignment and thought I had a complete picture of traffic flowing by on the network. Thinking about it now, there does not seem to be enough traffic for this. I took Johnny's comment into account for the other network detects.

My words:
```
> It's possible that servers were compromised but this
> data is just unavailable.   However, it seems more likely –
> since there is some outgoing data available from this network - >
that no attacks were successful.
```

Johnny's comment:
```
You seem to be contradicting yourself here; confuses the reader.
You may want to explain your reasoning and why the information is
unavailable.  What would you look for if the information was
available? ex. Web Server Logs, traffic dumps, firewall/router logs,
etc.
```

Answer:
The point I was trying to make was that, in the outgoing data from the protected network, there is no evidence the attacks in the log were successful. I would be looking for outbound Code Red alerts or attack responses indicating other attacks may have been successful. If other data was available, I would look in the web server logs for evidence of a successful attack, as well as making sure the server was up to date on patches. I'd also revisit firewall logs and ACLs to make sure the firewall is blocking what I want it to block.

Johnny's comment:

```
You have a solid analysis overall, but I would have like to seen a
better correlation section, rather than a list of references. You may
want to expand on why you used each reference.
```

Answer:
Good point. I added some explanations to my correlation section after reading this.

Network Detect 2
DNS Zone Transfer

1. Source of Trace:
This trace was obtained from http://www.incidents.org/logs/Raw. It is from the 2002.10.3 file. The protected network range appears to be 207.166.0.0/16. The MAC addresses again belong to Cisco Systems but some are the same as the one I analyzed in the 2002.10.17 file – suggesting that they have been modified. There is also broadcast traffic in the file that would normally be filtered by an external router.

Attempts at passive fingerprinting of the source did not work. The initial TTL appears to be 255 (meaning the source is probably 6 hops away) but taken with the window size of 32850 – which may correspond to Windows NT – the alerts do not match any p0f fingerprints. This could indicate that something in the packet has been modified – which is likely given the incorrect IP and TCP checksums – or it could be that the attacker has modified his TCP/IP stack settings.

The sensor appears to be on the network perimeter. While this is by no means certain, it is suggested by the variety source of addresses in the log file. There is also broadcast traffic from 255.255.255.255 - which most external routers would filter out.

2. Detect Generated by:
Snort 1.9.1(Build 231), using the rule set included with the Linux snort-1.9.1-snort.i386.rpm obtained from www.snort.org running on Redhat 8.0. I also used Ethereal version 0.9.11 from RPMs ethereal-0.9.11-1.80.0 and ethereal-gnome-0.9.11-1.80.0**.** The command used to extract the data was:

```
snort -r 2002.10.3 -c /etc/snort/snort.conf -l log -A console -q -k none -d
```

IP header and TCP checksums in this log file were consistently incorrect. It doesn't look like there were any UDP packets present. Either they didn't cause any snort alerts or they've been filtered out.

Snort signature:

```
alert tcp $EXTERNAL_NET any -> $DNS_SERVERS 53 (msg:"DNS zone transfer
TCP"; flow:to_server,established; content: "|00 00 FC|"; offset:14;
reference:cve,CAN-1999-0532; reference:arachnids,212;
```

```
classtype:attempted-recon; sid:255; rev:7;)
```

Snort alerts:

```
[**] DNS zone transfer TCP [**]
11/02-23:05:40.766507 167.206.112.181:40541 -> 207.166.87.159:53
TCP TTL:249 TOS:0x0 ID:45261 IpLen:20 DgmLen:66 DF
***AP*** Seq: 0x44F8E361  Ack: 0x7AA4D5A2  Win: 0x8052  TcpLen: 20
6C EE 00 00 00 01 00 00 00 00 00 00 04 58 58 58  l............XXX
58 03 63 6F 6D 00 00 FC 00 01                    X.com.....

+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
**] DNS zone transfer TCP [**]
11/03-17:09:50.396507 167.206.112.181:39333 -> 207.166.87.159:53
TCP TTL:249 TOS:0x0 ID:18843 IpLen:20 DgmLen:66 DF
***AP*** Seq: 0xCD0FE662  Ack: 0x79216BAF  Win: 0x8052  TcpLen: 20
0B 93 00 00 00 01 00 00 00 00 00 00 04 58 58 58  .............XXX
58 03 63 6F 6D 00 00 FC 00 01                    X.com.....
```

Looking at the hex data in the snort alerts we can see the DNS ID in the first 2 bytes of the packet (0B 93), followed by no flags (00 00), followed by a 00 01 – signifying 1 question. The 6 pairs of zeros that follow mean that there are no answer, authority or additional resource records. According to Stevens this is normal for a DNS query. The 04 refers to the number of bytes in the label, which has been obscured to XXXX or hex 58 58 58 58 in this case. The obscured label is followed by an 03, indicating the number of bytes in "com". Taken together XXXX and com form the name being looked up. This could be something like fred.com. Finally, we come to the 00 00 FC that triggered the alert. This corresponds to AXFR or a zone transfer query type. Note how it starts on byte 21 of the payload. The offset 14 rule seems to refer to the second byte of the first label, after the count (the second 58 in the hex dump). This is a well written signature. Because of the predictable layout of DNS packets the chance of a false positive is very low.

3. Probability the Source Address was spoofed

It seems very unlikely the source address was spoofed. In order to obtain useful information from a zone transfer the attacker needs to be in a place to receive the response. It's possible the source was spoofed and the originator waits somewhere along the return path with a sniffer to obtain the response – this seems unlikely given the simplicity of a zone transfer and the difficulty involved in ensuring the return packet takes the appropriate path. It's also possible this is merely a test to see how resilient the firewall or the DNS server is – what is allowed and what is not. This also seems unlikely since zone transfers are the only activity seen from this host in the log file and the resiliency information would need a lot of correlation data to be worthwhile.

4. Description of the Attack

A zone is a separately administered domain in the DNS hierarchy. Some examples of

zones are microsoft.com and cisco.com. Zones can be divided into smaller zones such as gates.microsoft.com or chambers.cisco.com. Each zone generally has a primary and at least one secondary or backup DNS server. These servers contain databases, which hold among other things, the name to IP address mapping for the zone. Typical queries to a DNS server are for one name or IP address.

Usually only the primary server is updated when entries are added to the DNS database. The secondary server(s) are expected to update themselves in some automated fashion. The purpose of a zone transfer is to perform the update efficiently using one query. Unfortunately a zone transfer can provide an attacker with a wealth of information about a network – making it commonly used command for attackers.

Even though DNS traffic usually uses UDP, because zone transfers typically require more than 512 bytes – which is the limit for UDP DNS – they are generally performed using TCP.

5. Attack Mechanism

There are seven alerts that seem related to this attack. One interesting thing about the alerts is that there is about 3-3.5 hours between the timestamps. It's difficult to imagine why an attacker would do this. A zone transfer is not a very subtle attack. Trying to come in low and slow seems pointless – if the owner of the DNS server is paying any attention at all, the attacker has already been spotted, if not – then why not just do all the attempts at once? Perhaps this isn't an attack at all.

ARIN provides this information about the source:

```
Cablevision Systems Corp (167.206.112.0/24)
111 New South Road
Hicksville NY 11801
```

And destination:

```
I-Link Worldwide Inc (207.166.64.0/19, 207.166.96.0/20)
13751 S Wadsworth Park Dr, Suite 200
Draper UT, 84020
```

The address ranges for the destination do not agree with what's seen in the log file, nor does there appear to be a DNS server at 207.166.87.159. Most likely this means that the addresses have been obfuscated. This makes it difficult to tell if there is a business relationship between the organizations that might suggest this is legitimate traffic.

Other things to note are that all the packets are the same length (80 bytes according to ethereal), none are fragmented and all use the same TCP flags (ACK and PSH). The source ports are all above 30000 and while this isn't the best thing to use for passive fingerprinting it does suggest the OS isn't Linux or Windows, since they start at 1024 by default. It might be Solaris, but it might also be going through NAT – which would make

the source port meaningless.

Since a zone transfer between primary and secondary servers is a routine occurrence, it would be odd not to create a pass rule to prevent it from generating an alert. This and the lack of any evidence of a relationship between the source and destination leads me to believe this is an attack and not legitimate traffic.

6. Correlations

[1] CAN-1999-0532 "Zone Transfers"
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0532
This is candidate for inclusion in the CVE list. It was rejected by Steven Northcutt – he says it is quite appropriate for split DNS implementations. I disagree. CVE stands for Common Vulnerabilities and Exposures. If it were limited to vulnerabilities – Northcutt would be right. Zone transfers are a feature of DNS, not vulnerability. However, in many cases, allowing zone transfers is an unintentional exposure. Even with a split implementation it still makes sense to restrict who can do a zone transfer.

[2] IEEE Organizationally Unique Identifier Listing
http://standards.ieee.org/regauth/oui/oui.txt
Provides a reference for MAC address allocation

[3] Lao, Steven. "Why is Securing DNS zone transfer necessary?"
http://www.giac.org/practical/GSEC/Steven_Lau_GSEC.pdf
Talks about defensive measures available to prevent zone transfers.

[4] Lowe, Scott. "Block DNS Zone Transfers to your Servers." Feb 24, 2003.
http://asia.cnet.com/itmanager/netadmin/0,39006400,39113741,00.htm
Covers Microsoft DNS on Windows NT or Windows 2000 complete with pretty pictures.

[5] Roesch, Martin. "Snort Users Manual", 2003.
http://www.snort.org/docs/writing_rules/
This page has lots of guidance for the rules that make up a signature.

[6] Seifried, Kurt. "Passive OS Detection"
http://www.seifried.org/security/network/20011009-passive-os-detection.html
There are lots of articles on this subject out there but this one has a very good section on source ports and what they might indicate.

[7] Stevens, Richard. TCP/IP Illustrated. Addison Wesley, 1994.
There's an excellent discussion of the layout of DNS packets in here. This explains how to interpret each hex field in a DNS packet. It also explains what a zone transfer is, how it works and in what circumstances it's legitimately performed. In addition, Stevens talks about why and when TCP and UDP are used for DNS.

[8] Zelewski, Michael; Stearns, William. p0f - Passive OS Fingerprinting Tool
http://www.stearns.org/p0f/
A handy tool for analyzing libpcap formatted data to find operating systems.

Dshield Reports:
IP: `167.206.112.181`
Hostname: `olympus.srv.hcvlny.cv.net`
`No reports were available.`

7. Evidence of Active Targeting:
This IP looks like it was actively targeted. There is no evidence of any other scanning from this source in the log file. This may have been an information gathering attempt as a prelude to further attacks on this site. The attacker may have obtained the address of the DNS server from ARIN, by querying the domain of a public web site or any number of other means that wouldn't trigger an alert. He may also have had previous familiarity with the site. This is likely since we can tell from the TTL that he was probably only six hops away.

Perhaps the attacker was just looking for any DNS server and not this one in particular. The openness of the attack suggests this. If he was interested in being more subtle, he could have done the equivalent of a zone transfer by simply querying all the addresses in the domain one by one using an automated script. Most likely this would not have been noticed in all the other traffic seen by a DNS server – particularly if it was done over a long period of time.

8. Severity
The severity of the attack will be calculated with the following formula:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Each value will be ranked on a scale from 1 (lowest) to 5 (highest).

Criticality – A site is pretty helpless without a DNS server. Almost all incoming and outgoing traffic must go to the DNS server for IP/hostname resolution first. Shutting down DNS would criple a site. There is also a great deal of sensitive information on a DNS server that could give an attacker a map of the internal network. Even with multiple secondary servers – a DNS server is still a critical system. It rates a **5.**

Lethality – The attacker wasn't going for compromise here, he was just getting information. This attack would have yielded a great deal of useful information if successful but much more would be needed to cause damage to the organization.

Therefore, this rates a **2.**

System Countermeasures – It would seem logical that the attacker would have stopped attempting to perform a zone transfer if any attempts had succeeded. Since the attacker tried seven times, it seems likely that all attempts failed. Given that there was no additional traffic from this server over the two days spanned by the log file it can't be the most desirable of targets. This suggests at least some system countermeasures and calls for a **3**.

Network Countermeasures - The number of outbound GNUTella requests suggest that this site either needs a better acceptable use policy or needs to enforce its existing one. However, there is no evidence of any outbound attack, suggesting that none of the exploit attempts worked. Since the site is employing some sort of intrusion detection (hence the existence of this log file) there appears to be some concern for security. This seems to rate a **3**.

Severity
(5+2) – (3+3) = 1

9. Defensive Recommendations:
Probably the most important thing to do to prevent successful zone transfers by unauthorized parties is to ensure DNS servers are only configured to allow transfers from authorized hosts. In Bind 8 and 9 this can be done with the allow-transfer statement in the options section of the Bind configuration file

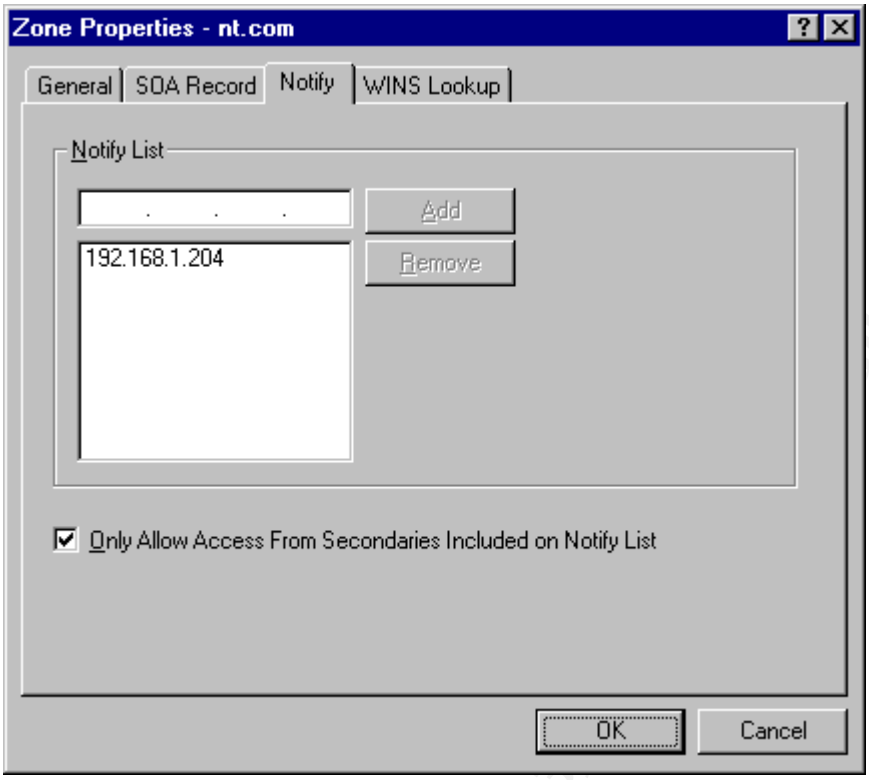
```
options {
   allow-transfer {192.168.1.1; };
};
```

This option can also be set for individual zones. The statement below creates zone bedrock.org, which does not allow zone transfers.

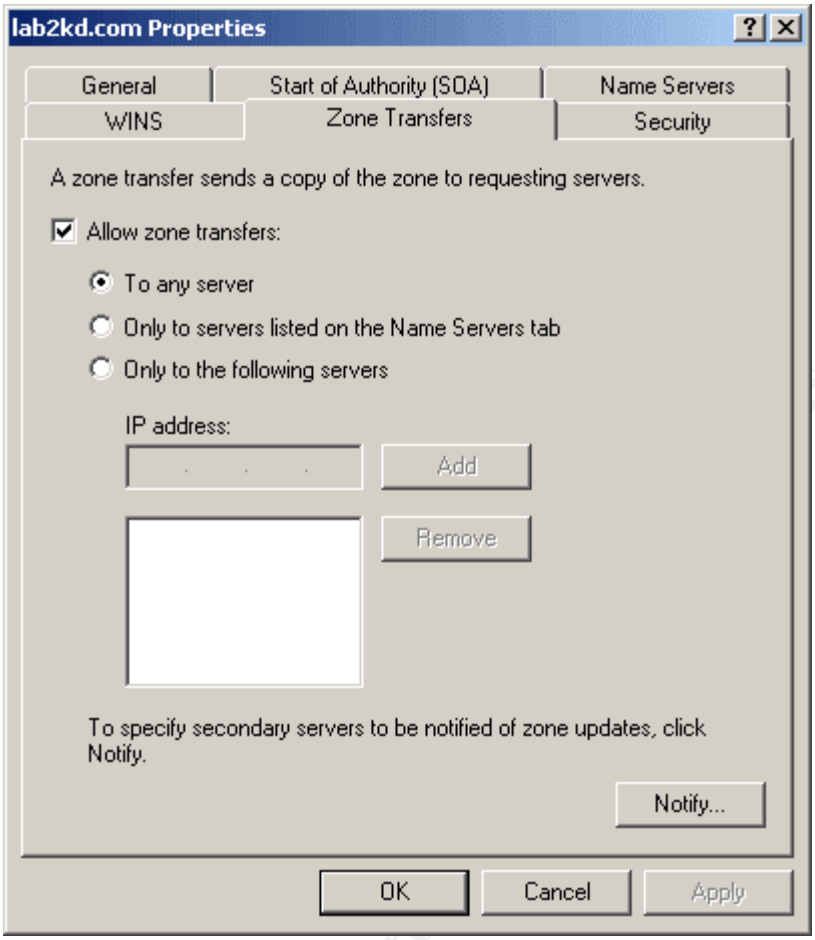
```
zone bedrock.org {
   type slave;
   master { 192.168.1.1; };
   file bedrock.zone.cache;
   allow-transfer { none; };
};
```

In Bind 4.9 an administrator can use the xfrnets directive
`xfrnets 192.168.1.1&255.255.255.255`

With Windows NT DNS use the Notify Tab of the Zone Properties window and check:
`Only Allow Access from Secondary servers included on Notify List.`



On Windows 2000 uncheck the Allow zone transfers box in the Zone Transfers tab or use the radio buttons for `Only to servers listed in the name servers tab` or `Only the following servers`



Another thing to consider would be to prevent inbound TCP port 53 requests to DNS servers via a firewall or filtering router. This will disallow some legitimate traffic in addition to zone transfers so some thought should be put into it prior to implementation. Split DNS servers are also valuable. Many sites have at least two sets of DNS servers: primary and secondary internal DNS servers as well as primary and secondary external DNS servers. This way, only the internal DNS servers need to have the layout of the internal network and these servers can be tucked away safely behind the firewall. Only the external DNS servers would be accessible to the outside world and all they would need to list are publicly available systems such as the web server and mail server. The site may also need to include throw-away hostnames for user workstations to satisfy remote sites that do reverse DNS lookup.

10. Multiple Choice Question:

Why is TCP normally used for zone transfers?
a. Zone transfers are important transactions and UDP is not reliable enough
b. Zone transfers usually contain too much data to use UDP
c. Because of the way the packet is structured – it can be handled more efficiently with TCP
d. UDP needs to be kept available to handle other types of DNS queries

Answer: b

Questions:
Since I learned something from the first post, I went ahead and posted this one as well (5 Jun 03). Donald Smith was kind enough to respond (also 5 Jun 03).

My words:
```
> Attempts at passive fingerprinting of the source did not work. > The
initial TTL appears to be 255 (meaning the source is
> probably 6 hops away) but taken with the window size of 32850 - >
which may correspond to Windows NT - the alerts do not
> match any p0f fingerprints. This could indicate that something
> in the packet has been modified - which is likely given the
> incorrect IP and TCP checksums - or it could be that the
> attacker has modified his TCP/IPstack settings.
```

Donald's question:
```
Did you try googling for the hex version of the window size Win:
0x8052? It may not be common but it is a known size:-)
```

Answer:
The Window size does in fact correspond to a p0f fingerprint from Windows NT. I guess from the way I worded things this was unclear. However, taken with the TTL, it doesn't match any known fingerprints. After reading this, I did try googling. There is a page with this window size (http://www.corfu1.com/eee/h.TXT) however, the fingerprint seems different from our host. In this URL the TTL is 41 (vs our 249) indicating a probable initial TTL of 64 and the ephemeral port used is 1729 (vs our >30000). This seems to indicate a different type of OS.

My words:
```
> Even though DNS traffic usually uses UDP, because zone
> transfers usually require more than 512 bytes - which is the
> limit for UDP DNS - they are generally performed using TCP.
```

Donald's question:
What happens when a zone transfer doesnt fit in a udp packet?

Answer:
When a zone transfer doesn't fit in a UDP packet, TCP is used. This happens fairly frequently since most zones have more than 512 bytes of data. I have occasionally seen UDP zone transfers and there is a snort signature for them so I guess other people have seen them as well. This is why it's important to configure your DNS server to only allow zone transfers from authorized hosts and not rely on a firewall that blocks TCP DNS connections.

My words:
```
> Dshield Reports:
>
> IP:  167.206.112.181
>
> Hostname: olympus.srv.hcvlny.cv.net
```

Donald's comments:
```
Ok does that host name mean anything?
srv could be a server.

hcvlny. cv.net

cv is probably Cablevision Systems Corp
ny probably newyork

So this could be a part Cablevision systems corp.

I googled hcvlny.cv.net and found LOTS of pages all refering to
systems like
mta9.srv.hcvlny.cv.net

Odviously a mail server. Seems like Cablevision systems puts their at
least some of thier servers in a domain named srv.hcvlny.cv.net. So
your it could be normal traffic (not an attack) is not a bad guess.

If it was normal traffic what do you think it is.
What kind of system would attempt to do a zone transfer about
every 3 hours or so?
```

Answer:
This could be a slave server going to a master server. The time between zone transfers is configurable so while every three hours seems a little too frequent to me, maybe it is suitable for this organization.

Network Detect 3
Front Page Extensions

1. Source of the Trace
http://www.incidents.org/logs/raw/2002.10.14
This file appears to be from a sensor on the 170.129.0.0/16 network, the same network as the 2002.10.17 log file. It uses the same MAC addresses and seems to have some of the same servers. Attempts at automated passive fingerprinting on the source were again unsuccessful and there are no outbound packets from the destination to fingerprint.

Looking at what we can from the source – the original TTL appears to have been 128 – suggesting that the source is 16 hops from the destination. This would most likely indicate a MS Windows NT/2000 system but the initial window size of 64894 corresponds to no p0f fingerprint. Since the window size does increase between the two packets – it may have started off as a different number. There are Windows NT 5.1 fingerprints with the right TTL, a similar window size, and the Don't Fragment bit set. Also, the source ports of 3336 and 3340 suggest a Microsoft Windows system. Given that the port and sequence numbers increase between the two packets it would seem that they are not crafted, or that the crafter is just more clever than average.

Both IP header checksums are correct while the TCP checksums are incorrect. The web site has been obfuscated which is a little puzzling if the headers are in fact accurate and 170.129.50.3 is the web site that was attacked.

2. Generated by:
Snort 1.9.1(Build 231), using the rule set included with the Linux snort-1.9.1-snort.i386.rpm obtained from www.snort.org running on Redhat 8.0. I also used Ethereal version 0.9.11 from RPMs ethereal-0.9.11-1.80.0 and ethereal-gnome-0.9.11-1.80.0. The command used to extract the data was:

```
snort -r 2002.10.14 -c /etc/snort/snort.conf -l log -A console -q -k none -d
```

Snort signatures:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-IIS
_vti_inf access";flow:to_server,established;
uricontent:"_vti_inf.html"; nocase; classtype:web-application-
activity; sid:990;  rev:5;)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-
FRONTPAGE _vti_rpc access"; flow:to_server,established;
uricontent:"/_vti_rpc"; nocase; reference:bugtraq,2144; classtype:web-
application-activity; sid:937;  rev:6;)
```

The signatures are pretty straightforward. They cause alerts to trigger when someone (in this case $EXTERNAL_NET was set to any) sends _vti_inf.html or _vti_rpc respectively to $HTTP_SERVERS – in this case also any. Note the flow and nocase rules, causing the alerts to trigger only when the traffic is sent to (not from) the server with either upper or lowercase characters. Taken singly these signatures could be better since in most situations when an alert is triggered there is no indication as to whether the attacker succeeded or failed. This may not be possible if IIS does not reliably give a unique response indicating success.

Snort alerts:

```
[**] WEB-IIS _vti_inf access [**]
11/14-16:55:07.816507 12.217.179.244:3336 -> 170.129.50.3:80
TCP TTL:112 TOS:0x0 ID:65414 IpLen:20 DgmLen:305 DF
***AP*** Seq: 0xB7198FF  Ack: 0x28042406  Win: 0xFD7E  TcpLen: 20
47 45 54 20 2F 5F 76 74 69 5F 69 6E 66 2E 68 74  GET /_vti_inf.ht
6D 6C 20 48 54 54 50 2F 31 2E 31 0D 0A 44 61 74  ml HTTP/1.1..Dat
65 3A 20 54 68 75 2C 20 31 34 20 4E 6F 76 20 32  e: Thu, 14 Nov 2
30 30 32 20 32 31 3A 35 31 3A 33 33 20 47 4D 54  002 21:51:33 GMT
0D 0A 4D 49 4D 45 2D 56 65 72 73 69 6F 6E 3A 20  ..MIME-Version:
31 2E 30 0D 0A 41 63 63 65 70 74 3A 20 2A 2F 2A  1.0..Accept: */*
0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F  ..User-Agent: Mo
7A 69 6C 6C 61 2F 32 2E 30 20 28 63 6F 6D 70 61  zilla/2.0 (compa
74 69 62 6C 65 3B 20 4D 53 20 46 72 6F 6E 74 50  tible; MS FrontP
61 67 65 20 34 2E 30 29 0D 0A 48 6F 73 74 3A 20  age 4.0)..Host:
77 77 77 2E 58 58 58 58 58 58 58 58 0D 0A 41 63  www.XXXXXXXX..Ac
63 65 70 74 3A 20 61 75 74 68 2F 73 69 63 69 6C  cept: auth/sicil
79 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74  y..Content-Lengt
68 3A 20 30 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E  h: 0..Connection
3A 20 4B 65 65 70 2D 41 6C 69 76 65 0D 0A 43 61  : Keep-Alive..Ca
63 68 65 2D 43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D  che-Control: no-
63 61 63 68 65 0D 0A 0D 0A                       cache....

[**] WEB-FRONTPAGE _vti_rpc access [**]
11/14-16:55:08.076507 12.217.179.244:3340 -> 170.129.50.3:80
TCP TTL:112 TOS:0x0 ID:2695 IpLen:20 DgmLen:430 DF
***AP*** Seq: 0xB719B9B  Ack: 0x2811ADC9  Win: 0xFFFF  TcpLen: 20
50 4F 53 54 20 2F 5F 76 74 69 5F 62 69 6E 2F 73  POST /_vti_bin/s
68 74 6D 6C 2E 65 78 65 2F 5F 76 74 69 5F 72 70  html.exe/_vti_rp
63 20 48 54 54 50 2F 31 2E 31 0D 0A 44 61 74 65  c HTTP/1.1..Date
3A 20 54 68 75 2C 20 31 34 20 4E 6F 76 20 32 30  : Thu, 14 Nov 20
30 32 20 32 31 3A 35 31 3A 33 34 20 47 4D 54 0D  02 21:51:34 GMT.
0A 4D 49 4D 45 2D 56 65 72 73 69 6F 6E 3A 20 31  .MIME-Version: 1
2E 30 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20  .0..User-Agent:
4D 53 46 72 6F 6E 74 50 61 67 65 2F 34 2E 30 0D  MSFrontPage/4.0.
0A 48 6F 73 74 3A 20 77 77 77 2E 58 58 58 58 58  .Host: www.XXXXX
58 58 58 0D 0A 41 63 63 65 70 74 3A 20 61 75 74  XXX..Accept: aut
68 2F 73 69 63 69 6C 79 0D 0A 43 6F 6E 74 65 6E  h/sicily..Conten
74 2D 4C 65 6E 67 74 68 3A 20 34 31 0D 0A 43 6F  t-Length: 41..Co
6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 70 70 6C  ntent-Type: appl
69 63 61 74 69 6F 6E 2F 78 2D 77 77 77 2D 66 6F  ication/x-www-fo
```

```
72 6D 2D 75 72 6C 65 6E 63 6F 64 65 64 0D 0A 58  rm-urlencoded..X
2D 56 65 72 6D 65 65 72 2D 43 6F 6E 74 65 6E 74  -Vermeer-Content
2D 54 79 70 65 3A 20 61 70 70 6C 69 63 61 74 69  -Type: applicati
6F 6E 2F 78 2D 77 77 77 2D 66 6F 72 6D 2D 75 72  on/x-www-form-ur
6C 65 6E 63 6F 64 65 64 0D 0A 43 6F 6E 6E 65 63  lencoded..Connec
74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C 69 76 65  tion: Keep-Alive
0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F 6C 3A  ..Cache-Control:
20 6E 6F 2D 63 61 63 68 65 0D 0A 0D 0A 6D 65 74   no-cache....met
68 6F 64 3D 73 65 72 76 65 72 2B 76 65 72 73 69  hod=server+versi
6F 6E 25 33 61 34 25 32 65 30 25 32 65 32 25 32  on%3a4%2e0%2e2%2
65 32 36 31 31 0A                                e2611.
```

3. Probability the source was spoofed:
It's very unlikely the source was spoofed. This is an attack that requires an established TCP connection, which is what both of these packets appear to be part of. The attacker is still in the information gathering stages so she would need to be able to get data back. While it's possible that the IP has been spoofed and the attacker has arranged to have the traffic pass through a sniffer – this is unlikely because of its complexity and the dynamic nature of traffic flow.

4. Description of the Attack:
Microsoft FrontPage was originally developed by Vermeer Tech Inc. - a company that was purchased by Microsoft shortly after FrontPage 1.0 emerged. FrontPage extensions allow a departure from the usual method of updating a web server which involves editing files on a local workstation and then uploading them to the server. Instead the extensions permit an administrator to update content using the HTTP POST method in combination with the Common Gateway Interface (CGI). FrontPage extensions can be used in conjunction with Netscape, Apache or IIS servers. Full functionality can only be utilized in conjunction with Microsoft IIS.

There are twelve CVE entries relating to FrontPage extensions. They can be found here:
http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=frontpage+extensions

The snort signature for _vti_rpc references Security Focus bugID 2144 which describes a denial of service that can be exploited by supplying unexpected data to one of Front Page server extension functions.

Several of the above could have been the goal of our attacker. It appears she was trying to obtain information about the server and FrontPage extensions. This may have been part of an attempt to upload new content to the server or it could have been an effort to exploit one of vulnerabilities above.

5. Attack Mechanism:
The attacker attempted to access two files. _vti_inf.html contains basic configuration

information about Front Page extensions including the version and location on the server. _vti_rpc is a server binary that can be used to establish a protocol for communication between client and server and informs the client of what functions the server provides. It is generally accessed prior to modifying web server content.

The post data in the _vti_rpc alert corresponds exactly to that listed in Sonzi's "Using Webfolders" paper. The expected response includes the server version (the client may have already obtained this in _vti_inf.html) which can be used to communicate with the web server and possibly modify content or perform administrative tasks.

6. Correlations:

[1] Edward, Perry. "Re: More Microsoft debri".
http://www.insecure.org/sploits/Microsoft.frontpage.insecurities.html
A short email that describes _vti_inf.html and _vti_rpc. This email is archived all over the place and I kept bumping into it while I was searching for information on Google.

[2] Fugjostle. "Guide to IIS Exploitation"
http://g0tr00t.mson.org/releases/fugjostle/docs/ReD2.txt
Provides a short description about _vti_inf.html and _vti_rpc and explained the origins of Front Page,

[3] Microsoft. "Files and permissions on Apache"
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/sharepnt/proddocs/admindoc/owsk02.asp
This has some information on front page extensions and a bunch of charts describing what permissions should look like on a Unix system.

[4] "Microsoft IIS Front Page Server Extension DoS Vulnerability". Dec 22, 2000.
http://www.securityfocus.com/bid/2144/discussion/
Discusses a front page DoS vulnerability. This is what the snort alert references but it doesn't seem to have much to do with the alerts captured in our log file.

[5] Microsoft. "Security"
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vidref98/html/viconsecurity.asp
Best practices for using front page extensions. Explains appropriate permissions settings for dll files.

[6] Sonzi. "Using webfolders"
http://www.xato.net/Reference/webfolders.txt
An excellent reference. Explains different components of Front page and how the relate. Also goes into how to secure front page extensions and how to exploit insecure configurations. This article shows what POST data to _vti_rpc might look like (exactly as it does in our alert) and explains what goals an attacker might have for doing a GET request on _vti_inf.html.

[7] Zelewski, Michael; Stearns, William. p0f - Passive OS Fingerprinting Tool
http://www.stearns.org/p0f/
A handy tool for analyzing libpcap formatted data to find operating system information.

DShield Reports:
IP Address: 12.217.179.244
HostName: 12-217-179-244.client.mchsi.com
No reports.

AT&T WorldNet Services (12.0.0.0/8)
400 Interpace Parkway
Parsippany NJ 07054

The layout of the hostname suggests dialup or DSL. Given that this IP is registered to AT&T either seems likely.

7. Evidence of Active Targeting:
Since there are only two alerts from the source address it seems likely that it is actively targeting the web server. Given that there is a web server at this address, 170.129.50.3 appears to be the actual IP. Perhaps the attacker is a disgruntled customer or employee. It's also possible the attacker didn't intend to attack at all but was merely using FrontPage to browse. It's difficult to tell from the alerts since at most the attacker gained information about the web server. Most likely the alerts were a prelude to modification of server content since that is Front Page's primary purpose and there is no evidence of a relationship between the source (looks like a dialup or DSL user) and the destination (a corporate web site).

8. Severity:

The severity of the attack will be calculated with the following formula:

severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Each value will be ranked on a scale from 1 (lowest) to 5 (highest).

Criticality – The destination IP address does not appear to have been modified since it currently refers to a corporate web server. This web server seems to be a means for the corporation to market its products. This makes it a critical asset and rates a **5**.

Lethality – The attacker was only gathering information that may or may not have led to a future attack. The information would only have been useful for this particular web

server. However, given the number of vulnerabilities in Front Page – the attacker would have had a number of choices in how best to continue toward her goal. This calls for a **2.**

System Countermeasures – It is suggested by the way the second packet follow the first in standard FrontPage query form that the request for _vti_inf.html was successful. The TCP ack numbers do increase between the two packets but the increase is by 887235. This would mean that 887235 bytes were passed from the destination to the source in the .26 seconds between the _vti_inf.html query and the _vti_rpc POST. That is unlikely to be accurate given that the maximum Ethernet packet size is 1500 bytes and that would involve at least 592 packets. Also, other ack and sequence numbers for other entries in the log file appear to have been modified.

Since a number of other hosts tried to perform the same attack on this server's front page extensions and some of them tried multiple times – it appears that they failed. However, given the number of attacks from so many different sources, it seems safe to say that this server does have Front Page extensions loaded. Given the number of vulnerabilities in the extensions and the difficulty in securing them this site shows a disturbing willingness to trade security for convenience. This calls for a **3.**

Network Countermeasures – The number of outbound GNUtella alerts indicates either a poorly enforced or an inadequate acceptable use policy. However, GNUtella appears to be the only outbound alert. There is no evidence that any attack performed in this file succeeded. While it's still possible some attacks were successful, given their simplicity and visibility it seems unlikely we wouldn't have seen something. The site also seems to have some form of intrusion detection. This seems to rate a **3.**

Note that this is different from the first detect even though it is the same network. This is because I made the first assessment based on the assumption that I could see all the traffic and all the attacks had failed. Johnny Calhoun corrected this, pointing out that only traffic generating snort alerts was captured. While the lack of outbound alerts suggests the attacks failed, there is much more room for doubt than I thought in the first detect.

Severity:
(5+2) – (3+3) = 1

9. Defensive Recommendations:
- Do not use or even install Front Page extensions on a FAT file system. FrontPage determines who may perform a given request by the ACLs on dynamic linked libraries (dll) in the _vti_bin directory. Since there are no ACLs on a FAT file system, these files are impossible to secure.
- Remember that FrontPage extensions only need to be installed to be exploited. Even if

they have never been used, the site may still vulnerable.
- Ensure the FrontPage web is password protected. If the authorized author doesn't need to enter a password, then neither does anyone else.
- Ensure ACLs on dll files in the _vti_bin directory are restricted to authorized users. Even read access to some files (particularly author.dll and admin.dll) is dangerous.
- Consider uploading web content using secure shell or an encrypted terminal server session.

10. Multiple choice question:
How do you securely operate Front Page extensions on a FAT file system?
a. Ensure permissions on dll files in the _vti_bin directory are set so only authorized users may access the dll files
b. Password protect the FrontPage web
c. Put the Front Page web on a different partition than operating system files
d. All of the above
e. It's not possible to secure Front Page on a FAT file system

Answer: e

Hopefully everyone will get this one – I think most people applying for this certification know FAT file systems don't have ACLs - but at least we got to sneak in some ways to secure a FrontPage server. Even c is a good best practice.

Questions:
I posted this one to the intrusions mailing list as well (6 Jun at 1454) and got a response from Rocker on 7 Jun at 1355. Many thanks to him. He had this to say:

```
Samuel,

Judged from the snort alert, I suspect that two mentioned alarts ar of 2
different TCP stream from the host. The logic is that

(1) TCP source ports are different. ==>2 different TCP stream
(2) it is detecte by the snort rule because of established session (after
TCP handshake). ==> this is not SCAN.
(3) Most of the modern OS , the ISN of TCP will be highly randomized.
(http://razor.bindview.com/publish/papers/tcpseq.html)

The logic of victim's response to the stimulus based SEQ/ACK is not
conclusive.
Hope it helps....
```

Answer:
Perhaps I could have worded the 3rd part of this detect better. I did not mean to imply that I thought both alerts were part of the same TCP session. I meant that both packets appeared to be part of established TCP sessions, not necessarily the same one. In

Perry Edward's paper he describes the process the Front Page software goes through when posting to a server with Front Page extensions. First the software goes for _vti_inf.html, then tries to post to _vti_rpc. The _vti_inf.html files contains data that FrontPage needs to post data to the server. FrontPage processes the information and the does the post to vti_rpc.

Part 3 - Analyze This

Executive Summary

The following logs were reviewed as part of a security audit of GCIA University.

Alert Files	Size	Scan files	Size	Out of Spec Files	Size
alert.030501.gz	493999	scans.030501.gz	187806	OOS_Report_2003_05_01_31055	701443
alert.030502.gz	1289933	scans.030502.gz	1128995	OOS_Report_2003_05_02_28431	1387523
alert.030503.gz	5217006	scans.030503.gz	2195075	OOS_Report_2003_05_03_7239	993283
alert.030504.gz	3648489	scans.030504.gz	2585730	OOS_Report_2003_05_04_21395	1141763
alert.030505.gz	1455662	scans.030505.gz	1711158	OOS_Report_2003_05_05_25821	747523

The scan and alert files contain information captured by snort during the first 5 days of May 2003. The OOS files cover 30 April to 4 May. There were 1277731 alerts, 846774 scans and 12192 OOS alerts. Only 876658 alerts and 846660 scans were analyzed. The others entries were corrupted.

The most important thing that came to light during the audit is the two hosts that are probably compromised: MY.NET.97.48 and MY.NET.97.181. These systems should be investigated immediately.

There is a great deal of peer to peer traffic occurring on University networks. If it isn't in place already, the University should have a signed copy of the acceptable use policy for each network user. This policy should include a prohibition against using University networks to share copyrighted material. GCIA University should also ensure that network users are fully aware of the risks associated with the use of peer to peer applications.

The Snort rule set and preprocessors appear to be very out of date. A signature based IDS like Snort is much less effective without a current rule set. Plus, recent upgrades to Snort's engine and preprocessors, such as fragment and stream reassembly, have drastically improved the IDS's functionality. Upgrading to the latest version of Snort should be a priority.

It appears that ingress and egress filtering have not been implemented on the University perimeter. This is highly recommended to prevent address spoofing and abuse of protected system trust relationships from hosts outside the University. Also, a security policy where protocols are denied by default is highly recommended.

There appear to be some systems that aren't up to date and may have insecure configurations. Scanning systems with ISS or Nessus would be advisable to pinpoint vulnerabilities and insecure configurations.

Alerts:

	Internal Src	Internal Dst	External Src	External Dst	Total
Alerts	**Uniq Hosts**				
Incomplete Packet Fragments Discarded	2	66	99	3	355259
TCP SRC and DST outside network	0	0	198880	2192	208267
SMB Name Wildcard	0	40906	22474	1	174110
High port 65535 udp - possible Red Worm - traffic	78	109	163	222	27260
CS WEBSERVER - external web traffic	0	2	5319	1	24934
High port 65535 tcp - possible Red Worm - traffic	64	68	83	92	23629
Tiny Fragments - Possible Hostile Activity	1	21	21	900	13531
TFTP - Internal TCP connection to external tftp server	11	12	33	31	9337
spp_http_decode: IIS Unicode attack detected	528	174	275	706	8770
EXPLOIT x86 NOOP	0	147	168	0	6019
connect to 515 from outside	0	4873	3	0	5033
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC	12	0	0	21	5023
Null scan!	0	109	115	1	2460
spp_http_decode: CGI Null Byte attack detected	119	6	35	127	1823
Queso fingerprint	0	123	328	1	1577
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.	0	63	74	0	1562
MY.NET.30.4 activity	0	1	294	0	1343
Possible trojan server activity	21	179	48	24	921
MY.NET.30.3 activity	0	1	44	0	804
CS WEBSERVER - external ftp traffic	0	1	147	0	781
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC	3	0	0	11	746
IDS552/web-iis_IIS ISAPI Overflow ida nosize	0	580	455	0	717
SUNRPC highport access!	0	23	30	0	520
TFTP - Internal UDP connection to external tftp server	31	19	14	33	395
[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot	0	5	9	0	271
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.	0	11	11	0	194
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize	2	0	0	167	188
IRC evil - running XDCC	15	0	0	14	168
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot	0	5	6	0	149
External RPC call	0	149	4	0	149
NMAP TCP ping!	0	60	41	0	145
EXPLOIT x86 setuid 0	0	106	118	0	128
SNMP public access	0	9	5	0	98
NIMDA - Attempt to execute cmd from campus host	2	0	0	58	60
EXPLOIT x86 setgid 0	0	48	50	0	53
EXPLOIT x86 stealth noop	0	6	12	0	51

TCP SMTP Source Port traffic	0	12	3	0	34
Notify Brian B. 3.54 tcp	0	1	21	0	26
Back Orifice	0	26	2	0	26
Notify Brian B. 3.56 tcp	0	1	20	0	22
SMB C access	0	9	11	0	13
Probable NMAP fingerprint attempt	0	9	8	0	12
Attempted Sun RPC high port access	0	3	3	0	10
RFB - Possible WinVNC - 010708-1	4	4	4	4	8
[UMBC NIDS IRC Alert] K:lined user detected, possible trojan.	0	4	6	0	7
FTP passwd attempt	0	2	3	0	7
TFTP - External UDP connection to internal tftp server	1	4	4	1	6
DDOS shaft client to handler	0	2	2	0	4
TFTP - External TCP connection to internal tftp server	1	2	2	1	3
NIMDA - Attempt to execute root from campus host	1	0	0	3	3
SYN-FIN scan!	0	2	2	0	2
EXPLOIT x86 NOPS	0	1	1	0	2
[UMBC NIDS IRC Alert] Possible trojaned machine detected	0	1	1	0	1
site exec - Possible wu-ftpd exploit - GIAC000623	0	1	1	0	1
Bugbear@MM virus in SMTP	0	1	1	0	1
DDOS TFN Probe	0	1	1	0	1

The chart above lists the alert messages, unique sources and destinations and total analyzed. This served as a useful reference during the analysis. The alert messages that correspond to messages in the Snort 1.9.1 rule set are italicized.

Top 10 Alert Sources (by number of alerts)

	Internal SRC	Alerts	External Dest	Alerts	DNS Reverse Lookup
1	MY.NET.210.114	354801	216.39.48.127	14010	buildrack52.sv.av.com
2	MY.NET.201.58	13423	133.82.241.150	8412	cuapfs0.imit.chiba-u.ac.jp
3	MY.NET.235.110	9161	12.207.10.226	4965	12-207-10-226.client.attbi.com
4	MY.NET.201.38	4026	128.46.117.76	4872	civl1240pc2.ecn.purdue.edu
5	MY.NET.198.221	3926	67.161.246.193	3294	c-67-161-246-193.client.comcast.net
6	MY.NET.226.250	3457	24.45.157.41	2966	ool-182d9d29.dyn.optonline.net
7	MY.NET.201.42	1721	216.78.180.128	2639	adsl-78-180-128.lft.bellsouth.net
8	MY.NET.226.206	1427	218.141.54.99	2551	YahooBB218141054099.bbtec.net
9	MY.NET.223.114	1071	195.167.225.233	2032	Unresolved
10	MY.NET.233.134	890	143.248.115.88	1898	xide.kaist.ac.kr

Queries:

```
select srcip,count(*) as count from alerts where srcip LIKE 'MY.NET%' group by srcip order by count desc limit 10;
select srcip,count(*) as count from alerts where srcip NOT LIKE 'MY.NET%' group by srcip order by count desc limit 10;
```

Top 10 Alert Destinations (by number of alerts)

	Internal SRC	Alerts	External SRC	Alerts	DNS Reverse Lookup
1	MY.NET.100.165	25831	213.97.198.23	354775	23.Red-213-97-198.pooles.rima-tde.net
2	MY.NET.201.58	10637	64.202.103.12	106932	giving.head.for-money.net
3	MY.NET.234.82	4972	65.116.88.75	43804	75.88.116.65.sharpnet.net
4	MY.NET.201.38	3384	146.100.53.56	29559	Unresolved
5	MY.NET.226.250	2552	216.200.173.18	25217	Unresolved
6	MY.NET.24.34	1811	67.161.246.193	3944	c-67-161-246-193.client.comcast.net
7	MY.NET.226.206	1739	205.188.149.12	3926	undernet.irc.aol.com
8	MY.NET.201.42	1538	218.141.54.99	3456	YahooBB218141054099.bbtec.net
9	MY.NET.30.4	1347	65.120.111.17	1992	65-120-111-17.velocity.net
10	MY.NET.233.134	1245	66.42.68.210	1678	66-42-68-210.stkn.mdsg-pacwest.com

Queries:

```
select dstip,count(*) as count from alerts where dstip LIKE 'MY.NET%' group by dstip order by count desc limit10
select dstip,count(*) as count from alerts where dstip NOT LIKE 'MY.NET%' group by dstip order by count desc limit10
```

Alert #1

Incomplete Packet Fragments Discarded (355259 alerts)

There were 354768 Incomplete Packet Fragment Discarded alerts from MY.NET.210.114 to 213.97.198.23. This is 99.8% of the total for this alert. These alerts were generated between 11:45 on 3 May and 16:53 on 4 May. Most of the time there were 2 or 3 alerts per second. 213.97.198.23 resolves to 23.Red-213-97-198.pooles.rima-tde.net - this name strongly suggests a dial-up or broadband user. dshield.org reported no prior activity. The RIPE Whois registry says this IP is part of a /16 CIDR block registered to Telefonica De Espana in Spain.

In answer to a question about alerts like this posted to the snort users mailing list Marty Roesch explains that seeing lots of these alerts

```
"...means that you're using the defrag preprocessor instead of the newer
frag2 preprocessor..."[2]
```

He goes on to recommend upgrading to frag2 since the defrag preprocessor fails frequently and unpleasantly. Since this preprocessor has been deprecated for some time now it was not surprising that documentation on what these alerts might mean was a bit sparse. We can guess that the preprocessor attempts to perform fragment reassembly and if it is unable to find all the fragments for a given fragment ID then it will generate an alert. Since this preprocessor was deemed broken and has since been replaced it's hard to tell if these alerts are something worth investigating or a false positive.

Doug Kite noted activity like this in his practical and suggested that the packets were a result of hostile activity[3]. He correlated them with 155 portscan alerts and suggested that fragmented packets might be used to bypass a firewall or IDS. In our log files there were 64663 outbound scans from MY.NET.210.114. 64601 of these scans were to 213.97.198.23 and all were UDP scans to various ports. To a degree, the times correlate with our alerts. They began at 11:53 on 3 May and ended at 14:14 on 4 May. Unfortuneately there was nothing in the OOS files from either IP.

It is worth noting that there were also 4 outbound UDP tftp alerts (3 on 3 May and 1 on 4 May) from MY.NET.210.114 to 213.97.198.23 and 1 inbound from 213.97.198.23 to MY.NET.210.114. The outbound udp alert on 4 May correlates exactly with a scan alert at the same time. Perhaps the fragmented packets were generated as part of a data transfer between 2 very distant hosts.

Recommendations:
Given the number of alerts and outbound scans it might be worthwhile to talk to the owner of this system and find out what software she's running. This may be some multimedia application. It would also be a good time to upgrade snort, particularly the defrag preprocessor.

Alert #2

TCP SRC and DST outside network (208267 alerts)

This appears to be a custom alert designed to detect traffic with both addresses outside the protected range.

[2]Roesch, Marty. "http://www.ultraviolet.org/mail-archives/snort-users.2001/3408.html"
[3]Kite, Doug. Page 28

DstIP	Alerts	%
64.202.103.12	106932	51
65.116.88.74	43804	21
146.100.53.56	29995	14
216.200.173.18	25217	12

Alerts to the four destination IP addresses in the table above total 205948 (99%). All but 8 have port 6667 as a destination port. The alerts occurred between 11:43 and 11:55. There were 152947 distinct source IP addresses.

According to ARIN 64.202.103.12 (giving.head.for-money.net) is registered to an ISP in Western Australia, 65.116.88.74 (74.88.116.65.sharpnet.net) is registered to an ISP in Ohio, 146.100.53.56 (did not resolve) is registered by RIPE in Italy and ARIN says 216.200.173.18 (did not resolve) is from an ISP in Seattle Washington. Given the destination port for all the traffic and that all of these addresses appear to be assigned to ISPs they are most likely IRC servers.

This activity appears to be a scan in reverse. With the number of mangled alerts in the alerts files it's difficult to say with certainty that these alerts were accurately reported. Assuming they were, perhaps the srcIPs were spoofed and someone on MY.NET was trying to cause a denial of service on four IRC servers. This would explain the huge number of alerts in such a short period of time. This should not be possible with proper egress filtering but since this alert is included in our rule set - it is unlikely that proper egress filtering is in place.

Michael Wilkinson suggested that this alert would only be generated if a local system was misconfigured.[4] This does appear to be what's going on for about 500 or so of the alerts in our log files but not for these alerts. There are simply too many alerts from too many different sources. Plus having such a large number in such a short time - with very little at any other time - does not suggest a misconfiguration.

Recommendations:
Implement proper ingress and egress filtering on all perimeter routers. If this is not possible for some reason, (one would think this would have been implemented by now if it were possible) it might be worthwhile to track this person down using his MAC address. It does appear to be one person (or at least a few people in coordination) given the time from and the targeting of IRC servers. Perhaps by tracking the MAC address for these alerts and correlating it with the MAC address for other traffic the owner of this system could be tracked down and discouraged from doing this kind of thing.

[4]Wilkinson, Michael. Page 66

Alert #3

SMB Name Wildcard (174110)

Most likely the signature for this rule looked something like this:

alert udp any any -> $HOME_NET 137 (msg:"SMB Name Wildcard";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)[5]

It appears to have been taken out of the current snort rule set. Becky Bogle suggests that this is because this alert is frequently caused by benign activity, such as a Windows system attempting to obtain the Netbios name of other systems it communicates with[6].

There were 22474 distinct source IPs and 40907 distinct destination IPs associated with this alert. The top talker among the source IPs 133.82.241.150 with 8412 alerts (5%). These alerts occurred between 10:24 on 3 May and 22:55 on 4 May. The source port was consistently 54799. This is worth noting since queries using nbtstat generally use port 137 as source and destination port. 133.82.241.150 was associated with no other alerts, scans or OOS alerts.

133.82.241.150 resolved to cuapfs0.imit.chiba-u.ac.jp. dshield.org reported no activity. The JPNIC database reports that the IP is part of a block belonging to Chiba University - which agrees with the hostname.

dshield.org lists port 137 as the most popular port for network scans. It also lists 5 CVE vulnerabilities associated with port 137. The activity does not appear to be a scan at first glance since the destination addresses are not sequential and it takes place over a long period of time. However, there are 7862 different destinations and the traffic only occurs during two days - suggesting the source either achieved his goal or gave up.

Recommendations:
If possible prevent all Netbios traffic (ports135,137,138,139) from crossing the network perimeter. Another idea would be to prevent anonymous access to Netbios shares. Many attackers try to establish a null or anonymous session in an attempt to access improperly secured network shares. This reference explains how to disable anonymous session access: http://securecomputing.stanford.edu/alerts/lioten.html. Note that following the instructions in the reference may render necessary services unusable - some experimentation may be required.

[5]Staniford-Chen, Stuart
[6]Bogle, Becky. Page 26

Top 5 Alert Source ports under 1024 (internal source IP)

	SRC Port	Protocol/Top SrcIP(#alerts)	Alert Message(s)	Count
1	80	**HTTP** MY.NET.24.34(9)	Possible Trojan Server/possible Red Worm	29
2	25	**SMTP** MY.NET.24.22(12)	High port 65535 tcp - possible Red Worm	17
3	0	**N/A** MY.NET.210.114 (10)	Incomplete Packet Fragments Discarded	10
4	143	**IMAP** MY.NET.12.4 (5)	Possible Trojan Server Activity/Red Worm	5
5	443	**HTTPS** MY.NET.24.33 (5)	High port 65535 tcp - possible Red Worm	5

Sample queries

```
select srcport,srcip,dstip,count(*) as count from alerts where srcport < 1024 AND NOT dstport = 0
  AND srcip LIKE 'MY.NET%' group by srcport order by count desc limit 10;
select message,srcip,count(*) as count from alerts where srcport = 25 AND srcip LIKE MY.NET%'
  group by srcip order by count desc;
```

12 Selected Alert Destination Ports (internal dest IP)

	Dest Port	Protocol/Top Dest IP (#alerts)	Top Alert Message(s)	Count
1	137	**Netbios-**NS MY.NET.24.34 (1797)	SMB Name Wildcard (174088)	174092
2	80	**HTTP** MY.NET.100.165 (24931)	CS Webserver - external web traffic (24931)	32106
3	515	**Printer** MY.NET.70.199 (160)	connect to 515 from outside (5032)	5033
4	524	**NCP** MY.NET.30.3 (723)	MY.NET.30.3 (723)/MY.NET.30.4 (247)	970
5	21	**FTP** MY.NET.100.165 (781)	CS Webserver - external ftp traffic (781)	794
6	32771	**SunRPC** MY.NET.194.187(353)	SUNRPC Highport Access! (520)	530
7	139	**Netbios-SSN** MY.NET.190.93 (415)	EXPLOIT x86 NOOP (414)	428
8	1214	**Kazaa** MY.NET.194.13	Queso Fingerprint (249)	415
9	25	**SMTP** MY.NET.24.22 (70)	Queso Fingerprint (281)	318
10	110	**POP3** MY.NET.6.7 (190)	Queso Fingerprint (190)	201
11	27374	**SubSeven** MY.NET.202.14 (3)	Possible Trojan Server Activity (166)	166
12	111	**SunRPC** MY.NET.53.222(1)	External RPC call (149)	149

Sample Queries:

```
select dstport,dstip,count(*) as count from alerts where dstport < 1024 AND NOT srcport = 0
  AND dstip LIKE 'MY.NET%' group by dstport order by count desc limit 15;
select message,dstip,count(*) as count from alerts where dstport = 137 AND dstip LIKE 'MY.NET%'
  group by dstip order by count desc limit 50
select message,dstip,count(*) as count from alerts where dstport = 137 AND dstip LIKE 'MY.NET%'
  group by message order by count desc limit 50
```

7 Selected Alert Destination Ports (external dest IP)

	Dest Port	Protocol/Top Src IP (#alerts)	Top Alert Message(s)	Count
1	6667	**IRC** MY.NET.198.221 (3925)	TCP Source and Dest Outside Network (205501)	210531
2	5121	**Neverwinter** MY.NET.201.58(12952)	High port 65535 udp possible Red Worm traffic	12952
3	80	**HTTP** MY.NET.84.218 (398)	spp_http_decode: IIS Unicode attack (8228)	10261
4	69	**TFTP** MY.NET.201.42 (1721)	TFTP - Internal TCP connection to external server	5168
5	6661-6669	**IRC** MY.NET.97.128 (389)	Possible sdbot floodnet detected attempting to IRC	847
6	27374	**Subseven** MY.NET.220.50 (461)	Possible Trojan Server Activity (570)	571
7	33450	**Unk** MY.NET.238.78 (214)	High port 65535 tcp - possible Red Worm (214)	214

Sample Queries:

```
select message,dstip,dstport,count(*) as count from alerts where dstip NOT LIKE 'MY.NET%' group by dstport
  order by count desc limit 50;
select message,srcip,count(*) as count from alerts where dstport = 6667 AND dstip NOT LIKE 'MY.NET%'
  group by dstip order by count desc limit 50;
```

Alert IDS552

IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize

As suggested by the IDS552 (Arach-NIDS identifier) at the beginning of the alert message, the signature that generated this alert probably looks like this:

```
alert TCP $INTERNAL any -> $EXTERNAL 80 (msg: "IDS552/web-iis_IIS ISAPI
Overflow ida"; dsize: >239; flags: A+; uricontent: ".ida?"; classtype:
system-or-info-attempt; reference: arachnids,552;)[7]
```

This is similar to the signature in the standard snort ruleset except that it includes a dsize option and uses flags instead of the flow option. Since our alert message says "nosize", most likely the dsize option has been eliminated. This signature is designed to detect an exploit of a buffer overflow in Microsoft IIS's Index Server.

The two source IPs associated with this alert are MY.NET.97.181 (184 alerts) and MY.NET.97.48 (4 alerts). MY.NET.97.181 was most likely compromised since it generates alerts going to multiple, widely varying IP addresses over the course of a few seconds. It's difficult to imagine a legitimate reason for traffic like that, particularly since it continues for 2 hours. The status of MY.NET.97.48 is more difficult to determine. The four alerts could have occurred under legitimate circumstances but they are all to different destinations and the times are within 3 minutes of each other. There are also 3 NIMDA cmd attempts originating from this host.

Recommendations:
One or both of these hosts have probably been compromised. An investigation of these systems would be worthwhile. It might also be a good idea to scan MY.NET with ISS or Nessus to make sure other systems do not have well known vulnerabilities.

[7]Whitehats. http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552&view=signatures

Portscan Top 10s

	External Src	Reverse DNS Lookup	Count
1	152.1.193.6	chjlpc4.chem.ncsu.edu	15962
2	217.88.231.137	pD958E789.dip.t-dialin.net	13949
3	217.84.122.16	pD9547A10.dip.t-dialin.net	11688
4	198.144.65.56	nt-001-00055.greenapple.com	9160
5	64.212.144.139	unresolved	8313
6	80.161.34.13	0x50a1220d.kd4nxx15.adsl-dhcp.tele.dk	8244
7	66.130.208.97	modemcable097.208-130-66.que.mc.videotron.ca	7663
8	213.204.66.141	unresolved	7040
9	208.163.46.185	port0185-cvx-pmbk.cwjamaica.com	6939
10	12.16.131.99	unresolved	6932

	Internal Src	Count
1	MY.NET.210.114	64663
2	MY.NET.240.62	39797
3	MY.NET.87.50	32582
4	MY.NET.250.98	29274
5	MY.NET.97.190	26833
6	MY.NET.1.3	21844
7	MY.NET.234.158	20909
8	MY.NET.205.150	16738
9	MY.NET.153.152	15298
10	MY.NET.225.230	13686

	External Dst	Reverse DNS Lookup	Count
1	213.97.198.23	23.Red-213-97-198.pooles.rima-tde.net	64601
2	64.39.186.133	dsl-cust-133.openweb.ca	1779
3	66.66.126.241	roc-66-66-126-241.rochester.rr.com	1732
4	66.167.144.245	h-66-167-144-245.MCLNVA23.covad.net	1624
5	24.42.0.66	unresolved	1620
6	68.165.25.243	h-68-165-25-243.PHLAPAFG.covad.net	1570
7	68.13.93.150	ip68-13-93-150.om.om.cox.net	1217
8	12.245.31.155	12-245-31-155.client.attbi.com	1212
9	68.81.50.22	pcp01413723pcs.potshe01.pa.comcast.net	1185
10	68.82.22.172	pcp03173548pcs.fairvw01.pa.comcast.net	1178

	Internal Dest	Count
1	MY.NET.132.26	15967
2	MY.NET.234.82	923
3	MY.NET.249.194	457
4	MY.NET.238.230	310
5	MY.NET.207.254	285
6	MY.NET.86.66	235
7	MY.NET.218.254	213
8	MY.NET.6.7	206
9	MY.NET.211.26	145
10	MY.NET.252.110	122

Sample Queries:

```
select srcip,count(*) as count from scans where srcip NOT LIKE '130.85%' group by srcip order by count desc limit 10;
select dstip,count(*) as count from scans where dstip LIKE '130.85%' group by dstip order by count desc limit 10;
```

Scan #1

152.1.193.6 (15962)

This is a comprehensive port scan that took place between 15:29 and 15:47 on 5 May. All scan activity was directed against MY.NET.132.26. From alerts that where generated while this scan was going on it appears that ports 69 (tftp) and 139 (Netbios-ssn) were open. Port 69 appears open since there was an outbound alert from MY.NET.132.26 on port 69 that was generated in the same second that the port was scanned. Port 139 appears open since an SMB C$ access alert was generated and the current signature requires an established connection to generate an alert. It is also possible that ports 111, 515, 445 and 80 were open. We can deduce this from alerts on the former 111 and 515 and scans from other sources to ports 445 and 80. Port 111 is unlikely to be open on a windows system since it is usually used for SunRPC.

According to ARIN 152.1.193.6 (chjlpc4.chem.ncsu.edu) belongs to a /16 CIDR block

registered to North Carolina State University. The host name suggests the Chemistry department. Dshield.org had no record of the IP. There was no activity from 152.1.193.6 that did not involve MY.NET.132.26, nor was there any sign of a catalyst from MY.NET that may have caused this.

Recommendations:
Considering the depth of this scan it might be worthwhile to check MY.NET.132.26 to make sure it's been properly secured. The presence of a tftp server and access to port 139 is definitely cause for concern. The SMB C$ Access alert is designed to detect attempts to access the C: file system[8]. Since this is usually the primary file system where system files are stored an attacker with access can cause a great deal of damage.

Scan #2

MY.NET.240.62 (64663)

MY.NET.240.62 scanned 25339 different destinations from 157 source ports to 3361 different destination ports. There were 39630 UDP scans and 167 TCP scans. Several of the TCP ports were known IRC server ports such as 6666 (8 scans) and 7000 (10 scans). The udp destination ports varied widely but the source ports were limited to 6257 (WinMX - 22777 scans) and 2468 (qip_msgd - 16853 scans). WinMX is a popular peer to peer file sharing utility but there is little information available on qip_msgd. Since the scans on port 2468 sometimes occur within seconds of the port 6257 scans perhaps 2468 is a custom configured file sharing port for WinMX or for some other utility with similar functionality.

MY.NET.240.62 was involved in 2 OOS alerts. The first was from 210.153.216.10. This address resolved to pl010.nas921.yokkaichi.nttpc.ne.jp. There was no record of activity at dshield.org. According to the JPNIC this IP is from a block belonging to Infosphere - a communications company (possibly an ISP) in Japan. The TTL of 107 (original was likely 128 - usually Windows or Linux) and source port of 2257 (Windows systems generally start at 1024 whereas Linux usually uses 32768-61000) suggest a Windows system[9]. The data is all unreadable but there are no TCP flags and the TCP length is 0. While this exact IP does not appear in any scans, 11 other addresses from Infosphere do appear. This may be a subtle attack on MY.NET.240.62 but most likely it's peer to peer traffic that became corrupted. It is worth noting that there are 13 SMB Name Wildcard alerts from Infosphere to various MY.NET IPs. These occur over several days and do not appear malicious.

[8]Whitehats. http://www.whitehats.com/info/IDS339
[9]Seifried, Kurt

The second OOS alert was from 65.24.141.137. This address resolved to dhcp065-024-141-137.columbus.rr.com. dshield.org had no record of activity. According to ARIN this IP is part of a /14 CIDR block belonging to Road Runner (AOL's broadband ISP subsidiary) in Virginia. In this case the destination port is 2468 but the protocol is TCP. The source port of 4872 and TTL of 111 again suggest a Windows system. There are no flags and the acknowledgement number and TCP length are both 0. The data portion contains 8 bytes of unreadable data. An acknowledgement number of 0 in a packet other than the initial SYN sometimes signifies an early version of the Nmap scanner. Nmap is an unlikely source since there are no other scans or alerts from this address. Most likely this is an attempt at file sharing that was somehow garbled.

MY.NET.240.62 was also the destination for 152 alerts. 147 of these were SMB Name Wildcard alerts - possibly attempts to gather information. Most likely the peer to peer traffic drew attention to the IP and made it a target for multiple attacks.

Recommendations:
Peer to peer file sharing has become so widespread that it may be unrealistic to try to shut it down. To insulate the University from possible lawsuits due to copyright infringement, an acceptable use policy prohibiting the use of peer to peer applications might be worthwhile. It also might be wise to inform students of the risks of file sharing. These risks include the potential for downloading trojans and viruses, possible legal repercussions of sharing copyrighted material and the possibility of making a student's PC a into a target for information gathering or attack.

OOS #1

68.54.93.181 (1528)

In addition to generating the largest number of OOS alerts this address generated 190 Queso fingerprint alerts and 189 scans all to MY.NET.6.7 and all on port 110 (pop3).

68.54.93.181 resolves to pcp01781292pcs.howard01.md.comcast.net. According to ARIN the IP is part of a /20 CIDR block assigned to Comcast Cable in Baltimore - an ISP. All packets from this address had the same fingerprint characteristics. The window size of 5840, MSS of 1460, Selective acknowledgement OK, unset window scale and packet length of 60 bytes suggest a Linux 2.4 kernel.

A Possible trojan server activity alert generated by a response from MY.NET.6.7 on port 80 to 66.157.117.78 tells us MY.NET.6.7 is probably a web server. This alert is most likely a false positive since it appears to be triggering on the 27374 (SubSeven) destination port. This was probably just selected as the ephemeral port for a connection to MY.NET.6.7 on port 80. This correlates with ??? The 167 SMB Name Wildcard alerts with MY.NET .6.7 as a destination suggest this is a Windows system with Netbios

services open.

The OOS alerts took place between 30 April at 00:09 and 4 May 13:11. Queso fingerprint alerts began on 12:37 on 1 May ended at 12:45 on 4 May. None of the times on the OOS alerts appear to correlate with the Queso fingerprint alerts. The OOS alerts do have both ECN bits set - which may have been enough to generate the alerts even though this is common for traffic from recent Linux systems.

The signature for the Queso Fingerprint attempt probably looks much like this:

```
alert tcp any any -> any any (msg: "Possible Queso Fingerprint attempt";
flags: S12;)
```
[10]

This signature was found to generate numerous false positives and an alternative was suggested by Max Vision to the snort-users mailing list and added to the Arach-NIDS signature database.[11]

```
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS29/scan_probe-Queso
Fingerprint attempt"; ttl: >225; flags: S12; classtype: info-attempt;
reference: arachnids,29;)
```
[12]

The new signature would eliminate the alerts above. Queso usually starts with a TTL of 255 whereas most Linux systems have a default TTL of 64. This is mostly academic since Nmap has replaced Queso as the tool of choice for remote OS fingerprinting.

Recommendations:
All these alerts seem to have been false positives. Looking at our OOS alerts it appears that the vast majority (10886 - 89%) have the ECN bits along with the SYN flag set. Most of these seem to have been generated by Linux systems doing something other than OS fingerprinting. This traffic seems to add another reason to look into upgrading snort and updating the signature set.

OOS #2

148.64.48.213 (223)

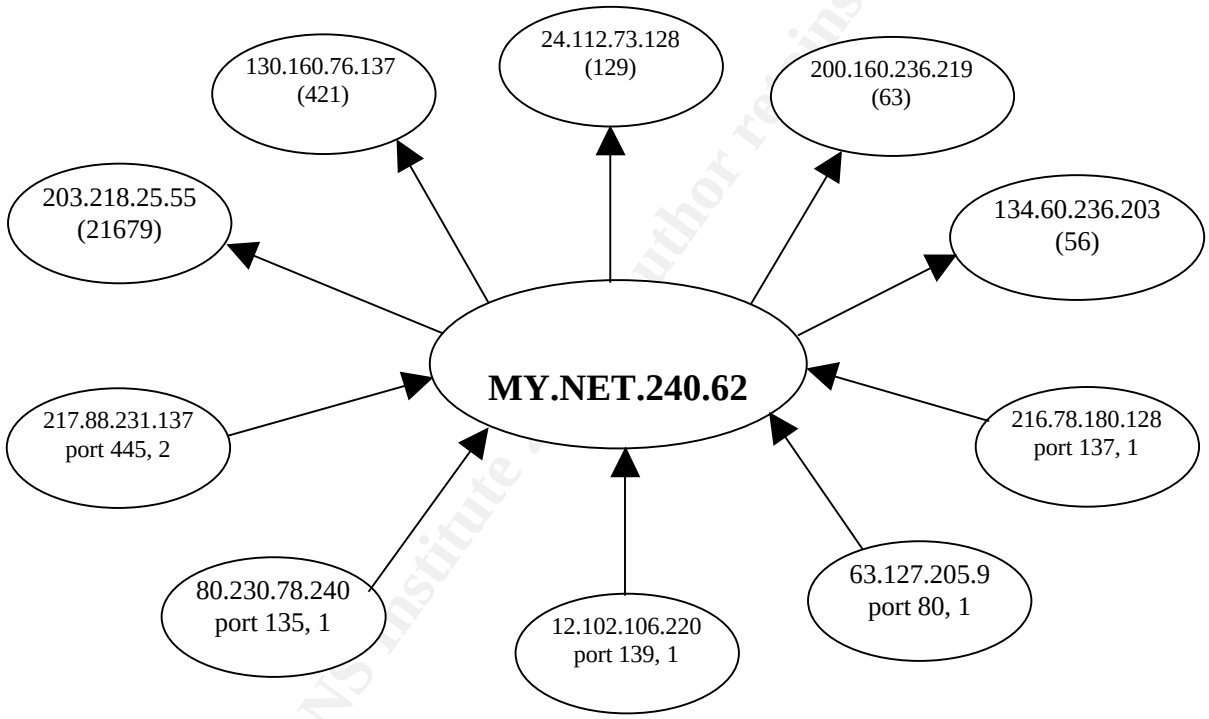
148.64.48.213 resolves to vsat-148-64-48-213.c050.t7.mrt.starband.net. dshield.org has no record of activity from this address. According to the ARIN whois registry this address is part of several CIDR blocks belonging to Spacenet Inc in Virginia. Their web site indicates that they provide commercial grade Satellite connectivity.

[10] Cipherdyne
[11] Neohapsis Archives
[12] Whitehats. http://whitehats.com/cgi/arachNIDS/Show?_id=ids29&view=signatures

There were other addresses that generated more OOS alerts but they appear to have been very similar to OOS #1. This IP generated no alerts or scans. However, there were 732 total OOS alerts, 4 alerts and 105 scans originating from the addresses registered to Spacenet Inc.

All traffic from 148.64.48.213 goes to MY.NET.235.202 on port 3516. This port is not listed on any port lists and a search on google.com yields little of value. dshield.org reported a spike in activity on this port on June 3 and 4 but has no insight on what services might run on it. The data portion of the OOS alerts indicates Kazaa - a peer to peer file sharing tool.

Link Diagram:



This diagram illustrates the risk of peer to peer file sharing. MY.NET.240.62 is used as an example. Arrows pointing out represent outbound peer to peer scans. Numbers in parentheses for outbound arrows indicate a connection count. Incoming scans show a port number and a count. Because of his extensive file sharing, MY.NET.240.62 has become a target. This graph could be shown to students to help them understand the dangers of the use of peer-to-peer applications.

Analysis Process:

Tools:
Redhat Linux 8.0
- MySQL 3.23.56
- Perl 8.0
- Perl Database drivers for MySQL 2.1017
- OpenOffice 1.0.1

Windows 2000
-Microsoft Excel 2000
-Microsoft Word 2000

I used the following shell script commands similar to these to concatenate, sort, eliminate duplicates and eliminate obviously mangled alerts in the alert files:

```
gzcat alert* > alert.all
sort -n alert.all | uniq > alert.uniq
grep "^05" alert.uniq > alert.whole
```

A similar process was done to concatenate, sort and eliminate duplicates among the scans. Then, using Gary Morris' perl scripts as a starting point, I wrote perl scripts to parse the files and insert them into MySQL. The database was created using Les Gordon's create_gciadb.sql script. I also used a simple drop_gcia.sql script so I could easily destroy and recreate the database while I was working on the scripts to parse the data. I used query.pl to create the table of alert messages. Brandon Newport's sql queries were extremely helpful in extracting data from the database. I also used Hee So and Les Gordon's practicals as a guide for making tables and the layout of the practical.

Lessons Learned:
I would have used a MySQLversion after 4.0 so I could have taken advantage of using the union keyword in my queries. This would have made it much easier to combine data from multiple tables.

I spent too much time on the parsing scripts. I think it would have been easier to turn all the files into comma separated lists and then load them into the database the way Les Gordon did. I also should have spent more time studying the alert files before writing the scripts. The alerts were mangled in more ways than I anticipated and I had to rewrite the parsing script several times to take unforeseen input into account.

Appendix
Perl Scripts

```
#!/usr/bin/perl
# parseAlerts.pl
# Written by Gary Morris
#modified by sca

use DBI;
use strict;
use warnings;

my $inFile = shift;
my $dbUser = "root";
my $dbPassword = "";
my ($line, $date, $time, $temp, $datetime, $srcIP, $srcPort, $dstIP, $dstPort,
$srcString, $dstString, $query);
my ($raw1, $raw2, $raw3);
my @alert;
my $info;
my $hosts;
my $sth;

open (MYFILE, "$inFile") || die "Cannot find file $inFile: $!\n";
open (DEBUG, ">/usr/local/sans/status.alert") || die "Can't write to status: $!\n";
open (MANGLED, ">/usr/local/sans/mangled.alert") || die "Can't write to mangled:
$!\n";

my $dbh = DBI->connect('DBI:mysql:gcia', $dbUser, $dbPassword) or die 'OUCH
$DBI::errstr\n';

while (<MYFILE>) {
        chomp;
        @alert = (split/\[\*\*\]/);
        unless (@alert == 3) {
                print MANGLED "Alert array is: $#alert $_\n";
                next;
        }
        ($raw1, $raw2, $raw3) = @alert;

        # parse date and time
        ($date, $time) = split/\-/, $raw1;
        ($time, $temp) = split/\./, $time;
        $date = "2003/" . "$date";
        $datetime = $date . " " . $time;

        $raw2 =~ s/\s//;
        $raw2 =~ s/\s$//;
        if ($raw2 =~ /spp_portscan/) {
                #Ignore portscan unless we got all of it
                unless ($raw2 =~ /End of portscan/) {
                        print MANGLED "Not end of portscan $_\n";
                        next;
                }

                ($srcIP,$info) = (split(/:/, $raw2))[1,2];
                $srcIP =~ s/.*from\s+//;
                unless (checkIP($srcIP)) {
```

```
                        print MANGLED "Invalid src or dst IP for portscan src: $srcIP
$_\n";
                        next;
                }

                $info =~ s/\s//;
                $info =~ s/\s$//;
                $info =~ /hosts\((\d+)\)/;
                $hosts = $1;
                $query = "INSERT INTO spp_scan_alerts (dttime,alert,hosts,srcip,info)
                                        VALUES
('$datetime','$raw2','$hosts','$srcIP','$info')";
        }
        else {
                ($srcString, $dstString) = split/\-\>/, $raw3;
                if (!defined($srcString) || !defined($dstString)) {
                        print MANGLED "Undefined src or dst string $_\n";
                        next;
                }

                ($srcIP, $srcPort) = split(/:/, $srcString);
                ($dstIP, $dstPort) = split(/:/, $dstString);

                $srcIP   =~ s/\s//g;
                $dstIP   =~ s/\s//g;

                unless (checkIP($srcIP) && checkIP($dstIP)) {
                        print MANGLED "Invalid src or dst IP src: $srcIP dst: $dstIP
strS: $srcString dstS: $dstString\n";
                        next;
                }

                if (!defined($srcPort) or $srcPort eq "") {
                        $srcPort = 0;
                }
                else {
                        $srcPort =~ s/ //;
                }
                if (!defined($dstPort) or $dstPort eq "") {
                        $dstPort = 0;
                }
                else {
                        $dstPort =~ s/ //;
                }

                $raw2 =~ s/\'//;
                $query = "INSERT INTO alerts
(dttime,message,srcip,srcport,dstip,dstport)
                VALUES ('$datetime','$raw2','$srcIP',$srcPort,'$dstIP',$dstPort)";
        }

        print DEBUG "Query is: $query\n";

        $sth = $dbh->prepare($query) or die "Line: $. - Can't prepare: $query. Reason:
$!";
        $sth->execute;
}
```

```
($dbh->disconnect or die "Can't disconnect from database. Reason: $DBI::errstr" and
undef $dbh);
close MYFILE;

sub checkIP {
        my $ip = shift;
        return ($ip =~ /^[\w\d]{1,3}\.[\w\d]{1,3}\.\d{1,3}\.\d{1,3}$/);
}

#!/usr/bin/perl
# parsescan.pl
# Written by Gary Morris
#   modified by sca

use DBI;
use strict;
use warnings;

my $inFile = shift;
my $dbUser = "root";
my $dbPassword = "";
my
($line,$month,$day,$time,$datetime,$srcIP,$srcPort,$dstIP,$dstPort,$srcString,$dstStri
ng,$info,$scantype,$flags);

open (MYFILE, "$inFile") || die "Cannot find file $inFile: $!\n";
open (DEBUG, ">/usr/local/sans/status.scans") || die "Can't write to status: $!\n";
#open (MANGLED, ">/usr/local/sans/mangled") || die "Can't write to mangled: $!\n";

my $dbh = DBI->connect('DBI:mysql:gcia', $dbUser, $dbPassword) or die 'OUCH
$DBI::errstr\n';

while (<MYFILE>) {
        $month = "05"; #hardcode 05 since all our data is from May
        my ($day,$time,$srcString,$dstString,$info) = (split(/\s+/,$_,7))[1,2,3,5,6];

        chomp($info);
        $day = sprintf("%02d",$day);
        $datetime = "2003/$month/$day $time";

        ($srcIP,$srcPort) = split(/:/,$srcString);
        ($dstIP,$dstPort) = split(/:/,$dstString);
        ($scantype,$flags) = (split(/\s+/, $info))[0,1];

        $flags = 0 unless defined($flags);

        my $query = "INSERT INTO scans
(dttime,srcip,srcport,dstip,dstport,scantype,flags,info)
                                        VALUES
('$datetime','$srcIP',$srcPort,'$dstIP','$dstPort','$scantype','$flags','$info')";
        print DEBUG "Query is: $query\n";

        my $sth = $dbh->prepare($query) or die "Line: $. - Can't prepare: $query.
Reason: $!";
        $sth->execute;
}

($dbh->disconnect or die "Can't disconnect from database. Reason: $DBI::errstr" and
```

```
undef $dbh);
close MYFILE;

#!/usr/bin/perl
# ParseOOS.pl
# Written by Gary Morris
#   Modified by sca
use DBI;
use warnings;
use strict;
use English;

my $inFile = shift;
my $dbUser = "root";
my $dbPassword = "";

open (MYFILE, "$inFile") || die "Cannot find file $inFile: $!\n";
open (DEBUG, ">/usr/local/sans/status.oos") || die "Can't write to status: $!\n";
open (MANGLED, ">/usr/local/sans/mangled.oos") || die "Can't write to mangled: $!\n";

my $dbh = DBI->connect('DBI:mysql:gcia', $dbUser, $dbPassword) or die 'OUCH
$DBI::errstr\n';

my $line;
my
($timestr,$srcString,$dstString,$datetime,$date,$time,$srcIP,$srcPort,$dstIP,$dstPort)
;
my ($proto,$TTL,$TOS,$ID,$headlen,$packlen,$ipopt);
my ($flags,$seq,$ack,$win,$paylen);
my ($tcpopt,$data);

while ($line = <MYFILE>) {
        $tcpopt = 0;
        $data = 0;

        #Get Time and IP/port info from first line
        if($line =~ m#^\d+/\d+#) {
                ($timestr, $srcString, $dstString) = (split(/\s+/,$line))[0,1,3];
                $datetime = gettime($timestr);
                ($srcIP, $srcPort) = split(/:/, $srcString);
                ($dstIP, $dstPort) = split(/:/, $dstString);
        }
        else {
                next;
        }

        #Forget about alert if any of this is missing
        unless (defined($datetime) && defined($srcIP) && defined($dstIP)) {
                print MANGLED "$line";
                while (<MYFILE>) {
                        print MANGLED;
                        last if /^=\+=\+=*/;
                }
                next;
        }

        $line = <MYFILE>;
        ($proto,$TTL,$TOS,$ID,$headlen,$packlen,$ipopt) = split(/\s+/,$line,7);
```

```
        $TTL = (split(/:/,$TTL))[1];
        $TOS = (split(/:/,$TOS))[1];
        $ID = (split(/:/,$ID))[1];
        $headlen = (split(/:/,$headlen))[1];
        $packlen = (split(/:/,$packlen))[1];
        chomp($ipopt);

        $line = <MYFILE>;
        if($line =~ /^Frag/) {
                $srcPort = 0;
                $dstPort = 0;
                $flags = 0;
                $seq = 0;
                $ack = 0;
                $win = 0;
                $paylen = 0;
        }
        else {
                ($flags,$seq,$ack,$win,$paylen) = (split(/\s+/,$line))[0,2,4,6,8];
        }

        $line = <MYFILE>;
        if($line =~ /^TCP/) {
                $tcpopt = $line;
                $tcpopt =~ s/TCP Options //;
                chomp($tcpopt);
        }
        while (<MYFILE>) {
                last if /^=\+=\+=*/;
        }
#Putting hex data in the database is not useful - decided to take it out
$data = 0;
#       if($line =~ /^[\dA-F]{2}/) {
#               $data = $line;
#               while (<MYFILE>) {
#                       last unless /^[\dA-F]{2}/;
#                       $data = "$data" . "$_";
#               }
#               $data =~ s/\'//g;
#       }

        my $query = "INSERT INTO oos
(dttime,srcip,srcport,dstip,dstport,ip_ttl,ip_tos,ip_id,ip_len,length,ip_flags,tcp_fla
gs,tcp_seqno,tcp_ackno,tcp_win,tcp_options,hexdata)
                VALUES
('$datetime','$srcIP',$srcPort,'$dstIP',$dstPort,'$TTL','$TOS','$ID','$headlen','$pack
len','$ipopt','$flags','$seq','$ack','$win','$tcpopt','$data')";
        print DEBUG "$query\n";

        my $sth = $dbh->prepare($query) or die "Line: $INPUT_LINE_NUMBER - Can't
prepare: $query. Reason: $!";
        $sth->execute;
}

($dbh->disconnect or die "Can't disconnect from database. Reason: $DBI::errstr" and
undef $dbh);
close MYFILE;
```

```
sub gettime {
        my $str = shift(@_);
        my $date;
        my $time;

        ($date, $time) = split/\-/, $str;
        ($time) = (split(/\./, $time))[0];
   $date = "2003/" . "$date";
   return "$date $time";
}

#!/usr/bin/perl
#Takes as input a file with all alert messages
#  Commented out lines can be used to query external hosts

use strict;
use warnings;
use English;
use DBI;

my $dbUser = "root";
my $dbPassword = "";
my ($query,$squery,$dquery,$bquery,$total);
my ($dbh,$sirc,$dst,$both);
my ($scnt,$dcnt,$bcnt,$ext);
#my ($scnt,$dcnt,$bcnt,$int);
my @row;

my $qsrc = shift;
open (MESSAGES, "$qsrc") || die "Cannot find file $qsrc: $!\n";
open (OUTPUT, ">/usr/local/sans/part3/queries/uniq") || die "Can't write to
output file: $!\n";
#open (OUTPUT, ">/usr/local/sans/part3/queries/extuniq") || die "Can't write
to output file: $!\n";
open (ALERTS, ">/usr/local/sans/part3/queries/alerts") || die "Can't write to
alerts file: $!\n";
$dbh = DBI->connect('DBI:mysql:gcia', $dbUser, $dbPassword) or die 'OUCH
$DBI::errstr\n';

while (<MESSAGES>) {

      next unless /\t/;

      ($query,$total) = (split(/\t/));
      $squery = "select count(distinct srcip) from alerts where srcip LIKE
'MY.NET%' and message = \'$query\'";
      $dquery = "select count(distinct dstip) from alerts where dstip LIKE
'MY.NET%' and message = \'$query\'";
# $squery = "select count(distinct srcip) from alerts where srcip NOT LIKE
'MY.NET%' and message = \'$query\'";
#   $dquery = "select count(distinct dstip) from alerts where dstip NOT LIKE
'MY.NET%' and message = \'$query\'";

   print ALERTS "$query\n";
```

```
      $src = $dbh->prepare($squery) or die "Line: $. - Can't prepare:
$squery. Reason: $!";
      $dst = $dbh->prepare($dquery) or die "Line: $. - Can't prepare:
$dquery. Reason: $!";
      $src->execute;
      $dst->execute;

      @row = $src->fetchrow_array;
      $src->fetchrow_array;
      $scnt = shift(@row);

      @row = $dst->fetchrow_array;
      $dst->fetchrow_array;
      $dcnt = shift(@row);

      print OUTPUT "$scnt,$dcnt,$total";
}
$dbh->disconnect;
```

References (for Analyze This):

Bogle, Becky. “GIAC Certification Practical.” 2001.
http://www.giac.org/practical/Becky_Bogle_GCIA.doc

Cipherdyne. “Diff for /psad/psad_signatures between version 1.4 and 1.5”. April 16 2001.
http://www.cipherdyne.com/cgi/viewcvs.cgi/psad/psad_signatures.diff?r1=1.4&r2=1.5

Common Vulnerabilities and Exposures. Mitre Corporation. Jun 2003.
http://www.cve.mitre.org

Distributed Intrusion Detection System.
http://www.dshield.org

Kite, Doug. “Intrusion Detection in Depth.” July 2002.
http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf

So, Hee. “Giac Intrusion Detection In Depth.” Feb 2002.
http://www.giac.org/practical/Hee_So_GCIA.doc

Gordon, Les. “Intrusion Analysis – The Director’s Cut.” May 2002.
http://www.giac.org/practical/GCIA/Les_Gordon_GCIA.doc

Morris, Gary. “Contemporary Intrusion Detection and Analysis.” Oct 2002.
http://www.giac.org/practical/GCIA/Gary_Morris_GCIA.doc

Neohapsis. “Ports List.” 20 May 2003.
http://www.neohapsis.com/neolabs/neo-ports/neo-ports.html

Neohapsis Archives. “Snort Users”. Mar 15 2001.
http://archives.neohapsis.com/archives/snort/2001-03/0317.html

Newport, Brandon. Level Two Intrusion Detection In Depth. May 2001.
http://www.giac.org/GCIA_500.php

Seifried, Kurt. “Passive OS Detection and Source Ports”. 10 Sep 2001.
http://www.seifried.org/security/network/20011009-passive-os-detection.html

Staniford-Chen, Stuart. “Re: IDS: Source port of Samba Scans?” 11 Mar 2000
http://www.shmoo.com/mail/ids/mar00/msg00065.shtml

ITS Information Security. “Yale University Windows 2000 Workstation Security Guidelines”.
http://www.yale.edu/its/security/new-

index.html?http://www.yale.edu/its/security/Procedures/Securing/NT/w2k/

Whitehats. “IDS29 Probe-Queso Fingerprint Attempt”. 2001.
http://whitehats.com/cgi/arachNIDS/Show?_id=ids29&view=signatures

Whitehats. “IDS 339 Netbios-SMB-C$Access.” 2001.
http://www.whitehats.com/info/IDS339

Whitehats. “IDS552 IIS ISAPI OVERFLOW IDA" . 2001.
http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids552&view=signatures

Wilkinson, Michael. GCIA Practical for SANS Darling Harbour.
http://www.giac.org/practical/michael_wilkinson_gcia.doc

Many thanks to my colleagues Dan Russell, Scott Higgins, Craig Taylor, and Chris Koentopp for proofreading this paper and for their helpful suggestions.