Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment Version 3.3

Andrew Evans

Turning the Worm - Preventing the spread of malicious code

In recent years malicious self-propagating code has become a significant threat to information infrastructure. Worms such as Code Red, Nimda and Slammer, to name a few, have cost organisations around the world millions of dollars and countless man hours. Some of the latest viruses such as Sobig and Fizzer have been found automatically connecting to websites to download keyloggers, irc bots, and proxy servers that can change at any time. We can expect to see more of this activity in future, worms will become more modular and will be able to exploit multiple vulnerabilities and even multiple system types (we have already seen this with the IIS/sadmind worm that targeted Microsoft IIS and Linux servers).

In 2001 and 2002 there was much discussion on the concept of the "Warhol" or "Flash" worm that can compromise the entire set of vulnerable hosts on the internet in a matter of minutes. A Warhol worm is one that is given a predetermined set of vulnerable or potentially vulnerable IP addresses when loaded onto its initial host. The set of vulnerable IP addresses is determined by traditional scanning methods and is intended to significantly accelerate the infection rate of the worm. The initial host compromises a group of these vulnerable addresses, dividing up the remaining addresses and assigning each compromised machine a set of addresses, this will continue until the list of vulnerable addresses has been exhausted. When all addresses in the list have been scanned and/or compromised, the infected machines begin scanning random IP addresses for other victims. The flash worm is an extension of this concept whereby the initial list of IP addresses comprises all vulnerable machines on the Internet (Staniford et al, 2002). Although it did not utilise these mechanisms, the Slammer worm showed the damage a fast propagating worm can have on the internet.

This document discusses the effects of active worms and the intrusion detection and prevention methods that may be used to prevent the spread of these worms. In the network intrusion detection arena, most attention is focused on the use of separate intrusion detection and prevention systems attached to the network. During my research into effective mechanisms for preventing the spread of active worms, I came to the conclusion that host based intrusion prevention systems were under-utilised and have a high efficiency rate in preventing or slowing down the spread of malicious code. I have given an overview of the different types of host based intrusion prevention systems, their efficiency in preventing worm infection or propagation and the issues involved in their deployment.

The Worms

Code Red

On the 12th of July 2001 the first Code-Red worm (Code-Red version 1) began spreading across the Internet, this version had limited impact due to the fact that it used a static seed in generating the random target IP addresses. On the 19th of July the second Code-Red variant appeared in the wild (Code-Red version 2). The second variant used a random seed in generating target IP addresses and affected hundreds of thousands of servers across the internet. Both of these worms were memory resident, and could be removed by rebooting the infected server (although

the vulnerability remained until patched).

On August 4th the worm known as CodeRedII was released, although this worm is entirely different to the original Code-Red version 1&2 worms the string "CodeRedII" appears in the worms code. While the original Code-Red worms had no lasting effects and were memory resident only, the CodeRedII worm installed a backdoor into the infected IIS server. CodeRedII is not memory resident, so unlike the original Code Red worms it is not removed by rebooting. (CAIDA, 2003)

Nimda

The Nimda worm was detected on September 18th, 2001 and was notable due to the fact it utilised five distinct propagation methods. These were: from client to client via email; from client to client via open network shares; from web server to client via browsing of compromised websites; from client to web server via exploitation of Microsoft Internet Information Server (IIS) directory traversal vulnerabilities and from client to web server via scanning for backdoors left behind by CodeRedII and sadmind/IIS worms. (CERT/CC, 2001) Although the Nimda worm was a significant threat, its existence was somewhat overshadowed by the events of September 11th, so it did not receive the media attention that many other worms did.

There have been a number of variants of the Nimda worm, and it is still active across the Internet, examination of any organisations web server logs will show scans from Nimda variants on a regular basis. On the 14th of July the Nimda worm is listed as being the second most prolific worm or virus of the past 24 hours on the website of Antivirus company Trend Micro (Trend Micro Enterprise Homepage, 2003).

Slammer

The Slammer worm began spreading at around 05:30 UTC on the 25st of January 2003. This worm was the fastest spreading worm ever seen in the wild; within 10 minutes of its initial release it had compromised 90% of all estimated vulnerable hosts on the Internet. The extremely fast spread of the Slammer worm can be attributed to two main reasons, firstly, the exploit code was encapsulated in a single UDP packet, and secondly, the worms scan rate was restricted only by the network bandwidth available to the infected machine.

The Slammer worm was restricted in the damage it caused in that it had no destructive payload and the worm was memory resident only, if the infected machine was rebooted the infection was removed. Also, as the slammer worm spread via the MS-SQL Monitor port (UDP 1434), a port rarely used for server communication on the Internet, it was easy for Internet Service Providers (ISPs) to simply filter traffic to this port at their border routers.

Although the Slammer worm caused no real damage to the systems that were infected, the Denial of Service (DoS) effects of the Slammer worm's propagation were so severe that Republic of Korea's internet infrastructure was paralysed and latency levels on the rest of the internet became severe. The DoS effects of the worm caused problems with a number of other systems, including Bank of America ATMs (Automatic Teller Machines) and airline reservation systems (Moore et al,

2003b).

Detection and Prevention

Standard network intrusion detection systems have limited usefulness in the case of another worm like Slammer. Recent studies have shown that it is next to impossible for human analysts to respond in time to prevent a worm reaching its 'critical mass'. Researchers have developed a model for the spread of typical active worms called the Analytical Active Worm Propagation (AAWP) model. They have also developed a variation of the model called the Local AAWP model to simulate those active worms that preferentially scan local networks over randomly selected networks (such as CodeRedII). These models are fairly accurate when tested against worms of recent past. The models show that while it is possible to detect a worm early in its propagation cycle, it would require the dedication of a significant number of unused IP addresses monitored by intelligent sensors capable of separating worm activity from network scans and denial of service attacks (Chen et al, 2003).

Chen, Gao and Zwat theorise that in order to detect a Code Red v2 like worm, the average time to detect the worm would be over 2 hours if monitoring 2^{16} IP addresses (1 class B network), and decreases to around 2 minutes when monitoring 2^{24} IP addresses (1 class A network). They also state that it would probably be necessary to require the sensor to receive several scans before alerting to reduce the number of false positives. This requirement further reduces the performance of such a detection system.

Three primary methods of reducing the threat posed by active worms have been identified; these are prevention, treatment and containment. Prevention is defined as 'a method of reducing the size of the vulnerable population, thereby limiting the spread of a worm outbreak' (Moore et at, 2003b). This can basically be translated to mean reducing the number of vulnerabilities inherent in today's software. Although this is a laudable goal, it is unlikely we will see the end to widespread software vulnerabilities for quite some time, if ever.

Treatment is simply the removal of the infection or vulnerability from a system, patching and malware removal programs are both methods of treatment with respect to active worms. These mechanisms, which are currently the most commonly used methods of defending against worm outbreaks, have so far failed to protect us from the widespread effects of active worms. Although most vulnerabilities exploited by worms are well known, there are still many unpatched systems on the internet, and the time taken to disinfect and patch all infected or vulnerable hosts after an outbreak is virtually infinite. This is proven by the fact that there are still significant numbers of hosts on the Internet infected with Nimda and Code Red variants.

The third method of reducing the threat, containment, means the isolation of infected hosts from vulnerable hosts, using such methods as firewalls and content filters. Containment has been an effective strategy in preventing the spread of active worms in the past, for example, a significant factor in mitigation of the Slammer worm's effects was the fact that many Internet Service Providers (ISP's) and backbone providers began blocking traffic to UDP port 1434 across the internet (Moore et al, 2003a).

Intrusion Analysts and Incident Response Teams are now far more likely to be dealing with the effects of self-propagating malicious code than with the direct actions of hackers. Over the past few years there have been a number of highly dangerous worms and we can expect worse to come. For this reason, the focus of real time Intrusion Prevention Techniques must be on preventing the spread of malicious code.

Network Intrusion Prevention has, to some, been a buzzword that vendors have bandied about in the hope of selling highly priced black boxes to clients stunned by a barrage of jargon. This is, for the most part, no longer the case. There are now a number of companies selling Network Intrusion Prevention solutions that have a place within today's organisations, and there are even a few free, reliable solutions that can be used. In addition to these, there are a multitude of host based intrusion prevention technologies that can be used to increase the 'defence in depth' of a network environment.

In recent years Network Intrusion Detection Systems (NIDS) have become much more mature and therefore are more of a commodity product (although the cost of deploying an IDS can still be significant). Network Intrusion Prevention Systems (NIPS) pose a completely different set of challenges and techniques for vendors, the issues that cause problems in NIDS deployment are catastrophic problems in an NIPS. A successful Intrusion Prevention implementation requires 100% availability, resistance to false positives, and the ability to handle high bandwidth network traffic with zero packet loss. For these reasons, IPS technologies are often built as hardware appliances based on Application Specific Integrated Circuits (ASIC) rather than as software solutions. As a result, where Network Intrusion Detection in high bandwidth environments is relatively easily performed using free, open source software on commodity hardware, NIPS in this sort of environment requires commercial solutions (Wickham, 2003).

Some of the problems associated with Network IPS are cost, network latency and packet loss. The specialised nature of the Integrated Circuits in IPS hardware devices make them expensive, and they are best deployed in dual hot standby configurations to provide 100% availability, further increasing the cost of deployment. The time a Network IPS takes to process the packets can increase network latency, and the NIPS must be carefully tuned to the environment to minimise this.

Network Intrusion Prevention

Some of the most common technologies being promoted for Network Intrusion Prevention in enterprise networks are Layer 7 Switches and Inline-NIDS (Network Intrusion Detection Systems). There are also some IDS response mechanisms that seek to prevent intrusions without directly filtering traffic on the network. The different types of solution, the options available for deploying them, and their usefulness in preventing the spread of malicious code are described below.

Layer 7 Switches

Layer 7 switches have been available for the past three to four years and were

developed for intelligent switching, load balancing and bandwidth management. These switches can examine the application layer of a packet to make switching or routing decisions based on packet content. Recently a number of vendors have began to build Intrusion Prevention capabilities into these switches or provide software to upgrade existing switches for Intrusion Prevention. Layer 7 switches have the advantage that they are optimised for packet inspection in high bandwidth, they are built on custom hardware and often can be deployed in pairs in a fail-over configuration so if one fails the other can take its place.

Companies offering Layer 7 switches include Foundry Networks and TopLayer, TopLayer now focuses entirely on Intrusion Prevention (Desia, 2003). The effectiveness of these systems in preventing the spread of active worms is difficult to judge. Due to the high cost of the systems, procurement of a system for testing was impractical. I was unable to find any independent documentation on the effectiveness of these systems in preventing worm outbreaks, although Network Computing Magazine is currently evaluating NIPS systems in preparation for publishing an article in September (Fratto, 2003). I suspect that these systems will be found to be extremely effective in preventing the propagation of active worms, however it may be advisable to wait until the technologies are more mature and well evaluated before deploying them in an enterprise environment.

Inline-NIDS

Inline-NIDS is an extension of Network Intrusion Detection technology and is currently the most common type of Intrusion Prevention System available. There are a number of commercial solutions available from NIDS vendors who have adapted their existing technology for inline use and a number of vendors are developing ASIC based solutions for inline-NIDS (Desia, 2003).

There are also two open-source inline-NIDS solutions based on the Snort Intrusion Detection System. The first and earliest of these is called Hogwash. Hogwash was developed in response to the dilemma that some administrators faced when the application of security patches to IIS web servers broke critical functionality on the server. In order to prevent frequent compromise of an unpatched server it was necessary to block or alter packets attempting to exploit the vulnerabilities. On a filtering machine in front of the vulnerable server, it has the capability to drop packets based on standard snort signatures or to modify the packet to neutralise the destructive content (Hogwash, 2003).

The second solution, snort-inline, is a version of snort combined with an IPtables firewall. Snort-inline was developed for the Honeynet project to filter outgoing malicious traffic from a honeypot without blocking normal traffic (making the honeypot appear to be a normal machine while preventing it from participating in attacks on other machines). The Honeynet project provide a precompiled version of snort-inline for redhat Linux and a set of rules to drop malicious outgoing traffic from a honeypot while allowing any traffic in. Although snort-inline was developed to prevent outgoing malicious traffic, it is only a matter of swapping a couple of variables in the rule set to allow it to filter incoming traffic (The Honeynet Project, 2003).

Commercial inline-NIDS products vendors include Internet Security Systems, Network Associates and TippingPoint technologies.

Inline-NIDS products suffer from many of the same drawbacks as traditional NIDS systems; in fact, many of the drawbacks are enhanced. False positives are a significant hurdle in the effective use of Intrusion Prevention. Because a traditional NIDS does not alter traffic in any way false positives are acceptable to some extent, however an inline-NIDS should not alter or block normal or benign traffic at all. In spite of this, Inline-NIDS systems based on traditional signatures could have a role in preventing the spread of active worms. Most worms exploit well known vulnerabilities, if we have reliable IDS signatures specific to the vulnerability then we can stop worms attempting to exploit those vulnerabilities. In addition, it could be possible to detect and stop worms attempting to spread from internal hosts by using the portscan detector capability provided by most Intrusion Detection Systems.

IDS Response Mechanisms

Some Intrusion Detection Systems can be configured with a pseudo-Intrusion Prevention functionality to close a malicious connection by either blocking the connection at a firewall or responding to actively close the connection via TCP reset or ICMP unreachable packets. Recent versions of the Snort IDS have come with the flexible response functionality. Snort's flexible response mechanisms allow for the following responses to malicious activity:

RST_SND, this response generates a TCP reset directed at the source of the threat, effectively causing the source to terminate the current connection.
RST_RCV, this response generates a TCP reset directed at the destination of the threat, preventing the destination from responding to the event.
RST_ALL, this response generates a TCP reset in both directions causing the source and destination to close the connection.
The ICMP_NET response generates an ICMP net unreachable
(ICMP Code 3 Type 0) message to the sender, advising the sender that the host it has attempted to communicate with is unreachable.
The ICMP_HOST response generates an ICMP host unreachable
(ICMP Code 3 type 1) message to the sender of the packet, informing the sender that the host they wish to communicate with is not reachable.
The ICMP_PORT response generates an ICMP port unreachable
(ICMP Code 3 type 3) message informing the sender that the UDP port they are trying to connect to at the destination is not listening.
The ICMP_ALL sends all of the above ICMP messages
(Roesch & Green, 2003)

The other IDS flexible response mechanism, interaction with a firewall, can be implemented in Snort with Snortsam. Snortsam is an extension to the Snort Intrusion Detection system that enables blocking or shunning of IP addresses at the firewall and is compatible with a variety of firewalls including Checkpoint Firewall-1, Cisco Pix, IPFilter, Iptables and IPChains. It is also capable of interacting with Cisco Routers by changing ACL's. Snortsam is compiled into the IDS and an agent is run on the firewall, IDS rules can then be written to shun IP addresses as required (Knobbe, 2003).

The major disadvantage of IDS Response mechanisms in preventing the spread of active worms is that we are shutting the gate after the horse has bolted, so to speak. The malicious traffic has already passed by the sensor and most likely reached the target host before the IDS can respond. Also, the spoofed TCP Reset and ICMP unreachable packets are not guaranteed to reach their destinations or to be accepted by the destination hosts. IDS Response is also considered extremely risky when blocking or closing connections before they are fully established. It is possible for a malicious person who is aware of the response mechanisms to cause blocking of vital IP addresses such as DNS servers and routers. In the case of a worm such as Slammer, IDS response will have no chance of stopping the initial infection; the exploit code is encapsulated in 1 UDP packet, so ICMP unreachable packets and firewall blocking will be useless. However, after the initial infection, the firewall blocking mechanisms could be useful in preventing the infected host from infecting other hosts outside the network.

Host based Intrusion Prevention

Host Based Intrusion Prevention systems (often referred to as host based firewalls) have been around for a long time, but have not received a lot of attention despite the fact that a well implemented Host based solution can be of great value in preventing intrusions. Many host-based solutions are application and operating system aware and are capable of tracking system calls and enforcing security configurations. Host based solutions are of great value in that they add an extra layer of defence within the organizations firewall. Many intrusions that can be traced to firewall misconfigurations could have been stopped or the misconfiguration detected by a host based IDS. Host based IPS products range from simple packet filters such as OpenBSD's IPFilter to Host firewalls such as ZoneAlarm that monitor which applications are allowed to open sockets, through to high level products such as Entercept, which monitor and protect hosts at the kernel level and operate on a behavioural basis.

Host based IPS are a logical extension to traditional host based IDS products such as Tripwire, however the significant management overhead and increased cost mean that they are not widely deployed. One of the major problems of most lower cost Host IPS is that they are can be difficult to integrate into an organisations security infrastructure (Desai, 2003).

Packet filters

Most operating systems now come with some form of IP packet filtering functionality, or it is easily installed. Windows 2000 has very basic packet filtering built in to the IP options. Systems such as OpenBSD's IPFilter, IPTables and the Internet Connection Firewall in Windows XP are stateful packet filtering firewalls. Stateful packet filtering simplifies rule definitions and allows for greater security as rules need only be defined for the establishment of a session, after which communications relating to that session are allowed. IPFilter and IPTables are equally well suited for inline packet filtering or host based packet filtering, and there are straightforward instruction sets on the internet for deploying these systems as host based firewalls (Price, 2002).

Even the simple functionality offered by these systems would be of some help in preventing the spread of worms. In the case of a standard web worm similar to the Code Red variants a system like this would not prevent infection. However, the worm would be preventing from initiating outgoing scans (if the filter ruleset was properly configured) and administrators could be quickly alerted as the scan attempts appear in the logs of blocked traffic. This worm would be very efficient in preventing Slammer infection, as the vast majority of Microsoft SQL Servers are not expected to receive connections from external computers at all. Those computers that are (perhaps unwittingly) running Microsoft SQL Server Desktop Engine (MSDE), which may accept external connections, are highly unlikely to need to accept them on the MS-SQL Monitor Port (UDP 1434). Packet filters would be unlikely to stop the spread of a worm such as Nimda, due to different the propagation mechanisms it uses. The packet filter would be able to stop an infected web server from initiating connections out to spread the worm, however clients could still be infected by viewing the compromised website, and infected desktop machines could then spread further by email, exploitation of IIS vulnerabilities, and via open network shares. A packet filter would prevent the infected computer from scanning for backdoors from Code Red II and sadmind/IIS as these operate on ports 69 and 600 (CERT/CC, 2001).

Host Firewalls

There are a variety of more advanced host based Intrusion Prevention solutions provided for Windows operating systems in particular. These systems incorporate the functionality of simple packet filters, but are more application aware. Most host firewalls have the ability to restrict what applications are able to bind to certain TCP and UDP ports and which applications are permitted to initiate external network connections. They also usually incorporate application verification whereby the IPS maintains a hash of the executable files that access the internet and alert if the hash value changes (indicating infection or replacement of the executable). Examples of this type of system include ZoneAlarm, Tiny Personal Firewall and Blackice.

These systems are reasonably cheap to deploy across every workstation and server in an organisation (Zone Labs offer volume licences for their ZoneAlarm Pro software for under $30 per user per year) and many host firewalls available in corporate editions with centralised administration and monitoring capabilities. They also have a high probability of preventing the spread of worms due to their control over which applications are allowed to open network connections. Host Firewalls would show similar performance to a packet filter in blocking a standard web server worm, and in preventing infection by the Slammer worm. Host Firewall software would be more effective in preventing the spread of the Nimda worm than a packet filter as it would prevent the software from using its own mass mailer and its scanning functions, and would prevent any infected application from initiating connections out without an administrator first confirming the change in the executables hash.

Application Firewall

Application Firewalls are the most advanced of all the host based IPS solutions, they

are primarily intended for high value servers (and are priced accordingly). These systems can monitor system calls, and are often available in versions specific to certain types of server such as web or database servers. Application Firewalls protect the operating systems and applications at the kernel level on the server rather than blocking the attack at the IP stack (Desai, 2003). They usually operate by profiling the normal activity of the server and intercepting system calls. Some of these systems also implement direct protection of the memory stack in order to prevent buffer overflows, one of the most common vulnerabilities exploited in application attacks (Entercept, 2003). Application Firewalls require considerable initial setup, profiling the server can take days or weeks to complete. Often they provide management consoles that allow remote configuration and monitoring of the protection systems on all servers across an organisation. Examples of these systems include Entercept and Okena's Stormwatch.

These systems theoretically offer the greatest possibility of preventing infection by an active worm, however, they are very new and are somewhat of an unknown quantity. There is little independent documentation on how these systems have performed in the worm outbreaks of the past few years. Although they can be expected to perform very well in protecting servers, the high cost of some of the products would be a barrier for many organisations and they should primarily be deployed on high value, high profile servers only. The potential of Application Firewalls can be seen in the fact that some of the companies offering them have been acquired by much bigger companies in the security field (Entercept is now owned by Network Associates and Okena is owned by Cisco). Network computing performed a review of the performance of these systems and found that many of them performed well in preventing attacks (Fratto, 2002).

In conclusion, Intrusion Prevention needs to be considered as a part of every organisations Defence in Depth strategy. Although the risks and cost of deploying Network Intrusion Prevention Systems can be high, they have the potential to significantly increase the level of protection on the network. The free Network Intrusion Prevention options are currently not sufficiently advanced enough for use in most enterprise networks, however the free or low cost Host based Intrusion Prevention systems are sufficiently mature to be deployed on all computers in an organisation. Host based Intrusion Prevention systems are rarely considered in discussions of Intrusion Prevention, however the more advanced options show great promise in protecting high value servers against the threat of active worms.

References

Cooperative Association for Internet Data Analysis. "CAIDA Analysis of Code-Red." 8 Apr. 2003. URL:http://www.caida.org/analysis/security/code-red/ (3 Jun. 2003).

CERT/CC. "CERT Advisory CA-2001-26 Nimda Worm." 25 Sep. 2001. URL:http://www.cert.org/advisories/CA-2001/26.html (19 Jun. 2003).

Chen, Zesheng. Gao, Lixin. Kwiat, Kevin. "Modeling the Spread of Active Worms." 25 May 2003. URL:http://www.hackbusters.net/AAWP.pdf (20 Jun. 2003).

Desia, Neil. "Intrusion Prevention Systems: the Next Step in the Evolution of IDS." 27th Feb. 2003. URL:http://www.securityfocus.com/infocus/1670 (6 Jun. 2003).

Entercept. "Entercept Standard Edition – Product Overview." URL: http://www.entercept.com/products/entercept/prodinfo/overview.asp (15 Jun. 2003).

Fratto, Mike. "Hip Check." Network Computing Magazine. 21 Oct. 2002. URL:http://www.nwc.com/1322/1322f2.html (5 Jul. 2003).

Fratto, Mike. "Network-Based Intrusion-Prevention System (NIPS)." Network Computing Magazine. 2 Jun. 2003. URL:http://www.nwc.com/1417/1417p1.html (14 Jul. 2003).

Hogwash. "Hogwash – Documentation." URL:http://hogwash.sourceforge.net/docs/overview.html (14 Jul. 2003).

Knobbe, Frank. "SnortSam." URL:www.snortsam.net (27 May 2003).

Moore, David. Paxson, Vern. Savage, Stefan. Shannon, Colleen. Staniford, Stuart. Weaver, Nicholas. "The Spread of the Sapphire/Slammer Worm." 4 Feb. 2003. URL:http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html (27 May 2003).

Moore, David. Shannon, Colleen. Voelker, Geoffrey M. Savage, Stefan. "Internet Quarantine: Requirements for Containing Self-Propagating Code." 5 Jun. 2003. URL:http://www.caida.org/outreach/papers/2003/quarantine/worm-infocom03.pdf (5 Jul. 2003).

Price, Dana. "IPFilter: A Unix Host-Based Firewall." 1 Jun. 2002. URL:http://www.sans.org/rr/papers/21/815.pdf (1 Jul. 2003).

Roesch, Marty. Green, Chris. "Snort Users Manual – Snort 2.0.0". URL:http://www.snort.org/docs/writing_rules/chap2.html#tth_chAp2 (6th July 2003)

Staniford, Stuart. Paxson, Vern. Weaver, Nicholas. "How to 0wn the Internet in Your Spare Time." Proceedings of the 11th USENIX Security Symposium. 2002. URL:http://www.icir.org/vern/papers/cdc-usenix-sec02/ (1 Jun. 2003).

The Honeynet Project. "Honeynet Project – Tools for Research." 17 Jun. 2003. URL:http://www.honeynet.org/papers/honeynet/tools/index.html (13 Jul. 2003).

Trend Micro. "Trend Micro Enterprise Home Page." URL:http://www.trendmicro.com (14 Jul. 2003).

Wickham, Timothy D. "Intrusion Detection is Dead. Long live Intrusion Prevention!" 21 Apr. 2003. URL:http://www.sans.org/rr/paper.php?id=1028 (3 Jun. 2003).

Network Detects

RPC mountd/pcnfsd scan

1. Source of Trace

The source of this trace was the log files at http://www.incidents.org/logs/Raw/2002.9.23
Note that the timestamp on the packets show a date of 10/24, this is in keeping with other GCIA detects where the packet timestamps have been 1 month greater than the date of the filename.

[**] RPC portmap request mountd [**]
10/24-10:46:45.956507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:41759 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request mountd [**]
10/24-10:46:45.966507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:42015 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request pcnfsd [**]
10/24-10:46:50.776507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:42527 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request pcnfsd [**]
10/24-10:46:50.926507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:42783 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request mountd [**]
10/24-10:46:59.046507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:44831 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request mountd [**]
10/24-10:46:59.226507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:45087 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request pcnfsd [**]
10/24-10:47:03.756507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:45343 IpLen:20 DgmLen:84
Len: 56

[**] RPC portmap request pcnfsd [**]
10/24-10:47:04.116507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x62
66.1.161.243:600 -> 32.245.170.117:111 UDP TTL:109 TOS:0x0 ID:45599 IpLen:20 DgmLen:84
Len: 56

Examining the entire log file shows that all packets originate from 2 MAC addresses, both of which are allocated to Cisco Systems. The sensor must be located between the two devices. . Further examination of the log file shows that the traffic coming from the internal network all has the source address of 32.245.166.236, suggesting that either there is one very busy machine in the network, or the more likely reason that this network is using Network Address Translation (NAT). Traffic destined to addresses from 32.245.0.42 to 32.245.253.117 are being seen being to this network, suggesting that this site is allocated the entire class B netblock of 32.45.0.0/16.

Internet facing router Firewall/NAT
0:3:e3:d9:26:c0---------- 0:0:c:4:b2:33
|
IDS

2. Detect generated by

This detect was generated by Snort 2.0.0 (Build 72) using the snort-stable ruleset downloaded from www.snort.org on the 23/6/03.

The snort rules that generated this detect were:
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request mountd"; content:"|01 86 A5 00 00|";offset:40;depth:8; reference:arachnids,13; classtype:rpc-portmap-decode; sid:579; rev:2;)
alert udp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"RPC portmap request pcnfsd"; content:"|02 49 f1 00 00|";offset:40;depth:8; reference:arachnids,22; classtype:rpc-portmap-decode; sid:581; rev:2;)

3. Probability Source address was spoofed

Although this is a UDP packet and spoofing the source of this request is trivial, this is a reconnaissance scan, to have any use it needs to receive the response to its request. The source address of the trace resolves to a netblock assigned to Sprint Broadband Direct, a provider of wireless broadband connectivity. This type of address is usually assigned statically so that makes it worth checking the history of the address. Mynetwatchman.com shows that there have been 2 incidents registered in the past against this IP address, however it does not show any further details regarding the incidents or the date the incident was reported. This IP does not show in the Dshield database www.dshield.org. A traceroute and ping attempt to this address does not reach the address.

The fact that there has been malicious activity in the past from address decreases the probability of spoofing, so I would assess the probability that this address is spoofed as low. Although it is possible that the attacker is spoofing the address and sniffing the traffic or is using the source address as a proxy, that is less likely due to the increased difficulty and time required and the fact that this appears to be a reconnaissance scan only.

4. Description of Attack

This is a set of RPC queries. The attacker sends a request to the RPC portmapper on port 111 of the target host asking for the udp port that the mountd and pcnfsd programs are running on. There are two probable reasons for this scan, either the scanner is looking for insecure NFS servers to steal information from or they are looking to exploit vulnerabilities in these programs (they are both known to have contained buffer overflows).

5. Attack Mechanism

The attacker has issues a series of 8 RPC GetPort requests to the RPC portmapper (UDP port 111) on the target host. The intention of these requests is to elicit a response confirming that the mount and pcnfs programs are running on the target host. There are 4 pairs of request, each pair contains 2 identical requests with duplicate request Identification numbers. The source host issues a pair of requests for the mount program (program 100005), then a pair for the pcnfsd program (program 150001), then repeats the pattern. The likely reason for the repeated scans is the unreliable nature of the UDP protocol, by sending 4 requests for each program, the source host is attempting to ensure at least one request arrives at the destination. If the scan was successful, the portmapper on the target host would return a value for the UDP port number of the programs.

If the scan was unsuccessful, there are a number of different possibilities for responses depending on the circumstances. The request could be blocked by the firewall, which could drop traffic silently or send an ICMP unreachable packet depending on configuration. If the target host did receive the request but was not running the RPC portmapper we would expect to see a ICMP unreachable message back, however it is likely that a firewall would drop this traffic. Due to the fact that only packets that violate the ruleset are logged, it is impossible to verify whether or not the scan was successful. A RPC response back to the scanning host would not be logged by snort, and neither would an ICMP unreachable message (ICMP messages have been removed from the logs provided). Due to the apparent use of NAT by the internal network, I would consider it unlikely that the request even reached the source host. After checking the traffic logged it appears that no traffic from trusted source ports (1023 and below) is leaving the network, leading me to believe that if the request did arrive at the target host, the response would have most likely been blocked at the firewall.

It would, however, be very useful to check the firewall logs and those of the target host to be certain.

It is worth noting that the IP identification number in this scan is incrementing by exactly 256 each time, suggesting that the attacking host may be scanning another 255 addresses, on different netblocks. Another possible, but less probable

explanation for this is that the IP Identification is crafted to obscure active targeting.

6. Correlations

The mynetwatchman database (www.mynetwatchman.com) shows 2 closed incidents have been registered against this IP address. As these incidents have been closed some time ago there are no further details available. Dshield does not have this IP address in its database.
Identical traffic was reported to SANS in both 2000 and 2001, note the source port of 600 which is consistent with the signature of our traffic
http://www.sans.org/y2k/102400.htm
http://www.sans.org/y2k/012301.htm
Similar traffic was reported to the incidents mailing list in may 2001 (http://lists.jammed.com/incidents/2001/05/0055.html), however the author of the post mentions that most of the scanning hosts were found to be running SunOS 5.6, whereas the TTL values of 109 on the packets logged in this trace seem to indicate that they originated from a Windows Operating system with an initial TTL of 128 (it is difficult to determine more from a UDP packet with the limited information it provides).
Vulnerable RPC services are the number one unix vulnerability in the SANS/FBI top 20 list (http://www.sans.org/top20/), as these services often run with root priveliges.

7. Evidence of Active Targeting

This is a difficult, there is only one target host for this scan in the logs, so the scan appears to be targeted, however the changes in IP Identification number suggest that a number of other hosts are also being scanned. I would suggest that this is merely part of a larger scan, where the scanner is targeting a number of distributed hosts in an attempt at stealth. Also, correlations show similar scans at a number of other sites from a variety of source addresses, suggesting that this may be automated scanning activity by a worm attempting to spread through insecure NFS implementations. I think the most likely explanation is that the source host is utilising an RPC scanning tool to scan a number of hosts for vulnerabilities.

There is other traffic directed at the target IP address in these logs, there are a number of proxy scans to ports 8080 and 3128, however these scanners appear to be going through the entire netblock probing 32.45.X.117 addresses.

8. Severity

Severity=(criticality + lethality)-(system countermeasures + network countermeasures)

Criticality – As we know nothing about the target host, or even whether the host actually exists, we will assume an criticality of 3, particularly given that it is likely to be a unix host.

Lethality – While this appears to be an information gathering attack only, it could provide information that would lead to loss of information or system compromise, so I have assigned it a lethality of 3.

System Countermeasures – It is impossible to ascertain the system countermeasures, but the lack of host logs do not bode well for security so I have

assigned this a value of 2

Network countermeasures – There appears to be a firewall in place using NAT, however, without firewall logs or logs of all traffic passing the sensor it is impossible to verify if the scan was successful of not, I have assigned this a value of 3.

Severity=(3+3)-(2+3)=1
Although this is not high severity traffic in of itself, it could be leading to a root level compromise of the target host. More investigation is required into how successful the scan was and the vulnerability of the target host.

9. Defensive Recommendations

RPC services are well know for security holes and RPC vulnerabilities rate as the number one unix vulnerability in the SANS/FBI Top 20 list. The fact that many of these services run as root mean that the vulnerabilities can lead to a full system compromise. There are a number of ways to limit exposure to RPC vulnerabilities.
1. Turn off RPC services when not needed, Unix machines that are not running GUIs or NFS often do not require RPC functionality at all.
2. Ensure patches for RPC services are kept up to date.
3. Block the RPC portmapper port (111) and the higher ports that RPC services run on (32770-32789) at the firewall.
4. Deploy host based protection systems such as TCPwrappers and the secure portmapper developed by Weitse Venema, and use host based firewalls on systems where RPC services need to run (Stateful packet filters such as IPTables or IPfilter are easy to configure as host based firewalls).
5. As this scan may be searching for open NFS servers, secure these services by exporting file systems read only where possible and restricting access to internal hosts only.

10. Multiple Choice Question

Why would a host send duplicate RPC GetPort requests over UDP to a target host?
a. The RPC portmapper requires multiple requests before it will send a response
b. The source host is attempting to overload the Portmapper on the target host
c. The source host is compensating for the unreliable nature of UDP
d. The traffic is backscatter from a Distributed Denial of Service attack
The correct answer is c. UDP is connectionless, so there is no mechanism for ensuring packets reach their destinations. By sending multiple packets the source host can increase the likelihood of the request reaching the target host intact.

11. References

ArachNIDS. “IDS13 – PORTMAP-REQUEST-MOUNTD." URL: http://www.whitehats.com/info/IDS13 (16 May 2003).

Eeye Digital Security. “RPC vulnerabilities.” URL: http://www.eeye.com/html/Products/Retina/RTHs/Rpc_Services/ (15 May 2003).

ArachNIDS. “IDS22 – PORTMAP-REQUEST-PCNFSD." URL: http://www.whitehats.com/info/IDS22 (16 May 2003).

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Addison Wesley. 1994.

Distorted NetBIOS scan

1. Source of Trace

This source of this trace was http://www.incidents.org/logs/Raw/2002.9.29
Although the timestamps on the packets show a date of 20/6, this is in keeping with other GIAC log files which have timestamps different to that on the name of the packet. All IP addresses have been obfuscated by SANS.

[**] [116:97:1] (snort_decoder): Short UDP packet, length field > payload length [**]
10/30-07:16:27.826507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5C
211.223.8.214:0 -> 32.245.161.79:0 UDP TTL:109 TOS:0x0 ID:25767 IpLen:20 DgmLen:78
Len: 129

[**] [116:97:1] (snort_decoder): Short UDP packet, length field > payload length [**]
10/30-07:16:33.676507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x5C
211.223.8.214:0 -> 32.245.161.117:0 UDP TTL:109 TOS:0x0 ID:62887 IpLen:20 DgmLen:78
Len: 129

Packet Dump
07:16:27.826507 211.223.8.214.1026 > 32.245.161.79.0: udp 129
0x0000 4500 004e 64a7 0000 6d11 33e8 d3df 08d6 E..Nd...m.3.....
0x0010 20f5 a14f 0402 0000 0089 003a 0696 0100 ...O.......:.....
0x0020 0010 0001 0000 0000 0000 2043 4b41 4141 ....CKAAA
0x0030 4141 4141 4141 4141 4141 4141 4141 4141 AAAAAAAAAAAAAAAA
0x0040 4141 4141 4141 4141 4141 4100 0021 AAAAAAAAAAA..!
07:16:33.676507 211.223.8.214.1026 > 32.245.161.117.0: udp 129
0x0000 4500 004e f5a7 0000 6d11 a2c1 d3df 08d6 E..N....m.....
0x0010 20f5 a175 0402 0000 0089 003a 0670 0100 ...u.......:.p..
0x0020 0010 0001 0000 0000 0000 2043 4b41 4141 ....CKAAA
0x0030 4141 4141 4141 4141 4141 4141 4141 4141 AAAAAAAAAAAAAAAA
0x0040 4141 4141 4141 4141 4141 4100 0021 AAAAAAAAAAA..!

Examination of the log files show that all packets originate from 2 MAC addresses, both of which are allocated to Cisco Systems. The sensor must be located on the network between these two devices.

The address of the internal network appears to be 32.245.0.0/16, as traffic to addresses between 32.245.6.200 and 32.245.252.181 are being routed to it. All traffic coming from the internal network has the source address of 32.245.166.236, suggesting that this network is probably using Network Address Translation (NAT) to protect internal hosts.

Internet facing router Firewall/NAT
0:3:e3:d9:26:c0---------- 0:0:c:4:b2:33
|
IDS

2. Detect was generated by:

This detect was generated by Snort version 2.0.0, using the snort-stable ruleset downloaded from the www.snort.org website on 23/6/03.

These traces were not generated by a snort rule, however the snort decoder flagged the traffic as having a length field greater than the actual payload length, so I deemed it worthy of more attention. Examination of the packet reveals that the actual UDP datagram size is 58 bytes whereas the length field of the UDP header has a value of 137. It is interesting to note that the snort alert actually provides incorrect information regarding this packet, it reports the source port as being 0, when according to the packet dump the source port for both packets is actually 1026.

Had the length field of the packet been correct, this packet would have triggered the BAD TRAFFIC UDP Port 0 rule in the snort ruleset.

3. Probability the source address was spoofed

The source address in these packets has a medium probability of spoofing, they are UDP packets, so spoofing the address is trivial. Also, the target port of 0 and invalid length field could mean that this could be a packet designed merely to provoke an intrusion analysts interest and waste time. However, the evidence suggests that this packet is actually a malformed NetBIOS name query, in which case the source host would require a response for the packet to be of any use.

The mynetwatchman and Dshield databases have no incidents registered against this IP address. Whois queries show that the address is registered to Kornet, an ISP in the Republic of Korea. A traceroute to this address does connect, however it is possible that this is a dynamically allocated IP address, in which case the address history and connectivity are meaningless.

4. Description of Attack

This is almost definitely a malformed NetBIOS Wildcard scan, specifically a NetBIOS name service node status request to two hosts on the internal network. The scanning host has sent a name request with a wildcard (*) to the target hosts. If the target hosts accept these requests, they will respond with the NetBIOS name of the host, the windows workgroup or domain name and the login names of users who are logged in to the target host. This type of traffic is often seen in internal network communication between windows hosts using file sharing, it can be used to determine a hosts NetBIOS name when only the IP address is known. External traffic of this type is usually a reconnaisance scan, particulary in this case where the RPC Transaction ID of the queries are the same for both packets, in normal traffic this value should increment.

5. Attack Mechanism

The source host has sent 2 UDP packets to port 0 on two target hosts. The length field in the UDP header is 137, which is much greater than the actual UDP datagram length of 58 bytes. The destination port in itself would usually generate a BAD TRAFFIC UDP Port 0 snort alert, however in this case the snort packet decoder has alerted on the incorrect length field. Examining the hex and ASCII dump of the the packet below shows the string, CKAAAAAAAAAA(30xA)! in the payload, a classic characteristic of a NetBIOS wildcard scan. Also, the length field in the UDP header

is 137, the NetBIOS Name Service port number. As I did not consider this to be a coincidence, I examined the hex dump of the packet.

```
07:16:27.826507 211.223.8.214.1026 > 32.245.161.79.0: udp 129
0x0000    4500 004e 64a7 0000 6d11 33e8 d3df 08d6   E..Nd...m.3.....
0x0010    20f5 a14f 0402 0000 0089 003a 0696 0100    ...O.......:....
0x0020    0010 0001 0000 0000 0000 2043 4b41 4141   ..........CKAAA
0x0030    4141 4141 4141 4141 4141 4141 4141 4141   AAAAAAAAAAAAAAAA
0x0040    4141 4141 4141 4141 4141 4100 0021        AAAAAAAAAAA..!
```

After removing the 20 byte IP header we are left with a 58 byte UDP datagram

```
0x0014              0402 0000 0089 003a 0696 0100    ...O.......:....
0x0020    0010 0001 0000 0000 0000 2043 4b41 4141   ..........CKAAA
0x0030    4141 4141 4141 4141 4141 4141 4141 4141   AAAAAAAAAAAAAAAA
0x0040    4141 4141 4141 4141 4141 4100 0021        AAAAAAAAAAA..!
```

The UDP Header makes up the first 8 bytes of the datagram, as is made up as below.
[Source Port (2 bytes][Destination Port (2 Bytes][Length (2 bytes][Checksum (2 bytes)]
[0x0402=1026][0x0000=0][0x0089=137][0x003a=58]
Note that the Checksum of the UDP datagram is 58, which is the actual length of the datagram. It appears from this that the Destination Port appears to have been inserted into the packet in error, leading to all other fields in the datagram being shifted 2 bytes to the right.

Further examination of the entire packet shows that this must be the case, by removing the 0x0000 value from the destination port field and shifting all other packets back I get a packet that looks identical to other traces of NetBIOS Scans. A decode of the RPC Data is shown below:

Bytes 0-1: NAME_TRN_ID – Transaction ID
01 00 = Transaction ID of 256
Bytes 2-3: OPCODE, NMFLAGS & RCODE – packet type, operation flags and response code
00 10 = request packet, name query, broadcast or multicast
Bytes 4-5: QDCOUNT - number of entries in the question section of the packet
00 01 = 1 name query
Bytes 6-11: ANcount, NScount, ARcount
00 00 00 00 000
All these fields are used in response packets, this is a request packets
Byte 12: Size of Name field
20 = 32 Bytes for name in packet
Bytes 13-45: Name Field
43 4B 41 41 41 41...
This translates to the ASCII string CKAAA...
This is the NetBIOS string for *. The Translation to the string above is performed by splitting the hex value for each character in the name into separate values then

added 0x41 to each. * in ASCII is 2A, Adding 0x2 to 0x41 and 0xA to 0x41 respectively yeilds 0x43 and 0x4B (as above), the rest of the name field is null so each value is just 0x41.
Byte 46: Field Delimiter
00
Bytes 47-48: Question Type
00 21 = Node Status Request
Bytes 49-50: Question Class
This field does not exist in these packets. Due to the malformed request the last 2 bytes have been lost from the packet. Normally we would expect this field to have the value 0x0001 for Internet Class (Northcut et al, 2001).

This packet would definitely be rejected by the target host. The UDP checksum value will no longer be valid as it contains the value that is supposed to be in the Datagram length field. This means that it is most likely that the corruption of the packet is accidental rather than intentional.

6. Correlations

Similar traffic to this has been seen in bulk since early 2000. A correctly formatted NetBIOS Wildcard query is featured in the Intrusion Signatures and Analysis book, and this was of great value in decoding the packet trace and identifying the attack signature.

Reto Baumann analysed similar traffic from the incidents.org log files of the http://www.incidents.org/logs/Raw/2002.10.11 at (http://cert.uni-stuttgart.de/archive/intrusions/2003/03/msg00090.html). It is interesting to note that his snort version (1.9.1) alerted on the UDP Port 0 rule, rather than the incorrect UDP datagram length field. Although this analysis came to a different conclusion to me (that the traffic was an attack on CheckPoint FireWall-1), responses to the analysis indicate that this is a corrupted NetBIOS query.

The source host of the request in the 2002.10.11 log files has an IP of 211.194.68.39, although this is different to the source host in my detect, the IP is also allocated to Kornet, suggesting that it is possible that the traffic is coming from the same host with a different dynamically allocated IP address.

Scans of this type can be generated in bulk by tools such as NBTscan (http://www.inetcat.org/software/nbtscan.html), or manually on windows machines with the “nbtstat -A [ipaddress]” command. They can also be produced by the multitude of worms that spread via unprotected network shares, the scanning activity of these worms usually involves NetBIOS Wildcard queries.

7. Evidence of Active Targeting

This traffic is only directed at 2 hosts on the internal network, meaning that it is likely that prior reconnaisance has already been undertaken to identify those hosts open to port 137. However, looking at the IP Identification number of the packets shows a significant increase between scans, leading to the possibility that the source host is scanning several thousand randomly selected hosts across the internet. If the activity were caused by a worm this is they type of activity we would expect to see. For this reason I do not think that this is a case of active targeting, particularly when

taking into account the similar traffic seen on 2002.10.11.

8. Severity

Target Criticality – It is difficult to judge the criticality of the target host, assuming that it is a windows workstation or server, I will average the criticality between the two and give it a criticality of 3.

Lethality – This is a reconnaissance attempt primarily, and possibly the precursor to an worm infection, however, the corruption in the NetBIOS query means this attack cannot succeed, so lethality is assessed as 1.

System Countermeasures – This is difficult to judge, system countermeasures will generally have no effect on this type of scan unless denied by host based firewalls. I will assume a countermeasures value of 2, as we do not know anything about the host, and have no access to host logs.

Network Countermeasures – This is also difficult to verify, however traffic to an usual port such as port 0 should usually be blocked at the border router, as this is a reserved port and should not be used for general traffic, the fact that the traffic was not blocked does not bode well for network countermeasures. Also, the fact that we do not have access to firewall logs, rulesets or full packet traces to verify the whether this attack would have succeeded if it was not corrupted. I will give network countermeasures a score of 3.

Severity = (3 + 1) – (2+3) = -1
This attack is not of great concern to us, the attack has no possibility of succeeding. However, this trace should serve as a reminder to review the Access Control List (ACL) on the border router, not only does this improve our security level but it serves to reduce the amount of traffic the IDS sees, allowing us to focus on more serious attacks.

9. Defensive Recommendations

Ensure traffic inbound to and outbound from TCP and UDP ports 135, 137, 138, 139 and 445 is blocked at the external router or firewall.
is being blocked at the firewall.
Review border router ACL and block inbound UDP and TCP port 0 to reduce bad traffic noise.
Disable network shares where not needed and ensure that all shares are protected with secure passwords
Install host based firewall software on windows workstations
Add rules to IDS ruleset to alert on traffic to NetBIOS ports (135, 137, 138, 139 and 445) from external hosts.
Restrict Null session access on windows machines in your environment (this is available in Windows versions from NT 4.0 service pack 3).

10. Multiple Choice Question

What response will be expected after sending a UDP packet with an invalid checksum?
a. The destination host will silently discard the packet
b. The destination host will reply with an ICMP Parameter Problem packet

c. The first router on the transmission path will silently discard the packet
d. The first router on the transmission path will reply with an ICMP Parameter Problem packet

The correct answer is a. UDP checksums are not checked until they reach the destination host (unlike IP header checksums). If the destination host finds that the checksum is invalid, the packet is silently discarded.

This analysis was posted to the incidents.org mailing list for comment. Don Murdoch provided comments on my analysis on the 22nd July 2003. Some of his comments are shown below, with my responses.

>Also, the target port of 0 and invalid length field could mean that this
>could be a packet designed merely to provoke an intrusion analysts
>interest and waste time. However, the evidence suggests that this
>packet is actually a malformed NetBIOS name query, in which case the
>source host would require a response for the packet to be of any use.

> Don - and what could be gained by a netbios query to a port
> not listening to NB?

A NetBIOS query to a port not listening to NetBIOS would elicit a ICMP Port Unreachable response from the target host. This could be used for OS identification and network mapping. If a host responds with a port unreachable packet, we can determine that this host is not running a Windows OS. However, this could just as easily be performed using an empty UDP packet.

>Scans of this type can be generated in bulk by tools such as NBTscan
>(http://www.inetcat.org/software/nbtscan.html), or manually on windows
>machines with the "nbtstat -A [ipaddress]" command. They can also be
>produced by the multitude of worms that spread via unprotected network
>shares, the scanning activity of these worms usually involves NetBIOS
>Wildcard queries.

> Don - If an automated tool ran this scan, why would the tool use
> a dest port of zer0? Does a windows PC listen on 0 for NBNS/Dgram/Stream
> traffic?

I cannot think of a good reason why the tool would use a destination port of zero except for the network mapping purposes mentioned above. Windows hosts do not listed for NetBIOS traffic on this port. That is the one of the main reasons why I am quite sure that this is a corrupted NetBIOS wildcard query.

>System Countermeasures will generally have no effect on this type of scan
>unless denied by host based firewalls. I will assume a countermeasures
>value of 2, as we do not know anything about the host, and have no
>access to host logs.

> Don = Not so. There are well publicized options in NT/2000/XP that control if it
> will give out informaiton on NBNS. "Allow Anonymous" and all that.

As far as I am aware, restricting Null Session will not prevent a host from responding to a NetBIOS name query. It will however, prevent a host logging into the Windows host using a blank username and password. A Null session allows an attacker to collect information on all users, groups and shares on a host and these connections are initiated on the NetBIOS-ssn port, not the NetBIOS-ns port. However, there is a way of preventing a host from responding to a NetBIOS name query, this involved disabling NetBIOS over TCP in the TCP/IP options.

11. References

ArachNIDS. "IDS177 – NETBIOS NAME QUERY." URL: http://www.whitehats.com/info/IDS177 (20 May 2002).

Internet Engineering Task Force – NetBIOS Working Group. "RFC1002 - PROTOCOL STANDARD FOR A NetBIOS SERVICE ON A TCP/UDP TRANSPORT: DETAILED SPECIFICATIONS." Mar. 1987. URL: http://www.ietf.org/rfc/rfc1002.txt (20 May 2002).

Northcutt, S. Cooper, M. Fearnow, M. Frederick, K. Intrusion Signatures and Analysis. Indianapolis: New Riders, 2001. 156-159.

Noisy CGI Scan

1. Source of Trace

This trace comes from a web server I administer for an IT Security organisation.

```
naughty.cable.modem.user - - [02/Mar/2003:17:20:58 +1300] "HEAD / HTTP/1.0" 200 12529
naughty.cable.modem.user - - [02/Mar/2003:17:20:58 +1300] "GET /cfdocs/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:58 +1300] "GET /scripts/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "GET /cfcache.map HTTP/1.0" 200 180
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "GET /cfide/Administrator/startstop.html
HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "GET /cfappman/index.cfm HTTP/1.0"
404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "GET /cgi-bin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /_vti_inf.html HTTP/1.0" 404
392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /_vti_pvt/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /mall_log_files/order.log
HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /PDG_Cart/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /quikstore.cfg HTTP/1.0" 404
392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /orders/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /Admin_files/order.log
HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /WebShop/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /pw/storemgr.pw HTTP/1.0" 404
392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /bigconf.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /icat HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /cgi-local/ HTTP/1.0" 404 392
```

naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /htbin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /cgibin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:20:59 +1300] "HEAD /cgis/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /cgi/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /cgi-win/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /manage/cgi/cgiproc HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /wwwboard/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /logs/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /database/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /databases/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /.htaccess HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /~root/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /ws_ftp.ini HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /WS_FTP.ini HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /search97.vts HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /search.vts HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /search97cgi/s97_cgi HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /webcart/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /webcart-lite/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /reviews/newpro.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /piranha/secure/passwd.php3 HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:00 +1300] "HEAD /srchadm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /users/scripts/submit.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /bb-dnbd/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /session/admnlogin HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /wwwthreads/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /ss.cfg HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /ncl_items.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /test/test.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /php/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /mlog.phtml HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /mylog.phtml HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /HyperStat/stat_what.log HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /Stats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /WebTrend/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /analog/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /cache-stats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /easylog/easylog.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /hit_tracker/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:01 +1300] "HEAD /hitmatic/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /hitmatic/analyse.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /hyperstat/stat_what.log HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /log/ HTTP/1.0" 404 392

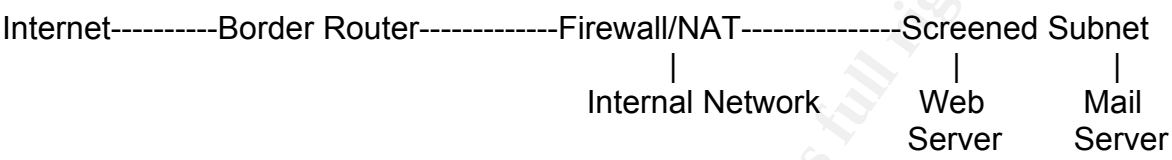
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /logfile/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /logfiles/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /logger/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /logging/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /logs/access_log HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /ministats/admin.cgi HTTP/1.0" 500 305
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /scripts/weblog HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /server_stats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /stat/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /statistics/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /stats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /super_stats/access_logs HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /trafficlog/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /ustats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /w3perl/admin HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:02 +1300] "HEAD /webaccess/access-options.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /weblog/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /weblogs/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /webstats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /wstats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /wusage/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /wwwlog/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /wwwstats/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /access-log HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /access.log HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /awebvisit.stat HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /dan_o.dat HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /hits.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /log.htm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /log.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /log.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /logfile HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /logfile.htm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /logfile.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /logfile.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:03 +1300] "HEAD /logger.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /stat.htm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /stats.htm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /stats.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /stats.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /webaccess.htm HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /wwwstats.html HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /bin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /admin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /Admin_files/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /DMR/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /StoreDB/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /Web_store/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /access/ HTTP/1.0" 404 392

naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /account/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /accounting/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /administrator/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /app/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /apps/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /archive/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:04 +1300] "HEAD /asp/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /atc/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /backup/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /bak/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /beta/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /buy/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /buynow/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /c/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /cart/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /ccard/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /config/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /counter/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /credit/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /customers/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /dat/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /data/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /db/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /dbase/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /doc-html/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /docs/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /down/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:05 +1300] "HEAD /download/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /downloads/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /employees/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /exe/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /file/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /files/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /forum/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /fpadmin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /ftp/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /guestbook/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /guests/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /home/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /htdocs/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /html/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /ibill/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /idea/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /ideas/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /incoming/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /info/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:06 +1300] "HEAD /install/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /intranet/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /jave/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /jdbc/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /lib/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /library/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /login/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /mail/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /mall_log_files/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /manual/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /marketing/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /msql/ HTTP/1.0" 404 392

naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /new/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /odbc/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /old/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /oracle/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /order/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /outgoing/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:07 +1300] "HEAD /pages/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /passwords/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /perl/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /private/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /pub/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /public/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /purchase/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /purchases/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /pw/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /register/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /registered/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /reseller/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /retail/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /root/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /sales/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /search/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /sell/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /setup/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /shop/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /shopper/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:08 +1300] "HEAD /site/iissamples/ HTTP/1.0" 404
392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /software/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /source/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /sql/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /store/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /support/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /temp/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /test/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /test-cgi/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /tmp/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /tools/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /tree/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /updates/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /usage/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /user/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /users/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /web/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /web800fo/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /webadmin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /webboard/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:09 +1300] "HEAD /webdata/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /website/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /www/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /www-sql/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /wwwjoin/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /import/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /zipfiles/ HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /passwd HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /passwd.txt HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /password HTTP/1.0" 404 392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /password.txt HTTP/1.0" 404
392
naughty.cable.modem.user - - [02/Mar/2003:17:21:10 +1300] "HEAD /status/ HTTP/1.0" 404 392

naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET / HTTP/1.1" 200 12529
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /styles.css HTTP/1.1" 200 1039
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /images/image1.gif HTTP/1.1" 200 5671
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /images/image2.jpg HTTP/1.1" 200 13300
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /images/image3.jpg HTTP/1.1" 200 12983
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /images/image4.gif HTTP/1.1" 200 96
naughty.cable.modem.user - - [02/Mar/2003:17:21:36 +1300] "GET /images/image5.jpg HTTP/1.1" 200 9941
naughty.cable.modem.user - - [02/Mar/2003:17:21:37 +1300] "GET /images/image6.jpg HTTP/1.1" 200 515

The network is set up as below.



I chose this trace to analyse for a number of reasons, this is the first real attack the site has come under (excluding the usual Nimda and Code Red Traffic) and the active targeting aspects I will discuss later made it an interesting exercise in intrusion analysis and incident response. Also, it is a good example of intrusion detection in an environment without access to raw packets or firewall logs.

2. Detect was generated by:

This detect was generated by iPlanet Web Server version 6.0 Service Pack 4. The server generates an access log file and an error log file (the error log file is not included here as it does not add any useful information to the detect). The IP of the host that scanned the web server has been obfuscated at the request of the web server owner.

The log file format is the Common Log File format also used by the Apache Web Server.

Example:
127.0.0.1 - - [02/Mar/2003:17:21:37 +1300] "GET /images/image6.jpg HTTP/1.1" 200 515

Field descriptions:
127.0.0.1 - This is the IP address of the client (remote host) that made the request to the server.

[02/Mar/2003:17:21:37 +1300] - The time that the server finished processing the request. The format is [day/month/year:hour:minute:second UTC Offset]

"GET /images/image6.jpg HTTP/1.1" - The request line from the client is given in double quotes.

200 - This is the status code that the server sends back to the client.

515 - The size of the object returned to the client, not including the response headers.

3. Probability the Source Address was spoofed

This traffic was not spoofed. The fact that the HTTP requests appear in the web server logs show that the TCP three-way handshake had been completed. Also, because this was a reconnaissance scan, the scanning host would need to receive the response from the web server for the attack to be of any use.

Although it is possible that the source host is an open proxy that is being used by the scanner to hide his or her identity, the evidence of active targeting I will discuss below leads me to believe that the source host is genuine.

4. Description of Attack

The logs show a very long and noisy vulnerability scan aimed at our web server. The scanning host is looking for vulnerable CGI scripts and files that may contain useful information (such as password and log files). This scan is completed in 12 seconds, and is almost definitely a standard web vulnerability scanner that the attacker has downloaded from the Internet. The website being scanned contains static content only, and has no e-commerce function so most of the items being scanned for are of no relevance. The attack is over 4 months old now, and many of the web server scanning tools available have changed over this period. After some investigation, I discovered a scan database for Whisker that looks like it would generate the requests we see in our web server logs. Unfortunately, Whisker has now been retired and replaced by Nikto. I tested the latest version of Nikto, however this has been updated since the scan and the signature of the attack was quite different to that of our log file.

5. Attack Mechanism

The attacker is trying to verify the existence of certain files, directories and scripts on the target web server by the use of the HEAD http request. The HEAD request returns only the header information from the server, and its most common use is to test web links. It is probably used in this scan for the purpose of speed. The scanner can then report back which files exist to the attacker who can use them to attempt to gain privileged access to the web server. Some of the queries in the scan use the GET request instead, this is probably used where the page contents have information that is useful to the scanner. I have identified the scanning tool used as Whisker, this tool is listed as number 10 in Fyodor's list of top 75 network security tools, and was an extremely popular HTTP scanning tool. Whisker is no longer being developed and has been superseded by Nikto.

Most of the status code responses the server returned to the attacker are 404, which translates to “Client error – file not found'. This is exactly the response we expect and hope for in this case. However, a number of the responses are 500 and 200. The status code 500 means 'Server error – internal server error'. Further examination shows that the server only returns this when a .cgi file is requested. The server does not have CGI directories defined, therefore it is returning a server error. This is not a problem as we do not require CGI scripts on the web server. The

status code 200 means 'OK, request succeeded'. The web server returns this status code when the file 'cfcache.map' is requested. I knew that this file did not exist on the web server, and after more investigation discovered that requesting any file with a .map extension from the server returns an Imagemap error page and different to the standard 404 error page and results in a status 200 being returned to the client, therefore this was nothing to be concerned about either.

Note that 26 seconds after the scan was completed, the attacking host visits our website again, perhaps to check if it is still there after the barrage it was just subjected to?

6. Correlations

I was able to retrieve a copy of the whisker 1.4.0 scan database from the Anti Hacker Toolkit book published by Osborne/McGraw Hill. There is also a copy available at http://security.testtubebabies.org/vulnerability-scanning/whisker/scan.db

This database file lists the scans performed, in order, by the Whisker scanner, and they match those in our web server logs. There is also a version 2.1 of whisker, however after examining a copy of the whisker 2.0 database from http://sourceforge.net/projects/whisker/, which appears slightly different to our scan, I believe that Whisker 1.4.0 was used to scan the web server.

Web vulnerability scans are among the most common attacks on the internet, with thousands of attackers daily searching for vulnerabilities to exploit for a number of reasons. Some of the major reasons for exploiting web servers are for defacement purposes and to steal confidential information. Defacement is mainly a 'bragging rights' issue, with attackers submitting their defacements to a mirror such as www.zone-h.org.

Stealing confidential information usually targets e-commerce sites and is focused on stealing credit card information stored on insecure servers or compromising the servers in order to collect credit card information submitted to the servers.

7. Evidence of Active Targeting

Further examination of the log files shows an earlier attempt at running the same scan the day before from the same IP address, however the scan was aborted after a second or so.

The IP address of naughty.cable.modem.user belongs to the cable modem pool of a large national ISP. These IP addresses are allocated statically so, after checking that a traceroute connects to the IP address, I used the safe web browser on www.samspade.org to see if there was a web server running on this IP address. A web server was running on this IP address and appears to be a local organization in the same city as the web server. A quick DNS lookup on the website domain name confirms this, showing that the attacking machine is probably located a few suburbs away, this lookup also provides a name, address and phone number for the server owner.

This is unlikely to be a coincidence, given that the web server is located in a small city by global standards. The IP does not appear on the Dshield or Mynetwatchman

databases.

8. Severity

Target Criticality – This is the organisations web server, and, although 100% uptime is not crucial for day to day operations, compromise of the server could lead to loss of reputation given that the organisation is in the IT security field. Criticality = 4

Lethality – This is a reconnaissance scan, and one that is fairly unlikely to be successful against the web server. Lethality = 2

System Countermeasures – The web server in question is fully patched and is hardened to the maximum extent possible. The web server hosts static content only and has no example scripts. It is not running host based firewall software, however it is running Tripwire and these reports are reviewed regularly. The log files for the server were not examined for some months after the attack.
System Countermeasures = 4

Network Countermeasures – The firewall ruleset is well defined, however firewall logs are not available for the period in question and the organisation is not running an IDS. In relation to the vulnerability scan above, the firewall would have no effect on the scan.
Network Countermeasures = 3

Severity = (4+2)-(4+3) = -1
This attack is not of great concern to us as it poses little threat.

9. Defensive Recommendations

Although the server in question is highly unlikely to show any vulnerabilities during scans of this type, it is of concern that the server log files are not reviewed on a regular basis. In addition, the lack of firewall and IDS logs leaves the organisation without the ability to be aware of malicious activity directed at the Internet facing servers. I recommended implementing regular log checking procedures, including the use of swatch or a similar package to monitor the logs. I also recommended the installation of an IDS with properly configured ruleset in the screened subnet.

In order to head off any future malicious activity from the address in question it is worthwhile contacting the individual responsible for the scanning host and the abuse contact at his or her ISP.

SANS also gives a number of recommendations for securing web servers, although this organisation uses only passive content and has removed any sample scripts from the server, some of the main recommendations relevant to this organisation are:

> Configure your web server to use CGI alerting scripts for Error Responses. WebAdmins need to keep tabs on all of these security related issues with their web servers. To assist with this monitoring, the web server should be configured to use custom CGI error response pages for server response codes 401, 403, 413 and 500. The error pages are PERL CGI scripts that are initiated every time the server issues either of these response codes.

These scripts accomplish many important tasks including issuing an html warning banner to the client and immediately sending an e-mail notification to the WebAdmin. The e-mail message automates the process of manually collecting security related session information from the web server access and error logs for the request.

Create CGI Alerting Scripts to catch CGI Scanners. Use a CGI alerting script and rename it to vulnerable script names such as: test-cgi, phf, php.cgi, etc. When a CGI Vulnerability scanner is run against your web server, these scripts will be executed and the WebAdmin will be notified via email. (Sans/FBI, 2003)

10. Multiple Choice Question

What is the most likely purpose of the GET requests in the scan when most of the queries use the HEAD request?

aIt is an attempt to exploit a vulnerability in the page or script
bThe head request will not return a result to the client for that page or script
cThe scanning tool has been misconfigured
dThe page contents may have information that is useful to the scanner

The correct answer is (d). The GET request retrieves the entire contents of the page, the HEAD request retrieves only the headers. As this is a scan only it is most likely that the scanner is trying to access some information on the page or script.

11. References

Apache Software Foundation. "Log Files – Apache HTTP Server." URL: http://httpd.apache.org/docs/logs.html (24 May 2003).

Jones, Keith J. Shema, M. Johnson, Bradley C. Anti-Hacker Toolkit. Berkely, California: McGraw-Hill/Osborne, 2002. 172-179.

Northcutt, S. Cooper, M. Fearnow, M. Frederick, K. Intrusion Signatures and Analysis. Indianapolis: New Riders, 2001. 46-57.

SANS/FBI. "SANS/FBI Top 20 List." 29 May 2003. URL: http://www.sans.org/top20/ (2 Jun. 2003).

Stevens, W. Richard. TCP/IP Illustrated, Volume 3. Addison-Wesley, 1996. 161-176.

Analyse This

Executive Summary

This document is an analysis of a set of Snort log files for the period 15th June 2003 to 19th June 2003. These files were broken up into three distinct types, alerts generated by Snort rules, scans detected by the Snort portscan pre-processor, and Out-of-Spec files containing packet logs for anomalous traffic.

The log files posed a significant challenge to an analyst; the alert files alone contained over 1 million separate alerts. The log files showed signs of corruption that signifies overloading of the Snort sensor or logging systems.

The analysis shows significant malicious traffic both directed at and originating from the University's internal network. There is at least one Distributed Denial of Service agent on the network that participated in an attack during the period analysed; there is also evidence of significant trojan activity and port scanning from the internal network. In addition to this, the University has a large amount of file sharing activity on the network involving XDCC bots on Internet Relay Chat and peer-to-peer applications such as Gnutella and Kazaa.

In summary, the university needs to address the problems of malicious or undesirable software on the internal network. Consideration should be given to defining policy regarding the use of IRC and peer-to-peer applications and implementing safeguards to prevent their use or improve the security of these services.

Files Analysed

Table 1 – Files

Alerts	Port Scans	Out-of-Spec Files
alert.030615	scans.030615	OOS_Report_2003_06_16_7208
alert.030616	scans.030616	OOS_Report_2003_06_17_10478
alert.030617	scans.030617	OOS_Report_2003_06_18_18347
alert.030618	scans.030618	OOS_Report_2003_06_19_2056
alert.030619	scans.030619	OOS_Report_2003_06_20_9601

Each set of logs (Alerts, scans and OOS) were concatenated into one large file for ease of analysis. The OOS_Report files contained data from the day prior to that indicated in the name, for this reason, I used the OOS log files starting from the 16/6 rather than the 15/6.

Alert Analysis

The alert logs for this 5 day periods contain over 1.3 million alerts (including over 775,000 portscan alerts). The portscans make up over 50% of all alerts, so it appears that the portscan detector needs some tuning. In the remainder of the analysis I have excluded portscan alerts, as these are addressed in the scans files.

Excluding the portscan alerts, there is an average of over 4,500 alerts per hour. This is obviously a very difficult level of alert traffic to deal with, both for human analysts and the Intrusion Detection System (IDS) itself. It appears from the numerous errors

in the logs files that the IDS is having difficulty keeping up with its logging, so the first priority in the analysis is to identify and analyse the top five high volume alerts by volume/

Figure 1 shows that over 50% of the alert traffic is generated by only 2 rules, and over 75% of alerts are generated by 5 rules in total. By tuning or eliminating these rules, it will be possible to significantly decrease the alert traffic and reduce the load on the IDS. This will also have the advantage of making analysis of the log files produced by the IDS much easier, as the extremely large nature of the current log files makes even automated analysis a difficult and time consuming process.

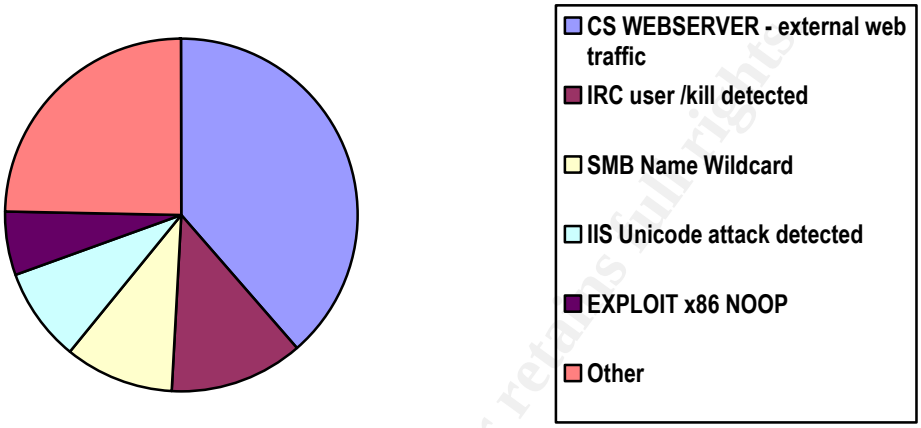


Figure 1 - Alert Traffic Volume

Table 2 - List of Alerts

No. of Alerts	Alert Name
209437	CS WEBSERVER - external web traffic
65703	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
54538	SMB Name Wildcard
46570	spp_http_decode: IIS Unicode attack detected
32316	EXPLOIT x86 NOOP
31908	MY.NET.30.4 activity
20796	Queso fingerprint
13097	MY.NET.30.3 activity
12042	CS WEBSERVER - external ftp traffic
11482	spp_http_decode: CGI Null Byte attack detected
6398	High port 65535 tcp - possible Red Worm - traffic
4489	TCP SRC and DST outside network
4383	High port 65535 udp - possible Red Worm - traffic
1563	IDS552/web-iis_IIS ISAPI Overflow ida nosize
1235	Null scan!
923	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected .
900	Tiny Fragments - Possible Hostile Activity
868	NMAP TCP ping!
556	connect to 515 from outside
452	Incomplete Packet Fragments Discarded

No. of Alerts	Alert Name
427	SUNRPC highport access!
402	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected
364	connect to 515 from inside
363	Possible trojan server activity
285	NETBIOS NT NULL session
265	SNMP public access
255	DDOS mstream handler to client
244	IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
229	SMB C access
163	NIMDA - Attempt to execute cmd from campus host
79	EXPLOIT x86 setuid 0
79	FTP passwd attempt
57	EXPLOIT x86 setgid 0
57	ICMP SRC and DST outside network
56	Notify Brian B. 3.54 tcp
54	Notify Brian B. 3.56 tcp
34	IRC evil - running XDCC
31	EXPLOIT x86 stealth noop
28	NIMDA - Attempt to execute root from campus host
27	Probable NMAP fingerprint attempt
24	RFB - Possible WinVNC - 010708-1
23	EXPLOIT NTPDX buffer overflow
20	Attempted Sun RPC high port access
19	TCP SMTP Source Port traffic
19	DDOS shaft client to handler
17	TFTP - External TCP connection to internal tftp server
14	[UMBC NIDS IRC Alert] K\:line'd user detected , possible trojan.
14	TFTP - Internal UDP connection to external tftp server
8	External FTP to HelpDesk MY.NET.70.49
7	External FTP to HelpDesk MY.NET.70.50
6	TFTP - External UDP connection to internal tftp server
5	Traffic from port 53 to port 123
4	External FTP to HelpDesk MY.NET.53.29
4	[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot.
3	SYN-FIN scan!
2	External FTP to HelpDesk MY.NET.83.197
2	DDOS mstream client to handler
2	External RPC call
2	[UMBC NIDS IRC Alert] User joining Warez channel detected. Possible XDCC bot
1	[UMBC NIDS IRC Alert] Possible trojaned machine detected
1	Fragmentation Overflow Attack
1	CS WEBSERVER - external ssh traffic
1	[UMBC NIDS IRC Alert] Possible trojaned box detected attempting to

No. of Alerts	Alert Name
	IRC
Total=542,583	Unique Alerts = 65

Top Five Alerts by Volume

For the initial analysis of IDS Alerts, I decided to examine the top 5 alerts by volume, as these rules generate 75% of the alert traffic. These alerts are:

CS WEBSERVER - external web traffic
[UMBC NIDS IRC Alert] IRC user /kill detected - possible trojan.
SMB Name Wildcard
spp_http_decode: IIS Unicode attack detected
EXPLOIT x86 NOOP

CS WEBSERVER – external web traffic
Alert Volume: 209437 (39%)
Current standard Snort rule: No
Possible Snort Alert Rule
alert TCP $EXTERNAL any -> MY.NET.100.165 $HTTP_PORTS (msg: "CS WEBSERVER - external web traffic";)

Details
This rule generated over 200,000 alerts over the 5 day period analysed, around 40% of all non-portscan alerts. This is not a current standard Snort rule, but the rule name seems to be fairly self explanatory. The "CS WEBSERVER - external web traffic" rule triggers when there are any connections from external addresses to port 80 on the CS Webserver (this probably means the Computer Science Webserver). The address of the CS Webserver is MY.NET.100.165.

There are a number of corrupted alerts for this rule with multiple destination addresses that initially made it appear that the rule triggered for other traffic as well, however further examination of the alert log files revealed the problem (see the Analysis Process section for more information on the log file corruption).

A quick examination of the source IP addresses generating this alert shows that the source address generating the highest number of alerts (216.39.48.127) belongs to the Altavista Company, a well known internet search company. The address resolves to buildrack52.sv.av.com, which is a known address for Altavista's Scooter 3.3 web crawler (Project BanBots, 2003).

The second most prolific alert generator is 66.77.73.164, this IP address belongs to Fast Search, Inc, another large internet search company. The address resolved to cr005r01-3.sac2.fastsearch.net, which is similar to addresses used by the Lycos web crawler in the past (IceHouse Designs, 2002).

The traffic generated by these two hosts is the normal activity of web bots indexing the University's web server, and the alerts for other hosts accessing the web server

are probably normal web traffic.

There is no question that either this alert rule should not be in the rule set, or the University border devices are configured incorrectly, allowing connections from external hosts to a web server intended solely for internal use.

Due to the extreme number of alerts over the 5 day period, It is most likely that the web server is intended for use by external clients, particularly as the web server appears to be being indexed by search engines. Search engine web crawlers do not usually scan hosts that do not have links elsewhere or have not been submitted for indexing.

There is no indication of this alert rule in previous GCIA practicals, indicating that this is a new rule.

Recommendations
The real purpose of this rule is unclear, although if the web server is supposed to be externally accessible then its most likely purpose is non-security related. It is possible that the University is using the IDS to measure visitors to the CS Webserver. If this is the case, the rule should be removed from the ruleset as soon as possible to reduce the load on the IDS logging mechanisms and the university should set up another mechanism to measure web traffic. If the CS Webserver is not supposed by be accessible to external hosts, then the University's firewalls need to be configured to block web traffic from external hosts to MY.NET.10.165.

[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan
Alert Volume: 65703 (12%)
Current standard Snort rule: No
Possible Snort Alert Rule
alert tcp $EXTERNAL_NET 6660:7000 -> $HOME_NET any (content: "ERROR \:Closing Link\: "; nocase; flow: established; msg: "IRC user /kill detected, possible trojan.";)

Details
This rule generates over 65,000 alerts over the 5 day period analysed, 12% of the non-portscan alert traffic. From the alert description I assume that it is triggered by Kill messages being sent to an Internet Relay Chat (IRC) client from an IRC server. This is not a current standard Snort rule, and appears to be part of a set of IRC related rules in the University's rule set with the [UMBC NIDS IRC Alert] heading. Although this is not a current standard rule, I found a set of rules very similar to that used by the University at (http://arpa.com/~nick/snort). The fact that the University has a set of IRC alerts shows that the university has had problems with malicious or illegal IRC activity in the past and has made this a priority for detection.

Kill messages are generated automatically when users attempt to join an IRC channel using a nickname that is already in use on the server (this is known as a Nick Collision kill). They are also generated by Channel Operators (ChanOps) when users on a channel are being abusive or breaching the Acceptable Use Policies of the IRC network. ChanOps can also add kill lines (K-lines) or autokill lines (A-kill) to

irc configuration files to ban users, hosts or entire networks from an IRC channel or network (resulting in the user being kicked off the channel immediately on connection) (http://www.valinor.sorcery.net/glossary/kline.html). The sheer volume of IRC kill alerts to some hosts makes it most likely that they have a bot installed and that has been A-line'd or K-line'd and are automatically trying to log back on to an IRC channel.

A bot is a program that logs into an IRC channel as a user to perform various operations like file sharing or information serving. There are also bots with malicious purposes, most Distributed Denial of Service attacks are now performed by zombie computers with bots installed that are controlled over IRC. There is evidence that some bot networks contain tens or even hundreds of thousands of hosts.

There are a variety of reasons why a K-line or A-line may be added, examples include where a trojan or backdoor program is detected in the IRC client, when the IP address or netblock of the client has been involved in Denial of Service attacks or when the client attempts to connect to a network multiple times (cloning). Some IRC networks also set restrictions on what bots are allowed to log in and K-line hosts when they are detected logging in using a non-authorised bot (R00ters, 2003).

Looking at the list of internal destination IP addresses for this alert, we find that the majority of the traffic is directed at only a few internal addresses. This is definitely symptomatic of the behaviour of an IRC bot, as it is unlikely that a real user on IRC would attempt to join a channel more than a few times if rejected by a kill message. I will examine the behaviour of the three destination hosts generating the majority of the traffic to further identify the problems.

Working with the top three destination IP addresses for these alerts, I found that the vast majority of the traffic relating to the alert is from 66.207.164.23 (irc1.aniverse.com - an IRC server specialising in Anime) to MY.NET.190.95. This pair of addresses make up 46,952 (71%) of all the IRC /kill alerts.

There are a multitude of other alerts relating to MY.NET.190.95, including 36 SMB Name Wildcard alerts, 7 SMB C Access, 2 NetBIOS NT Null sessions and an sdbot floodnet access attempt. These SMB and NetBIOS alerts show attempts to gain information from or access insecure file shares on a Windows host. The NetBIOS Null Session and SMB alerts are usually part of an already established TCP session, so the target machine is almost definitely a windows host listening on the NetBIOS ports (135-139), and from the distribution of alerts relating to the address, it appears to have been scanned and compromised between the 15th and 16th of June and may have been used in DDoS attacks on external hosts. It is also possible that the connections to the Anime IRC channels mean that MY.NET.190.195 is hosting Anime movie files and scanned images, although this host has not triggered any alerts for XDCC traffic.

The next most prolific host is MY.NET.83.100, generating 11,768 IRC /kill alerts. There are also 16,759 "[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC" alerts generated for this IP. XDCC is a Direct Client to Client connection (it does not operate through the IRC server), this is a popular method of sharing files via IRC and is often used for sharing software, movies, music and adult material,

both legal and illegal. It is a common occurrence for XDDC bots to be installed on compromised hosts to share such materials.

XDCC activity has been found to be particularly troublesome for University environments, which are prime targets due to the high bandwidth they usually have available. Universities are also very vulnerable to this sort of activity as the large number of hosts, permissive firewalling and difficulties in maintaining configuration control often make it easy for both network intruders and authorised users to install XDCC software on internal hosts (TonikGin, 2002).

MY.NET.83.100 appears to be successfully connecting to 4 IRC servers, generating the XDCC alert message, however it is being blocked by a fifth IRC server (205.160.101.121) generating the IRC kill alerts. 205.160.101.121 resolves to irc.rma.edu, an IRC server at the Randolph-Macon Academy, a school affiliated with the US Air Force. Irc.rma.edu is also generating a large number of Queso fingerprint alerts as well, it is likely that the IRC server is setting the ECN bits in the TCP Header, this generates a false positive alert in older versions of the Queso alert rule (Miller, 2000).

MY.NET.83.48 is the target host in 3955 of the IRC kill alerts. These kill messages are being generated by a variety of IRC servers, with no one server standing out from the rest. MY.NET.83.48 is also triggering a variety of other alerts. The most relevant alerts regarding this host are the K:\line's user detected, EXPLOIT x86 setuid 0 and Possible Incoming XDCC Send Request alerts.

The “EXPLOIT x86 setuid 0” rule is equivalent to the most recent “SHELLCODE x86 setuid 0” Snort rule. This rule has a high rate of false positives, and is often triggered by large binary transfers. Both destination and source port numbers are greater than 1024, this is the behaviour we would expect from client to client file transfers. From this it appears likely that MY.NET.83.48 is also hosting an XDCC bot and is actively serving files on a number of IRC channels.

The “[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan” alert rule generates a high proportion of noise, however it does provide insight into the IRC bot activity within the university network. This rule shows significant file sharing activity on the University's network. The existence of the alert rules for IRC XDCC activity in the university's Snort ruleset implies that XDCC activity is banned on the University's network. Interestingly enough, there are no alerts for other types of Peer-to-Peer file sharing such as Gnutella, suggesting that either the University has succeeded in preventing this traffic on the network (unlikely) or that they have accepted it's existence and are not using the Snort rules for detecting peer to peer activity.

Again, there are no correlations to this alert from previous GCIA practicals, indicating that the university has only recently begun focusing on IRC traffic.

Recommendations

Analysis of the top 3 destination hosts for this alert traffic show that the university appears to have a major problem with IRC activity on the internal network. There are a number of hosts that have file sharing bots installed, and at least one is possibly part of a DDoS botnet. Some of the IRC /kill alerts will be related to normal

IRC activity, however any host generating a significantly more alerts than most should be singled out for closer inspection. In addition, the university needs to carefully consider their policy on IRC activity on the internal network. With recent legal developments in the USA, it is possible that the University could find itself liable for copyrighted material that is being hosted on the internal network.

SMB Name Wildcard
Alert Volume: 54538 (10%)
Current standard Snort rule: No
Possible Snort Alert Rule
alert UDP $EXTERNAL any -> $INTERNAL 137 (msg: "SMB Name Wildcard"; content: "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA00 00";)

Details
This alert rule generated 54538 alerts in the set of alert files. Although the SMB Name Windcard rule is not part of the latest Snort ruleset, in older rulesets it alerts on a NetBIOS wildcard query when an external host sends a NetBIOS query to any host with a wildcard in the question field(*).

The presence of the string "CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA00 00" in the packet indicates a mangled name format where the ASCII value for each character in the name is split into two hex characters, then 0x41 is added to each value (Alexander, 2000)

This type of activity is often seen in internal network traffic when a windows host accesses a network share on another windows host. However, NetBIOS traffic should almost never be seen coming into the network from external hosts and is generally filtered, both incoming and outgoing, at the border devices. NetBIOS name queries are often used by attackers or worms scanning for vulnerable Windows hosts with insecure file sharing properties. A Windows host that receives a NetBIOS wildcard query will respond with information about the properties of the workstation including host name, Domain or Workgroup name, and a list of currently logged on users.

The fact that Windows hosts respond to a NetBIOS wildcard query is not generally considered a vulnerability and these queries are expected to be seen in normal internal network traffic (hosts use this type of query to resolve a NetBIOS name from an IP address). This type of query can easily be generated on a windows host using the nbtstat -A [IP Address] command on a Windows host.

There are a variety of sources for this alert, and the scanning activity is directed at over 1400 hosts. MY.NET.137.7 and MY.NET.132.24 are each the target of over 1400 NetBIOS wildcard queries, while no other host is the target of more than 200 of these queries. It seems likely that these two hosts are being actively targeted, while the rest of the traffic is random scanning activity.

The other alerts relating to the first host, MY.NET.137.7 are many and varied. There is a lot of traffic directed at ports 53 and 80. There are some Queso alerts for traffic to port 6346, these alerts are most likely due to a host setting the ECN bits in the

TCP header, but the fact that this traffic is directed at port 6346, and that some of the other alerts show odd port combinations, make it likely that the alerts are being triggered by Gnutella or similar peer-to-peer traffic. Peer-to-Peer networks are well known for using unusual flag combinations or TCP settings that can trigger IDS alerts (http://www.mcabee.org/lists/issforum/Aug-01/msg00132.html).

A more in-depth examination shows that it is likely that the majority of the alerts related to MY.NET.137.7 are for Gnutella traffic, there are connections between ports 80 and 53, an unusual combination in normal circumstances, however some versions of software on the Gnutella network allows users to set their own port. 80 and 53 are popular ports used to get around firewall restrictions as many networks let DNS and Web traffic into the network on a blanket basis (http://www.wired.com/news/business/0,1367,42438,00.html). I could not determine whether the two IIS related alerts, "IDS552/web-iis_IIS ISAPI Overflow ida nosize" and "spp_http_decode: IIS Unicode attack detected" were related to Gnutella activity, the ISAPI alerts are most likely relating to scanning activity by Code Red worms, however the Unicode alert is possibly related to Gnutella activity, as it is directed at port 8080 and this is not a port scanned by the Nimda and Code Red worms which commonly attempt to exploit this vulnerability.

MY.NET.132.24 has a number of more serious alerts associated with it, and appears to be hosting an sdbot trojan. This will be examined further in the analysis of the alerts for sdbot traffic.

This alert was the highest reported in Al Maslowski-Yerges' GCIA practical (Maslowski-Yerges, 2003), and is an extremely common scan type.

Recommendations
NetBIOS traffic should never be allowed to enter or leave the internal network and should therefore be blocked at the Border Routers as well as at the firewall (this protects against misconfiguration of either border device). In addition, internal hosts should be regularly scanned with vulnerability scanning tools to identify insecure file shares or Windows accounts with no password.

spp_http_decode: IIS Unicode attack detected
Alert Volume: 46570 (10%)
Current standard Snort rule: Yes
Snort Alert Rule
None, this alert is generated by the Snort http preprocessor and detects attempts at directory traversal using unicode to exploit vulnerabilities in the way IIS handles Unicode characters.

Details
Microsoft IIS 4 and 5 are both vulnerable to double dot "../" directory traversal exploitation if extended UNICODE character representations are used in substitution for "/" and "\". This allows access to files anywhere on the IIS host. The Unicode vulnerability is exploited by the Code Red v1, Code Red v2, CodeRed II, Nimda and Sadmind worms.

The Snort http preprocessor detects the Unicode representations of "/", "\" and ".".

This alert rule is sometimes triggered by dynamically generated URL's, search engines, websites in foreign languages, cookies and SSL traffic . It is possible that peer to peer traffic over HTTP ports could generate false positives as well, but I was unable to find any evidence of this. Two of the top three external destination hosts in these alerts belong to Netscape Incorporated and the other is a Korean ISP, this supports the theory regarding search engines and foreign languages (Berkers, 2001).

This is an extremely high number of alerts, and examination of the alert files shows that most of the alerts are generated by internal hosts. No single source or destination host stands out in the log files with significantly more traffic than any other, so either the internal network is infested with Worms, or most of the alerts for internal source hosts are false positives.

It is likely that a lot of the traffic with external source addresses is real worm traffic, Code Red and Nimda variants are still very common on the internet and a large network can expect to see many scans of this type on a daily basis.

Recommendations
Due to the fact that this vulnerability is now over 2 years old, and most of the traffic relating to it is from known worms, detection of unicode directory traversal should probably be disabled in the preprocessor using the -unicode switch in the snort.conf file. Snort.org state in their FAQ that one effective way of reducing false positives from the preprocessor is to use a BPF to ignore outbound HTTP traffic (Snort.org, 2003). In the university's case, this would result in the inability to monitor any outgoing HTTP traffic, so the best workaround would be to disable unicode detection in the portscan preprocessor.

Unicode alerts have been analysed in a number of past GCIA practicals, it was the highest alert generator in Doug Kite's analysis (Kite, 2002)

EXPLOIT x86 NOOP
Alert Volume: 32316 (6%)
Current standard Snort rule: Yes
Possible Snort Alert Rule
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE x86 NOOP"; content: "90 90 90 90 90 90 90 90 90 90 90 90 90 90"; depth: 128; reference:arachnids,181; classtype:shellcode-detect; sid:648; rev:5;)

Details
The EXPLOIT x86 NOOP alert is triggered by the detection of a series of x86 architecture NOOP instructions within a datastream (this is known as a NOOP sled). The NOOP sled is a series of "No Operation" instructions to the CPU that are often used in buffer overflow attempts to pad the exploit and increase the chance of successful execution of the exploit code. This particular alert is detecting a string of NOOP instructions specific to the Intel x86 type processor, there are also rules for detecting NOOP instructions for a variety of other processor types such as SPARC and SGI.

In the current Snort ruleset, this rule is now part of the SHELLCODE group, which

are disabled by default due to the high false positive rate generated by the alerts. The rule in the University's ruleset is an older version, the rule in the latest version of the Snort ruleset will only generate an alert when traffic is directed at a shellcode port, which is defined as any port other than 80. 31759 (98%) of the "EXPLOIT x86 NOOP" alerts in the University's network are for traffic directed at port 80.

The sequence of bytes that triggers this alert could be found in any binary file, and the majority of these alerts are likely to be false positives, particulary in the University's network where evidence suggests that there is significant file sharing activity. Any alerts for this rule that are valid will be impossible to verify without access to raw datastreams or examination of the target hosts of the alert traffic (Fitzgerald, 2001).

Fred Thiele mentions in his GCIA practical that this rule also commonly triggers for encrypted traffic such as SSL web traffic (Thiele, 2002).

Recommendations
In an environment like a university, the rule for detecting shellcode are probably not appropriate due to the high false positive rates and the traffic involving transfers of large binary files. If this rule was to be used, it needs to be restricted to only apply to a few high value or high profile internal hosts that are highly unlikely to be transferring large binary files.

If the university does need to continue using these rules, they should update to a newer version of the rule that excludes web traffic. This would significantly reduce the number of alerts that this rule creates and make it easier to detect real buffer overflow attempts.

Top Alerts by Severity

Having examined the five rules that generated the highest volume of alerts. I will next examine those alerts that may signify compromised internal hosts as I consider these to be of the highest severity.

These are the alert groups likely to be generated by compromised internal hosts:

TCP SRC and DST outside network
ICMP SRC and DST outside network
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
NIMDA - Attempt to execute cmd from campus host
NIMDA - Attempt to execute root from campus host
DDOS mstream client to handler
DDOS mstream handler to client
DDOS shaft client to handler

TCP SRC and DST outside network
Alert Volume: 4487 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rule

alert tcp $EXTERNAL_NET any -> $EXTERNAL_NET any (msg:"TCP SRC and DST outside network";)

ICMP SRC and DST outside network
Alert Volume: ICMP 57 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rule
alert icmp $EXTERNAL_NET any -> $EXTERNAL_NET any (msg:"ICMP SRC and DST outside network";)

Details
These alerts are two of the most serious in the log files. This traffic must be being generated by hosts inside the University's network, as otherwise it would not reach the IDS. Traffic with both Source and Destination addresses external to the network imply that the source host is spoofing its address. This type of activity may indicate that there is a host or hosts on the University's network infected with DDoS bots that are spoofing their IP addresses and participating in attacks on external hosts

Examination of the traffic generating the alerts shows a lot of traffic from source addresses in the following netblocks. 192.168.0.0/16, 169.254.0.0/16, 10.0.0.0/8, 172.16.0.0/12. These IP addresses are all special use Ipv4 addresses as defined by RFC3330. The 192.168.0.0/16. 10.0.0.0/8 and 172.16.0.0/12 netblocks are all RFC1918 type addresses allocated for private networks, they are non-routable on the internet. The 169.254.0.0/16 block is the 'link local' block allocated for use in communiation between hosts on a single link. Hosts commonly allocate themselves an IP address in this netblock when they cannot contact a DHCP server (http://www.rfc-editor.org/rfc/rfc3330.txt).

The traffic listed above is not a problem from a security perspective, as these addresses are not routable on external networks and therfore cannot be part of an attack. Traffic from special use addresses is most likely to be generated by misconfigured hosts on the internal network.

The traffic that should be of concern is the traffic directed at 67.80.77.94. There are over 3800 alerts for traffic directed at TCP port 6112 on this host. The traffic directed at this host has a variety of different, seemingly random source IP addresses, and appears to be definitely spoofed. There are two possible targets for the attack on this port. One possibility is that the target is a Unix host running the Common Desktop Environment (CDE). CDE's desktop subprocess control process (dtspc) uses this port, and is known to have contained buffer overflows in the past (http://www.cert.org/advisories/CA-2001-31.html). However, I could find no evidence that this service is vulnerable to any kind of flooding DoS attack. The other possible target could be a games server, Blizzard Entertainment games such as Starcraft, Warcraft and Diablo use port 6112 for the battle.net service to run multiplayer games (Cardoso, 2000).

It is most likely that the target host is a gaming server, these types of server are known to be among the most popular targets for Denial of Service activity. (Moore et al, 2001)

The target host in the DDoS attack is a cable modem system connected to an ISP called Optimum Online. Registration details for this host are below.

IP address: 67.80.77.94
Host name: ool-43504d5e.dyn.optonline.net
OrgName: Optimum Online (Cablevision Systems)
OrgID: OPTO
Address: 111 new south RD
City: Hicksville
StateProv: NY
PostalCode: 11801
Country: US

NetRange: 67.80.0.0 - 67.87.255.255
CIDR: 67.80.0.0/13
NetName: NETBLK-OOL-4BLK
NetHandle: NET-67-80-0-0-1
Parent: NET-67-0-0-0-0
NetType: Direct Allocation
NameServer: NS.CV.NET
NameServer: NS.CVNET.COM
Comment: ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
RegDate: 2001-11-09
Updated: 2003-01-29

TechHandle: OH4-ORG-ARIN
TechName: OOL Hostmaster
TechPhone: +1-516-803-3000
TechEmail: hostmaster@cv.net

OrgAbuseHandle: OOLAB-ARIN
OrgAbuseName: OOL Hostmaster
OrgAbusePhone: +1-516-803-2400
OrgAbuseEmail: abuse@cv.net

OrgTechHandle: OH4-ORG-ARIN
OrgTechName: OOL Hostmaster
OrgTechPhone: +1-516-803-3000
OrgTechEmail: hostmaster@cv.net

ARIN WHOIS database, last updated 2003-07-21 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

There are no correlations for this alert from previous GCIA practicals, it is likely that the university has just implemented the rule.

Recommendations
Implement egress filtering at all routers, both internal and external. It will probably be necessary to use packet sniffers on the internal network to detect which host is

generating the DDoS traffic as the source IP addresses are spoofed. By using sniffers it will be possible to identify the MAC address of the malicious host and from this identify it's correct IP address.

[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
Alert Volume: 923 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rule
Unknown

Details
This alert rule is a custom one used by the University to detect sdbot trojan activity on IRC. The description of the rule is a little vague, and it is difficult to determine what it is triggering on. If these alerts are not false positives, then it appears there are hosts in the internal network infected with the sdbot trojan.

The sdbot trojan was first discovered in April 2002, it spreads via IRC, and is usually presented on IRC channels as files that will be popular to download (such as movies, music etc). One discussion on its use shows it being presented as a password cracker for gaining free access to adult websites (Ryan1918, 2003).

When the file is downloaded and executed, the trojan installs itself and attempts to log into a predefined IRC channel from which it will receive commands. There are a variety of functions that the sdbot trojan will perform after receiving commands on the IRC channel. These include updating the installed Trojan, sending the Trojan to other IRC channels in an attempt to compromise more computers, performing DDoS attacks, and uninstalling itself. There are a number of sdbot variants avaliable, and new variants are still being released (Symantec, 2003).

There are 8 internal hosts generating the sdbot alert traffic. The majority of the traffic is generated by MY.NET.84.228 and MY.NET.132.24. MY.NET.84.228 generated 861 alerts, and MY.NET.132.24 generated 56 alerts.
The following hosts each generated 1 alert:
MY.NET.97.97
MY.NET.97.76
MY.NET.97.247
MY.NET.153.120
MY.NET.97.74
MY.NET.97.72

The alert traffic from MY.NET.84.228 is all related to communication with 206.167.75.78. This IP address resolves to crcri.quest.net , an IRC server on the Efnet IRC network. This pair of addresses has also generated a high number of "[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan" alerts. It is most likely that MY.NET.84.228 has been banned from the IRC server because of malicious activity or because it is hosting the sdbot trojan.

The alert traffic from MY.NET.132.24 is all related to communication with 217.211.72.145, a host that belongs to Telia Network Services. I was unable to find any information on this host, but it is probably a standalone IRC server that is not

part of the larger IRC networks. MY.NET.132.24 also generates a large number of other alerts including SMB Name Wildcard, NetBIOS NT NULL Session, IRC evil – running XDCC and SMB C access. This host is almost definitely compromised in some manner and is highly likely to be running the sdbot, the XDCC requests are evidence that it is spreading itself on IRC by masquerading as a tempting file.

Because of the variety of traffic relating to this host, a link graph is useful to better understand the alert traffic.

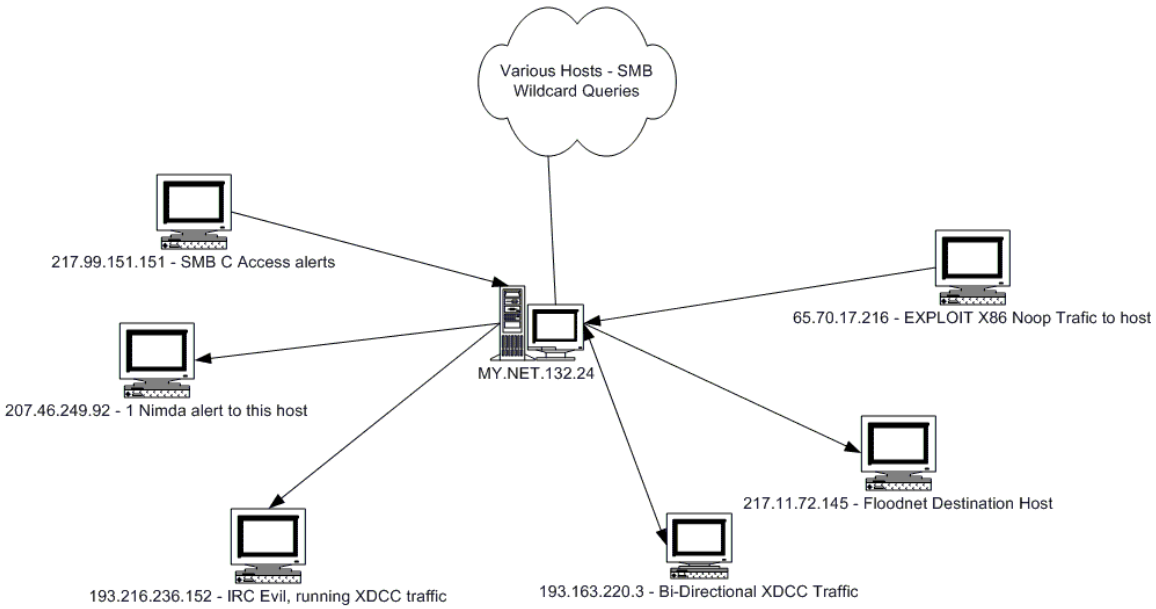


Figure 2 - Traffic to/from MY.NET.132.24

The link graph shows a lot of activity to and from this host. There are a large number of SMB Name Wildcard requests coming from a variety of different hosts, this traffic is suspicious as most of it comes from a number of random hosts within a group of class B netblocks, none of which appear to be related to one another. It is possible that a lot of this activity is using spoofed addresses to mask the address of the real scanner of intruder. This activity is trivial to spoof as the wildcard queries use UDP packets.

There are four IRC servers that MY.NET.132.24 is logging in to or attempting to log in to, if this host is infected with the sdbot trojan, this is most likely propagation activity. There are two “SMB C Access” alerts from 217.99.151.151 signifying attempts to access the administrative share of the hosts C drive, these are among the most significant with relation to compromise of the host.

Another traffic flow of concern is the EXPLOIT x86 NOOP traffic from 65.70.17.216 directed at port 445 of the target host. It is possible that this host is either attempting a buffer overflow attack against MY.NET.132.24 or is uploading a binary file to the host. If this is the case, it is possible that this is the method used to infect the host with the sdbot trojan.

Registration information for Host 217.99.151.151, source of “SMB C Access” alerts.

```
% This is the RIPE Whois server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/ripencc/pub-services/db/copyright.html

inetnum:      217.99.151.128 - 217.99.151.255
netname:      NOWA-HUTA-SDI
descr:        TP S.A. SDI
descr:        Nowa Huta Krakow
country:      PL
admin-c:      ZP299-RIPE
tech-c:       PB7294-RIPE
status:       ASSIGNED PA
mnt-by:       TPNET
changed:      tkielb@cst.tpsa.pl 20011012
changed:      tkielb@cst.tpsa.pl 20011022
source:       RIPE

route:        217.99.0.0/16
descr:        TPNET
descr:        for abuse: abuse@tpnet.pl
origin:       AS5617
mnt-by:       AS5617-MNT
changed:      nabn@tpnet.pl 20030228
source:       RIPE

person:       Zbigniew Pacut
address:      Telekomunikacja Polska S.A.
address:      Zaklad Telekomunikacji Krakow-Centrum
address:      Oddzial Teleinformatyki
address:      ul. Rakowicka 51
address:      31-510 Krakow
address:      Poland
phone:        +48 12 4239238
fax-no:       +48 12 4239123
e-mail:       zpacut@krakow.tpsa.pl
nic-hdl:      ZP299-RIPE
mnt-by:       TPNET
changed:      tkielb@cst.tpsa.pl 20000321
source:       RIPE

person:       Piotr Bialka
address:      TPSA, Zaklad Telekomunikacji Krakow
address:      POLAND
phone:        +48 12 4239238
fax-no:       +48 12 4239123
```

e-mail: pbialka@krakow.tpsa.pl
nic-hdl: PB7294-RIPE
mnt-by: TPNET
changed: tkielb@cst.tpsa.pl 20000219
source: RIPE

Registration information for 65.70.17.216, source of “EXPLOIT x86 NOOP” traffic.

CustName: Paragould City Light & Water
Address: 2701 W. 15th
City: Plano
StateProv: TX
PostalCode: 75075
Country: US
RegDate: 2001-08-30
Updated: 2001-08-30

NetRange: 65.70.16.0 - 65.70.23.255
CIDR: 65.70.16.0/21
NetName: SBCIS-101829-131419
NetHandle: NET-65-70-16-0-1
Parent: NET-65-64-0-0-1
NetType: Reassigned
Comment:
RegDate: 2001-08-30
Updated: 2001-08-30

TechHandle: ZS44-ARIN
TechName: IPAdmin-SBIS
TechPhone: +1-888-212-5411
TechEmail: IPAdmin-SBIS@sbcis.sbc.com

OrgAbuseHandle: ABUSE6-ARIN
OrgAbuseName: Abuse - Southwestern Bell Internet
OrgAbusePhone: +1-877-722-3755
OrgAbuseEmail: abuse@swbell.net

OrgNOCHandle: SUPPO-ARIN
OrgNOCName: Support - Southwestern Bell Internet Services
OrgNOCPhone: +1-888-212-5411
OrgNOCEmail: support@swbell.net

OrgTechHandle: IPADM2-ARIN
OrgTechName: IPAdmin-SBIS
OrgTechPhone: +1-888-212-5411
OrgTechEmail: IPAdmin-SBIS@sbis.sbc.com

ARIN WHOIS database, last updated 2003-07-23 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

Recommendations
All eight hosts on the internal network triggering this alert should be taken offline and scanned to determine whether they are infected and with what. The university appears to have serious problems with malicious or illegal IRC traffic on the network, and needs to formulate some policy regarding it's use and how to control or monitor it effectively. These hosts should be considered a top priority, as it is likely at least one of them has already been involved in a DDoS attack on external hosts.

IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize
Alert Volume: 244 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rule
alert tcp $INTERNAL_NET any -> EXTERNAL_NET $HTTP_PORTS (msg:"IDS/552/web-iis IIS ISAPI Overflow ida INTERNAL nosize"; uricontent:".ida?"; nocase; flow:to_server,established;)

Details
This rule triggers an alert when the IDS detects an attempt to access a .ida file on a web server. Attempts to access a .ida file show that a host is attempting to exploit a buffer overflow vulnerability in the Indexing service on Microsoft Internet Information Server (IIS) 4 and 5 (CERT/CC, 2002) Successful exploitation of the vulnerability can result in the intruder gaining system level priveleges on the compromised host. Activity that triggers this alert is generally that of one of the Code Red worms scanning for other vulnerable hosts to exploit. In this case, the rule triggers when the source of the scanning activity is an internal host, this makes it a high severity alert as it means that an internal host has been infected with the worm. There are three internal hosts that generate the alert, so it is most likely that all three are infected with a variant of the Code Red worm. The infected hosts are MY.NET.98.59, which generates 206 alerts, MY.NET.98.41, which generates 35 alerts and MY.NET.97.208, which generates 3 alerts.

Most of the alert traffic is directed at external hosts in the 130.0.0.0/8 netblock, with a few other external addresses scattered throughout the logs. This makes it most likely that the alert traffic is probing from the CodeRedII worm. The CodeRedII worm uses preferential local subnet scanning, an infected host has a 12.5% probability of probing a completely random IP address, a 50% probability of probing an address in the same Class A netblock, and a 37.5% probability of probing an address on the same Class B netblock (CAIDA, 2003). Because the rule has only alerted on traffic to external network addresses, we only see the random probes and those in the 130.0.0.0/8 netblock, there will be many more probes to hosts on the internal network.

Recommendations
The internal hosts that generate these alerts must be immediately taken offline. CodeRedII installs a backdoor program on an infected server, so for preference, these hosts should be completely reinstalled and secured, as removal of the worm does not guarantee the security of the host. If these hosts do not act as web servers the firewall should block incoming connections to port 80 and the Microsoft IIS application should be removed.

NIMDA - Attempt to execute cmd from campus host
Alert Volume: 163 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rules
alert tcp $INTERNAL_NET any -> EXTERNAL_NET $HTTP_PORTS (msg:"NIMDA - Attempt to execute cmd from campus host "; uricontent:"cmd.exe"; nocase; flow:to_server,established;)

NIMDA - Attempt to execute root from campus host
Alert Volume: 28 (<1%)
Current standard Snort rule: No
Possible Snort Alert Rules
alert tcp $INTERNAL_NET any -> EXTERNAL_NET $HTTP_PORTS (msg:"NIMDA - Attempt to execute cmd from campus host "; uricontent:"root.exe"; nocase; flow:to_server,established;)

Details
These alerts trigger on detection of probes from internal hosts by the Nimda worm. Nimda attempts to exploit directory traversal vulnerabilities in IIS and also attempts to access backdoors left by the CodeRedII worm.

There are two hosts in the internal network that are highly likely to be infected with the Nimda worm, MY.NET.98.59 which generates 147 alerts and MY.NET.98, which generates 34 alerts. These hosts seem to be scanning random external addresses. MY.NET.97.208 may also be infected, however, it only generates 2 alerts and most Nimda infected hosts would be more prolific in their probing activity.

There are 7 other hosts that generated alerts for this traffic. None of these hosts generated more than 2 Nimda alerts. The destination addresses in all alerts from these hosts were Microsoft web servers, and many of them appear to be Windows Update sites. These factors make it likely that the alerts for this traffic are false positives, as Nimda probe traffic is completely random in nature.

Recommendations
The Nimda worm uses multiple propagation techniques, so it is difficult to know whether the infected host is a workstation or a server with Microsoft IIS installed. The three possibly infected hosts need to be taken offline and examined to confirm whether they are infected with the Nimda worm, then they should be rebuilt and properly secured. Because Nimda has now been around for over 2 years, it is likely that these are newly installed hosts that are vulnerable by default. Systems administrators on the University network need to be educated on proper system hardening and patch management procedures.

DDOS mstream client to handler
Alert Volume: 2 (<1%)
Current standard Snort rule: Yes
Snort Alert Rule
alert tcp $EXTERNAL_NET any -> $HOME_NET 15104 (msg:"DDOS mstream client

to handler"; flags: S; reference:arachnids,111; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:249; rev:1;)

DDOS mstream handler to client
Alert Volume: 255 (<1%)
Current standard Snort rule: Yes
Snort Alert Rule
alert tcp $HOME_NET 12754 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to client"; content: ">"; flags: A+;reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:248; rev:1;)

DDoS shaft client to handler
Alert Volume: 19 (<1%)
Current standard Snort rule: Yes
Snort Alert Rule
alert tcp $EXTERNAL_NET any -> $HOME_NET 20432 (msg:"DDOS shaft client to handler"; flags: A+; reference:arachnids,254; classtype:attempted-dos; sid:230; rev:1;)

Details
These three alert rules are all used for detecting communication to or from DDoS agents. Mstream and Shaft are both well known DDoS attack tools, similar in structure and function. Both use a heirachical network of clients, handlers and agents to launch DDoS attacks. The attacker controlling the network sends commands to the handler(s) from the client. The handler then relays the commands to agents under its control, which launch the DDoS attack (Ditrich et al, 2000).

The “DDOS mstream client to handler” alert rule is a simple one, triggering on an attempted initial TCP connection from an external host to port 15104 on an internal host, this is likely to have a generate false positives for normal traffic. Neither of the destination hosts in these alerts show evidence of further DDoS communication, so it is unlikely that they are actually mstream handlers.

There are 255 “DDOS mstream handler to client” alerts, neither are related to the “DDOS mstream client to handler” alerts. These alerts are all being triggered by traffic from MY.NET.84.235 to 80.100.101.176. The destination port for the traffic is 4662, one of the default server ports for eDonkey2000 peer to peer traffic according the Incidents.org, it is most likely that binary file transfers over the eDonkey2000 network are triggering this alert.

Most of the “DDOS shaft client to handler” alerts appear to be false positives as well, the rule is a simple one that alerts on established connections to port 20432 on the internal network, port 20432 is a legitimate client port that appears to be being used here in web and smtp traffic for normal purposes. The traffic from ports 4661 and 4662 are most likely to be e-donkey2000 peer-to-peer file sharing traffic, as these are default server ports for e-donkey. The only connection that could possibly be real DDoS agent traffic is the connection from 213.46.226.54 on port 23504 to MY.NET.97.74. It is possible that this host is a shaft handler, however we would probably expect to see more traffic to or from the host involved in the alert.

Recommendations
It is unlikely that any of the hosts listed are really involved in DDoS activity. The alert rules are quite broad, and are likely to generate false positives. 213.46.226.54 should be investigated to determine the nature of the traffic on port 23504. These rules should continue to be monitored, as if an internal host does become infected with one of these DDoS agents the alert traffic would increase significantly.

Scans

There were over 9,271,000 scans recorded in the scans log files over the 5-day period. These scans can be broken down into a number of different types. More than 99% of the traffic in the scans files consists of UDP and SYN scans. These types of scans are often false positives if the portscan pre-processor is not tuned correctly. Also, over 80% of the scan traffic originated from internal hosts. This makes it highly likely that the portscan pre-processor is badly configured and needs to be tuned. The spp_portscan alerts in the alerts files show that the portscan processor implementation used by the University triggers an alert if 12 unique connections are detected from any 1 host. The portscan pre-processor seems to be using a long time period for detections, as some alerts are triggered for 12 connections in as much as 49 seconds. The internal IP addresses in the scans files are in the format 130.85.x.x, rather than MY.NET.x.x as in the alerts files.

Table 3 - List of Scans

Volume	Scan Type
7194830	UDP scan (Internally-based)
1519589	SYN scan (Externally-based)
523120	SYN scan (Internally-based)
31194	UDP scan (Externally-based)
640	NULL scan (Externally-based)
413	UNKNOWN scan (Externally-based)
389	INVALIDACK scan (Externally-based)
317	NOACK scan (Externally-based)
235	FIN scan (Internally-based)
217	FIN scan (Externally-based)
101	VECNA scan (Externally-based)
84	UNKNOWN scan (Internally-based)
38	XMAS scan (Externally-based)
32	NMAPID scan (Externally-based)
11	FULLXMAS scan (Externally-based)
10	scan (Externally-based)
6	SPAU scan (Externally-based)
5	SYNFIN scan (Externally-based)
4	NULL scan (Internally-based)
2	scan (Internally-based)
1	INVALIDACK scan (Internally-based)

For the purpose of this analysis I will focus on the UDP and SYN scans, the top ten generators of this UDP and SYN scan traffic are all internal hosts, and the scans traffic from each will be analysed below to identify the nature of the traffic in the

scans files.

130.85.1.3 and 130.85.1.4
These hosts appears to be a DNS servers on the internal network, it is initiating connections to multiple hosts on port 53. These host are also initiating connections to external hosts on port 123, port 123 is the port for Network Time Protocol, this makes it more likely that these is a DNS server which periodically syncronises its clock with an external NTP server.

130.85.83.170
The traffic from this host has a source port of 1992 and is destined to a variety of destination hosts on a variety of ports. There is little information available regarding this port, it is used as a server port for Cisco STUN-P3 and ipsendmsg. Investigation of this host is required to determine whether this traffic is malicious as I was able to find little information regarding these services.

130.85.153.223, 130.85.153.190 and 130.85.84.178
The traffic from these hosts is made up of connections to and from port 6257, this is the default port for a peer-to-peer application called WinMX (WinMX, 2002). It is most unlikely that this traffic is malicious, but the sheer volume of peer-to-peer activity means that these hosts should be investigated. This traffic is also seen in Al Maslowski-Yerges' analysis, and also triggered a number of "Possible Red Worm – traffic" alerts (Maslowski-Yerges, 2003).

130.85.97.16 and 130.85.97.145
Both of these hosts are scanning the NetBIOS name service on external hosts, the traffic is definitely malicious, they are scanning all hosts in a number of subnet sequentially. 130.85.97.145 is also scanning a number of hosts on UDP port 3036, I could find no information on services or trojan programs running on this port, but it could be a customised version of any common piece of malicious software.

130.85.100.230
This host is initiating multiple connections to external hosts on port 25, this makes it highly likely that the host is a mail server on the internal network. Within the scans traffic there is the occasional connection to port 113, this port runs the ident service, and some mail transfer agents connect to this port by default before accepting mail from an external host (Baldwin, 2001).

130.85.88.198
The majority of traffic from this host has a source port of 3456, these are UDP packets with a variety of external destination addresses and ports. This indicates that the activity is in response to traffic to port 3456 on 130.85.88.198. According to incidents.org this port is the default port for Terrortrojan and VAT default data. VAT is a multicast audio conferencing application developed by the Network Research Group of Lawrence Berkeley National Laboratory (Jacobson and McCanne, 2003). It is quite possible that this host is involved in using this application, however more investigation is required.

Scans Recommendations

Tune the portscan pre-processor to alert after a short period of time, this will require

some research and monitoring to determine the background level of traffic on the network. The University should use the ignorehosts option on the pre-processor to ignore traffic from known DNS and Mail servers, which by nature will be initiating multiple connections in a short period of time. This will reduce the level of false positives scan detections significantly. There are also at least two hosts on the network involved in NetBIOS scans against external hosts. These should be immediately investigated to determine whether the cause is a worm or an individual at the University.

Out-of-Spec Analysis

There are just over 43,000 alerts for Out of Spec (OOS) packets in the alert files. Out of Spec packets are usually caused by either packet corruption or packet crafting. On this occasion, all but 93 of the packet logs are being caused by implementation of the ECN standard (Explicit Congestion Notification). This standard is defined in RFC2481 and uses the two reserved bits in byte 13 of the TCP header to signal network congestion. The bits are both set to indicate that the source host is ECN capable. These packets should no longer be considered out of spec, ECN is a standard that will be implemented by more systems in future, and the presence of these packets in the OOS logs are just noise.

After removing this traffic and examining the rest of the alerts, I found that most of the other OOS packets were caused by peer-to-peer traffic (both Kazaa and Gnutella) and web traffic. There were some packet traces that may be malicious, however, without full logs it is difficult to determine the purpose of the packets. See below for examples of the types of packet traces seen in the OOS logs.

Kazaa Traffic

```
06/16-00:10:07.092128 148.64.168.64:3270 -> MY.NET.98.26:3376
TCP TTL:114 TOS:0x0 ID:51787 IpLen:20 DgmLen:441 DF
****P*** Seq: 0xA0B9B00A  Ack: 0x0  Win: 0x2000  TcpLen: 20
47 45 54 20 2F 2E 68 61 73 68 3D 30 32 34 36 32  GET /.hash=02462
33 62 31 35 35 31 32 66 38 32 62 36 35 64 65 66  3b15512f82b65def
62 37 62 39 38 61 61 36 38 65 35 33 36 37 63 63  b7b98aa68e5367cc
39 35 32 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  952 HTTP/1.1..Ho
73 74 3A 20 31 33 30 2E 38 35 2E 39 38 2E 32 36  st: MY.NET.98.26
3A 33 33 37 36 0D 0A 55 73 65 72 41 67 65 6E 74  :3376..UserAgent
3A 20 4B 61 7A 61 61 43 6C 69 65 6E 74 20 4E 6F  : KazaaClient No
76 20 20 33 20 32 30 30 32 20 32 30 3A 32 39 3A  v  3 2002 20:29:
30 33 0D 0A 58 2D 4B 61 7A 61 61 2D 55 73 65 72  03..X-Kazaa-User
6E 61 6D 65 3A 20 53 68 61 64 6F 77 6F 6E 74 68  name: Shadowonth
65 73 75 6E 0D 0A 58 2D 4B 61 7A 61 61 2D 4E 65  esun..X-Kazaa-Ne
74 77 6F 72 6B 3A 20 4B 61 5A 61 41 0D 0A 58 2D  twork: KaZaA..X-
4B 61 7A 61 61 2D 49 50 3A 20 31 34 38 2E 36 34  Kazaa-IP: 148.64
2E 31 36 38 2E 36 34 3A 33 33 38 37 0D 0A 58 2D  .168.64:3387..X-
4B 61 7A 61 61 2D 53 75 70 65 72 6E 6F 64 65 49  Kazaa-SupernodeI
50 3A 20 32 30 38 2E 36 31 2E 32 37 2E 36 30 3A  P: 208.61.27.60:
32 34 31 37 0D 0A 52 61 6E 67 65 3A 20 62 79 74  2417..Range: byt
65 73 3D 31 36 30 35 36 33 32 2D 32 30 30 37 30  es=1605632-20070
33 39 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20  39..Connection:
```

63 6C 6F 73 65 0D 0A 58 2D 4B 61 7A 61 61 2D 58 close..X-Kazaa-X
66 65 72 49 64 3A 20 38 34 31 31 34 31 31 0D 0A ferId: 8411411..
58 2D 4B 61 7A 61 61 2D 58 66 65 72 55 69 64 3A X-Kazaa-XferUid:
20 70 62 53 32 6C 35 42 70 65 65 4A 71 44 6B 75   pbS2l5BpeeJqDku
46 5A 37 5A 7A 42 2F 37 2B 59 37 34 52 4F 45 6D FZ7ZzB/7+Y74ROEm
6A 50 41 2F 36 65 4C 67 53 2B 78 59 3D 0D 0A 0D jPA/6eLgS+xY=...
0A

Web Traffic with TCP Options
06/16-13:36:21.514252 198.92.125.250:80 -> MY.NET.81.4:3331
TCP TTL:49 TOS:0x0 ID:11124 IpLen:20 DgmLen:806 DF
TCP Options (1) => Opt 172 (40): A236 58D1 0083 FABB 57DB D16B ADAC 143B
691E CEDB 04FF 00F4 4DEC 04BA B6FB ADA9 8181 8181 8181
81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ....
81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 ....
81 81 81 81 81 81 31 20 47 4D 54 0D 0A 43 6F 6E ......1 GMT..Con
74 65 6E 74 2D 54 79 70 65 3A 20 69 6D 61 67 65 tent-Type: image
2F 67 69 66 0D 0A 41 63 63 65 70 74 2D 52 61 6E /gif..Accept-Ran
67 65 73 3A 20 62 79 74 65 73 0D 0A 4C 61 73 74 ges: bytes..Last
2D 4D 6F 64 69 66 69 65 64 3A 20 54 75 65 2C 20 -Modified: Tue,
32 33 20 4F 63 74 20 32 30 30 31 20 31 38 3A 34 23 Oct 2001 18:4
35 3A 30 30 20 47 4D 54 0D 0A 45 54 61 67 3A 20 5:00 GMT..ETag:
22 30 39 65 39 63 64 31 66 32 35 62 63 31 31 3A "09e9cd1f25bc11:
39 34 66 22 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 94f"..Content-Le
6E 67 74 68 3A 20 35 34 31 0D 0A 0D 0A 47 49 46 ngth: 541....GIF
[Packet contents truncated]

Unknown traffic, possibly malicious
06/18-06:27:43.805523 64.105.9.226:61636 -> MY.NET.83.170:61636
TCP TTL:111 TOS:0x0 ID:3124 IpLen:20 DgmLen:1372 DF
******** Seq: 0x7C8111F Ack: 0x5A0329C5 Win: 0x5010 TcpLen: 36
TCP Options (1) => EOL
[Full Packet Trace removed]

Possible Gnutella, possibly malicious
06/18-18:03:12.047245 211.229.105.244:4662 -> MY.NET.153.144:0
TCP TTL:109 TOS:0x0 ID:58219 IpLen:20 DgmLen:1362 DF
**UA*RSF Seq: 0x63B0278 Ack: 0x48E5A915 Win: 0x5018 TcpLen: 52 UrgPtr:
0xC1B7
TCP Options (1) => EOL
[Full Packet Trace removed]

The Web traffic with TCP options from 198.92.125.250 seemed to be very peculiar, so I looked up the registration information for this host.

OrgName: Millennium Systems
OrgID: MILLEN-74
Address: 18003 Sky Park Circle
City: Irvine
StateProv: CA
PostalCode: 92614

Country: US

NetRange: 198.92.120.0 - 198.92.127.255
CIDR: 198.92.120.0/21
NetName: MILLSYS-120-26
NetHandle: NET-198-92-120-0-1
Parent: NET-198-92-0-0-1
NetType: Reassigned
Comment:
RegDate: 2001-12-27
Updated: 2001-12-27

TechHandle: BL790-ARIN
TechName: Lewis, Brian
TechPhone: +1-949-252-8772
TechEmail: brian@nextmill.net

OrgTechHandle: BL790-ARIN
OrgTechName: Lewis, Brian
OrgTechPhone: +1-949-252-8772
OrgTechEmail: brian@nextmill.net

ARIN WHOIS database, last updated 2003-07-23 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

Millennium systems appears to be a legitimate Web Hosting company, so it is likely that is traffic is benign.

Top ten generators of Traffic separated by type

Alerts

Host Address	Volume
MY.NET.83.100	16759
66.207.164.23	47047
216.39.48.127	45305
205.160.101.121	23496
68.49.35.0	8985
212.202.40.149	7243
24.35.62.222	5197
193.225.219.29	4474
MY.NET.153.185	4392
66.77.73.164	4168

Scans

Host Address	Volume
130.85.1.3	2396759
130.85.83.170	495810
130.85.1.4	370425
130.85.153.223	319790
130.85.97.16	251963
130.85.100.230	221260
130.85.153.190	202614
130.85.97.145	193730
130.85.88.198	191746
130.85.84.178	188017

Out-of-Spec

Host Address	Volume
205.160.101.121	26251
216.95.201.21	726
216.95.201.24	638
216.95.201.23	628
216.95.201.25	588
216.95.201.22	566
216.95.201.34	557
216.95.201.20	555
216.95.201.30	490
216.95.201.33	439

Defensive Recommendations

Firewall or Border Router rules should also be configured to drop traffic directed at NetBIOS ports (135-139 and 449, TCP and UDP), as there is usually no reason for external computers to initiate connections to internal computers on these ports. This would significantly reduce the exposure of University Windows hosts to intrusions, as evidence suggests that insecure file shares are being used to compromise internal hosts. In addition, the university needs to set up strong configuration and change control mechanisms, as there are hosts on the internal network that have been infected with CodeRedII and Nimda worms due to poor setup and maintenance.

One of the most important tasks the University should undertake is to formulate a policy on the use of IRC in the university network. If XDCC traffic is to be banned, it may be worth investigating active response or Intrusion Prevention mechanisms to prevent the XDCC agents from logging in to IRC servers. Also, the high level of peer-to-peer traffic is a drain on network resources, exposes the university to greater risk of compromise, and wastes analyst time by generating IDS alerts. The University should investigate the possibilities of setting up legal, internal file sharing systems. Although this would have a setup and licensing costs, it is likely that the cost savings generated in Internet traffic costs alone would balance these out.

The university needs to implement egress filtering at all external routers, router logs should be reviewed regularly to confirm that there is no IP spoofing activity on the internal network. Priority needs to be given to identifying the host on the internal network that is participating in DDoS attacks on external hosts.

Analysis Process

The first step in the analysis involved concatenating each type of file together to get a single alert file, scans file and OOS_Report file. After this, I used grep to remove all spp_portscan entries from the alert files as the scans are already detailed in the scans file.

After beginning my analysis I identified that the logs were in fairly bad shape, with around 1500 alerts run together on 1 line, a number of alerts missing fields such as source or destination port, and some even appeared to have two destination addresses or ports in the alert. The following are examples of the corruption I encountered in the log files provided.

No Destination address
06/15-00:45:40.629217 [**] SMB Name Wildcard [**] 202.64.208.161

2 Destination Addresses (plus 2 destination ports for the second address
06/19-15:24:21.710220 [**] CS WEBSERVER - external web traffic [**]
62.232.9.130:4211 -> 205.188.149.12:10340 -> MY.NET.100.165:6667:80

2 Alerts run together on 1 line, second alert spread over 2 lines
06/15-00:20:02.339740 [**] Queso fingerprint [**] 150.101.112.7506/15-
00:08:05.628216 [**] SMB Name Wildcard [**] 169.254.45.176:137 ->
MY.NET.197.207:137
:53869 -> MY.NET.111.197:4662

This type of corruption is, I would say, symptomatic of IDS overload. The logging systems are unable to keep pace with the alerting process of the IDS. It is also worth noting that some of the events in the log files are out of order.

I wrote a fix.pl script to repair the third type of corruption where the entire alert was still intact and correctly formatted, however there is no effective way of fixing the other types of corruption. I did not filter out these errors, instead choosing to leave the alerts in place to give me an overall picture of the environment. However, when analysing individual alerts, I did attempt to remove alerts that were corrupted from my analysis when considering IP addresses or alerts on an individual basis.

To analyse the three different types of files a used a number of different methods. The Alert files were concatenated together into one large file and analysed using the csv.pl and summarize.pl scripts written by Tod Beardsley for his GCIA practical (Beardsley, 2002) with a few minor modifications, these were also used in combination with grep for more information on specific alerts or IP addresses.

The scans file was analysed using the same method as the alerts file, this required some minor alterations to the summarize.pl script as the addresses in the scans file are not in the format MY.NET.x.x but instead have the actual IP address of

130.80.x.x.

The OOS files were a lot smaller than the others, most of the traffic was related to ECN. These were removed using grep leaving a much smaller log file that could easily be parsed manually to identify the types of OOS traffic.

A wide variety of information sources were used in this research, primary sources of information on port numbers and snort rule properties were www.incidents.org, www.snort.org and www.whitehats.com.

References/Bibliography

Alexander, Bryce. "SANS Intrusion Detection FAQ: Port 137 Scan." 10 May 2000. URL: http://www.sans.org/resources/idfaq/port_137.php (12 Jul. 2003).

Baldwin, Lawrence. "Mail Server slow to respond." 21 Mar. 2001. URL: http://www.mynetwatchman.com/KB/NetKb/Mailservers/mailatlident.htm (15 Jul. 2003)

Beardsley, Tod. "GIAC GCIA Practical (version 3.1)." 8 May 2002. URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc (10 Jul. 2003)

Berkers, John. "RE: [Snort-users] spp_http_decode rules." 3 Aug. 2001. URL: http://archives.neohapsis.com/archives/snort/2001-08/0075.html (15 Jul. 2003)

CAIDA. 8 Apr. 2003. "CAIDA Analysis of Code Red." URL: http://www.caida.org/analysis/security/code-red/#crii (14 Jul. 2003)

CERT/CC. "CERT Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service DLL." 17 Jan. 2002. URL: http://www.cert.org/advisories/CA-2001-13.html (15 Jul. 2003)

Cardoso, Fernando. "Security Incidents: Re: Port 6112." 20 Mar. 2000. URL: http://lists.insecure.org/lists/incidents/2000/Mar/0198.html (14 Jul. 2003)

Dittrich, D. Weaver, G. Dietrich, S. Long, N. "The mstream distributed denial of service attack tool." 1 May 2000. URL: http://staff.washington.edu/dittrich/misc/mstream.analysis.txt (14 Jul. 2003)

Fitzgerald, Nick. "Security Incidents: Re: SHELLCODE x86 NOOP." 4 Oct. 2001. URL: http://lists.insecure.org/lists/incidents/2001/Oct/0030.html (17 Jul. 2003).

IANA Network Working Group. "RFC3330: Special-Use IPv4 Addresses." Sep. 2002. URL: http://www.rfc-editor.org/rfc/rfc3330.txt (14 Jul. 2003)

IceHouse Designs. "Search Engine Spider Indentification." 19 Jul. 2002. URL:http://www.icehousedesigns.com/engines/spiderlist.php3 (15 Jul. 2003)

Jacobson, Van. McCanne, Steven. "LBNL Audio Conferencing Tool (vat)." URL: http://www-nrg.ee.lbl.gov/vat/ (17 Jul. 2003)

Kite, Doug. "GCIA Practical Assignment Version 3.3." Jul. 2002. URL: http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf (16 Jul. 2003)

Maslowski-Yerges, Al. "GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 3.3." 5 Jan 2003. URL: http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf (12 Jul. 2003).

Moore, D. Voelker, G. Savage, S. "Inferring Internet Denial-of-Service Activity." 2001. URL: http://www.caida.org/outreach/papers/2001/BackScatter/usenixsecurity01.pdf (20 Jul. 2003)

Project BanBots. "AltaVista: web crawlers, spiders and robots." 2003. URL: http://www.banbots.com/altavista.htm (17 Jul. 2003)

R00ters. "R00ters/Ejeet IRC Networks K-line/Akill Team." URL: http://intrepidengineering.com/r00ters/kline.html (16 Jul. 2003).

Ryan1918. "SDBot Kiddies Get A Dose Of My Boredom :)." 19 Apr 2003. URL: http://www.ryan1918.com/stuff/botnet.htm (15 Jul. 2003)
Snort.org. "The Snort FAQ." URL: http://www.snort.org/docs/FAQ.txt (20 Jul. 2003)

Symantec. "Symantec Security Resonse – Backdoor.Sdbot." 15 Jul. 2003. URL: http://securityresponse.symantec.com/avcenter/venc/data/backdoor.sdbot.html (18 Jul. 2003)

Thiele, Fred. "GCIA Practical v3.3." 2002. URL: http://www.giac.org/practical/GCIA/Fred_Thiele_GCIA.pdf (11 Jul. 2003)

TonikGin. "XDCC – An .EDU Admin's Nightmare." 11 Sep. 2002. URL: http://www.russonline.net/tonikgin/EduHacking.html (15 Jul. 2003).
Miller, Toby. "Global Incident Analysis Center: Special Notice – ECN and Intrusion Detection." 2000. URL: http://www.sans.org/y2k/ecn.htm (15 Jul. 2003).

WinMX. "Working around ISP port blocks." 2002. URL: http://winmx.2038.net/winmx/fr-blocked.html (21 Jul. 2003)

Appendices

Fix.pl Script

```
#!/usr/bin/perl
# Name: fix.pl
# Reads in a Snort alert log with newline characters in the wrong places
# and fixes it.
#This script is based on Tod Beardsley's, and my implementation is highly inefficient
#but it does the job.
#
#
# Usage: fix.pl infile [outfile]
{
unless ($ARGV[0]) {
  print "Need an input file!\n";
  die "No input";
}

unless ($ARGV[1]) {
  $outfile = "$ARGV[0].fixed";
} else {
  $outfile = "$ARGV[1]";
}
open(INFILE,"$ARGV[0]") || die "Can't open $ARGV[0] for reading!\n";
open(OUTFILE,">$outfile") || die "Can't open $ARGV[1] for writing!\n";
print "Transforming $ARGV[0] into $outfile.\n";
print "Just a moment.";
@badfile = <INFILE>;
close(INFILE);
#Insert a newline before each instance of 06/
for ($i = 1; $i <= $#badfile; ++$i)
{
                $badfile[$i] =~ s/06/\n06/g;
                if ($badfile[$i] !~ /^06/)
                {
                                chomp($badfile[$i-1]);#Remove newlines for each line
                                previous to the short ones
                }
                print OUTFILE "$badfile[$i-1]";
}
print OUTFILE "$badfile[$#badfile]";
}
```