# Global Information Assurance Certification Paper

# GIAC Intrusion Detection In Depth

GCIA Practical Assignment v3.3

By Marshall S. Heilman

5 Dec 2003

SANS Hammersmith – London 2003
June 23 - 28

# Table of Contents

**Assignment #1**

## Real-time Network Awareness (RNA): the Future or the Fad?


### Abstract

This paper will examine one of the emerging intrusion detection system (IDS) technologies, real-time network awareness (RNA). Although not strictly a new technology, Sourcefire has combined the capabilities of such tools as Nessus (vulnerability analysis), Snort (behavioral profiling) and p0f (passive operating system fingerprinting) to create a new comprehensive capability it has named RNA. RNA has been hailed as both the savior and the destroyer of IDS. One thing is for certain, RNA will definitely re-shape the IDS community and the business at large.


### IDS – The Background

IDSs suffer from a number of problems. Perhaps the largest problem of all is the misconception that once implemented, an IDS can be left alone to perform its duties without any maintenance. This type of thinking is along the same lines of an individual who configures a firewall once, puts it inline, and doesn't bother to configure it again. Of course the firewall will not provide all the functionality that it is supposed to, the individual did not configure it to do so! An IDS needs to be continually updated, configured, and monitored. The sheer amount of data that most IDSs produce is enough to turn away all except the true IDS analysts. It is safe to say that IDS is not for the faint of heart.

Another problem with IDS that RNA hopes to fix is the distinct advantage a savvy attacker has over an IDS. A savvy attacker has most likely performed some reconnaissance and knows something about the target host, and therefore knows more about the target host than the IDS, which is supposed to be protecting the host. In the words of Martin Roesch[3], "IDS operate in a contextual vacuum." Fundamentally, this means that the IDS makes guesses about the network it was put in place to defend because it has no concept of the network (see figure 1).

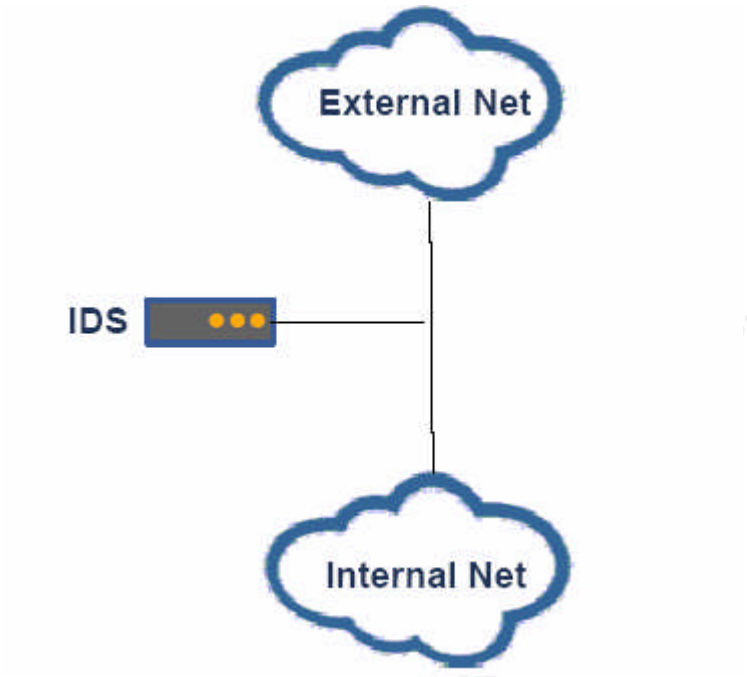

figure 1[2]

Another large problem with IDS technology is the amount of data an IDS produces and the requirement to sort through all the data and make sense of it. IDS will display alerts for anything that matches their signatures, be it normal traffic or malicious traffic; IDS have no way of differentiating between the two. The only way to determine whether or not traffic is malicious, or normal, is by administrative analysis of the traffic. Traffic analysis can be time consuming, especially when larger companies may receive hundreds of thousands of alerts a day. IDS are also not capable of preventing OS based false positives; a perceived Code Red attack on an Apache web server (either *nix or Win32 based) will trigger an alert even though an Apache web server will not be vulnerable to a Code Red attack.

Burak Dayioglu and Attila Ozgit attempted to address the immense amount of data produced by an IDS in their paper on the use of passive OS fingerprinting in conjunction with IDS[4]. They pointed out that the greatest reduction in false positives could come from the IDS possessing knowledge of the destination's OS. This could either be programmed manually or automatically. Obviously automatic is much more desirable. They wrote a Snort processor plug-in called osaffected that added passive OS fingerprinting capability to the rulesets. This caused a 6% increase in processor overhead[4]. The authors showed an average of around 11% reduction in false positives through the use of OS fingerprinting. RNA, a much more advanced version of this technology, starts to show its true potential when numbers like the previous statistic are shown. Marty Roesch even referred to Dayioglu and Ozgit's work in an email to the SecurityFocus IDS list[5].

**RNA**

What is RNA? According to its creator, Sourcefire, RNA gives IDS a compositional knowledge of the protected network by "providing continuously updated, persistent information about all the active components of a network"[3]. RNA greatly varies from the standard signature-based, anomaly detection, or statistical anomaly detection IDS engines in use today. As will be discussed in the following sections, RNA allows the IDS to provide better, smarter, alerts because it is now aware of the network it is defending, thus fixing the problem of the attacker knowing more about the target host, or network, than the IDS (see figure 2). RNA allows for new rogue services to be identified the moment they become active on the network, allowing an alert to be sent to administrators immediately. RNA provides real-time network awareness to the network administrators and to the IDS.

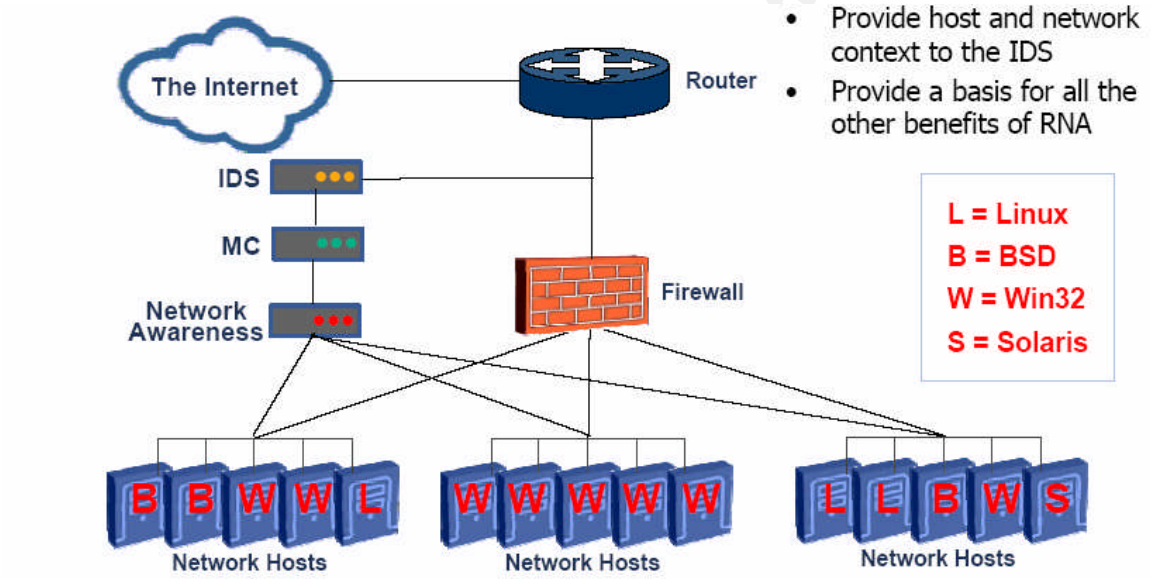

figure 2[2]

RNA works by combining passive network sniffing, behavioral profiling, and vulnerability analysis to generate a complete network picture. RNA does not consider a machine actually on the network unless it is transmitting or receiving traffic. By analyzing packets from network services and matching the header and payload data to known signatures, RNA is able to create a picture of the network service, to include specific patch and vulnerability information. A single RNA unit can handle an entire Class B network[3].

In an effort to make RNA as flexible as possible, RNA rulesets can be manually configured with the same type of ease that Snort rulesets offer. RNA also has built-in automatic configuration management, which can be configured to work hand-in-hand with Snort sensors and to trigger automatic responses based on new or updated network information. This allows for potential threats to be neutralized before becoming a real threat. Snort can be configured to populate its state information automatically from the network picture generated by RNA.

One major advantage is that it passively monitors a network in real-time. As a network service transmits or receives packets, RNA is passively monitoring those packets and updating its network map when appropriate. RNA will alert administrators to a policy rule violation when the network maps changes if configured to do so. RNA does not actually know about a rogue telnet service that has been started on one of its protected machines the moment the service is started. Instead, RNA becomes aware of that service as soon as that service becomes a presence on the network (i.e. it transmits or receives packets). One problem that RNA has no way of adequately addressing is whether the new network service is a rogue service or a legitimate one. As always, the network and security staffs should have a good working rapport.

Sourcefire is marketing RNA as a separate network component, one that complements the current IDS infrastructure, though they recommend buying it as part of their Intrusion Management System (IMS). This suite also includes their Management Console and Snort IDS sensors. An RNA unit could find a legitimate place on any network even without its IDS integration, though not the intended design[7].

**RNA - the Future**

RNA does not suffer from many of the problems that plague current active vulnerability scanners. Some of the problems include accidentally knocking over a machine or service, basing information off forced responses to abnormal packets (in OS fingerprinting), network saturation, invisible machines (i.e. some scanners require a successful ping before they will scan a machine; if the machine is blocking ICMP echo requests, the scanner will never see the machine), and the single point-in-time (snapshot) view of the network. RNA solves these problems by performing the following operations passively: fingerprinting, behavioral profiling, vulnerability analysis by inference, and by using a sniffing approach rather than an active scanning approach. By looking at certain information in the packets header and payload, RNA is able to identify the OS version, level, and patch status. To accurately assess a machine's network services, patch level, and vulnerabilities, the RNA unit needs to collect a number of packets from each of the services. Because this depends on a certain number of packets being transmitted or received by the network service, RNA has different levels of confidence in its network map. If RNA cannot gather enough information from the packets transmitted at one time, it must wait until more packets are seen before it can continue to update its map. This is where the confidence level of RNA is very important, and where administrators have to configure the rulesets accordingly. Obviously an administrator would not want to set up automatic response rules based off information that RNA was not 100% confident of. RNA will always error with false positives rather than false negatives. That means that if RNA is able to tell that a certain host is running IIS, but it cannot tell which version, it will flag all IIS attacks directed toward that host.

The point-in-time view of active vulnerability scanners is eliminated with RNA because RNA continually updates its network map and all related information. If a new vulnerability is discovered on a host after a vulnerability

scanner has scanned a machine, the report will never show the new information. RNA's report, which is always changing to reflect new information, will clearly show the new vulnerability.

Machines are not able to be invisible on the network from RNA because RNA does not attempt to actively seek them out by sending packets. Instead, as a host transmits or receives packets, RNA is able to capture and evaluate those packets, making it impossible for machines to hide from the view of RNA if they are actively engaged on the network. An incorrectly placed or misconfigured RNA unit could blind administrators or the IDS, so proper configuration and maintenance is crucial, perhaps even more so than on the IDS.

In addition to not suffering from any of the same problems that plague current active vulnerability scanners, RNA takes the technology one step further. Instead of simply providing vulnerability assessment reports to administrators, RNA also provides those reports to the IDS, which allows the IDS to tune its rulesets automatically to the current network map. For example, an IDS could be configured to automatically start alerting on incoming telnet requests if a new telnet service was discovered on the internal network. Otherwise, an administrator would have to see that a new telnet service had been started and create a new rule in the IDS to reflect the network change.

IDS no longer operate in a contextual vacuum when used in conjunction with RNA. This allows administrators to configure optimized rulesets to provide asset-specific protection and alerts. Instead of configuring a rule that applies to an entire network but only looks for a certain vulnerability in Microsoft machines, rules can now be created to only look at traffic directed to certain machines for that same Microsoft vulnerability. This also speeds up IDS performance as it does not have as many rules to search through for every packet it sees.

A big change from traditional IDS to IDS with RNA is that IDS would no longer be categorized as network-based IDS (NIDS), or even host-based IDS (HIDS) but would instead become target-based IDS[1]. Instead of a NIDS being configured to monitor all traffic coming into a network using generic rulesets and applying them to all hosts on a network, or a HIDS which is configured for each machine a sensor is placed on which results in many administrative correlating log information, the IDS with RNA can now be configured to specifically watch the targets of attacks using its knowledge of the network. This allows an IDS to only raise alerts that administrators really care about, helping to eliminate false positives. RNA can be configured to raise informational alerts about attacks directed at non-vulnerable machines (see figure 3). In an administrator's effort to not get bogged down underneath thousands and thousands of daily alerts, administrators attempt to configure rulesets as explicitly as possible, which allows for false negatives to occur. By using IDS with RNA, false negatives can be partially eliminated because the rulesets themselves are specific, the administrator does not have to narrow them down, which could lead to misconfiguration resulting in false negatives.

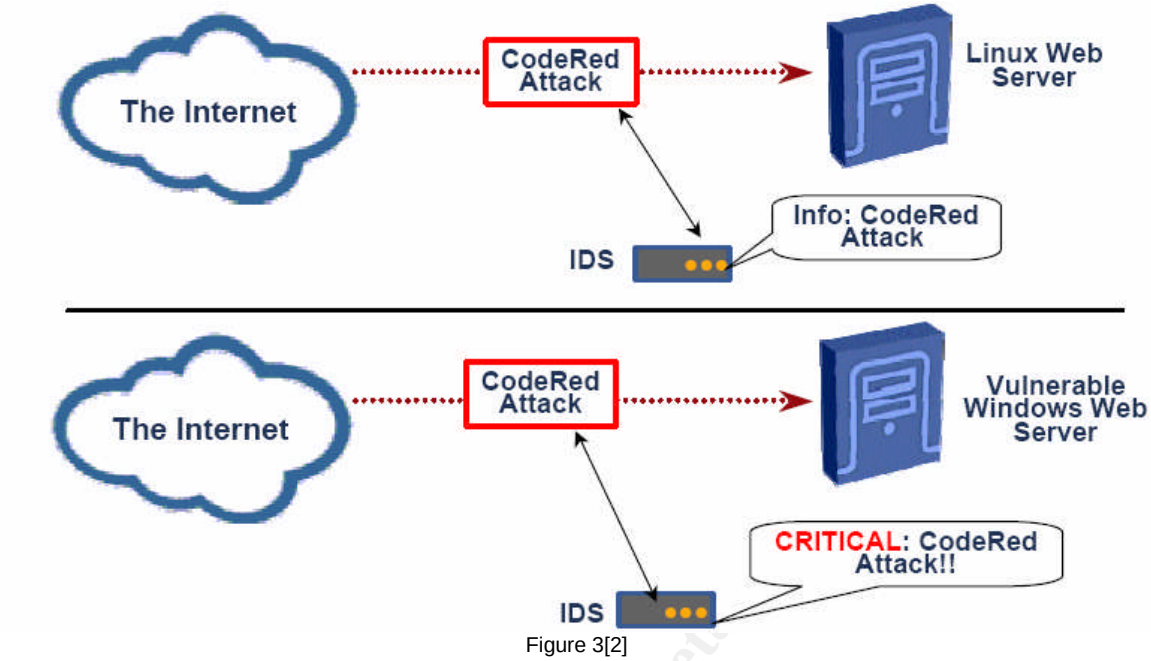

Figure 3[2]

RNA does not attempt to solve the problem of differentiating between normal and malicious traffic, but it does try to solve the problem of overwhelming amounts of data and IDS contextual vacuum. RNA "provides the contextual data required to disambiguate the environment in which the IDS operates, allowing the IDS to start protecting assets, not protocols or traffic"[3]. This is also where critics of the RNA technology make their case, as will be discussed later in this paper. An example where RNA would be very useful (other than the typical OS based vulnerability examples) would be the use of covert channels. If a malicious employee of the company the IDS was watching started a service to be used for covert channel communication, the IDS would most likely not catch the traffic. RNA would catch the traffic immediately, alerting security staff to the presence of an unauthorized new service.

During the most recent webinar, sponsored by SANS, in which Marty Roesch talked about RNA[2], a question was asked as to why host-based IDS are not employed on all machines with specific rules tuned to those systems. This seems like a more complete solution than a network sensor. In some ways, that is not a bad suggestion. As pointed out by Marty, that type of configuration would be incredibly expensive and would overwhelm the already overburdened security staff with alerts. The truth is that network sensors are easier to manage, provide less output, and are a more realistic solution. Another point discussed briefly by Marty is that if a machine with a host-based IDS gets hacked, the sensor cannot be trusted.

**RNA - the Fad**

RNA is most certainly not without its faults, as pointed out by its many critics. Perhaps the largest problem that critics see with RNA is that it takes traffic out of context - the whole picture is no longer seen, only the attacks the IDS is able to see. This was also an issue addressed by Dayioglu and Ozgit. They

stated “It is accepted that, total suppression of alerts going towards non-vulnerable hosts is not an acceptable strategy for intrusion detection and tracking at some installations”[4]. Todd Bennett also suggested some scenarios where RNA would not be appropriate[9], showing that RNA, like all security products, is not for every company. Dayioglu and Ozgit suggested that their processor plug-in could be modified slightly to downgrade the priority of the alerts (called dynamic de-prioritization) of attacks directed towards non-vulnerable machines. This would allow organizations to continue to have full context and make administration slightly easier. RNA addresses this issue in the exact same manner as suggested by Dayioglu and Ozgit. RNA allows for the alerts to be thrown out entirely or simply downgraded in priority.

As Security Management tools are the going trend, it makes sense that RNA would integrate with many of the more prevalent tools. RNA has the capability to provide its output to management tools such as HP OpenView, Tivoli, ArcSight, and in CSV format. RNA can also pull information from the above management tools. Active vulnerability scanners, such as ISS Scanner, Foundstone FS1000, QualsysGuard, and Nessus can populate RNA’s network map with their output.

According to Network World[7], linking scanning tools to IDS is a new idea, but one being pursued by many different companies. ISS plans to allow its scanning tool, Internet Scanner 7.0, to send information to its IDS, RealSecure 7.0[7]. SecurityProfiling, a patch management company and creator of SysUpdate, plans on correlating information with Snort IDS[7]. According to Wayne Jackson, CEO of Sourcefire, “If you have a completely unique idea, it’s probably not that good of an idea”[8]. Pete Lindstrom, a Spire Security LLC analyst said “Context is in. We’re still tackling the problem of false positives, and we’ve gotten to the point where there’s not much more you can do except bring in more context to help you make a better decision. This will be a big trend in the next year and a half”[8]. RNA attempts to do just that, bring more context to the IDS.

Assignment #2

*Detect #1 BAD-TRAFFIC bad frag bits*

**Source of Trace**

This detect was obtained from a file downloaded in binary form from www.incidents.org/logs/Raw. The file was 2002.10.6.

I believe that the IDS was between two Cisco routers. I believe this because the Ethernet headers were all either 00:00:0c:04:b2:33 or 00:03:e3:d9:26:c0, both of which are assigned as Cisco MAC addresses (the first three bytes of the MAC show this). This information can be gleaned from the IEEE (http://standards.ieee.org/regauth/oui/index.shtml). I further believe that the internal network was 207.166.0.0/16 because I was able to see packets going to or from a low network address of 207.166.5.x and a high network address of

207.166.243.x. The 00:00:0c:4:b2:33 MAC address is the external interface to the internal router and the 00:03:e3:d9:26:c0 MAC address is the internal interface to the external router.

**Detect was Generated by**
This detect was generated by Snort v2.0.2 using the most current ruleset as of 4 October, 2003. The setup includes Snort running on a Windows XP machine with MYSQL v4.0.13, Apache v2.0.47, and ACID v0.9.6b. Modified preprocessors include:

frag2
stream4: detect_scans, detect_state_problems
stream4_reassemble: both, ports all
rpc_decode: 111 32771 alert_fragments
conversation: allowed_ip_protocols all, timeout 60, max_conversations 3000
portscan2: scanners_max 256, targets_max 1024, target_limit 10, port_limit 20, timeout 60

The snort command used to generate the alerts was:
snort –c c:\snort\etc\snort.conf –k none –r c:\detects\detect#1 –vX
The –c <file> option specifies the snort config file and places snort in IDS mode, the –k none option directs snort to not perform checksum validation, the –r <file> directs snort to read from the input file, the –v option tells snort to be more verbose in its screen output, and the –X option tells snort to display the ASCII and HEX info for each packet.

The file detect#1 was a dump of all traffic coming from 80.5.184.140 or going to 207.166.211.223 ( windump –n –r c:\detects\2002.10.6 –w c:\detects\detect#1 host 80.5.184.140 or host 207.166.211.223 ). The –n option tells windump to not attempt name resolution, the –r <file> directs windump to read from the input file, the –w <file> option directs windump to dump the output in binary form to the input file, and the two host IP addresses tell windump to dump all packets that have either a source or destination host of either of those two IP addresses.

The alert generated, with packet dump, was:

[NOTE]: Packet shortened for brevity.

[**] [1:1322:5] BAD-TRAFFIC bad frag bits [**]
[Classification: Misc activity] [Priority: 3]
11/06-12:58:33.416507 80.5.184.140 -> 207.166.211.223
TCP TTL:112 TOS:0x0 ID:34053 IpLen:20 DgmLen:1468 DF MF
Frag Offset: 0x0000  Frag Size: 0x0014

12:58:33.416507 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 1482: IP (tos 0x0, ttl 112, len 1468) 80.5.184.140.4326 > 207.166.211.223.80: . [bad tcp cksum 24c3 (->bdb8)!]

```
1558397551:1558398979(1428) ack 1944854527 win 17520 (frag 34053:1448@0+)bad cksum
fe68 (->b41e)!
0x0000  4500 05bc 8505 6000 7006 fe68 5005 b88c     E.....`.p..hP...
0x0010  cfa6 d3df 10e6 0050 5ce3 426f 73ec 1fff     .......P\.Bos...
0x0020  5010 4470 24c3 0000 4745 5420 2f64 6566     P.Dp$...GET./def
0x0030  6175 6c74 2e69 6461 3f4e 4e4e 4e4e 4e4e     ault.ida?NNNNNNN
0x0040  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0050  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0060  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0070  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0080  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0090  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00a0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00b0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00c0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00d0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00e0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x00f0  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0100  4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e 4e4e     NNNNNNNNNNNNNNNN
0x0110  4e4e 4e4e 4e4e 4e4e 4e25 7539 3039 3025     NNNNNNNNN%u9090%
0x0120  7536 3835 3825 7563 6264 3325 7537 3830     u6858%ucbd3%u780
0x0130  3125 7539 3039 3025 7536 3835 3825 7563     1%u9090%u6858%uc
0x0140  6264 3325 7537 3830 3125 7539 3039 3025     bd3%u7801%u9090%
0x0150  7536 3835 3825 7563 6264 3325 7537 3830     u6858%ucbd3%u780
0x0160  3125 7539 3039 3025 7539 3039 3025 7538     1%u9090%u9090%u8
0x0170  3139 3025 7530 3063 3325 7530 3030 3325     190%u00c3%u0003%
0x0180  7538 6230 3025 7535 3331 6225 7535 3366     u8b00%u531b%u53f
0x0190  6625 7530 3037 3825 7530 3030 3025 7530     f%u0078%u0000%u0
0x01a0  303d 6120 2048 5454 502f 312e 300d 0a43     0=a..HTTP/1.0..C
0x01b0  6f6e 7465 6e74 2d74 7970 653a 2074 6578     ontent-type:.tex
0x01c0  742f 786d 6c0a 484f 5354 3a77 7777 2e77     t/xml.HOST:www.w
0x01d0  6f72 6d2e 636f 6d0a 2041 6363 6570 743a     orm.com..Accept:
0x01e0  202a 2f2a 0a43 6f6e 7465 6e74 2d6c 656e     .*/*.Content-len
0x01f0  6774 683a 2033 3536 3920 0d0a 0d0a 558b     gth:.3569.....U.
```

The snort signature used to generate the above alert was:

alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"BAD-TRAFFIC bad frag bits"; fragbits:MD; sid:1322; classtype:misc-activity; rev:5;)

This signature alerts when both the Don't Fragment (DF) and More Fragments (MF) IP options are set in a packet. The flags bits are the three high ordered bits of the 6$^{th}$ byte. Normally, the hex dump of the DF flag is 0x04 and the hex dump of the MF flag is 0x02. In this case, the hex dump showed 0x06, which means that both the DF and MF flags set.

The long string of "N's" is a dead giveaway that something else was not right with the packet, as well as the string "Host:www.worm.com." This packet was pretty easily identified as the CodeRed worm; the GET ./default.ida?NNNNNNN was a well known signature for the CodeRed worm [10], as is the "Host:www.worm.com". In this case, this is most likely the CodeRed.b worm because CodeRed.b was exactly identical to CodeRed.a except that it used a

random seed to generate the IP addresses it would attempt to infect whereas CodeRed.a used a static seed to generate potential target IP addresses [10] and CodeRed.c and beyond versions no longer have the “Host:www.worm.com” string, as well as using the letter “X” instead of the letter “N” as the buffer overflow character. Unlike CodeRed.b, CodeRed.c was not identical to the original versions of CodeRed in any way other than the exploit method used. One reason that the string “Host:www.worm.com” was removed from the CodeRed.c worm is because it does not attempt to deface the web site on the local server and it also does not attempt a denial of service attack on www.whitehouse.gov. Instead, it infects a system, installs a backdoor which allows code execution at the SYSTEM level privilege, propagates itself, becomes dormant for a single day, then reboots the machine [10]. Also unlike the previous two versions of CodeRed, which were resident in memory and were removed upon machine reboot, CodeRed.c is installed locally and is not removed by a reboot. Some versions also send the “GET” and then the rest of the get request in separate packets.

An interesting thing was that after disabling the bad-traffic.rules file, removing the flow line, and disabling both the stream4 and stream4_reassemble preprocessors, Snort still did not give a CodeRed v2 or other .ida alert. This is because Snort simply logs the packet which flagged a rule, in this case it was the first fragment that flagged the alert, the rest of the packets were not even processed. You can tell that the packet was fragmented, despite the DF flag being set, and that this was the first packet by looking at the header output which displays: (frag 34053:1448@0+). This shows that the fragID is 34053, the fragmented packet is 1448 bytes long, this is the first fragment (the 0+) and more fragments are expected. The fragID is used by the TCP/IP stacks to keep track of fragment streams (i.e. each fragment belonging to the same packet has the same fragID). The fragment offset is used to show where in the original packet this particular fragment fits in.

**Probability Source Address was Spoofed**

In this case, the packet was an HTTP packet, which means that the TCP three-way handshake would have already occurred. The CERT/CC describes the necessity of first completing the three-way TCP handshake before the worm is able to attempt to execute[10], therefore the probability that the source IP address was spoofed is very low. The only exception to this is that the possibility exists of an attacker guessing the sequence number of an established connection and injecting malicious packets appropriately, and the attack will still work. The possibility is still very low that the source IP address would have been spoofed.

There is also the possibility of an attacker owning a machine and using it to spread the worm. I think another good possibility is that the “attacking” host was simply an infected host. The worm tries to propagate itself from the 1$^{st}$ - 19$^{th}$ of

every month [10]. Since I used a dump file from the 6th, this falls within the range of time that infected hosts were attempting to propagating the worm.

**Description of the Attack**

This attack was a CodeRed attack using non-RFC 791 standard IP flags[12], DF and MF, which is generally associated with IDS evasion. The crafted packets could have been crafted using a packet crafting tools such as Hping[13] or packet Excalibur[14], or it they could have been routed through an automated IDS evasion tool, such as fragroute by Dug Song[15]. CodeRed attacks attempt to exploit a buffer overflow within the Indexing Service ISAPI filter (.ida), which does not properly perform bounds checking on input buffers[11]. If an attacker successfully runs this exploit against an IIS server, the attacker is given SYSTEM privileges, which means the attacker has complete access to the web server and can do anything he wants. The CodeRed worm also defaces the local web page, attempts to infect other hosts, and finally attempts a DoS against www.whitehouse.gov. The CVE number for the vulnerability CodeRed attempts to exploit is CVE-2001-0500.

**Attack Mechanism**

It is usually a safe guess that anytime you see a packet that is not conforming to the RFC's[16], then the packet has been crafted for some malicious purpose. That being said, it can never be ruled out that sometimes people are using a hacked version of a TCP/IP stack, or simply a poor implementation of one, and that the TCP/IP stack is merely sending out bad packets. Vijay Gullapalli states that he noticed the DF and MF flags set when packets were sent between two of his Linux boxes when the Path MTU was set[17].

Originally, I thought it most likely that the attacker had sent the CodeRed packet through fragroute in an attempt at IDS evasion, which would explain the DF and MF flags being set. After reading a post by Chris Russel[18] from the securityfocus incidents list, I believe this packet is simply a variant from the one he described in which the attacker added the MF flag to confuse an IDS that looked for the DF flag as part of the CodeRed signature. Chris had commented that even though the GET and the rest of the request were in two separate packets, the DF flag had been set. I couldn't find any analysis of the CodeRed worm that included both the DF and MF flags set, which means that this is something new, either getting routed through fragroute or intentionally crafted. I further believe that this was intentionally crafted, and not run through fragroute, because only the ACK flag is set. With a normal HTTP GET request, both the PSH and ACK flags will be set. In this case, I believe the attacker intentionally removed the PSH flag to make the packet appear as an innocent ACK reply to a previous packet from the web server, which firewalls not keeping state would allow through.

**Correlations**

According to www.ripe.net, the attacking IP, 80.5.184.140, is registered to an Internet Cable company in the Netherlands called NTL Birmingham. A search on www.dshield.org did not turn up anything either, indicating that this attacker has not been very busy lately, or at least has flown under everyone's radar.

1. A similar detect was done by Mike Wyman for his GCIA attempt. His analysis can be read at: http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00204.html.
2. A complete explanation, and disassembly, of the CodeRed worm was done by Eye's Marc Maiffret and Ryan Permeh and can be read at: http://cert.uni-stuttgart.de/archive/isn/2001/07/msg00055.html.
3. The CERT/CC CA-2001-19 can be read at: http://www.cert.org/advisories/CA-2001-19.html.
4. Symantec has an easily understand write up of the CodeRed worm at: http://securityresponse1.symantec.com/sarc/sarc.nsf/html/codered.worm.html .
5. Common Vulnerabilities and Exposures number – CVE-2001-0500. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0500.
6. Apache web server, for both *NIX and win32 platforms can be downloaded from: http://httpd.apache.org/download.cgi.
7. The Microsoft patch to fix this vulnerability is MS01-033.

**Evidence of Active Targeting**

This attack was targeted toward the web server that the IDS alerted on, though it was most likely part of a broader scan. According to the CERT/CC's analysis of the worm, "The Code Red worm attempts to connect to TCP port 80 on a randomly chosen host… "[10], the Code Red worm randomly chooses the IP addresses it will attack, making this attack most likely a randomly targeted server.

**Severity**

The following formula was used to calculate the severity of the attack (5 is the high end of the scale, 1 is the low end):

Severity = (Targets Criticality + Attacks Lethality) – (System Countermeasures + Network Countermeasures)

| | | |
|---|---|---|
| Target Criticality | 1 | I do not believe that this web server was very critical to its organization because, after searching through five days worth of binary dumps, I could not find a single other packet that was sent to this host. This leads me to believe that the web server was either not important, or perhaps set up as a test server and taken down. In addition, multiple web attacks were recorded going to other destination IP addresses. This leads me to believe that this host is probably not very important, so I am rating it a 1. |
| Attack Lethality | 5 | Giving the lethality a 5 was very easy in this case; if the buffer overflow initiated by the CodeRed worm were to succeed, the attacker would have SYSTEM privileges on the web server. |
| System Countermeasures | 4 | I decided to give the system countermeasures a rating of 4 because this was an exploit that had been out for a long time and a patch had been available for an even longer time. To protect against this attack, an administrator simply had to install a patch. As best practices dictate to run an antivirus scanner on all machines, an antivirus client was probably installed on the web server and would have easily detected the Code Red attack. Another reason this gets a 4 is because the destination host only received one packet all day, and it happened to be the Code Red packet. Other hosts received numerous port 80 packets, so I do not believe this host was even a web server at all. This would have been given a system countermeasure rating of 5 if I could verify that the appropriate patch was installed, that this was not an IIS web server, or that the host was not listening on port 80. |
| Network Countermeasures | 2 | Other than hardening the host against this vulnerability, there is not much that can be done in terms of network defense to guard against this worm. The best measure of protection would be to run an IDS, which this site did, as evidenced by the captured packet. In general, running a proxy firewall which proxies port 80 will help to reduce the number of web attacks. |

The final severity level, according to the above formula is 0 ( (1+5 )–(4+2)=0 ). This means that this attack does not pose a high priority threat to the targeted host based upon the information available from the binary dumps.

**Defensive Recommendations**

The best defense measures against this attack would be to not run an IIS web server, but to run something such as Apache, which also runs on a Win32 platform if that platform absolutely must be used. If there is no option other than to use IIS, the correct Microsoft patch, MS01-033, should be applied. In addition, all HTTP and HTTPS traffic should be filtered through a proxy firewall to ensure proper HTTP and HTTPS requests.

There are a number of guides available for securing Microsoft IIS 5.0. Microsoft has a Securing IIS 5.0 Resource Guide, which contains links to many resources and documents that help with securing IIS 5.0. Microsoft also has a secure IIS 5.0 checklist. Microsoft also has a few tools available, one of which is called IISLOCKDOWN v2.1, which turns off unnecessary features (administrator controlled), and another of which is called Urlscan 2.5, which blocks certain types of HTTP requests.

Other organizations have also released IIS 5.0 security guides such as Securityfocus, SANS includes a chapter on this topic in their Track 5 Securing Windows track, and O'Reilley has published a book called Securing Windows NT/2000 Servers for the Internet.

**Multiple Choice Question**
What is the problem with the following packet fragment?
12:33:34.416507 (tos 0x0, ttl 112, len 1468) 209.67.45.3.4531: . 1558397551:1558398979(1428) ack 1944854527 win 17520 (frag 23509:1448@64188+)

a. The length of the packet seems to change
b. The ttl value is too low to be correct
c. Fragment size exceeds 65535 bytes
d. More fragments are expected

Correct answer is C.

*DETECT #2 – WEB-IIS VIEW SOURCE VIA TRANSLATE HEADER*

**Source of Trace**
This detect was obtained from a file downloaded in binary form from www.incidents.org/logs/Raw. The file was 2002.5.13.

I believe the IDS was between two Cisco routers. I believe this because the Ethernet headers were all either 00:03:e3:d9:26:c0 or 00:00:0c:04:b2:33, both of which are assigned as Cisco MAC addresses (the first three bytes of the MAC show this). This information can be gleaned from the IEEE (http://standards.ieee.org/regauth/oui/index.shtml). I also believe that the 00:00:0c:04:b2:33 MAC address is the internal interface of the external router and that the 00:03:e3:d9:26:c0 MAC address is the external interface of the internal router. In addition, I believe the internal network was 46.5.0.0/16

because I was able to see packets going to a wide variety of 46.5.X addresses. I further believe that this site has a DMZ (46.5.180.X) with a very important web server (46.5.180.250). I believe this due to the high volume of HTTP traffic going to the 46.5.180.250 IP address, and best practices dictate having web servers on a DMZ.

**Detect was Generated by**
This detect was generated by Snort v2.0.2 using the most current ruleset as of 4 October, 2003. The setup includes Snort running on a Windows XP machine with MYSQL v4.0.13, Apache v2.0.47, and ACID v0.9.6b. Modified preprocessors include:

frag2
rpc_decode: 111 32771 alert_fragments
conversation: allowed_ip_protocols all, timeout 60, max_conversations 3000
portscan2: scanners_max 256, targets_max 1024, target_limit 10, port_limit 20, timeout 60

The snort command used to generate the alerts was:
snort –c c:\snort\etc\snort.conf –k none –r c:\detects\detect#2 –vX
The –c <file> option specifies the snort config file and places snort in IDS mode, the –k none option directs snort to not perform checksum validation, the –r <file> directs snort to read from the input file, the –v option tells snort to be more verbose in its screen output, and the –X option tells snort to display the ASCII and HEX info for each packet.

The file detect#2 was a dump of all traffic involving host 46.5.180.133 and either host 208.10.255.66 or host 218.145.63.95 ( windump –n –r c:\detects\2002.5.13 –w c:\detects\detect#2 ( host 208.10.255.66 or host 218.145.63.95 ) and host 46.5.180.133 ). The –n option tells windump to not attempt name resolution, the –r <file> directs windump to read from the input file, the –w <file> option directs windump to dump the output in binary form to the input file, the host IP portion tells windump to dump all packets with either of the first two addresses and the third IP address.

There were 19 WEB-IIS View Source Via Translate Header alerts generated, 7 from 208.10.255.66 and 12 from 218.145.63.95. One alert from each source address, along with a dump of the first packet from each source address, are:

[**] [1:1042:6] WEB-IIS view source via translate header [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
06/13-13:51:30.454488 208.10.255.66:3246 -> 46.5.180.133:80
TCP TTL:112 TOS:0x0 ID:4431 IpLen:20 DgmLen:221 DF
***AP*** Seq: 0x2F2530A6  Ack: 0x39EC8D81  Win: 0x2238  TcpLen: 20
[Xref => http://www.securityfocus.com/bid/1578][Xref =>
http://www.whitehats.com/info/IDS305]

13:51:30.454488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 235: IP (tos 0x0, ttl 112, id 4431, len 221) 208.10.255.66.3246 > 46.5.180.133.80: P [bad tcp cksum fe42 (->2e5d)!] 790966438:790966619(181) ack 971804033 win 8760 (DF)bad cksum 4cfa (->46f4)!

```
0x0000         4500 00dd 114f 4000 7006 4cfa d00a ff42      E....O@.p.L....B
0x0010         2e05 b485 0cae 0050 2f25 30a6 39ec 8d81      .......P/%0.9...
0x0020         5018 2238 fe42 0000 5052 4f50 4649 4e44      P."8.B..PROPFIND
0x0030         202f 6d61 696e 2048 5454 502f 312e 310d      ./main.HTTP/1.1.
0x0040         0a44 6570 7468 3a20 300d 0a74 7261 6e73      .Depth:.0..trans
0x0050         6c61 7465 3a20 660d 0a55 7365 722d 4167      late:.f..User-Ag
0x0060         656e 743a 204d 6963 726f 736f 6674 2d57      ent:.Microsoft-W
0x0070         6562 4441 562d 4d69 6e69 5265 6469 722f      ebDAV-MiniRedir/
0x0080         352e 312e 3236 3030 0d0a 486f 7374 3a20      5.1.2600..Host:.
0x0090         7777 772e 5858 5858 2e63 6f6d 0d0a 436f
        www.XXXX.com..Co
0x00a0         6e74 656e 742d 4c65 6e67 7468 3a20 300d      ntent-Length:.0.
0x00b0         0a43 6f6e 6e65 6374 696f 6e3a 204b 6565      .Connection:.Kee
0x00c0         702d 416c 6976 650d 0a50 7261 676d 613a      p-Alive..Pragma:
0x00d0         206e 6f2d 6361 6368 650d 0a0d 0a             .no-cache....
```

[**] [1:1042:6] WEB-IIS view source via translate header [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
06/14-01:50:32.754488 218.145.63.95:37155 -> 46.5.180.133:80
TCP TTL:49 TOS:0x0 ID:23200 IpLen:20 DgmLen:256
***AP*** Seq: 0x5C6A90E6  Ack: 0xD53C422E  Win: 0xFFFF  TcpLen: 32
TCP Options (3) => NOP NOP TS: 536228 7102634
[Xref => http://www.securityfocus.com/bid/1578][Xref => http://www.whitehats.com/info/IDS305]

01:50:32.754488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 270: IP (tos 0x0, ttl 49, id 23200, len 256) 218.145.63.95.37155 > 46.5.180.133.80: P [bad tcp cksum cd06 (->fd20)!] 1550487782:1550487986(204) ack 3577496110 win 65535 <nop,nop,timestamp 536228 7102634>bad cksum 37e3 (->31dd)!

```
0x0000         4500 0100 5aa0 0000 3106 37e3 da91 3f5f      E...Z...1.7...?_
0x0010         2e05 b485 9123 0050 5c6a 90e6 d53c 422e      .....#.P\j...<B.
0x0020         8018 ffff cd06 0000 0101 080a 0008 2ea4      ..............
0x0030         006c 60aa 5052 4f50 4649 4e44 202f 6d61      .l`.PROPFIND./ma
0x0040         696e 2048 5454 502f 312e 300d 0a44 6570      in.HTTP/1.0..Dep
0x0050         7468 3a20 300d 0a74 7261 6e73 6c61 7465      th:.0..translate
0x0060         3a20 660d 0a55 7365 722d 4167 656e 743a      :.f..User-Agent:
0x0070         204d 6963 726f 736f 6674 2d57 6562 4441      .Microsoft-WebDA
0x0080         562d 4d69 6e69 5265 6469 722f 352e 312e      V-MiniRedir/5.1.
0x0090         3236 3030 0d0a 486f 7374 3a20 7777 772e      2600..Host:.www.
0x00a0         5858 5858 2e63 6f6d 0d0a 436f 6e74 656e      XXXX.com..Conten
0x00b0         742d 4c65 6e67 7468 3a20 300d 0a50 7261      t-Length:.0..Pra
```

```
0x00c0      676d 613a 206e 6f2d 6361 6368 650d 0a43    gma:.no-cache..C
0x00d0      6163 6865 2d43 6f6e 7472 6f6c 3a20 6d61    ache-Control:.ma
0x00e0      782d 7374 616c 653d 300d 0a43 6f6e 6e65    x-stale=0..Conne
0x00f0      6374 696f 6e3a 2063 6c6f 7365 0d0a 0d0a    ction:.close....
```

In addition to the above alerts, 218.145.63.95 also generated these alerts, with packet dumps:

```
[**] [1:990:5] WEB-IIS _vti_inf access [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
06/14-01:49:56.664488 218.145.63.95:33471 -> 46.5.180.133:80
TCP TTL:49 TOS:0x0 ID:58374 IpLen:20 DgmLen:340
***AP*** Seq: 0xF9902FA6  Ack: 0xD34E339C  Win: 0xFFFF  TcpLen: 32
TCP Options (3) => NOP NOP TS: 536156 7099025

01:49:56.664488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 354: IP (tos 0x0, ttl 49, id
58374, len 340) 218.145.63.95.33471 > 46.5.180.133.80: P [bad tcp cksum e1e1
(->11fc)!] 4186976166:4186976454(288) ack 3545117596 win 65535
<nop,nop,timestamp 536156 7099025>bad cksum ae28 (->a822)!
0x0000      4500 0154 e406 0000 3106 ae28 da91 3f5f    E..T....1..(..?_
0x0010      2e05 b485 82bf 0050 f990 2fa6 d34e 339c    .......P../..N3.
0x0020      8018 ffff e1e1 0000 0101 080a 0008 2e5c    ..............\
0x0030      006c 5291 4745 5420 2f5f 7674 695f 696e    .lR.GET./_vti_in
0x0040      662e 6874 6d6c 2048 5454 502f 312e 300d    f.html.HTTP/1.0.
0x0050      0a44 6174 653a 2046 7269 2c20 3134 204a    .Date:.Fri,.14.J
0x0060      756e 2032 3030 3220 3030 3a35 303a 3430    un.2002.00:50:40
0x0070      2047 4d54 0d0a 4d49 4d45 2d56 6572 7369    .GMT..MIME-Versi
0x0080      6f6e 3a20 312e 300d 0a41 6363 6570 743a    on:.1.0..Accept:
0x0090      202a 2f2a 0d0a 5573 6572 2d41 6765 6e74    .*/*..User-Agent
0x00a0      3a20 4d6f 7a69 6c6c 612f 322e 3020 2863    :.Mozilla/2.0.(c
0x00b0      6f6d 7061 7469 626c 653b 204d 5320 4672    ompatible;.MS.Fr
0x00c0      6f6e 7450 6167 6520 342e 3029 0d0a 486f    ontPage.4.0)..Ho
0x00d0      7374 3a20 7777 772e 5858 5858 2e63 6f6d    st:.www.XXXX.com
0x00e0      0d0a 4163 6365 7074 3a20 6175 7468 2f73    ..Accept:.auth/s
0x00f0      6963 696c 790d 0a43 6f6e 7465 6e74 2d4c    icily..Content-L
0x0100      656e 6774 683a 2030 0d0a 4361 6368 652d    ength:.0..Cache-
0x0110      436f 6e74 726f 6c3a 206e 6f2d 6361 6368    Control:.no-cach
0x0120      650d 0a43 6163 6865 2d43 6f6e 7472 6f6c    e..Cache-Control
0x0130      3a20 6d61 782d 7374 616c 653d 300d 0a43    :.max-stale=0..C
0x0140      6f6e 6e65 6374 696f 6e3a 2063 6c6f 7365    onnection:.close
0x0150      0d0a 0d0a                                  ....

[**] [1:962:6] WEB-FRONTPAGE shtml.exe access [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
06/14-01:49:57.984488 218.145.63.95:33623 -> 46.5.180.133:80
TCP TTL:49 TOS:0x0 ID:62683 IpLen:20 DgmLen:467
```

***AP*** Seq: 0xFCCFFC07  Ack: 0xD38A07B9  Win: 0xFFFF  TcpLen: 32
TCP Options (3) => NOP NOP TS: 536158 7099158
[Xref => http://www.securityfocus.com/bid/1174][Xref =>
http://www.securityfocus.com/bid/1608][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0709][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2000-0413][Xref =>
http://cgi.nessus.org/plugins/dump.php3?id=10405]

01:49:57.984488 0:3:e3:d9:26:c0 0:0:c:4:b2:33 0800 481: IP (tos 0x0, ttl 49, id 62683, len 467) 218.145.63.95.33623 > 46.5.180.133.80: P [bad tcp cksum 1263 (->2c93)!] 4241488903:4241489318(415) ack 3549038521 win 65535 <nop,nop,timestamp 536158 7099158>bad cksum 9cd4 (->96ce)!

| | | |
|---|---|---|
| 0x0000 | 4500 01d3 f4db 0000 3106 9cd4 da91 3f5f | E.......1.....?_ |
| 0x0010 | 2e05 b485 8357 0050 fccf fc07 d38a 07b9 | .....W.P........ |
| 0x0020 | 8018 ffff 1263 0000 0101 080a 0008 2e5e | .....c.........^ |
| 0x0030 | 006c 5316 504f 5354 202f 5f76 7469 5f62 | .lS.POST./_vti_b |
| 0x0040 | 696e 2f73 6874 6d6c 2e65 7865 2f5f 7674 | in/shtml.exe/_vt |
| 0x0050 | 695f 7270 6320 4854 5450 2f31 2e30 0d0a | i_rpc.HTTP/1.0.. |
| 0x0060 | 4461 7465 3a20 4672 692c 2031 3420 4a75 | Date:.Fri,.14.Ju |
| 0x0070 | 6e20 3230 3032 2030 303a 3530 3a34 3120 | n.2002.00:50:41. |
| 0x0080 | 474d 540d 0a4d 494d 452d 5665 7273 696f | GMT..MIME-Versio |
| 0x0090 | 6e3a 2031 2e30 0d0a 5573 6572 2d41 6765 | n:.1.0..User-Age |
| 0x00a0 | 6e74 3a20 4d53 4672 6f6e 7450 6167 652f | nt:.MSFrontPage/ |
| 0x00b0 | 342e 300d 0a48 6f73 743a 2077 7777 2e58 | 4.0..Host:.www.X |
| 0x00c0 | 5858 582e 636f 6d0d 0a41 6363 6570 743a | XXX.com..Accept: |
| 0x00d0 | 2061 7574 682f 7369 6369 6c79 0d0a 436f | .auth/sicily..Co |
| 0x00e0 | 6e74 656e 742d 4c65 6e67 7468 3a20 3431 | ntent-Length:.41 |
| 0x00f0 | 0d0a 436f 6e74 656e 742d 5479 7065 3a20 | ..Content-Type:. |
| 0x0100 | 6170 706c 6963 6174 696f 6e2f 782d 7777 | application/x-ww |
| 0x0110 | 772d 666f 726d 2d75 726c 656e 636f 6465 | w-form-urlencode |

**Probability that the Source Address was Spoofed**
This type of attack relies on an established TCP connection, so the probability that the source IP address was spoofed is very low. In addition, this type of attack is used for information gathering, so the attacker will want to see the response. The exception to this would be if the host was vulnerable TCP sequence number predictions, which would mean that an attacker could potentially guess the sequence number of an established stream and send the malicious packets.

According to initial research done by Michal Zalewski[20] Windows 2000 and XP have weak ISN generators, which leads to improved sequence number guessing. As the referenced document above shows, Microsoft Windows 2000 SP2 and XP have a 12% chance of sequence number prediction. I was not able to find any information about this generator being improved in service packs 3 or 4. While this is not high, the chance of this occurring cannot be totally ruled out. This

having been said, the probability that the source IP address has been spoofed is still very low.

**Description of the Attack**
This attack exploits a vulnerability in the way IIS 5.0 handles WebDAV view source requests in the header of the request. This attack is native to Microsoft 2000, but is also applicable to IIS 4.0 if FrontPage Server Extensions 2000 has been installed. The vulnerability also exists in IIS 4.0 if the scriptable pages are stored on a shared directory. These are two separate vulnerabilities are vulnerable to the same attack, as explained by Russ Coope[24]. This attack has been assigned a Common Vulnerabilities and Exposures number of CVE-2000-0778.

When executed correctly, an attacker will be returned the source code of the requested script page. This is not an exploit that will result in the attacker being able to execute any type of commands on the web server, but it is useful for reconnaissance and account information grabbing. As Daniel points out, the most useful aspects of this exploit is to see SQL account names and information and database locations[23]. This information will give an attacker a great advantage when attempting to sabotage a business.

**Attack Mechanism**
(This entire section borrows heavily from SecurityFocus (Securityfocus translate: f) and Daniel[23])
The view source via translate header attack works because of a flaw in the way IIS 5.0 (and 4.0 in some circumstances) reacts to a crafted WebDAV header request for scriptable pages. Adding “Translate: f” to an HTTP GET request is a valid header for WebDAV to allow WebDAV components to bypass server-side processing and simply return the source code of the script page. This is a useful feature for a web administrator trying to edit the page or perform basic troubleshooting of a misbehaving script page. Unfortunately, due to a coding error, simply adding a “/” at the end of the request will return the scripting pages source code to anyone requesting it.

IIS has a dedicated scripting engine for handling and processing script pages. These pages are executed on the server and then the appropriate results are returned to the browser. When the malicious request is sent to the web server, IIS correctly locates the requested file but does not correctly process the file by sending it to a scripting engine. Instead, IIS returns the page to the browser because it does not recognize it as needing to be processed by the scripting engine.

The keywords WebDAV and PROPFIND in both attackers packets make them look like valid WebDAV requests. PROPFIND is used to request certain property information[21]. If one looks at the above packets and compares them to the

packet dump provided by Whitehats.com[22] (show below), it is easy to see the difference.

```
IIS-view source via translate\: F header
08/18-12:53:14.293999 attacker:15253 -> target:80
TCP TTL:53 TOS:0x0 ID:31083  DF
*****PA* Seq: 0x90F90192   Ack: 0x251D6AF2   Win: 0x2238
TCP Options => NOP NOP TS: 3060457 0
47 45 54 20 2F 70 61 67 65 73 2F 63 6F 6E 74 61  GET /pages/conta
63 74 2E 61 73 70 5C 20 48 54 54 50 2F 31 2E 30  ct.asp\ HTTP/1.0
0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A 20 6C 69  ..User-Agent: li
62 77 77 77 2D 70 65 72 6C 2F 35 2E 34 35 0D 0A  bwww-perl/5.45..
43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65  Content-Type: te
78 74 2F 68 74 6D 6C 0D 0A 54 72 61 6E 73 6C 61  xt/html..Transla
74 65 3A 20 66 0D 0A 58 2D 46 6F 72 77 61 72 64  te: f..X-Forward
65 64 2D 46 6F 72 3A 20 32 30 39 2E 31 38 37 2E  ed-For: 103.164.
31 34 30 2E 31 39 32 0D 0A 48 6F 73 74 3A 20 77  002.134..Host: w
77 77 2E 65 78 74 72 65 6D 65 6C 6F 67 69 63 2E  ww.aaaaaaaaaaa.
63 6F 6D 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E  com..Content-Len
67 74 68 3A 20 31 38 0D 0A 56 69 61 3A 20 31 2E  gth: 18..Via: 1.
31 20 6E 65 74 63 61 63 68 65 30 31 2E 67 77 2E  1 netcache01.aa.
74 6F 74 61 6C 2D 77 65 62 2E 6E 65 74 20 28 4E  aaaaaaaaa.net (N
65 74 43 61 63 68 65 20 34 2E 30 52 34 44 31 31  etCache 4.0R4D11
29 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B  )..Connection: K
65 65 70 2D 41 6C 69 76 65 0D 0A 0D 0A           eep-Alive....
```

As will become evident in a moment, I believe that a script was used and it modified the known exploit signature in an effort to be stealthy. The evidence is presented below.

The first attacker, 208.10.255.66, sent 7 packets to 46.5.180.133. The packets had a distinct script feel to them for the following reasons:
1. The packets arrived in groups of twos (except for the last packet). Each packet in a “group” were ~.5 seconds apart, with each group 3 seconds apart. The 7th packet was an oddball and arrived ~.5 seconds after the 6th packet to make a group of three.
2. Each group of two packets had the same source port, then the next group would arrive with a source port 2 numbers higher than the previous group (i.e. 3246, then 3248, etc…) Once again, the 7th packet was an oddball as it had the same source port as the third group (3250).
3. The first packet was 221 bytes, the second was 222. The difference was in the request, the first request was: ./main.HTTP/1.1 and the second was ./main/.HTTP/1.1. Once again, the third group was the oddball in that the 5th packet was simply ./.HTTP/1.1 and the 6th and 7th packets followed the above header protocol, except that they were only 203 and 204 bytes long respectively.

Here is a dump of the above packets without the ASCII/HEX output:

[NOTE: I have removed those fields from the output that I did touch upon above for ease of understanding and readability, to include removing the 13 from in front of the timestamp]

51:3**0.454**488 ( ttl 112, id **4431**, len **221**) 208.10.255.66.**3246** > 46.5.180.133.80
51:3**1.074**488 ( ttl 112, id **4436**, len **222**) 208.10.255.66.**3246** > 46.5.180.133.80:
51:3**3.294**488 ( ttl 112, id **4453**, len **221**) 208.10.255.66.**3248** > 46.5.180.133.80:
51:3**3.854**488 ( ttl 112, id **4458**, len **222**) 208.10.255.66.**3248** > 46.5.180.133.80:
51:3**6.004**488 ( ttl 112, id **4475**, len **188**) 208.10.255.66.**3250** > 46.5.180.133.80:
51:3**6.724**488 ( ttl 112, id **4480**, len **203**) 208.10.255.66.**3250** > 46.5.180.133.80:
51:3**7.264**488 ( ttl 112, id **4485**, len **204**) 208.10.255.66.**3250** > 46.5.180.133.80:

The second attacker, 218.145.63.95 was a bit craftier and did not exhibit any pattern matching behavior. The second attacker did display suspicious behavior by flagging three different Microsoft IIS alerts, WEB-IIS _vti_inf access, WEB-FRONTPAGE shtml.exe access, and WEB-IIS view source via translate header. All three of these alerts flag on various types of Microsoft IIS reconnaissance vulnerabilities. Each alert has a link to a brief explanation of what type of reconnaissance they allow an attacker to perform. This type of traffic definitely does not look right.

The “Evidence of Targeting” section provides more insight as to why this is certainly not two web administrators confused about which web server they were supposed to be working on.

**Correlations**
According to www.arin.net, the attacking IP, 208.10.255.66, is registered to Sprint Corporation. A search on www.dshield.org did not turn up anything, indicating that this attacker has not been very busy lately, or at least has flown under everyone’s radar. The second attacker, 218.145.63.95, is registered to Central Data Communications Office in Seoul Korea, according to the Asia Pacific NIC. This attacker also does not show in the Dshield database.


1. Common Vulnerabilities and Exposures number CVE-2000-0778 - http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0778
2. “Microsoft IIS 5.0 “Translate: f” Source Disclosure Vulnerability.” 14 Aug 2000. Securityfocus. 14 Oct 2003. <http://www.securityfocus.com/bid/1578>.
3. Snort WEB-IIS view source via translate header explanation - http://www.snort.org/snort-db/sid.html?sid=1042
4. Snort WEB-IIS _vti_inf access explanation - http://www.snort.org/snort-db/sid.html?sid=990
5. “Microsoft FrontPage Server Extensions Path Disclosure Vulnerability.” 6 May 2000. Securityfocus. 15 Oct 2003. <http://www.securityfocus.com/bid/1174/info/>.

**Evidence of Active Targeting**
Both of these attackers either were using automated tools or were script kiddies. A simple look at 46.5.180.133's response to other HTTP requests revealed that it was a Red Hat Linux machine running Apache version 1.3.12. The command used to find this out was: windump –n –vvv –XS –r 2002.5.13 –c 1 src port 80 and src host 46.5.180.133. The options used that were not already explained in the "Detect was Generated by" section were –vvv which tells windump to be very, very, verbose, -X which prints out ASCII and HEX information, -S which prints out absolute TCP sequence numbers, -c 1 which exits the program after one packet has been dumped, and the source port and source host this packet should come from.

I believe the first attacker was simply a script kiddy playing around with an exploit script. This attacker did not have any activity from 2002.5.11 – 2002.5.15, except for his 7 packets on the 13th. I believe the second attacker was an inexperienced attacker playing around with an automated script. Besides trying three different types of attacks on the 13th, he also sent eight packets to the same destination host, 46.5.180.133, on the 15th. Four of those packets flagged the _vti_inf access alert and four of them flagged the shtml.exe alert. These attacks were sent in an alternating sequence.

**Severity**
The following formula was used to calculate the severity of the attack (5 is the high end of the scale, 1 is the low end):

Severity = (Targets Criticality + Attacks Lethality) – (System Countermeasures + Network Countermeasures)

| | | |
|---|---|---|
| Target Criticality | 4 | I decided to give this target a criticality rating of 4 because web servers that have script pages often interact with databases which oftentimes contain sensitive personal or business information. This is exactly the type of information that an attacker would be eager to obtain. This targeted host received a decent bit of web activity on the other four days that I checked, so it is definitely an active host. |

| | | |
|---|---|---|
| Attack Lethality | 1 | This attack does not rate a very high lethality value because it does not perform any type of system damage, simply information leakage. Information leakage can be devastating to a company, but does not cause any type of system compromise, which would allow for the application of the gained information. This attack needs to be followed by another, more lethal attack, before a system is in trouble. |
| System Countermeasures | 5 | Since the targeted host is not running the vulnerable OS, there is no chance the attack could succeed. |
| Network Countermeasures | 2 | This is not the type of attack that can be blocked through network defenses, other than only allowing trusted IP addresses through the firewall. This site obviously had an IDS in place, which is a good countermeasure. Presence of a firewall is unknown. |

The final severity level, according to the above formula is -2 ( (4 +1)–(5+2)=-2 ). This means that this attack, against this host on this network, does not pose a real threat.

**Defensive Recommendations**
The best defense measure to take against this vulnerability is to install correct Microsoft patch. IIS 5.0 administrative should install MS00-058, or Microsoft Windows Service Pack 1 or higher. Best practices dictate the installation of Service Pack 4 as it is the most current service pack. IIS 4.0 administrators should install MS00-019 if they are using virtual directories. Other best practices that administrators should follow is to set proper permissions on all script pages and to not include sensitive information in script pages.

**Multiple Choice Question**
When a program retries a connection, the IPID:
a. remains the same
b. changes randomly
c. increments by 1
d. displays a + character indicating that this is not the first connection attempt

Correct answer is A.

*DETECT #3 – SNMP public access udp*

**Source of Trace**

This detect was obtained from a file downloaded in binary form from https://ids-europe.alchemistowl.org/. The file was 2003-08-19T15:32:35-snort.data.gz.

DSL Router --Switch Vlan1---IDS
Switch Vlan2---"Internal" machines (.98, .99, .100, .102, .103, .110)
Switch Vlan3---Router---DMZ Switch---DMZ machines(.105, .106)

I believe the network is something like the above diagram because all packets coming from external addresses had the same MAC address (which I believe is the external router), all packets going to or from .105 or .106 had the same MAC address (which I believe is the DMZ router), and all packets going to or from any of the other machines had a unique MAC address (which is why I believe those machines hang directly off the switch). I believe the IDS is part of VLAN1 and is monitoring the link to the router. I believe .105 offers SMTP and HTTP, and .106 offers SSH. I think this because the only traffic that goes to either of the DMZ servers other than the three listed above were 161, 1434 and 1433, which are the ports for SNMP, Slammer and SQL Snake worms respectively. The ports 1434 and 1433 seemed to be a large scan, as every IP address got hit repeatedly, not just the DMZ, so I do not believe those servers were actively targeted, meaning they may or may not be running Microsoft SQL server. The other machines which I believe are part of the internal network seemed to only receive HTTP responses and numerous ICMP echo requests other than the Slammer and SQL Snake attempts.

I believe the external router is a Siemens DSL router because the first three bytes of the MAC are 00:20:6f, which is registered to FlowPoint Corporation (according to the IEEE http://standards.ieee.org/regauth/oui/index.shtml), which lists its website as www.flowpoint.com, which is a Siemens DSL page. I believe the DMZ router is a *nix machine because the first three bytes of the MAC addresses, 02:26:02 were not recognized by a search on either the IEEE website, or Google, and *nix machines can easily change their MAC addresses.

**Detect was Generated by**
This detect was generated by Snort v2.0.2 using the most current ruleset as of 18 October, 2003. The setup includes Snort running on a Red Hat 9.0 machine with MYSQL v3.23.56, Apache v2.047, and ACID v0.9.6b. Modified preprocessors include:

frag2
rpc_decode: 111 32771 alert_fragments
conversation: allowed_ip_protocols all, timeout 60, max_conversations 3000
portscan2: scanners_max 256, targets_max 1024, target_limit 10, port_limit 20, timeout 60

The snort command used to generate the alerts was:

snort -c /etc/snort/snort.conf –k none –r /detects/detect#3
The –c <file> option specifies the snort config file and places snort in IDS mode, the –k none option directs snort to not perform checksum validation, and the –r <file> directs snort to read from the input file.

The file detect#3 was a dump of all traffic involving hosts 129.21.221.212 and 62.3.214.234 ( tcpdump –nn –r /detects/2003-08-19T15_32_35Z-snort.data –w /detects/detect#3 host 129.21.221.212 and host 62.3.214.234 and port 161). The –nn option tells tcpdump to not attempt name resolution for IP addresses or port numbers, the –r <file> directs tcpdump to read from the input file, the –w <file> option directs tcpdump to dump the output in binary form to the input file, the host IP portion tells tcpdump to dump all packets with either of the two addresses and a port of 161 (SNMP).

There were a total of 32 SNMP public access udp alerts, 24 were from 62.3.214.234 and 8 were from 129.21.221.212. One alert from each source address, along with a dump of the offending first packet from each source address are:

[**] [1:1411:3] SNMP public access udp [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/25-14:37:37.440233 62.3.214.234:2278 -> 195.82.120.98:161
UDP TTL:117 TOS:0x0 ID:11434 IpLen:20 DgmLen:69
Len: 41
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0013][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0012][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0517]

14:37:37.440233 62.3.214.234.2278 > 195.82.120.98.161:  [udp sum ok]
|30|27|02|01{ SNMPv1 |04|06|a0|1a{ GetRequest(26) |02|02R=1180
|02|01|02|01|30|0e |30|0c|06|08.1.3.6.1.2.1.1.2.0|05|00} }  (ttl 117, id 11434, len 69)
0x0000  4500 0045 2caa 0000 7511 c85b 3e03 d6ea  E..E,...u..[>...
0x0010  c352 7862 08e6 00a1 0031 40b6 3027 0201  .Rxb.....1@.0'..
0x0020  0004 0670 7562 6c69 63a0 1a02 0204 9c02  ...public.......
0x0030  0100 0201 0030 0e30 0c06 082b 0601 0201  .....0.0...+....
0x0040  0102 0005 0011 e0aa 04                   .........

[**] [1:1411:3] SNMP public access udp [**]
[Classification: Attempted Information Leak] [Priority: 2]
09/27-21:42:06.694870 129.21.221.212:32770 -> 195.82.120.98:161
UDP TTL:240 TOS:0x0 ID:59869 IpLen:20 DgmLen:66 DF
Len: 38
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0013][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0012][Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0517]

21:42:06.694870 129.21.221.212.32770 > 195.82.120.98.161:  [no cksum] |30|24|02|01{ SNMPv1 |04|06|a1|17{ GetNextRequest(23) |02|01R=1 |02|01|02|01|30|0c |30|0a|06|06.1.3.6.1.2.1.1|05|00} }  (DF) (ttl 240, id 59869, len 66)

```
0x0000          4500 0042 e9dd 4000 f011 062e 8115 ddd4    E..B..@.........
0x0010          c352 7862 8002 00a1 002e 0000 3024 0201    .Rxb........0$..
0x0020          0004 0670 7562 6c69 63a1 1702 0101 0201    ...public.......
0x0030          0002 0100 300c 300a 0606 2b06 0102 0101    ....0.0...+.....
0x0040          0500 46bf aae8                             ..F...
```

The first attacker also sent 48 packets which alerted on the SNMP private access udp rule and 8 packets which alerted on the SCAN SolarWinds IP attempt rule. No other packets were received by or transmitted to either of the attackers.

**Probability that the Source Address was Spoofed**

According to the CERT/CC [26], by exploiting a vulnerable SNMP implementation, an attacker can gain unauthorized privileges, perform a denial of service (DoS) attack, or cause erratic behavior. If the attacker was looking to gain unauthorized privileges, then the IP address is not going to be spoofed because the attacker is going to want to be able to use the machine. Even if the attacker wanted to perform a DoS, or cause erratic behavior, the IP address would be hard to spoof because of the TCP three-way handshake which needs to be completed prior to SNMP communication.

In this case, there was no attempt at a buffer overflow, no attempt at a DoS, and no sign of crafted packets which might cause erratic behavior. This has the feel of a reconnaissance attempt. If that is the case, then the attacker is going to want to see the information. Either the IP address is not spoofed, the attacker “owns” other machines and is using them to hide his IP address, or the attacker is spoofing the IP address and is monitoring responses to the spoofed IP address. I would say that the IP address is most likely not spoofed, but is most likely not the actual malicious users IP address.

**Description of the Attack**

A malicious user scans UDP port 161 requesting certain information (System and SysObjectID in this case) using the default read-only community string, public. The attacker hopes to find out if SNMP is running, and if it is, he hopes that the default read-only community string was used and that he will get responses to his requests. The responses will provide the attacker with useful information, which will help toward further attacks. This is a reconnaissance attack used to gather information about potential targets.

**Attack Mechanism**

The first attacker, 62.3.214.234 scanned for both the read-only public default community string and the read-write private default community string. The

attacker sent packets to the network in the following order (the addresses shown are the only active addresses at this site, they were the only addresses to receive packets within a five day span): .98, .99, .100, .102, .103, .105, .106, .110, .98, .99, .100, .98, .99, .102, .103, .105, .106, .110, .100, .102, .103, .105, .106, and .110. It looks like the attacker attempted to scan the network three times, and some of the packets arrived out of order. I believe the attacker used a script because all IP addresses were scanned within a four second time frame, all from the same source port, all packets were identical, except for IPID, and because the entire network was scanned three times. Each packet was 69 bytes. An attacker physically sitting at his computer attempting to gather this information would not have sent three requests to each IP address in a four second timeframe.

The second attacker, 129.21.221.212, scanned the site for the read-only public default community string over a period of one minute and thirty seven seconds. Each request was sent approximately eight seconds apart. When an IP was skipped (i.e. it didn't exist), the next packet was received eight seconds later (as long as only one IP address was skipped). This also feels like a script. Each packet was also identical, except for the IPID, and were 66 bytes in length, which is a variation from the length of the 1st attacker's packets. The variation is because this attacker sent a GetNextRequest for the System object whereas the first attacker sent a GetRequest for the SysObjectID object. According to RFC 1157[27], the GetRequest is issued to request a specific attribute, such as IP address, host name, etc., and GetNextRequest requests the next value in a list of attributes, such as the next IP address on an interface with multiple IP addresses assigned. The source port also stayed the same for each packet from this attacker.

I do not believe this is legitimate SNMP traffic, and is therefore a scan, because it seems unlikely that a college in the US and an English company (see Correlations) would both require SNMP information from hosts on someone's personal DSL connection.

**Correlations**

The first attackers IP address falls in the range of IP addresses assigned to Nildram-Morrisn, a company located in Great Britain, according to Reseaux IP Europeens (RIPE). The second attackers IP address falls in the range of IP addresses assigned to the Rochester Institute of Technology (RIT), according to the American Registry for Internet Numbers (ARIN). A quick search on Dshield turned up nothing for either IP address. Either they have not been very busy, or have flown under everyone's radar.

1. Counterpane Security Alert, "SNMP Vulnerability Update." - http://www.counterpane.com/alert-snmp2.html

2. The Microsoft specific vulnerability in its implementation of SNMP was assigned CAN-2002-0053 by Mitre. Mitre also has two other CVE candidates for SNMP, CAN-2002-0012 and CAN-2002-0013.
3. An analysis of the SNMPNetStat vulnerability can be found at: http://www.securityfocus.com/bid/3780/info/
4. The Oulu University PROTOS publication for snmpv1 - http://www.ee.oulu.fi/research/ouspg/protos/testing/c06/snmpv1/index.html

**Evidence of Active Targeting**

I believe the first attacker specifically targeted the network because of the repeated scanning in a short timeframe of the IP range. If this were part of a larger scan, I would most likely have seen a delay in the time of the start of the second and third scans. I believe the scan coming from the second attacker may have been part of a larger scale scan because the network was not scanned again and because of the slow and steady rate that the packets were received.

**Severity**

The following formula was used to calculate the severity of the attack (5 is the high end of the scale, 1 is the low end):

Severity = (Targets Criticality + Attacks Lethality) – (System Countermeasures + Network Countermeasures)

| | | |
|---|---|---|
| Target Criticality | 5 | I am going to give the target criticality a rating of 5 because the entire network was targeted. |
| Attack Lethality | 2 | I am going to give the attack lethality a rating of 2 because it was only a scan and not an attack. I do believe it deserves more than a 1 rating because if the scan is successful, the attacker will receive valuable information about the OS of the target, which can help him in a future, more devastating attack. |
| System Countermeasures | 2 | I am going to give the system countermeasures a rating of 2 because SNMP is widely used and I have no way of knowing whether the targeted hosts are using SNMP or not, so I will assume that most of them are. Also, if SNMP is used, there is a good chance the default community strings would have been used, allowing this attack to succeed. |

| | | |
|---|---|---|
| Network Countermeasures | 2 | SNMP is very easy to block at a firewall, just blocks incoming ports 161 and 162 udp/tcp. In this case the network seemed to be a home network, so a firewall may not be installed. On the other hand, if the user has the presence of mind to set up an IDS, a firewall is most likely inline somewhere. I am going to give the network countermeasures a rating of 2 because SNMP is most likely being blocked if a firewall is in place, but I have no proof of a firewall. |

The final severity level, according to the above formula is 3 ( (5 +2)–(2+2)=3 ). This means that this attack, against this network, poses a potentially high threat.

**Defensive Recommendations**

To protect against this type of scan, and the multiple vulnerabilities associated with SNMP, there are a number of choices available to the network administrator. The easiest way is to not use SNMP, though the CERT/CC claims that some of vendors affected by the SNMP vulnerabilities[26] will still have problems even after SNMP has been disabled[26]. Ingress and egress filtering should also be applied at the firewall level to prevent unauthorized SNMP access from an intruder as well as preventing a compromised host from being used as a launch point for further infections outside your network. Ports 161 and 162 UDP and TCP should be blocked, or limited to trusted networks. The simplest security measure that can be taken, and best practices dictate that this step should be taken, is to change the default community string names. To protect machines from the vulnerabilities listed in the CERT/CC's report, appropriate vendor patches should be applied (available at the end of the CERT/CC's report). The Microsoft patch that fixes the vulnerability described in CAN-2002-0053 can be downloaded from: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-006.asp. Appendix A of the Oulu Universities PROTOS Test-Suite write up contains five links for securing SNMP in a variety of environment, this would be an ideal place to start in trying to secure SNMP.

**Multiple Choice Question**

SNMP traps use which of the following port(s)?
a. udp/tcp 161 and 162
b. udp 22
c. udp/tcp 161
d. udp/tcp 162

Correct answer D.

Assignment #3

## Executive Summary

One of the most difficult tasks when preparing an analysis of any type is how to reach out to the readers of the analysis and ensure that they are able to easily interpret the analysis, regardless of their status within the company. This means that high-level management should be able to understand the analysis right along with the mid-level managers and the technicians. In addition, there has to be enough content to give the technicians enough information to work with. This analysis will attempt to bridge the gaps just mentioned.

When a third-party organization performs almost any type of analysis, they want to prove that the analysis was worth paying for and that [hopefully] the hiring company will hire them again in the future. The third-party organization will present the most shocking, or critical, data first, leaving the mediocre data to the end. This does not mean that the analysis is primarily bad (even though it might seem that way from the beginning of the analysis), but only that the third-party organization wishes to justify the job they performed though shock value.

This analysis will focus on the form mentioned in the above paragraph, presenting the most critical data first, leaving the rest to fall into place at the end of the report. The overall assessment and defensive recommendation for this network will be placed at the end, with the machines most likely compromised near the beginning.

After removing all corrupted lines [614] and spp_portscan entries (covered in the portscan logs), there were a total of 375,398 alerts, 11,400,868 scans and 18345 Out of Spec (OOS) log entries.

## Files Analyzed

The list of files used, obtained from www.incidents.org/logs, are shown in figure 4 (after having been uncompressed).

| *Alert Files* | *Scan Files* | *OOS Files* |
|---|---|---|
| alert.031020 | scans.031020 | OOS_Report_2003_10_20 |
| alert.031021 | scans.031021 | OOS_Report_2003_10_21 |
| alert.031022 | scans.031022 | OOS_Report_2003_10_22 |
| alert.031023 | scans.031023 | OOS_Report_2003_10_23 |
| alert.031027 | scans.031027 | OOS_Report_2003_10_24 |

figure 4

Notice that logs from the 24th through the 26th were missing from the alert and scan logs, so the 27th was substituted. Also notice that the 24th was not missing from the OOS logs, so it was used.

## Immediate Action

In this section of the analysis, the items most relevant to the universities network security will be reviewed. Items that will be included are a list of possibly

compromised machines and a list of the internal top-talkers, which includes alerts, scans, and OOS packets.

Possibly Compromised Machines

The list presented in figure 5 was created from analysis the generated alerts and correlating that information with an analysis of the ports which internal machines were using to communicate (from the scans files), to see if any known malicious ports were being over-utilized and to see if a malicious pattern would emerge. Machines appearing in this list are not guaranteed to have become infected, but are showing signs of probable infection and should be looked at. On the flip-side, machines not appearing in this list may be compromised but did not display enough activity to have been either 1) flagged by the Intrusion Detection System (IDS) or 2) noticed in this analysis.

| IP Address | Possible Compromise | Reason Compromised is Believed |
|---|---|---|
| MY.NET.24.20 | Adore worm (listed as Red Worm in IDS rule) | Lots of ports 25 and 80 traffic with port 65535, correlating to Adore activity. |
| MY.NET.24.44 | Adore worm and Ramen worm | Lots of port 80 and port 65535 traffic, correlating to Adore activity. Also, lots of port 25 and 27374 traffic, correlating to Ramen activity. |
| MY.NET.163.249 | SubSeven trojan and Blaster worm | Lots of port 25 and port 27374 activity, correlating to SubSeven activity (I don't think Ramen is present because I believe this host to be compromised with Blaster as well, meaning it has to be a Microsoft machine). This machine also triggered an "Internal MSBlast Infection Request" alert, indicating that it is probably infected with the Blaster worm. |
| MY.NET.24.34 | Ramen worm | Lots of port 80 and port 27374 activity correlating to Ramen activity. |
| MY.NET.27.103 | NETBIOS exploit | Many "EXPLOIT x86 NOOPs" followed by both "SMB Name Wildcard" and "SMB C Access" alerts. |
| MY.NET.29.3 | NETBIOS exploit | Many "EXPLOIT x86 NOOPs" followed by "SMB Name Wildcard," "SMB C Access," and a "NETBIOS NT NULL session" alerts. |
| MY.NET.12.6 | MiMail | Some port 25 traffic, correlating to MiMail activity, to include "[UMBC NIDS] External MiMail alert" alerts. |
| MY.NET.6.15 | Back Orifice | Lots of port 31337 (hacker speak for eleet) activity, correlating to the Back Orifice trojan. |

figure 5

Internal Top Talkers

Figure 6 is a list of internal top talkers compromised not of the machines that generated the most noise (i.e. generated the most alerts), but by the machines that generated the most unique alerts (i.e. these machines generated more than one different type of alert). If the name is in bold, then that machine also appears in the possibly compromised machines list (figure 5).

| # | Uniq Alerts | Src Address | Alerts | # Alerts |
|---|---|---|---|---|
| 1 | 6 | MY.NET.163.249 | 1. Possible trojan server activity<br>2. SMB Name Wildcard<br>3. Tiny Fragments - Possible Hostile Activity<br>4. TFTP - Internal UDP connection to external tftp server<br>5. [UMBC NIDS IRC Alert] Possible sdbot floodnet detected<br>6. [UMBC NIDS] Internal MSBlast Infection Request | 409<br>26<br>8<br>4<br>4<br>2 |
| 2 | 4 | MY.NET.84.143 | 1. SMB Name Wildcard<br>2. TFTP - Internal TCP connection to external tftp server<br>3. High port 65535 tcp - possible Red Worm - traffic<br>4. TFTP - Internal UDP connection to external tftp server | 9<br>6<br>3<br>3 |
| 3 | 3 | MY.NET.84.198 | 1. SMB Name Wildcard<br>2. High port 65535 tcp - possible Red Worm - traffic<br>3. High port 65535 udp - possible Red Worm - traffic | 19<br>5<br>1 |
| 4 | 3 | MY.NET.80.105 | 1. High port 65535 tcp - possible Red Worm - traffic<br>2. SMB Name Wildcard<br>3. High port 65535 udp - possible Red Worm - traffic | 1113<br>4<br>1 |
| 5 | 3 | MY.NET.70.176 | 1. High port 65535 udp - possible Red Worm - traffic<br>2. SMB Name Wildcard<br>3. NIMDA - Attempt to execute cmd from campus host | 80<br>2<br>1 |
| 6 | 3 | MY.NET.29.3 | 1. SMB Name Wildcard<br>2. Possible trojan server activity<br>3. High port 65535 tcp - possible Red Worm - traffic | 109<br>11<br>2 |
| 7 | 3 | MY.NET.24.44 | 1. High port 65535 tcp - possible Red Worm - traffic<br>2. Possible trojan server activity<br>3. TFTP - external TCP connection to internal tftp server | 23<br>8<br>1 |
| 8 | 3 | MY.NET.153.94 | 1. High port 65535 tcp - possible Red Worm - traffic<br>2. High port 65535 udp - possible Red Worm - traffic<br>3. SMB Name Wildcard | 11<br>3<br>2 |
| 9 | 3 | MY.NET.153.195 | 1. TFTP - Internal UDP connection to external tftp server<br>2. SMB Name Wildcard<br>3. [UMBC NIDS] Internal MSBlast Infection Request | 19<br>13<br>1 |
| 10 | 3 | MY.NET.112.159 | 1. SMB Name Wildcard<br>2. TFTP - Internal UDP connection to external tftp server<br>3. High port 65535 udp - possible Red Worm - traffic | 30<br>2<br>1 |

figure 6

Figure 7 is a list of internal top talkers with the most number of scans logged. Scans can either be legitimate scanning attempts, or just a server that is attempting to contact another server and retrying the connection many times. For

example, Simple Mail Transport Protocol (SMTP) servers often flag SYN scans when attempting to contact other mail servers.

| Rank | # Scans | IP Address | Uniq Ports | Rank | # Scans | IP Address | Uniq Ports |
|---|---|---|---|---|---|---|---|
| 1 | 2109617 | 130.85.1.3 | 5914 | 6 | 289674 | 130.85.70.129 | 289674 |
| 2 | 1130592 | 130.85.70.154 | 3976 | 7 | 273705 | 130.85.42.1 | 274699 |
| 3 | 841854 | 130.85.163.107 | 3875 | 8 | 217312 | 130.85.1.5 | 217312 |
| 4 | 775242 | 130.85.84.194 | 3974 | 9 | 181376 | 130.85.111.72 | 181376 |
| 5 | 669973 | 130.85.163.249 | 3977 | 10 | 175961 | 130.85.80.149 | 175961 |

figure 7

Figure 8 is a listing of the top, and only, six internal IP addresses and the number of OOS packets. These packets all have something wrong with them, as seen from the IDS point of view. In this case, packets with Explicit Congestion Notification (ECN) set were flagged as being OOS packets, though many of them are probably legitimate packets as ECN is becoming more commonplace. Packets with the ECN bit(s) set accounted for 98% of all the OOS packets.

| Rank | # Packets | IP Address | Rank | # Packets | IP Address |
|---|---|---|---|---|---|
| 1 | 93 | MY.NET.216.50 | 6 | 1 | MY.NET.12.7 |
| 2 | 18 | MY.NET.12.4 | | | |
| 3 | 12 | MY.NET.12.6 | | | |
| 4 | 8 | MY.NET.15.49 | | | |
| 5 | 2 | MY.NET.12.2 | | | |

figure 8

**Relavent Information**

This section focuses on knowledge that is important, but does not necessarily require immediate action. This section contains a list of all attacks flagged as well as some interesting statistics about each alert and an external top talkers list in terms of alerts, scans, and OOS.

List of all Alerts

Figure 9 is a listing of all alerts. While this section does not go into any depth, this list is informative and good information for the security managers to be aware of. This chart shows the number of unique source and destination internal and external IP addresses and ports.

| # Unique Chart | Internal Machines | | | | External Machines | | | |
|---|---|---|---|---|---|---|---|---|
| Name | src ip | src port | dst ip | dst port | src ip | src port | dst ip | dst port |
| SMB Name Wildcard | 906 | 153 | 0 | 0 | 0 | 0 | 100921 | 23 |
| SMB C Access | 0 | 0 | 959 | 1 | 630 | 7333 | 0 | 0 |
| MY.NET.30.4 activity | 0 | 0 | 1 | 36 | 447 | 3209 | 0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| EXPLOIT x86 NOOP | 0 | 0 | 947 | 105 | 1417 | 4582 | 0 | 0 |
| connect to 515 from inside | 4 | 5 | 0 | 0 | 0 | 0 | 1 | 1 |
| MY.NET.30.3 activity | 0 | 0 | 1 | 37 | 91 | 493 | 0 | 0 |
| External RPC call | 0 | 0 | 1830 | 1 | 6 | 1604 | 0 | 0 |
| High port 65535 tcp - possible Red Worm - activity | 43 | 23 | 43 | 22 | 57 | 7 | 74 | 10 |
| ICMP SRC and DST outside network | 0 | 0 | 0 | 0 | 114 | N/A | 1564 | N/A |
| Possible trojan server activity | 23 | 9 | 156 | 10 | 52 | 164 | 40 | 11 |
| NMAP TCP ping! | 0 | 0 | 61 | 57 | 153 | 13 | 0 | 0 |
| High port 65535 udp - possible Red Worm - traffic | 16 | 8 | 50 | 37 | 66 | 32 | 39 | 3 |
| SUNRPC highport access! | 0 | 0 | 23 | 1 | 18 | 9 | 0 | 0 |
| TFTP - Internal UDP connection to external tftp server | 7 | 15 | 13 | 13 | 11 | 1 | 17 | 1 |
| [UMBC NIDS IRC Alert] IRC user /kill detected | 0 | 0 | 6 | 54 | 1 | 51 | 0 | 0 |
| Null scan! | 0 | 0 | 56 | 18 | 62 | 92 | 0 | 0 |
| [UMBC NIDS IRC Alert] XDCC client detected attempting to IRC | 5 | 241 | 0 | 0 | 0 | 0 | 3 | 1 |
| TCP SRC and DST outside network | 0 | 0 | 0 | 0 | 27 | 108 | 108 | 67 |
| FTP passwd attempt | 0 | 0 | 6 | 1 | 36 | 100 | 0 | 0 |
| [UMBC NIDS] External MiMail alert | 0 | 0 | 1 | 1 | 46 | 99 | 0 | 0 |
| Back Orifice | 0 | 0 | 84 | 1 | 2 | 2 | 0 | 0 |
| Incomplete Packet Fragments Discarded | 0 | 0 | 48 | 1 | 60 | 1 | 0 | 0 |
| EXPLOIT x86 stealth noop | 0 | 0 | 10 | 27 | 11 | 27 | 0 | 0 |
| Tiny Fragments - Possible Hostile Activity | 1 | 1 | 23 | 1 | 37 | 1 | 1 | 1 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC | 7 | 55 | 0 | 0 | 0 | 0 | 2 | 4 |
| NETBIOS NT NULL session | 0 | 0 | 13 | 1 | 3 | 48 | 0 | 0 |
| [UMBC NIDS IRC Alert] Possible drone command detected | 0 | 0 | 8 | 18 | 3 | 1 | 0 | 0 |
| DDOS shaft client to handler | 0 | 0 | 2 | 1 | 5 | 1 | 0 | 0 |
| EXPLOIT x86 setgid 0 | 0 | 0 | 20 | 23 | 21 | 15 | 0 | 0 |
| EXPLOIT x86 setuid 0 | 0 | 0 | 16 | 16 | 23 | 21 | 0 | 0 |
| EXPLOIT NTPDX buffer overflow | 0 | 0 | 12 | 1 | 8 | 13 | 0 | 0 |
| FTP DoS ftpd globbing | 0 | 0 | 1 | 1 | 5 | 7 | 0 | 0 |
| DDOS mstream client to handler | 0 | 0 | 2 | 2 | 3 | 2 | 0 | 0 |
| TFTP - Internal TCP connection to external tftp server | 1 | 1 | 2 | 2 | 2 | 1 | 1 | 1 |
| Attempted Sun RPC high port access | 0 | 0 | 7 | 1 | 5 | 6 | 0 | 0 |
| TFTP - External UDP connection to internal tftp server | 0 | 0 | 6 | 1 | 6 | 6 | 0 | 0 |
| RFB - Possible WinVNC - 010708-1 | 4 | 5 | 4 | 5 | 2 | 2 | 2 | 1 |
| HelpDesk MY.NET.70.49 to External FTP | 1 | 8 | 1 | 1 | 1 | 1 | 4 | 1 |
| NIMDA - Attempt to execute cmd from campus host | 4 | 4 | 0 | 0 | 0 | 0 | 3 | 1 |
| [UMBC NIDS IRC Alwert] K\:line'd user detected | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| [UMBC NIDS] Internal MSBlast Infection Request | 2 | 1 | 0 | 0 | 0 | 0 | 3 | 3 |
| Traffic from port 53 to port 123 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TFTP - External TCP connection to internal tftp server | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| connect to 515 from outside | 0 | 0 | 2 | 1 | 1 | 2 | 0 | 0 |
| [UMBC NIDS IRC Alert] Possible trojaned box detected attempting to IRC | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| Probable NMAP fingerprint attempt | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| External FTP to HelpDesk MY.NET.70.50 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| External FTP to HelpDesk MY.NET.70.49 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| External FTP to HelpDesk MY.NET.53.29 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| Bugbear@MM virus in SMTP | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

figure 9

External top talkers

Figure 10 is a list of external top talkers compromised not of the machines that generated the most alerts noise (i.e. generated the most alerts), but by the machines that generated the most unique alerts (i.e. these machines generated more than one different type of alert). There were actually 23 machines that generated three or more alerts, but only the top ten are listed here. The criteria for the final ten (out of 23 competitors with three or more alerts), was the total number of all alerts per machine.

| # | Uniq Alerts | Src Address | Alerts | # Alerts |
|---|---|---|---|---|
| 1 | 5 | 208.153.50.192 | 1. High port 65535 udp - possible Red Worm - traffic<br>2. EXPLOIT NTPDX buffer overflow<br>3. Attempted Sun RPC high port access<br>4. TFTP - External UDP connection to internal tftp server<br>5. TFTP - Internal UDP connection to external tftp server | 16<br>6<br>2<br>1<br>1 |
| 2 | 4 | 64.209.74.229 | 1. connect to 515 from outside<br>2. External RPC call<br>3. SUNRPC highport access!<br>4. TFTP - External TCP connection to internal tftp server | 2<br>2<br>1<br>1 |
| 3 | 4 | 63.250.195.10 | 1. High port 65535 udp - possible Red Worm - traffic<br>2. EXPLOIT NTPDX buffer overflow<br>3. Attempted Sun RPC high port access<br>4. TFTP - Internal UDP connection to external tftp server | 100<br>8<br>7<br>5 |

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 193.114.70.169 | 1. External RPC call<br>2. NETBIOS NT NULL session<br>3. MY.NET.30.4 activity<br>4. MY.NET.30.3 activity | 2837<br>49<br>3<br>2 |
| 5 | 3 | 209.6.97.168 | 1. EXPLOIT x86 NOOP<br>2. MY.NET.30.4 activity<br>3. MY.NET.30.3 activity | 764<br>5<br>2 |
| 6 | 3 | 195.110.140.66 | 1. EXPLOIT x86 NOOP<br>2. MY.NET.30.3 activity<br>3. MY.NET.30.4 activity | 314<br>2<br>1 |
| 7 | 3 | 4.34.198.112 | 1. EXPLOIT x86 NOOP<br>2. MY.NET.30.3 activity<br>3. MY.NET.30.4 activity | 260<br>3<br>3 |
| 8 | 3 | 217.82.34.195 | 1. EXPLOIT x86 NOOP<br>2. MY.NET.30.3 activity<br>3. MY.NET.30.4 activity | 253<br>3<br>3 |
| 9 | 3 | 130.67.101.88 | 1. EXPLOIT x86 NOOP<br>2. TFTP - Internal UDP connection to external tftp server<br>3. MY.NET.30.4 activity | 242<br>4<br>2 |
| 10 | 3 | 200.29.18.227 | 1. EXPLOIT x86 NOOP<br>2. MY.NET.30.3 activity<br>3. MY.NET.30.4 activity | 205<br>1<br>1 |

figure 10

Figure 11 is the second list of external top talkers with the most number of scans.

| Rank | # Scans | IP Address | Uniq Ports | Rank | # Scans | IP Address | Uniq Ports |
|---|---|---|---|---|---|---|---|
| 1 | 74611 | 218.94.41.98 | 3973 | 6 | 16244 | 213.51.194.191 | 1514 |
| 2 | 30239 | 213.180.193.68 | 2 | 7 | 15688 | 200.168.78.213 | 3899 |
| 3 | 25234 | 63.250.195.10 | 6267 | 8 | 14827 | 219.121.66.87 | 3665 |
| 4 | 19164 | 193.114.70.169 | 2855 | 9 | 14549 | 217.158.99.5 | 3665 |
| 5 | 18246 | 68.85.216.188 | 4 | 10 | 14452 | 165.123.179.206 | 962 |

figure 11

Figure 12 is the third top talkers list of the top ten external IP addresses and the number of OOS packets.

| Rank | # Packets | IP Address | Rank | # Packets | IP Address |
|---|---|---|---|---|---|
| 1 | 1515 | 217.174.98.145 | 6 | 428 | 66.225.198.20 |
| 2 | 1100 | 195.111.1.93 | 7 | 358 | 195.208.238.143 |
| 3 | 872 | 158.196.149.61 | 8 | 335 | 195.101.94.101 |
| 4 | 750 | 212.16.0.33 | 9 | 320 | 195.101.94.208 |
| 5 | 627 | 82.82.64.209 | 10 | 294 | 195.101.94.209 |

figure 12

**Primary Alerts**

This section gives a listing of the top ten most frequent alerts on the universities network along with some statistical information, a general description

of the alert, cause for concern, IDS signature, correlative sources, and defensive recommendations. The signatures section does not give the recommended, or even Snort default signature, but the signature as it appears to be from the information provided from the university. The alerts presented in this section were deemed as the most interesting alerts due to the total number of times alerted. Ten alerts total will be analyzed from greatest to least, in terms of the total number of alerts. Ten seemed to be the magic number because each of these alerts alerted more than 1000 times (the next greatest was 752), as well as alerting more than 100 times on any given day. Figure 13 provides a graphical viewing of the statistics that will be presented below.

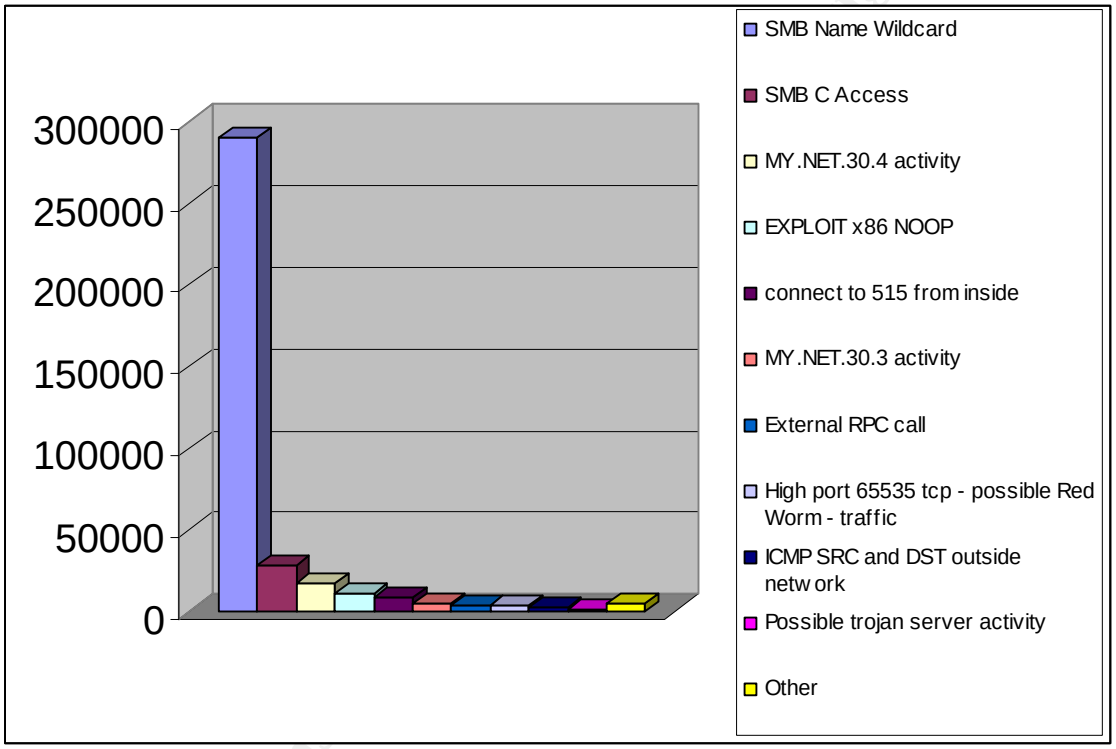


figure 13

### Alert #1 - SMB Name Wildcard

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 1 | 290578 | 77% | SMB Name Wildcard | 225971 | MY.NET.80.51 | 1265 | 198.62.205.6 |
| | | | | 53257 | MY.NET.150.133 | 1251 | 151.197.115.143 |
| | | | | 3100 | MY.NET.29.2 | 1208 | 193.114.70.169 |
| | | | | 1290 | MY.NET.84.224 | 878 | 199.181.134.74 |
| | | | | 454 | MY.NET.150.198 | 675 | 169.254.45.176 |

figure 14

Impact: Information Gathering
Protocol: UDP

Affected: Microsoft
Port: 137

Unique src addresses: 906 Unique dst addresses: 100921
Unique src ports: 153 Unique dst ports: 1

Description

This alert triggers whenever a Windows machine (or *nix machine running Samba) attempts a standard NetBIOS name table lookup. These types of queries are necessary to determine the NetBIOS name of the file or print hosting machine in a pre-Windows 2000 [native] environment. These types of requests are generally indicative of machines prior to Windows 2000, though Windows 2000 and XP machines can still perform those types of queries when not part of a native 2000 network; a native network is compromised entirely of 2000 and/or XP machines. If Windows 2000/XP machines are unable to perform Lightweight Directory Access Protocol (LDAP) queries when in a native environment, they fail over to NetBIOS to perform the name lookups.

This alert is labeled SMB Name Wildcard because the actual request is a wildcard request to the NetBIOS name service using the Server Message Block (SMB) protocol, which rides on top of NetBIOS. When a name query is requested via IP address (instead of NetBIOS name), the request is actually a wildcard request sent directly to the input IP address. Windows “mangles” the wildcard request by dividing each character of the 16 character NetBIOS name (which is an asterisk in this case because of the wildcard) into two hexadecimal characters and adds them to the value 0x41 (A)[28]. In the case of a wildcard request, the asterisk is 2A in hexadecimal, which becomes C and K (41 + 2 = 43 which is C and 41 + A = 4B which is K), and the other 14 characters are padded with nulls, which when added to 41 equals 41, or A[28]. This results in a distinct signature of “CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA.” The string is 32 characters long because each of the original 16 characters got broken into two hexadecimal characters.

Correlations

1. Judy Novak gives a nice description of how the connection associated with the SMB Name Wildcard alert actually works[28].
2. Carnegie Mellon’s CERT has an incident write-up on the network.vbs worm[30].
3. An interesting note is that Marcus Wu mentions in his GCIA practical[31] that he believes the rule had been re-written since Tod Beardsley’s practical[32] to only alert on external sources. It appears that the rule was changed back.
4. James Maher[33] lists eight different SMB vulnerabilities in his GCIA practical.

Signature

alert udp any any -> any 137 (msg:”SMB Name Wildcard”; flags: A+; content:”CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|”;)

The signature for this alert seems look for UDP port 137 requests with a payload of “CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA,” followed by two

bytes of zero's, or four zero's in a row. This signature also only flags on packets with the ACK flag and one other flag set.

Cause for Concern

According to Whitehats.com[29], "many Windows machines will send these NetBIOS name requests by default when negotiating various connections with other systems (not just NetBIOS)." When Windows machines make these types of requests, they break RFC protocol and use 137 as both the source and destination port. Any variation from this and it becomes either a badly mis-configured machine, or a reconnaissance attempt.

The cause for concern of this alert comes from the fact that out of 290578 total alerts, only 15864 have a source port of 137 and therefore appear to be valid NetBIOS name requests. The rest can only be assumed to be malicious. In this case, it is rather odd that every single alert came from an internal machine to an external machine, when the signature should be configured to look the other way (external to internal). It seems likely that $HOME_NET is set to any, though it seems strange that the university would not get a single hit from an external source. Another possibility is that the internal and external definitions defined in the Snort configuration file are reversed, but there is no evidence in the other alerts to suggest this. Another possibility is the presence of the network.vbs worm, which is a Visual Basic script that infects Windows machines and proliferates through unprotected shares on the C drive[30].

Recommendations

The top five offenders, all internal machines, should be checked for any abnormal activity which could explain the excessive amount of NetBIOS name requests to external machines. Special care should be taken to look for the presence of the network.vbs worm, a description of which can be found at the CERT[30]. If the worm is found, a large-scale scanning effort to locate and eradicate the worm should be undertaken.

While many organizations are very diligent about filtering ingress (incoming) traffic, many sites lack egress (outgoing) filtering. I recommend configuring the universities firewalls to perform egress filtering on NetBIOS traffic, which have no business being sent unencrypted across the internet. If this traffic is found to be legitimate, and deemed essential, I recommend tunneling the traffic through a Virtual Private Network (VPN). This will allow NetBIOS, a known insecure protocol, to pass securely through the internet. If a VPN is not a realistic solution to this problem, Open Secure Shell (OpenSSH) is a free solution. OpenSSH is freely available for Win32 machines from http://lexa.mckenna.edu/sshwindows/. A *nix version of OpenSSH is available from http://www.openssh.com/.

A final recommendation would be to tune the SMB Name Wildcard alert to only trigger on traffic the university needs to worry about. I would recommend configuring the rule to look like: alert udp !$EXTERNAL_NET any -> $HOME_NET 137 (the rest of the rule should remain the same). If configured this way, the rule will only trigger if an external source attempts to scan the internal

network. If egress filtering is not going to be applied, and the university wants to know whether the internal machines are scanning external machines, the rule would look like: alert udp any any -> any 137.

**Alert #2 - SMB C access**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 2 | 28523 | 8% | SMB C Access | 663 | 80.50.168.42 | 5088 | MY.NET.84.228 |
| | | | | 295 | 138.89.11.51 | 1146 | MY.NET.191.52 |
| | | | | 236 | 61.147.18.195 | 149 | MY.NET.152.166 |
| | | | | 224 | 61.223.139.116 | 123 | MY.NET.111.225 |
| | | | | 217 | 203.197.20.41 | 117 | MY.NET.110.220 |

figure 15

Impact: Possible Access Compromise
Protocol: TCP
Unique src addresses: 630
Unique src ports: 7333
Affected: Microsoft
Port: 139
Unique dst addresses: 960
Unique dst ports: 1

Description

This alert is generated when a connection attempt is made to the C$ default administrative share on a Windows machine. Only accounts with administrative rights on a machine are able to connect to the C$ share. If this connection were to succeed, the “C:” filesystem could be accessed. This could be a legitimate administrative connection, an illegitimate administrative connection, or an information gathering attempt. In order to open the C$ share, the source host must first complete the TCP three-way handshake on port 139 (NetBIOS session service) before the SMB protocol can take over. Unlike other NetBIOS ports, ephemeral ports (these are client port numbers, and are ports greater than 1023) are used in this connection.

Correlations

1. Hee So covers this alert as one of his network detects[44].
2. Al Maslowski-Yerges briefs covers this alert in his practical[45].
3. Daniel Wesemann covers this alert as his second network detect[46].

Signature

alert tcp any any -> $HOME_NET 139 (msg:”SMB C Access”;content:”5c|C$|00 41 3a 00|”;flow:to_server,established;)

The signature for this alert looks for tcp port 139 requests coming from an external source to an internal source with a payload of: “|5c|C$|00 41 3a 00|” from an established connection. This signature also only flags on packets with the ACK flag and one other flag set.

Cause for Concern

As can be seen in the stats section for this alert, the top five alerting sources were all external machines which should definitely not be happening, for any reason. There is no reason for an external machine to be connecting unencrypted to an administrative share on the internal network. If the connection was between two internal machines, it could easily be part of a normal administrative connection to the machine. If the campus policy is that all default shares will be disabled, as is recommended, then there is even more reason for concern. Another cause for these alerts could be the network.vbs worm, which was covered in the SMB Name Wildcard alert.

Recommendation

I recommend that the university blocks ingress and egress NetBIOS traffic, which are ports 135, 137, 138, 139, and 445 unless there is a specific need to allow that traffic. If NetBIOS traffic needs to pass through the firewall, I recommend tunneling the traffic through a Virtual Private Network (VPN). This will allow NetBIOS, a known insecure protocol, to pass securely through the internet. If an IPSEC VPN is not a realistic solution to this problem, Open Secure Shell (OpenSSH) is a freely available solution. OpenSSH is freely available for Win32 machines from http://lexa.mckenna.edu/sshwindows/. A *nix version of OpenSSH is available from http://www.openssh.com/. If OpenSSH is also not a realistic solution for the university, and administrative shares absolutely need to be accessed by sources outside the internal network, the firewall should be statically configured to only allow incoming connections from certain sources to certain destinations.

All destination machines from this alert should be thoroughly checked for root kits or other implanted malicious software, such as the network.vbs worm, or at the very least the top five destination machines should be checked. It is a good idea to reload the Operating System (OS) of every internal machine listed as a destination address from a known good, such as an image cd. While this is not a realistic action to perform on many networks, this is the safest way to ensure that the machines are not compromised, and to fix them if they were compromised.

**Alert #3 – MY.NET.30.4 activity**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 3 | 17590 | 5% | MY.NET.30.4 activity | 2934<br>2734<br>1124<br>997<br>959 | 68.55.85.180<br>68.54.91.147<br>172.142.110.232<br>151.196.19.202<br>151.196.10.97 | 17590 | MY.NET.30.4 |

figure 16

Impact: Unknown
Protocol: Any

Affected: MY.NET.30.4
Port: All

Unique src addresses: 447 Unique dst addresses: 1
Unique src ports: 3209 Unique dst ports: 37

Description

From looking at the provided log information, it seems that this alert is only generated when external traffic attempts to reach the MY.NET.30.4. Unfortunately, the actual type of traffic was not able to be observed. The top five destination ports were port 51443 (unknown), 80 (HTTP), port 524 (Novell Directory Service or NCP), 135 (WINS/DHCP), and 445 (SMB without NetBIOS). Unless specific rules were created to allow NDS or NCP through a firewall, port 524 traffic should not be reaching MY.NET.30.4. There is no reason that port 135 or port 445 traffic should be coming through the firewall and accessing MY.NET.30.4. It is unknown why traffic was sent to port 51443. Perhaps traffic was sent to ports 524, 135, and 445 in an OS fingerprinting attempt (Novell, WinNT, and Windows 2000 ports respectively), as well as web server identification (port 80).

Correlations

This alert appeared in almost every students practical, but this seems to be the first practical to analyze the alert, so there is very little in the way of correlating information. In addition, because this is a rule created by the university for their purposes, there is no documentation available on the internet.

Signature

alert any $EXTERNAL_NET any -> MY.NET.30.4 any (msg:”MY.NET.30.4 activity”;)

From the information provided, it looks like this alert is triggered whenever and external machine attempts to contact MY.NET.30.4, no matter which protocol or port is used.

Cause for Concern

It is very difficult to determine the cause for concern for this alert without being provided any justification for the alert. It is unknown why MY.NET.30.4 is considered special, in that it has its own alert. Since this alert indicates that the university wants to know whenever any machine attempts to talk to MY.NET.30.4, I would recommend that there is cause for concern because there were a total of 17590 MY.NET.30.4 activity alerts from 2075 different source addresses. Another note is that one other alert was flagged with a destination address of MY.NET.30.4, and that was the External RPC call. One reason for this could be that the External RPC call alert is checked before the MY.NET.30.4 rule (Snort stops processing rules after one has been matched).

Recommendation

Since the university seems to be concerned about external hosts communicating with this host, I would recommend that only connections

originating from MY.NET.30.4 be allowed through the firewalls, a technique known as keeping state. If the universities firewalls are not stateful, I recommend that the university purchase stateful firewalls at the first opportunity. Good firewalls to consider would be Checkpoints FW-1 or Cisco's PIX. An open source solution that is easily available and widely supported is iptables, which runs on most *nix platforms.

Another option freely available to the university is to run a tool such as Nmap against the internal machines. Nmap is generally considered the best port scanner/OS fingerprinter on the market, and is free, both the *nix and Win32 versions. The university may want to run this tool looking for open known server and file sharing ports.

**Alert #4 – EXPLOIT x86 NOOP**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 4 | 11362 | 3% | EXPLOIT x86 NOOP | 764 | 209.6.97.168 | 375 | MY.NET.15.198 |
| | | | | 418 | 24.87.153.94 | 366 | MY.NET.27.103 |
| | | | | 412 | 216.232.208.22 | 200 | MY.NET.80.16 |
| | | | | 314 | 195.110.140.66 | 190 | MY.NET.81.18 |
| | | | | 280 | 63.229.211.22 | 176 | MY.NET.29.2 |

figure 17

Impact: Buffer Overflow Attempt
Protocol: Any
Unique src addresses: 1417
Unique src ports: 4582

Affected: All Hosts
Port: Any
Unique dst addresses: 948
Unique dst ports: 106

Description

This alert triggers when the character 0x90 is detected in the payload because it represents a NOOP instruction. NOOP, or NOP stands for no operation and is used as padding for a buffer overflow attack. The technique of including many NOOPs before a buffer overflow is known as a NOOP slide and helps to position the return pointer in such a way that the attacker's code will be executed[35]. The presence of a NOOP slide is a good indication of a buffer overflow attempt. In order to successfully execute code on a smashed stack, or flooded buffer, the attacker must ensure that the memory return pointer actually points to a valid memory space, preferably to a space that contains data the attacker wants executed[35]. In order to place the return pointer "just right," the attacker will use a NOOP slide to increase the chances the return pointer points somewhere in the NOOP slide, which means the CPU will simply do nothing until it encounters another instruction, which will be the instruction(s) the attacker planted[35].

Correlations

1. The most well known paper on buffer overflows was written by Aleph One and is entitled "Smashing the Stack for Fun and Profit,"[36].

2. James Maher talks about this alert is his practical[33].
3. Terry Macdonald also covers this alert in his practical[37].

Signature

alert any $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP"; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90|";)

This alert looks for any traffic originating from an external source containing the infamous 0x90 NOOP slide signature.

Cause for Concern

This alert is known to produce many false positives. Large binary file transfers, such as can be found in ftp, NetBIOS, and HTTP, amongst others, oftentimes trigger this alert. The majority of the alerts were to port 135, which is commonly used by the Windows Internet Naming Service (WINS) and Windows Dynamic Host Control Protocol (DHCP), which are not known to easily produce false positives. Most of the rest of the alerts were for ports 80 (HTTP) and 445 (SMB without NetBIOS), which are known false positives. There is probably cause for concern for the five machines that made the top five destination addresses.

Recommendation

As is stated in the cause for concern section, I recommend that at least the top five recipients of this alert be checked for rootkits, modified system files, or other malicious logic that could have been planted after a successful buffer overflow. In an attempt to help reduce the number of false positives, the SHELLCODE_PORTS variable should be defined and used in the rule. The SHELLCODE_PORTS variable should look something like: SHELLCODE_PORTS !20 !80 !137 !138 !139 !445. I would recommend changing the rule to something like: alert any $EXTERNAL_NET any -> $HOME_NET SHELLCODE_PORTS (msg:"EXPLOIT x86 NOOP"; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90|";)

**Alert #5 - connect to 515 from inside**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 5 | 8056 | 2% | connect to 515 from inside | 8056 | MY.NET.162.41 | 8054 | 128.183.110.242 |

figure 18

Impact: Noise
Protocol: TCP
Unique src addresses: 1
Unique src ports: 1

Affected: *nix
Port: 515
Unique dst addresses: 1
Unique dst ports: 1

Description

From the given data given it seems that this alert only flags when an internal machine attempts to connect to an external machine on destination port 515, as the name implies. In the five days of examined traffic, the only machine to flag this alert was MY.NET.162.41 using source port 721 and destination address 128.183.110.242. According to RFC 1179, the source port must be between ports 721 and 731 in order to use the line printer (lpr) service, so this traffic looks like legitimate lpr usage. 128.183.110.242 falls under NASA's IP range, and a reverse DNS lookup on the machine returns tek924.gsfc.nasa.gov. Perhaps someone at the university was working on a project with NASA and sending research results to the space center, or perhaps they are trying to hack one of NASA's printers? There does not seem to be any type of time pattern fitting this traffic. Tod Beardsley called this alert noise in his practical[32] as it implies that a connection to the line printer service has been initiated.

Correlations

1. Peter Van Oosterom covered this alert in his practical[38].
2. Tod Beardsley covered this alert in his practical[32].
3. Terry Macdonald briefly covered this alert in his practical[37].

Signature

alert tcp $HOME_NET any -> $EXTERNAL_NET 515 (msg:"connect to 515 from inside"; flags: A+;)

This signature looks for all connections with an internal source IP address and a destination IP address of an external machine on port 515, the linux lpr port. This signature also only flags on packets with the ACK flag and one other flag set.

Cause for Concern

From the information provided, it looks like all 8056 alerts were from lpr connections. It is unclear why the university would want this alert; I am going to agree with Tod Beardsley that this alert is noise and should be removed[32]. If the university is concerned about students connecting to external printers, they need to block outgoing port 515. If the university is worried about its internal machines performing lpr attacks, the Snort rule "EXPLOIT LPRng overflow" should be sufficient[32]. As for the actual connection, the university must decide whether one of its machines should be communicating with a NASA printer or not. If the answer is that they should not be, then NASA should be notified that someone was apparently trying to access one of their printers. With the sporadic nature of the alerts, as well as the obvious nature of the packets, this is most likely legitimate traffic or a script kitty playing with a very noisy script.

Recommendation

I recommend that this alert be removed from the ruleset as it produces noise and does not seem to hold any real value to the university. I also recommend that outgoing port 515 be blocked at the firewall, unless the traffic was valid. If the traffic was valid, I recommend tunneling the traffic through a VPN, such as has been mentioned in the SMB alerts earlier.

**Alert #6 – MY.NET.30.3 activity**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 6 | 4374 | 1% | MY.NET.30.3 activity | 673<br>572<br>572<br>560<br>462 | 68.55.27.151<br>68.57.90.146.<br>141.157.6.106<br>68.55.62.79<br>68.55.105.5 | 4374 | MY.NET.30.3 |

figure 19

Impact: Unknown
Protocol: Any
Unique src addresses: 91
Unique src ports: 493
Affected: MY.NET.30.3
Port: All
Unique dst addresses: 1
Unique dst ports: 37

Description

From looking at the provided log information, it seems that this alert is only generated when external traffic attempts to reach the MY.NET.30.3. Unfortunately, the actual type of traffic was not able to be observed. The top five destination ports were port 524 (Novell Directory Service or NCP), 135 (WINS/DHCP), 80 (HTTP), 445 (SMB without NetBIOS), and 4000 (ICQ). Unless specific rules were created to allow NDS or NCP through a firewall, port 524 traffic should not be reaching MY.NET.30.3. There is no reason that port 135 or port 445 traffic should be coming through the firewall and accessing MY.NET.30.3. Perhaps traffic was sent to ports 524, 135, and 445 in an OS fingerprinting attempt (Novell, WinNT, and Windows 2000 ports respectively), as well as web server identification (port 80), and checking for the presence of ICQ (port 4000).

Correlations

This alert appeared in almost every students practical, but this seems to be the first practical to analyze the alert, so there is very little in the way of correlating information. In addition, because this is a rule created by the university for their purposes, there is no documentation available on the internet.

Signature

alert any $EXTERNAL_NET any -> MY.NET.30.3 any (msg:”MY.NET.30.3 activity”;)

From the information provided, it looks like this alert is triggered whenever and external machine attempts to contact MY.NET.30.3, no matter which protocol or port is used.

Cause for Concern

It is very difficult to determine the cause for concern for this alert without being provided any information as to why the rule was created in the first place. It is unknown why MY.NET.30.3 is considered special, in that it has its own alert. Since this alert indicates that the university wants to know whenever any machine attempts to talk to MY.NET.30.3, I would recommend that there is cause for concern because there were a total of 4374 MY.NET.30.3 activity alerts from 730 different source addresses. Another note is that two instances of the External RPC call alert had a destination address of MY.NET.30.3.

Recommendation

The recommendation for this alert is the exact same as for the MY.NET.30.4 activity alert.

**Alert #7 - External RPC call**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 7 | 3287 | <1% | External RPC call | 2837 | 194.114.70.169 | 18 | MY.NET.24.65 |
| | | | | 420 | 81.15.45.1 | 8 | MY.NET.6.15 |
| | | | | 21 | 202.56.195.237 | 6 | MY.NET.28.9 |
| | | | | 7 | 166.102.99.229 | 5 | MY.NET.75.140 |
| | | | | 2 | 64.209.74.229 | 5 | MY.NET.60.172 |

figure 20

Impact: Reconnaissance
Protocol: TCP/UDP
Unique src addresses: 6
Unique src ports: 1604
Affected: *nix
Port: 111
Unique dst addresses: 1831
Unique dst ports: 1

Description

The Remote Procedure Call (RPC) portmapper's designated port is 111 and is a service that runs on *nix machines. The RPC portmapper is used to identify which of certain services are running on which high numbered ports (generally in the 32700 range), "The RPC Portmapper is a server that converts RPC program numbers into TCP/IP (or UDP/IP) protocol port numbers."[48] Basically, when a client makes an RPC call, it first contacts the portmapper service, on port 111, to see which port it should send the request(s) to. For a listing of services that can be offered by RPC, Al Williams has a thorough listing in his practical[47]. There have been many vulnerabilities found with the RPC service, or with services that the RPC portmapper serves, so this can be a dangerous reconnaissance attempt, or an actual hack attempt.

Correlations

1. Sylvain Randier mentioned this alert in his practical[39]
2. Mario Ricci covered this alert in his practical[40]
3. Al Williams does an excellent job covering this alert in his practical[47].

Signature

alert tcp $EXTERNAL_NET any -> $HOME_NET 111 (msg:"External RPC call";)

This alert looks for packets with an external source address and an internal destination address with a destination port of 111 (RPC portmapper).

Cause for Concern

If the universities network is comprised mainly of Windows machines, then there is not much cause for concern. If the university is comprised mainly of *nix machines, then there is cause for concern. While this alert does not specifically alert on an attack, it serves as reconnaissance for an attacker as a starting point from which to launch further attacks, as is documented in Sylvain Randier's practical[39]. An attacker can easily discover which RPC services are being offered by querying the RPC portmapper as well as easily performing OS fingerprinting by looking for responses to portmapper queries. A quick search on www.cve.mitre.org lists five vulnerabilities with the portmapper service (CVE-1999-0168, CAN-1999-0195, CAN-1999-0632, CAN-2001-0617, and CAN-2001-1124). In addition, two other Common Vulnerabilities and Exposures (CVE) numbers, CA-1994-15, and CA-2001-27 (amongst a myriad of other, these are just given as proof of concept) deal with services that a query to the RPC portmapper could offer crucial information of to an attacker, making the service even more dangerous.

Recommendation

There shouldn't be a legitimate reason for non-trusted external hosts to need to connect to the RPC portmapper on internal machines. Incoming port 111 traffic should be blocked at the firewall. If a legitimate need exists for this type of traffic through the firewall, it should get tunneled through a VPN or, at the very least, a firewall entry should exist only allowing certain external IP addresses access to port 111. If the university wants to only look for exploit attempts, the rule should be re-written to include specific payload information, this rule is too general to provide much information.

**Alert #8 - High port 65535 tcp – possible Red Worm – traffic**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 3167 | <1% | High port 65535 tcp - possible Red Worm - traffic | 1113<br>1023<br>310<br>284<br>100 | MY.NET.80.105<br>200.96.13.157<br>MY.NET.153.141<br>66.66.71.92<br>63.250.195.10 | 1112<br>1022<br>320<br>268<br>90 | 200.96.13.157<br>MY.NET.80.105<br>66.66.71.92<br>MY.NET.153.141<br>MY.NET.70.176 |

figure 21

Impact: Possible compromise
Protocol: TCP
Unique src addresses: 100
Unique src ports: 27
Affected: *nix
Port: 65535
Unique dst addresses: 117
Unique dst ports: 29

Description

The Red Worm, now known as the Adore worm, operates on port 65535 and attempts to open a backdoor on the infected system, which grants root access[41]. In order to compromise a system, the trojan uses previously known vulnerabilities in wu-ftp, rpc.statd, LPRng, and BIND DNS[43]. Once the trojan finds a vulnerable system, it compromises the system and waits for a specially crafted Internet Control Message Protocol (ICMP) control packet (77 bytes long). Once that packet has been received, a backdoor is opened listening on port 65535. This alert is prone to false positives because port 65535 is a valid ephemeral port that can be used for valid (non worm) client/server communications. Port 65535 is also known to be used by the RC1 and Sins trojans[43]. Three of the top five talkers for this alert were pretty obviously having a conversation in which 65535 just happened to be the ephemeral port used, but MY.NET.24.20 and MY.NET.24.44 showed some suspicious behavior.

Correlations

There was no lack of correlation information on this alert. More than half of the student practicals viewed covered this alert.

1. Tyler Hudak covered this alert in his practical[41]
2. J. Anthony Dell describes the Adore worm in detail[42] (source gotten from Tyler Hudak[41])
3. James Maher does an excellent job of describing this alert in his practical[33].

Signature

alert tcp any any -> any 65535 (msg:"High port 65535 tcp – possible Red Worm – traffic"; flags: A+;)

This alert looks for any packets containing a destination port of 65535 and with the ACK flag and one other flag set.

Cause for Concern

MY.NET.24.20 had a conversation on Oct 21 on port 25 (SMTP) to 203.176.60.135 (registered in the Asia Pacific Network Information Center,

APNIC), another conversation on Oct 22 on port 25 to 202.71.28.81 (registered in APNIC), and a final conversation on Oct 22 on port 25 to 12.164.136.188 (registered in the American Registry for Internet Numbers, ARIN). It is not very probable that one machine would have that many SMTP conversations on port 65535 in that short a timeframe, and to both US and Asian sites when the Adore worm is known for communicating via port 25 to both US and Asian sites. This machine should definitely be checked for the Adore worm.

MY.NET.24.44 had a conversation on Oct 22 on port 80 (HTTP) to 198.86.10.116 (registered in ARIN), and a final conversation on Oct 22 on port 80 to 66.167.13.186 (registered in ARIN). It is also not very probable that one machine would have two HTTP connections with a source port of 65535. This machine should be checked for the Adore worm.

Other than these two machines, this alert generated mostly noise and should be modified to be more efficient, as recommended in the recommendations section. Both of the machines listed above are in the probably compromised machines list at the beginning of this analysis.

Recommendations

As mentioned in the cause for concern section, MY.NET.24.44 and MY.NET.24.20 should be scanned for the Adore worm. A scanning and removal tool is available from http://www.ists.dartsmouth.edu/IRIA/knowledge_base/tools/adorefind.htm. If the university is very worried about this worm, then the website http://go.163.com[42] should be blocked as well as the following four email addresses[42]: adore9000@21cn.com, adore9000@sina.com, adore9001@21cn.com, and adore9001@sina.com. In addition, all patches for the vulnerable services the Adore worm exploits, BIND, wu-ftp, rpc.statd and LPRng, should be applied.

As is mentioned in the correlations section, this alert is prevalent in more than half of student's practicals. This is a good indication that this alert is very noisy and needs to be modified to trigger less false positives. One way to accomplish this is to specifically tune the rules to only look for packet containing a payload known to match that of the worm, such as the signature offered by http://www.whitehats.org/ for the LPRng vulnerability (www.whitehats.com/cgi/arachNIDS/Show?_id=ids457&view=signatures):

alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg: "IDS457/lpr_LPRng-redhat7-overflow-security.is"; flags: A+; content: "|31DB 31C9 31C0 B046 CD80 89E5 31D2 B266 89D0 31C9 89CB|"; nocase;).

**Alert #9 - ICMP SRC and DST outside network**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 9 | 1564 | <1% | ICMP SRC and DST outside network | 129<br>98<br>98<br>82<br>62 | 192.168.0.235<br>172.171.216.138<br>172.138.40.37<br>192.168.0.209<br>172.152.79.107 | 129<br>82<br>55<br>17<br>3 | 211.150.195.212<br>211.150.203.67<br>192.168.0.2<br>211.150.211.203<br>67.105.78.198 |

figure 22

Impact: Illegitimate traffic
Protocol: ICMP
Unique src addresses: 114
Unique src ports: N/A

Affected: N/A
Port: N/A
Unique dst addresses: 1564
Unique dst ports: N/A

Description

This alert is another of the universities custom alerts, so little information is available, other than the very descriptive title. The university seems to want to know if its students are fooling around with IP spoofing. IP spoofing is when someone sends packets with a crafted source IP address. This technique has both advantages and disadvantages; on one hand the true source of an attack is hidden, but on the other hand, responses will be sent to the spoofed IP address, not to the true attacker. There are many reasons an attacker may wish to hide his true IP address, such as in a denial of service (DoS) attack, in preparing for a man-in-the-middle (MITM) attack, or perhaps for activating a service via a covert channel, such as was seen with the Adore worm earlier. There are many tools readily available on the market that will allow for packet crafting, two of the most popular tools are nmap and hping2.

Correlations

This alert showed up in many students practicals, but only as a statistical figure and not as an analyzed alert.

Signature

alert icmp $EXTERNAL_NET any -> $EXTERNAL_NET any (“msg:ICMP SRC and DST outside network”;)

This alert looks for ICMP packets that have both a source IP address and a destination IP address that are not within the universities range of valid IP addresses. This will also include reserved IP addresses.

Cause for Concern

This alert generates a lot of noise and can easily be made more efficient by following the guidance set forth in the recommendations section. Students playing around with packet crafting probably caused the majority, if not all of, the alerts. The cause for concern of this alert comes from the fact that about 15% of the alerts generated had destination IP addresses to IP addresses registered in Asia. This is probably not normal for a US university. Another 15% of the traffic was generated by IP addresses in the 192.168.0.0/16 reserved IP address

range. Perhaps the university has some machines using that reserved IP address range?

As none of the destination IP addresses were for valid internal university machines, the packets were all created inside the universities network, either by a misbehaving TCP/IP stack, a mis-configured program, or by packet crafting students (or faculty). This is not behavior the university should condone on a production network. Only 22 packets [total] were sent from a totally off-the-wall IP address, 0.0.0.0; they scanned 170.0.227.236-253 and then 170.0.228.1-15. Some alerts in those ranges were missing, probably due to the removal of corrupted log entries. All that activity happened on the 27 of Oct at 11:57:11. This is most likely an example of someone playing around with spoofing.

Recommendation

The easiest way to stop this type of misbehavior is by configuring the universities routers to drop packets with invalid IP addresses. This means packets with source or destination IP addresses of reserved IP address numbers, packets coming from the Internet with a source IP address of an internal machine, and packets coming from the internal network with a source IP address not valid within the universities IP range. By blocking these types of packets at the router, this alert becomes necessary only as part of the universities defense in depth strategy and will only start generating alerts when something very wrong is happening. Another way to help curb this type of behavior is to enforce policy that explicitly states that this type of activity will not be tolerated on the universities networks.

**Alert #10 - Possible trojan server activity**

Stats

| *Rank* | *Total #* | *%* | *Alert* | *Src # Alerts* | *Primary Sources* | *Dst # Alerts* | *Primary Destinations* |
|---|---|---|---|---|---|---|---|
| 10 | 1425 | <1% | Possible trojan server activity | 553<br>409<br>114<br>44<br>24 | 200.163.61.175<br>MY.NET.163.249<br>66.169.146.100<br>24.199.192.33<br>MY.NET.5.44 | 560<br>402<br>29<br>24<br>18 | MY.NET.163.249<br>200.163.61.175<br>MY.NET.12.6<br>209.40.150.118<br>64.41.183.130 |

figure 23

Impact: Possible Compromise
Protocol: TCP
Unique src addresses: 75
Unique src ports: 171

Affected: Microsoft/*nix
Port: 27374
Unique dst addresses: 196
Unique dst ports: 19

Description

This alert seemed to only flag when the source or destination port was equal to 27374. This port is known to be used by the following trojans: Bad Blood, Ramen, Seeker, SubSeven (all versions), and Tftploader (http://www.glocksoft.com/trojan_port.htm). Bad Blood, Seeker, and Tftploader all

piggyback off of SubSeven, so they will not be covered here. If the alert flagged on a Windows machine, then it was most likely by SubSeven, and if the alert flagged on a *nix machine, then it was most likely Ramen. Ramen is easy to notice because the web site running on the vulnerable server gets defaced[49]. Ramen exploits vulnerabilities found in wu-ftp, LPRng, and rpc.statd[49]. SubSeven is known to communicate on the following ports: 1243, 1999, 2773 (Key Logger), 2774, 6667, 6711, 6712, 6713, 6776, 7000 (IRC Bot), 7215 (Matrix), 16959, 27374, 27573, 54283 (ICQ Spy)[49]. Because this alert seems to flag on the port number, I cannot say that every machine that flagged this alert is compromised; more information is needed.

Correlations

1. Sylvain Randier provides an analysis of this alert along with signature recommendation[39].
2. Sunil Sekhri also covered this alert in his practical[50].
3. Peter Van Oosterom also covers this alert in his practical[38].

Signature

Some other students practicals have suggested that this alert also looks for payload information, but I disagree because it seems like many of the alerts generated by this university are by *nix machines, which would mean that they [the university] probably want this rule to be open enough to catch both Ramen and SubSeven activity. This pattern of open rules seems to be evident throughout many of the universities rules.

alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any (msg:"Possible trojan server activity";flags: A+;)

This rule looks for connections on port 27374, which is known for being a well used trojan port, and packets that contain the ACK and one other flag.

Cause for Concern

If this alert is not a false positive, then it becomes a very serious alert very fast. If any of the internal machines are actually compromised, then an attacker has root access to those machines, which means the attacker owns the machine. A list of machines will be given that are suspect to having become infected with either Ramen or SubSeven. I suggest that the recommendations section be followed. Suspect machines: MY.NET.163.249 (could possibly be IRC traffic), MY.NET.24.34 is almost certainly compromised by Ramen (connections to 10 different IP addresses using ports 80 and 27374), and MY.NET.24.44 (this was already suspect with Adore worm activity).

Recommendations

Sylvain Randier provides an excellent recommendation on how the signature should probably be written in order to cut back on the number of false positives[39]. If the university purposely left this rule open, and therefore prone to

false positives, so that it could cover both the Ramen worm and SubSeven trojan, I recommend that they create two rules, each of which look for specific payload information. It also would not be a bad idea to block incoming access to the ports listed above for SubSeven, if the university does not decide to go with the recommended "default deny" policy as is recommended later in this analysis.

The machines listed in the cause for concern section need to be thoroughly checked for either the SubSeven or Ramen trojan activity. http://www.hackfix.org/subseven has both information and removal tools available for each version of SubSeven[39]. Dartsmouth University has developed a tool for finding and removing the Ramen worm, located at: http://www.ists.dartsmouth.edu/IRIA/knowledge_base/tools/ramenfind.htm.

**Registration Information**

This section includes five external addresses and their registration information from www.arin.net and www.lacnic.net (Latin American and Caribbean IP address Network Information Center). All of this information was gleaned from www.samspade.org, an excellent website for gathering information about sites. All of these IP addresses were also run through www.mynetwatchman.com and www.dshield.org/ipinfo.php to see if there were any complaints against these IPs from other sources. Only one IP turned up a positive hit.

200.163.61.175 has valid reverse DNS of 200-163-061-175.cbabm7004.e.brasiltelecom.net.br

**whois -h magic 200.163.61.175**
**Trying whois -h whois.arin.net 200.163.61.175**

OrgName: Latin American and Caribbean IP address Regional Registry
OrgID: LACNIC
Address: Potosi 1517
City: Montevideo
StateProv:
PostalCode: 11500
Country: UY

ReferralServer: whois://whois.lacnic.net

NetRange: 200.0.0.0 - 200.255.255.255
CIDR: 200.0.0.0/8
NetName: LACNIC-200
NetHandle: NET-200-0-0-0-1
Parent:
NetType: Allocated to LACNIC
NameServer: TINNIE.ARIN.NET
NameServer: NS.LACNIC.ORG
NameServer: NS.DNS.BR
NameServer: NS2.DNS.BR
Comment: This IP address range is under LACNIC responsibility for further
Comment: allocations to users in LACNIC region.
Comment: Please see http://www.lacnic.net/ for further details, or check the
Comment: WHOIS server located at whois.lacnic.net

RegDate: 2002-07-27
Updated: 2003-06-12

TechHandle: LACNIC-ARIN
TechName: LACNIC Hostmaster
TechPhone: (+55) 11 5509-3522
TechEmail: abuse@lacnic.net

OrgTechHandle: LACNIC-ARIN
OrgTechName: LACNIC Hostmaster
OrgTechPhone: (+55) 11 5509-3522
OrgTechEmail: abuse@lacnic.net

This IP address was selected because it was the top offending IP address from the "Possible trojan server activity" alert, which means it was possibly involved with either SubSeven or Ramen activity. The registration information from the LACNIC is not shown for brevity (the list of information is repetitive and almost two pages long), though it is registered to brasiltelecom.net.br. This IP address also registered an incident with www.mynetwatchman.com, though www.dshield.org/ipinfo.php didn't have any problems with this IP address.

200.96.13.157 has no reverse DNS configured.

**whois -h magic 200.96.13.157**
**Trying whois -h whois.arin.net 200.96.13.157**

OrgName: Latin American and Caribbean IP address Regional Registry
OrgID: LACNIC
Address: Potosi 1517
City: Montevideo
StateProv:
PostalCode: 11500
Country: UY

ReferralServer: whois://whois.lacnic.net

NetRange: 200.0.0.0 - 200.255.255.255
CIDR: 200.0.0.0/8
NetName: LACNIC-200
NetHandle: NET-200-0-0-0-1
Parent:
NetType: Allocated to LACNIC
NameServer: TINNIE.ARIN.NET
NameServer: NS.LACNIC.ORG
NameServer: NS.DNS.BR
NameServer: NS2.DNS.BR
Comment: This IP address range is under LACNIC responsibility for further
Comment: allocations to users in LACNIC region.
Comment: Please see http://www.lacnic.net/ for further details, or check the
Comment: WHOIS server located at whois.lacnic.net
RegDate: 2002-07-27
Updated: 2003-06-12

TechHandle: LACNIC-ARIN

TechName: LACNIC Hostmaster
TechPhone: (+55) 11 5509-3522
TechEmail: abuse@lacnic.net

OrgTechHandle: LACNIC-ARIN
OrgTechName: LACNIC Hostmaster
OrgTechPhone: (+55) 11 5509-3522
OrgTechEmail: abuse@lacnic.net

This IP address was selected because it was the top offending IP address from the "High port 65535 tcp – possible Red Worm - traffic" alert, which means it was possibly involved with Adore worm activity. The registration information from the LACNIC is not shown for brevity (the list of information is repetitive and almost two pages long), though it is registered to brasiltelecom.net.br.

68.100.94.160 has valid reverse DNS of ip68-100-94-160.dc.dc.cox.net

whois -h magic 68.100.94.160
Trying whois -h whois.arin.net 68.100.94.160
Cox Communications Inc. NVRDC-68-100-0-0 (NET-68-100-0-0-1)
68.100.0.0 - 68.100.255.255
Cox Communications Inc. COX-ATLANTA-2 (NET-68-96-0-0-1)
68.96.0.0 - 68.111.255.255

This IP address was selected because it was the top offending IP address from the "[UMBC NIDS] External MiMail alert" alert, which means it was possibly involved with MiMail activity.

68.2.113.48 has valid reverse DNS of ip68-2-113-48.ph.ph.cox.net

whois -h magic 68.2.113.48
Trying whois -h whois.arin.net 68.2.113.48
Cox Communications Inc. COX-ATLANTA (NET-68-0-0-0-1)
68.0.0.0 - 68.15.255.255
Cox Communications Inc. PHRDC-68-2-0-0 (NET-68-2-0-0-1)
68.2.0.0 - 68.3.255.255

This IP address was selected because it was the top offending IP address from the "Back Orifice" alert, which means it was possibly involved with Back Orifice activity. Both this IP address and the previous IP address are registered to the Cox Communications company, though each IP belongs to a different sub-company.

205.243.60.4 has valid reverse DNS of warpspeed.megalink.net

whois -h magic 205.243.60.4
Trying whois -h whois.arin.net 205.243.60.4
Sprint SPRINT-BLKF (NET-205-240-0-0-1)
205.240.0.0 - 205.247.255.255
The Phone Store SPRINT-CDF33C (NET-205-243-60-0-1)
205.243.60.0 - 205.243.60.255

This IP address was selected because it was the top offending IP address from the "Bugbear@MM virus in SMTP" alert, which means it was possibly involved with Bugbear activity.

**Other Defensive Recommendations**

The biggest step the university could take in ensuring the integrity of its network is to apply a default deny policy on its firewalls. A default deny policy states that everything that is not explicitly allowed, is denied. This will help stop many of the new worms, trojans, and viruses running around the Internet. For example, if a default deny policy was in place and NETBIOS ports were denied, then MSBlaster would most likely not have been much of a problem. This default deny policy needs to be applied for both ingress and egress traffic. Preventing the spread of malicious traffic is just as important as preventing it from coming into the network.

One of the best ways to determine whether a machine has been compromised or not, is by using a tool called Tripwire. Tripwire creates a hash of every selected file, rehashes those files at given intervals, and checks the new hashes against the saved "known good" hashes. If the hashes differ, then the file has been changed. This gives administrators an easy way of knowing if sensitive files have been modified. More information can be gleaned from the commercial Tripwire website at www.tripwire.com, or from the freeware version of Tripwire at www.tripwire.org. The commercial version of Tripwire supports both Win32 and *nix platforms, whereas the open source version only supports *nix platforms.

All the hosts listed in the compromised hosts section need to be thoroughly checked for compromise using the tools given to detect the programs thought to have compromised the machines. Even if nothing is found, it is recommended that a full scan be performed on those machines, at the very least, and, if possible, a complete re-load from a known good state be performed on those machines. A re-load from a known good state is the only way to ensure that a compromised machine has gotten rid of all traces of compromise. One of the best commercial scanners is Eeye's Retina scanner, and the best open source scanner is Nessus.

The Snort configuration file should be checked to ensure that the HTTP preprocessor is running. In most other students practicals there was at least one HTTP type alert, whereas none were witnessed during this five day period. Whether or not the preprocessor was running could make a big difference in how an analysis is done. For example, the NIMDA alert by itself does not necessarily mean that a machine was compromised, but a NIMDA alert with an IIS Unicode alert means that the machine was compromised[39]. In this analysis, no machine was thought to have been compromised by NIMDA because of the lack of IIS Unicode alerts. If this was because the HTTP preprocessor was not running, then there could be some NIMDA compromised machines on the universities network.

As was pointed out in a number of the top ten alerts already, a VPN solution should be looked at for the university if they really want to allow such dangerous traffic into their internal network. A VPN will still allow the connections,

but the connections will be secure. The university should also look into purchasing a stateful firewall, also mentioned in the top ten alerts section. If the university can afford it, a proxy firewall would be better as they have the capability to perform packet inspection, whereas a stateful firewall just keeps track of state.

In general, all the alerts should be looked at for efficiency. Many of the rules seem to encourage false positives and cannot possibly contribute to an easier workload for the universities security staff. There were even two alerts that could easily be combined into one: SUNRPC highport access and Sun RPC high port. Many of the alerts would be eliminated if the problem traffic was blocked at the firewall.

**Analysis Process**

The first step I took to accomplish my analysis was to fret for a while over it and put it off for about a week. Once I had decided that putting it off wouldn't finish it, I sat down and started to tackle this challenge. I think that most students feel a bit lost when first starting this portion of the assignment. I know I would have felt much more comfortable using binary data instead of text logs, especially since data parsing and scripting is not one of my strong points. One of the comments in GCIA Study and Planning Guide[34] is that students shouldn't attempt to reinvent the wheel, but should look to other student's practicals for starting points. I downloaded about 20 practicals and starting reading through them, looking at their formats and most importantly, looking at the scripts they wrote. I played around with many, many different types of scripts before settling on a couple that I thought would benefit my analysis the most. In order to turn my alerts and scans into .csv files, I used Tod Beardsley's csv.pl script[32]. I couldn't find any scripts that were able to correctly parse the OOS logs, so I ended up grepping information out of them. I also used Chris Calabrese's[51] script ideas as a basis for my own scripts, which is how I gathered all my information.

One thing I realized a bit late is that a database would help me more than all the scripts in the world. I failed miserably at a couple of attempts to write scripts to import the alert files into the MySQL database. I even tried a couple of scripts from other student's practicals, but none of them worked with me. I would recommend to other students to get the information into a database first, as the queries are much faster than unix shell scripts. One program that I think would have been very helpful if I had played with it before I started, and not when I was almost done, is snortalog, written by Jeremy Chartier.

I noticed a custom looking alert named [UMBC NIDS IRC Alert] and I knew what the last three words were, but UMBC was unfamiliar to me and looked like a university name. A search at google turned up the University of Maryland, Baltimore County, www.umbc.edu. A search at the InterNIC's whois website showed that umbc.edu had a few name servers registered, with the following IPs: 130.85.1.3-5. I next browsed to the American Registry for Internet Numbers (ARIN) and searched for 130.85.1.3, which turned up UMBC owning a class B network, 130.85.0.0/16.

**Works Cited**


1. Roesch, Marty. Online Webinar. “Real-time Network Awareness – Redefining the Intrusion Detection Industry.” 1 Aug. 2003. Sourcefure Inc. 22 Aug. 2003. <The link has been removed and is no longer available>.
2. Roesch, Marty. Online Webinar. “Top 5 Ways to Make Your IDS Better.” 1 Oct. 2003. SANS Webcast. 1 Nov. 2003. <http://www.sans.org/webcasts/show.php?webcastid=90419>.
3. Roesch, Marty. “Real-time Network Awareness – Redefining the Intrusion Detection Industry.” Jun 2003. Sourcefire, Inc. 16 Aug. 2003. <http://www.sourcefire.com/technology/whitepapers.htm>.
4. Dayioglu, Burak, Ozgit, Attila. “Use of Passive Network Mapping to Enhance Signature Quality of Misuse Network Intrusion Detection Systems.” Nov. 2001. 15 Aug. 2001. <http://www.dayioglu.net> (requires registration).
5. Roesch, Marty. “Re: IDS is dead, etc.” 4 Aug. 2003. focus-ids@securityfocus.com. 1 Oct. 2003. <http://lists.insecure.org/lists/focus-ids/2003/Aug/0007.html>.
6. Heilman, Marshall. “RNA Questions.” Email to rob.cahill@sourcefire.com. 12 Aug. 2003.
7. Messmer, Ellen. “Sourcefire ignites scanning effort.” 2 Jun. 2003. Network World. 1 Oct. 2003. <http://www.nwfusion.com/news/2003/0602sourcefire.html>.
8. Fisher, Dennis. “Sourcefire Tool Aims to Help Intrusion Detection.” 9 Jun. 2003. eWeek. 1 Oct. 2003. <http://www.eweek.com/print_article/0,3668,a=43095,00.asp>.
9. Bennet, Todd. “Re: IDS is dead, etc.” 6 Aug. 2003. Securityfocus. 1 Oct. 2003. <http://archives.neohapsis.com/archives/sf/ids/2003-q3/0117.html>.
10. “Cert Advisory CA-2001-19 “Code Red” Worm Exploiting Buffer Overflow in IIS Indexing Service DLL.” 17 Jan 2002. CERT/CC. 8 Oct 2003. <http://www.cert.org/advisories/CA-2001-19.html>.
11. Maiffret, Mark. “[ISN] Full analysis of the .ida “Code Red” worm.” Online posting. 18 Jul 2001. ISN Securityfocus. 10 Oct 2003. <http://cert.uni-stuttgart.de/archive/isn/2001/07/msg00055.html>.
12. Information Sciences Institute. “Internet Protocol.” Sep 1981. RFC-Editor. 13 Oct 2003. <ftp://ftp.rfc-editor.org/in-notes/rfc791.txt>.
13. Hping - http://www.hping.org/
14. Packet Excalibur - http://www.securitybugware.org/excalibur/
15. Fragroute - http://www.monkey.org/~dugsong/fragroute/
16. RFC Homepage - http://www.rfc-editor.org/
17. Gullapalli, Vijay. “Can IP MF and DF flags be set simultaneously.” 20 Sep 2002. Netfilter. 8 Oct 2003. <http://lists.netfilter.org/pipermail/netfilter-devel/2002-September/009374.html>.
18. Russel, Chris. “Re: new codered worm penetrates content-filtering.” 1 Oct 2002. Securityfocus. 12 Oct 2003. <http://www.derkeiler.com/Mailing-Lists/securityfocus/incidents/2002-01/0077.html>.
19. CAIDA Analysis of Code-Red - http://www.caida.org/analysis/security/code-red/.

20. Zalewski, Michal. “Strange Attractors and TCP/IP Sequence Number Analysis – One Year Later.” 2002. 14 Oct 2003. <http://lcamtuf.coredump.cx/newtcp/>.
21. “HTTP Extensions for Distributed Authoring – WEBDAV.” Feb 1999. RFC 2518. 15 Oct 2003. ftp://ftp.rfc-editor.org/in-notes/rfc2518.txt.
22. Whitehat translate: f packet dump - http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids305&view=research.
23. Daniel. “Translate: f summary, history, and thoughts.” 15 Aug 2000. Bugtraq archive. 15 Oct 2003. <http://www.securityfocus.com/archive/1/76387>.
24. Cooper, Russ. “FW: Translate: f summary, history, and thoughts.” 15 Aug 2000. Bugtraq archive. 15 Oct 2003. <http://www.securityfocus.com/archive/1/76400>.
25. IEEE - <http://standards.ieee.org/regauth/oui/index.shtml>.
26. “Cert Advisory CA-2002-03 Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP).” 12 Feb 2002. CERT/CC. 19 Oct 2003. <http://www.cert.org/advisories/CA-2002-03.html>.
27. Case, J., Fedor, M., Schoffstall, M., Davin, J. “A Simple Network Management Protocol (SNMP), RFC 1157.” May 1990. RFC. 19 Oct 2003. < http://www.faqs.org/rfcs/rfc1157.html>.
28. Novak, Judy. “Detects Analyzed 6/15/2000.” 15 Jun 2000. Global Incidents Analysis Center. 9 Nov 2003. <http://www.sans.org/y2k/061500.htm>.
29. Whitehats. “IDS177 “NETBIOS-NAME-QUERY.” 2001. Whitehats. 9 Nov 2003. <http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids177&view=event>.
30. Carnegie Mellon University. “Exploitation of Unprotected Windows Networking Shares.” 7 Apr 2000. Carnegie Mellon CERT. 9 Nov 2003. <www.cert.org/incident_notes/IN-2000-02.html>.
31. Wu, Marcus. “Intrusion Detection: New Tools and Existing Theory.” 23 Jan 2003. SANS GCIA. <http://www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf>.
32. Beardsley, Tod. “Intrusion Detection and Analysis: Theory, Techniques, and Tools.” 8 May 2002. SANS GCIA. <http://www.giac.org/practical/GCIA/Tod_Beardsley_GCIA.doc>.
33. Maher, James. “Intrusion Detection In Depth GCIA Practical Assignment.” 16 July 2003. SANS GCIA. <http://www.giac.org/practical/GCIA/James_Maher_GCIA.pdf>.
34. Holland, Jeff., French, Jamie., Tan, Koon Yaw. “GCIA Practical Study and Planning Guide 3.3.” SANS GCIA. 18 July 2003. <http://www.giac.org/gcia_study_guide_v33.pdf>.
35. Skoudis, Ed. “Counter Hack.” 2002. pgs 259-268.
36. One, Aleph. “Smashing the Stack for Fun and Profit.” 11 Aug 1996. (Phrack 49, volume seven, issue 49, file 14 of 16). 21 Nov 2003. <http://www.phrack.org/show.php?p=49&a=14>.
37. Macdonald, Terry. “Intrusion Detection and Analysis: An Investigation.” 21 Jun 2003. SANS GCIA. <http://www.giac.org/practical/GCIA/Terry_Macdonald_GCIA.pdf>.

38. Van Oosterom, Peter. “GCIA Practical v3.3”. 31 Mar 2003. SANS GCIA. 21 Nov 2003. <http://www.giac.org/practical/GCIA/Peter_Van_Oosterom.pdf>.
39. Randier, Sylvain. “GCIA Practical Assignment.” 31 Jan 2003. SANS GCIA. 21 Nov 2003. <http://www.giac.org/practical/GCIA/Sylvain_Randier_GCIA.pdf>.
40. Ricci, Mario. “GIAC Certified Intrusion Analyst.” 18 Jun 2002. SANS GCIA. 21 Nov 2003. <http://www.giac.org/practical/GCIA/Mario_Ricci_GCIA.pdf>.
41. Hudak, Tyler. “GIAC Practical Assignment v3.3.” 31 May 2003. SANS GCIA. 21 Nov 2003. <http://www.giac.org/practical/GCIA/Tyler_Hudak_GCIA.pdf>.
42. Dell, J. Anthony.” Adore Worm – Another Mutation.” 6 Apr 2001. SANS GSEC. 22 Nov 2003. <http://www.giac.org/practical/gsec/Anthony_Dell_GSEC.pdf>.
43. Hayan, Saro. “Intrusion Analysis.” 27 Jun 2003. SANS GCIA. 24 Nov 2003. <http://www.giac.org/practical/GCIA/Saro_Hayan_GCIA.pdf>.
44. Se, Hee. “GIAC Intrusion Detection In Depth.” 16 Feb 2002. 24 Nov 2003. SANS GCIA. <http://www.giac.org/practical/GCIAHee_So_GCIA.doc>.
45. Maslowski-Yerges, Al. “GIAC Certified Intrusion Analyst (GCIA).” 5 Jan 2002. SANS GCIA. 24 Nov 2003. <http://www.giac.org/practical/GCIA/Al_Maslowski-Yerges_GCIA.pdf>.
46. Wesemann, Daniel. “LOGS: GIAC GCIA Version 3.3 Practical.” 11 Jan 2003. Incidents.org. 24 Nov 2003. <http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00084.html>.
47. Williams, Alan. “SANS GCIA Practical ver 3.3.” 23 Jan 2003. SANS GCIA. 24 Nov 2003. <http://www.giac.org/practical/GCIA/Al_Williams_GCIA.pdf>.
48. “5. The RPC Portmapper.” 28 Nov 2003. Linux HOWTO. <http://www.linux-nis.org/nis-howto/HOWTO/portmapper.html>.
49. “Ramen.” 28 Nov 2003. G-Lock Software. <http://www.glocksoft.com/trojan_list/Ramen.htm>.
50. Sekhri, Sunil. “Practical Assignment for SANS Big Apple.” 4 Mar 2003. SANS GCIA. 28 Nov 2003. <www.giac.org/practicals/GCIA/Sunil_Sekhri_GCIA.pdf>.
51. Calabrese, Chris. “SANS/GIAC Intrusion Detection In Depth GCIA Practical.” Dec 2001. SANS GCIA. 30 Nov 2003. <http://www.giac.org/practical/Chris_Calabrese_GCIA.html>.