Global Information Assurance Certification Paper

GIAC Certified Intrusion Analyst (GCIA)
Practical Assignment
Version 4.0

Derek A. Buelna

November 16, 2004

Table of Contents

Part I – Executive Summary

The alerts analyzed in the packet log include a variety of malicious traffic. Some is worse then others. We detected a variety of scans for an open proxy and several attempts to connect to a Windows share. We noticed attempts to compromise (hack) and also to communicate with potentially compromised hosts inside your network. There's also quite a bit of peer to peer file sharing activity, which places your organization at risk. Some of your internal hosts may have illegal or pirated software as well as pornography. We recommend against the use of such programs and that this should be defined as part of your acceptable use policy.

There's one internal host (207.166.87.157) that may be compromised. At the very least, the user of this system is up to no good, attempting to hack into other systems on the Internet. We recommend that the computer is confiscated and that someone with experience in computer forensics analyzes it. If it turns out that one of your employees is in control of this computer and that it hasn't been compromised, this employee needs to be disciplined at the very least.

As a follow-up to this analysis, we recommend that the firewall rule set is analyzed in order to determine which of the attacks could have made it through the firewall. Ports such as 8080, 1080, 3128, 139, 445 and 515 need to be closed. If they aren't, further analysis of internal hosts is suggested. Of course, the only ports on the firewall that should be open are those that are required to be open. Lastly, we want to communicate the importance of patch management. We don't know whether or not your hosts are up to date or not but you should know. It's a critical part of maintaining the security of your organization and its assets. Please reference the Defensive Recommendations for more suggestions on how to increase security.

Part II – Detailed Analysis

1. Scenario

The file http://isc.sans.org/logs/Raw/2002.9.30 has been analyzed. This binary capture file was generated by an unknown version of Snort with an unknown rule set. It is understood that every packet in the file is the result of an alert and that IP addresses and checksums have been sanitized in the file. The "–k none" option will be used with Snort so it will ignore the bad checksums. It was noted that the date on each of the logged packets is actually 10/30/02 and that there is a total of 15021 packets.

Snort 2.2.0 with 2.2 rules from 08/10/04, Tcpdump 3.81, Libpcap 0.83 and Ethereal 0.10.7 will be used for this analysis. Perl and various UNIX utilities were also used to manipulate the data.

2. General Analysis

Introduction

This section focuses on the intrusion analysis results. Our interpretation of the network topology is covered, then we dive into a general analysis of the logged packets. This is followed by a detailed analysis of three detects, along with some important statistics about the network. We reference the analysis of previous GCIA students in order to back up some of our conclusions, then we cover our concerns about internal hosts. Lastly, we provide defensive recommendations for your organization. For more information about the analysis process, please reference Part III: Analysis Process.

Network Topology

As there's only two unique MAC addresses among the logged packets, we can tell the IDS is watching a segment with only two devices.

tcpdump –ner 2002.9.30 | perl –ane 'print $F[1], "\n";' | sort –u
00:00:0c:04:b2:33
00:03:e3:d9:26:c0

We suspect that these are a Cisco router and a Cisco PIX firewall as IEEE (http://standards.ieee.org/regauth/oui/index.shtml) shows that the 00-00-0C and 00-03-E3 prefixes are registered to Cisco. We assume the IDS is monitoring traffic via a span port on a Cisco switch, however it's possible that the packet logs were obtained through the use of a vampire tap.

The examination of the MAC to IP address associations allowed us determine the inside and outside of the network. There's only one source IP address (207.166.87.157) among the packets where the source MAC address is 00:00:0c:04:b2:33.

tcpdump –ner 2002.9.30 ether src 00:00:0c:04:b2:33 | perl –ane 'print $F[9], "\n";' | sort –u
207.166.87.157

On the other hand, there's 71 source IP addresses among the packets where the source MAC address is 00:03:e3:d9:26:c0.

tcpdump –ner 2002.9.30 ether src 00:03:e3:d9:26:c0 | perl –ane 'print $F[9], "\n";' | perl -i -n -a -F\\. -e 'print join(".", @F[0..3]), "\n"' | sort –u
12.31.192.98
12.5.225.66
128.167.120.13
[snip]

80.67.66.49
80.67.66.7
80.67.68.9

We show 2160 IP destination addresses among the packets with source MAC address 00:03:e3:d9:26:c0.

tcpdump –ner 2002.9.30 ether src 00:03:e3:d9:26:c0 | perl –ane ‘print $F[9], “\n”;’ | perl -i -n -a -F\\. -e 'print join(".", @F[0..3]), "\n"' | sort –u | wc –l
2160

In browsing through these addresses, we started to think that the inside network was 207.166.0.0/16. The following command confirmed that.

tcpdump –ner 2002.9.30 ether src 00:03:e3:d9:26:c0 | perl –ane ‘print $F[11], “\n”;’ | perl -i -n -a -F\\. -e 'print join(".", @F[0..3]), "\n"' | sort –u | grep –v 207.166
[no results]

Based on the data obtained above, we produced the following basic network diagram.

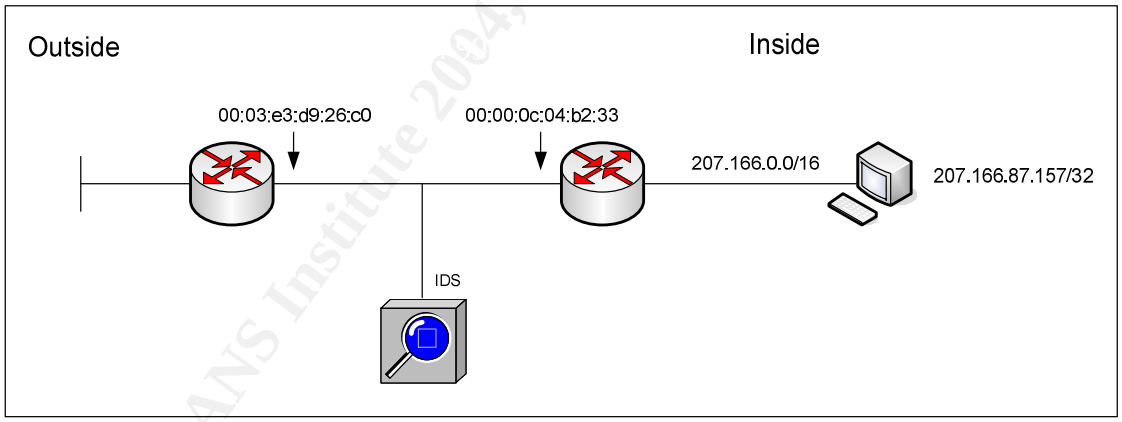


Figure 1

While the logged packets only list one source IP address on the inside, we’re certain several other hosts exist, based on the destination addresses and behavior of the packets coming from the outside.

Alert Analysis

Although it is understood that each of the 15021 packets in the 2002.9.30 file originally created a Snort alert and were thus logged, we were unable to obtain

15021 alerts when we ran Snort on the file. Depending on our configuration, we obtained between 500 and 2842 alerts. In order to solve this problem we analyzed the remaining 12179 packets manually. Breaking out the alerts by source IP and comparing the original alerts with the alerts that we received when running Snort against the packet log allowed us to begin doing so. Mapping out destination IP addresses, ports and in some cases source ports, enabled us to put together some Network Statistics, which is included below in section 4. In some cases (where our Snort run actually detected alerts) as with source IP 207.166.57.187, there were discrepancies with the number of alerts generated during our Snort run and those that were previously generated. This host originally generated 3625 alerts but only generated 2664 alerts when we ran Snort against the packet log. Even worse, 918 of the 2664 alerts that we received were redundant. In order to make sense of this we filtered out the packets from this source IP, broke it out by protocol and then by destination. This allowed us to determine which packets were not firing an alert for us. Additionally, some of the logged packets created multiple alerts when we ran Snort against the packet log. For example, 148.64.2.114 created 18 alerts when we ran Snort against the packet log but it originally generated 9 alerts.

The following link graph highlights the alerts that we detected in the packet log.

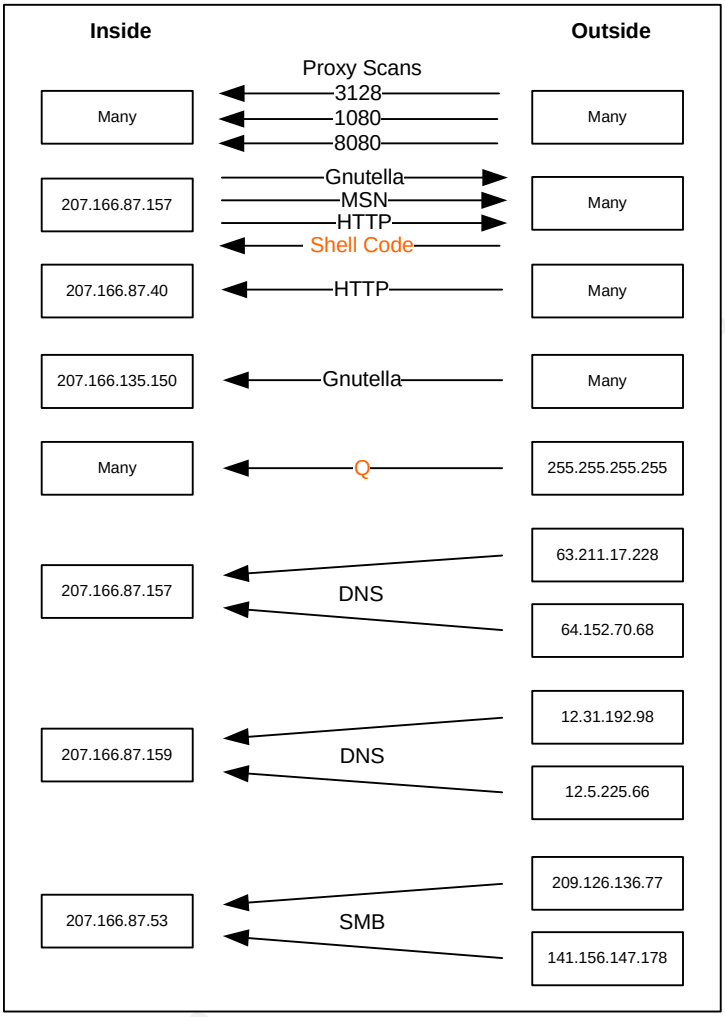


Figure 2

Although we consider it to be one of the least malicious attackers, the top talker (accounting for the most alerts) was 24.90.122.137. The host originally generated 10850 alerts while none of these were detected by our Snort run. The host sent 5425 packets to TCP destination port 8080 and another 5425 packets to TCP destination port 3128. We believe that these alerts may have been generated by a Snort rule like 1:620 that wasn't present in our rule set. The following rule would have picked up the traffic destined for TCP port 8080.

alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SCAN Proxy Port 8080 attempt"; flags:S,12; flow:stateless; classtype:attempted-recon; sid:620; rev:10;)

The alerts appear to be a scan for an open proxy. The scan started at 207.166.38.38 and ended at 207.166.45.246, covering close to a /21 range. Some hosts are only probed for each port once, while other are probed as many as six times per port.

Other outside hosts were found scanning for various proxy ports. TCP Destination port 1080 was scanned for by three outside hosts. 216.77.216.150 scanned 89 inside IP's, 216.77.219.225 scanned 51 inside IP's and 216.77.216.104 scanned 14 inside IP's. We noticed that the three outside hosts only managed to scan a total of 148 inside IP's although they sent out 244 packets. It's assumed that the 244 packets are coming from the same attacker.

Two more outside hosts were found to be scanning for an open proxy. 172.184.170.160 scanned 9 inside IP's for TCP destination port 3128 while 65.169.47.29 scanned 3 hosts for TCP destination port 3128 and also scanned 3 hosts for TCP destination port 8080.

The scans for open proxies are some of the least malicious attacks in the packet log. However, confirmation that these ports are blocked on the firewall should be obtained. If this is the case then the signatures firing these alerts could be disabled. Alternately, the ports could be blocked on the perimeter router and the IDS signatures could be left in place. Although we understand that in some cases it's preferable to block traffic on the firewall as it can be easier to manage a firewall rule set versus a router ACL, in this situation the IDS could provide router ACL validation. If these signature fire in this case, the router ACL would have been misconfigured or disabled.

207.166.87.157, an inside host, generated the second highest number of alerts. (3625) This host also generated the most different types of alerts. 2683 HTTP-related, 918 Gnutella and 24 MSN Messenger alerts were generated. Our Snort run detected 11 different types of HTTP-related alerts, some of which have no known false positives. In addition, 30 outside hosts generated alerts on what appear to be responses to traffic originating from 207.166.87.157. Here's what our Snort run of the packet log identified, although it really only picked up 1746 out of 3625 alerts due to the 918 duplicate GNUtella alerts.

Alert Type	# of Alerts
[119:2:1] (http_inspect) DOUBLE DECODING ATTACK	15
[1:882:5] WEB-CGI calendar access	1
[1:895:7] WEB-CGI redirect access	2
[1:972:8] WEB-IIS %2E-asp access	16
[1:1113:5] WEB-MISC http directory traversal	1
[1:1653:5] WEB-CGI campus access	4
[119:13:1] (http_inspect) NON-RFC HTTP DELIMITER	8
[1:540:11] CHAT MSN message	24
[119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING	500
[119:7:1] (http_inspect) IIS UNICODE CODEPOINT ENCODING	76
[119:15:1] (http_inspect) OVERSIZE REQUEST-URI DIRECTORY	116
[1:556:5] P2P Outbound GNUtella client request	918
[1:1432:6] P2P GNUtella client request	918
[119:12:1] (http_inspect) APACHE WHITESPACE (TAB)	65
Total	**2664**

Figure 3

While the Gnutella and MSN alerts are not necessarily malicious, some of the HTTP-related alerts have no known false positives. Additionally, there are three Snort alerts related to shell code (1:648:7. 1:649:8 and 1:1390:5) destined for 207.166.87.157, which is really bad. We'll take a closer look at one of the shell code alerts below. We have reason to believe that 207.166.87.157 has been compromised or at least the user of the system is up to no good. Based on this information, other inside hosts could have been compromised.

Note that the Snort signatures related to GNUtella [1:556:5] and [1:1432:6] are redundant as there were only 918 packets in the packet log originating from 207.166.87.157 that were not HTTP or MSN Messenger traffic.

There are two different types of GNUtella packets that generated alerts in the packet log. The first type includes 918 alerts originating from 207.166.87.157. These packets did not use a consistent TCP destination port. The second type includes 72 alerts generated from 9 outside hosts targeting inside host 207.166.135.150 on TCP destination port 9511. We also noticed an IP fragment from 217.36.23.34, destined for 207.166.135.150. The fragment was likely an attempt to get through the firewall. Our Snort run didn't detect the fragment but we assume that a custom rule was examining bytes 6 and 7 of the IP header.

Note that we recommend against the use of GNUtella inside your organization. Confidential information could have been leaked to the Internet and there may be files on inside hosts that you don't want. There could be pornography, illegally copied software and so on. You should consider prohibiting the use of peer to peer file sharing protocols as part of your acceptable use policy if it doesn't already.

The 24 [1:540:11] CHAT MSN message alerts originating from 207.166.87.157 were destined for two external hosts, 21 packets to 64.4.12.192 and 3 packets to 64.4.12.172. This chat was sent in the clear and there appears to be 2 separate conversations, although we're only seeing bits and pieces of the communications. The 21 packets destined for 64.4.12.192 appears to be a normal conversation and occurred between 01:46 and 01:58 while the second conversation, consisting of the 3 packets destined for 64.4.12.172 is a little more questionable. The latter 3 packets timestamps were between 06:02 and 06:08. We think that if an audit signature like 1:540 had been previously enabled, we should have seen the entire conversations, so we're not clear why only some of the packets generated alerts.

Inside host 207.166.87.47 appears to be a Web server, as there are alerts being fired on traffic destined for TCP port 80 to that host, from 13 different sources. We do not have any alerts originating from this host but we believe that the host exists. We noticed 4 outside hosts (see below) that generated the same types of attacks against 207.166.87.47. It's possible that these are the same attacker.

Source IP	[1:937:9]	[1:962:12]	[1:990:8]	[1:1288:7]	[119:13:1]
61.218.105.219	3	3	2	3	1
61.222.191.98	1	1	1	1	1
65.193.39.163	1	1	1	1	
61.170.244.77	1	1	1	1	

Figure 4

These signatures indicate attacks on Microsoft FrontPage. 207.166.87.47 needs to be examined to see if Microsoft FrontPage is installed and if so that it has the appropriate patches applied as per Microsoft Security Bulletin MS02-053. The FrontPage server configuration should be examined to validate that it is not password-less as Snort signature [1:990:8] detected an attack against this configuration. Snort signature 1:1288:7 detected reconnaissance of the directory structure as well.

In addition to the external attacks, it's likely that 207.166.87.47 is on the same subnet as 207.166.87.157, which may be compromised. This obviously isn't good. The front-end Web server should be on a DMZ and should have access controls in place for communication with other devices. Reference our Defensive Recommendations for more information.

18 packets destined for 18 different hosts in the 207.166.0.0/16 subnet fired the Snort alert [1:184:6] BACKDOOR Q access. These packets used TCP destination port 515 and source port 31337, which spells "eleet" in cracker speak. The source IP address on these packets was 255.255.255.255, which means this is a version of Q prior to 2.0, according to Les Gordon's paper What is the Q Trojan? It appears that the objective of this traffic is to send commands to hosts that have been compromised. We need to determine if the firewall allows TCP destination port 515. If so, further investigation is necessary.

4 external hosts sent a total of 7 packets to 207.166.87.157 and 207.166.87.159 on TCP port 53, normally used for DNS zone transfers. 3 of the 7 packets are directed at 207.166.87.159 and these are the only alerts destined for this host in the entire packet log. It's possible that these two internal boxes are DNS servers. The attack packets appear to be crafted, possibly attempting to bypass a simple firewall, as the source ports are set to 80 and 53.

Src IP	Src Port	Dst IP	Dst Port	Flags	TTL	IP ID
12.31.192.98	80	207.166.87.159	53	ACK	47	48266
12.5.225.66	80	207.166.87.159	53	ACK	47	48256
12.5.225.66	53	207.166.87.159	53	SYN	47	48258
63.211.17.228	80	207.166.87.157	53	ACK	54	22649
63.211.17.228	53	207.166.87.157	53	ACK	54	22650
64.152.70.68	80	207.166.87.157	53	ACK	54	21123
64.152.70.68	53	207.166.87.157	53	ACK	54	21124

The 4 external hosts exhibit very similar behavior, making us curious as to why. 12.31.192.98 and 12.5.225.66 targeted 207.166.87.159 while 63.211.17.228 and 64.152.70.68 targeted 207.166.87.157. The packets are nearly identical except that one of the packets from 12.5.225.66 had the SYN flag set. It appears that the first 2 sources are the same, while the latter two may also be the same. It could even be the same attacker across all of these alerts. We suspect that the attacker is trying to elicit a response from the internal hosts, attempting to circumvent access controls.

The packet log included 2 packets from 2 external sources, targeted at 207.166.87.53. These are the only packets in the packet log directed at this inside host. The target port is TCP 139. The packets show an attempt to connect to a Windows share \\b2b\c. It's odd that two different sources are trying to connect to the same share. Confirmation that NetBIOS traffic is blocked at the firewall should be obtained. We looked into why our Snort run did not detect these alerts and found a detect post by Daniel Wesemann to the intrusions mailing list. It appears that a Snort rule that checks for access to the Windows default shares may have been modified to look for any access to the c drive. According to Daniel, the following rule:

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB C$ access"; flow: to_server,established; content: "|5c|C$|00 41 3a 00|";reference:arachnids,339; c lasstype:attempted-recon; sid:533; rev:5;)

could have been modified to look like this:

alert tcp $EXTERNAL_NET any -> $HOME_NET 139 (msg:"NETBIOS SMB C access"; flow:to_server,established; content: "|5c|C|00 41 3a 00|";reference:arachnids,339; classtype:attempted-recon; sid:100533; rev:6;)

in order to alert on access attempts to a share called c, which makes sense.

In summary, it's clear that there's a variety of malicious code traversing the IDS. Some of it is malicious but much of it isn't terribly concerning. The proxy scans aren't a big deal. Crafted packets are concerning and deliberate attempt to compromise or issue commands to compromised hosts is even worse. It's also clear that the IDS is using a highly customized rule set. While it is readily apparent why most of the alerts originally generated did not generate an alert during our Snort run, we've had to make a few educated guesses in some cases.

3. Detect Analysis

This section provides an in-depth analysis of what we have determined to be the 3 most critical detects. This includes examining Snort alerts [1:184:6] BACKDOOR Q access, [1:649:8] SHELLCODE x86 setgid 0 and [1:648:7] SHELLCODE x86 NOOP. We felt that Q and the 3 shell code alerts were the most severe as these are deliberate attempts to compromise or issue commands to a compromised host. It's possible that the shell code alerts are false positives but further analysis is in order due to their criticality.

Detect 1: [1:184:6] BACKDOOR Q access (18 alerts)

Description
The is a Trojan that affects all UNIX operating systems. It offers the attacker remote access to the victim host. The Trojan is controlled by sending packets to the host that include commands to be run as root. The packets can be ICMP, TCP or UDP. In this case the 18 packets that caused alerts were destined for TCP port 515.

Reason Selected
This detect was selected due to it's criticality. These alerts are in reference to sending commands to already compromised internal host. While there are some attempts to compromise FrontPage on a server in the packet log and various crafted packets, we consider this detect worthy of further examination.

Detect Generation
The following rule detected this traffic.

alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; flow:stateless; dsize:>1; flags:A+; reference:arachnids,203; classtype:misc-activity; sid:184; rev:7;)

The signature is looking for TCP packets coming from 255.255.255.x using any source port, going to $HOME_NET on any port. The datagram size needs to be larger then 1 byte.

The following packet matches this rule. Note that we haven't included the Ethernet header as it isn't relevant.

```
01:43:06.876507 00:03:e3:d9:26:c0 > 00:00:0c:04:b2:33, ethertype IPv4,
length 60: IP (tos 0x0, ttl  15, id 0, offset 0, flags [none], length:
43, bad cksum ae17 (->63cd)!) 255.255.255.255.31337 > 207.166.120.90.515:
R [bad tcp cksum 633f (->18f5)!] 0:3(3) ack 0 win 0 [RST cko]
0x0000   4500 002b 0000 0000 0f06 ae17 ffff ffff        E..+............
0x0010   cfa6 785a 7a69 0203 0000 0000 0000 0000        ..xZzi..........
0x0020   5014 0000 633f 0000 636b 6f00 0000             P...c?..cko...
```

The packets match as the source IP is 255.255.255.255, the protocol is TCP and the payloads are 3 bytes. All of the datagrams are 43 bytes long. The IP header takes up 20 bytes, the TCP header takes up another 20 bytes, leaving 3 bytes for the payload.

Probability Source Address Spoofed
It is certain that the source address was spoofed as the source address is invalid. The source port appears to be crafted as well. 31337 is a well known cracker port that spells “eleet”.

Attack Mechanism
The attacker is trying to communicate with hosts that have been compromised. The article What is the Q Trojan? by Les Gordon describes the different versions of Q, indicating that the version used was pre 2.0, as using 255.255.255.255 as a source address is no longer supported. Go figure.

Correlations
Pete Storm’s GCIA practical includes analysis of Q. Pete analyzed the log file 2002.9.26, which is four days younger than the packet log we analyzed. Andrew Wagoner’s GCIA practical also analyzed Q, which was detected in the 2002.10.14 packet log, dated a few weeks after the log we analyzed. White Hats has the alert listed as IDS203 "TROJAN-ACTIVE-Q-TCP" while the CVE includes a reference to CAN-1999-0660.

Evidence of Active Targeting
It is hard to say if the attacker was trying to target specific hosts. There was a total of 18 alerts and each one includes a unique destination in the 207.166.0.0/16 range. We haven’t seen other alerts destined for these 18 inside host. It’s possible that these are random destinations.

Severity = 7
Severity is defined by the following equation. (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality = 4
We have no information on the internal hosts targeted by this attack and as such have assigned a 4 to the criticality of this detect. As it’s possible that one of the targeted boxes are critical, we’d prefer to a assign a higher value versus a lower value. Once further information about the inside hosts is obtained, this value may be adjusted.

Lethality = 5
If the attack succeeded, the host would likely execute commands on behalf of the attacker. This is about as bad as it can get so the lethality has been assigned a 5.

System Countermeasures = 1
There is no information available regarding system countermeasures on the hosts targeted by this attack. Therefore, we've assigned a 1 to the system countermeasures. If we had to guess, we would assume that no host-based firewall was present on the inside hosts, as we believe that host-based firewalls were not widespread in 2002. Given the nature of this attack, the targeted host should have already been compromised. Assuming that this was the case, the system countermeasures would have already been breached. It's likely that if a box gets owned, the attacker would configure the host-based firewall to allow the backdoor traffic, or disable it. Further information regarding patch management should be obtained in order to increase the accuracy of the value assigned here.

Network Countermeasures = 1
We know the perimeter router is letting in just about everything but we have limited visibility into the rule set of the inside router/firewall. We have to assign a 1 to network countermeasures until we can obtain further information about the rule set. This is one of the most critical things that needs to be determined. Many of the detects in the packet log, including Q, should have been blocked at the firewall.

Detect 2: [1:649:8] SHELLCODE x86 setgid 0 (1 alert)

Description
This alert detected an attempt to change the privileges of a current running process to having the privileges of root. One packet originating from 63.250.205.44 on port 1755, destined for 207.166.87.157 on port 62218 triggered the alert. IANA has TCP Port 1755 listed as ms-streaming.

Reason Selected
Even though it's likely that this alert is a false positive, it's severity warrants some attention. There's nothing worse then having an outsider obtain root access on an inside box.

Detect Generation
The following rule caused the packet to fire an alert.

alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 setgid 0"; content:"|B0 B5 CD 80|"; reference:arachnids,284; classtype:system-call-detect; sid:649; rev:8;)

It is unknown what $SHELLCODE_PORTS was defined as on the IDS generating the traffic. The default is to exclude port 80, as including it would apparently provide too many false positives. Basically, the signature is looking for "0xB0 0xB5 0xCD 0x80" to appear in the payload of the packet. It's possible that this could appear in many packets that aren't really trying to execute a setgid 0 attack. This

packet that generated this alert does have the “0xB0 0xB5 0xCD 0x80” but it looks like a binary. It’s very large as well, 1500 bytes. We wouldn’t expect it to be this large if it was an actual attack.

Probability Source Address Spoofed
It is unlikely that the source address was spoofed. The attacker would need to execute some form of buffer overflow in order to execute the attack and would likely need responses from the target in order to do so. As we believe that this is a false positive, it’s even more unlikely that the source address was spoofed.

Attack Mechanism
The getgid 0 attack is something that is used to target a process that has an effective gid of 0, as part of a buffer overflow attack. For example, Apache starts as root but then it changes privileges and runs with minimal privileges. It does however have the ability to become root if needed. The getgid 0 exploit injects code into process in order to increase the privileges of the process and then some other code will generally spawn a shell, which is very complicated to do as you need to bind t.

Correlations
Rich Helton posted a setgid 0 detect to the intrusions mailing list on 01/26/04. He found evidence of this attack from the 2002.8.30 packet log, dated one month earlier then the packet log we analyzed. The GCIA Practical submitted by James Affeld included analysis of the setgid 0 detect, from his analysis of alert logs dated 04/07/04 through 04/12/11.

Evidence of Active Targeting
This detect only includes one alert originating from one source, destined for one inside host. This could tell you that the attack wasn’t random in nature and that the attacker targeted this specific inside host. However, as we believe this is a false positive, we don’t believe that there is evidence of active targeting.

Severity = 8
Severity is defined by the following equation. (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality = 5
It is possible that the internal host 207.166.87.157 is a DNS server. Therefore we have assigned criticality a value of 5. Once further information about this inside host is obtained, this value may be adjusted.

Lethality = 5
Privilege escalation of a process to root is very lethal. The attacker may be able to spawn a shell, thus owning the box. Therefore, lethality has been set to 5.

System Countermeasures = 1

We don't know if the target is running a host-based firewall but we do know that the host is generating a considerable amount of malicious traffic and believe that the system is either compromised or that the user is up to no good. Based on this, we have assigned a value of 1 to system countermeasures.

Network Countermeasures = 1
Although the packet that generated this alert came from the outside, the session likely originated from the inside. Assuming that users are allowed to make outbound connections to TCP port 1755, there's not much that your average firewall could do against an attack like this. We have to assign a 1 to network countermeasures.

Detect 3: [1:648:7] SHELLCODE x86 NOOP (1 alert)

Description
A NOP helps an attacker execute code as part of a buffer overflow attack. When an application or service does not perform proper boundary checking on its variables, it's possible to submit more data then the application was expecting and cause the processor to start executing code that was provided by the attacker. In order to do this the attacker needs to tell the processor where to execute the code. The NOP tells the processor to skip over to the next code segment. Multiple NOPs allow attackers to direct the processor to execute code provided by the attacker.

Only one alert was generated due to a packet from 203.66.215.25 on port 1352, destined for 207.166.87.157 on port 63648.

Reason Selected
Although this shellcode alert also appears to be a false positive, attempts to overflow buffers should be considered severe and thoroughly analyzed. The packet log includes some attempts to compromise a FrontPage server and some crafted packets that are worthy of analysis but if the Q and the shell code attacks were actually real, we felt that these were the most severe.

Detect Generation
The following rule detected the alert:

alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any (msg:"SHELLCODE x86 NOOP"; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; depth:128; reference:arachnids,181; classtype:shellcode-detect; sid:648; rev:7;)

As with detect 2, the signature is looking for non port 80 traffic with a certain payload. In this case we're looking for 14 of 0x90 in the payload, essentially a NOP sled. The packet did include a bunch of 0x90's but it doesn't look like a sled as there are 0x51's mixed in with the 0x90's. This packet is very large, 1500 bytes, which is unexpected with this type of attack.

Probability Source Address Spoofed
It's unlikely that the source address was spoofed. The attacker would need to execute some form of buffer overflow in order to execute this attack and would likely need responses from the target in order to do so. As with detect 2, we believe that this is a false positive, so it's even more unlikely that the source address was spoofed.

Attack Mechanism
A NOP can appear as part of a buffer overflow, sliding the processor into the right place, in order to execute code provided by the attacker. This could provide access to a root shell, which is of course undesirable to the victim.

Correlations
We referenced the classic paper on buffer overflows, Smashing the Stack for Fun and Profit in order to gain some insight into how these attacks are executed and found it to be very complicated. Knowledge of Assembly, how the x86 processor works and how the target service works is essential, otherwise an attacker may end up just crashing the service.

Evidence of Active Targeting
This detect only includes one alert originating from one source, destined for one inside host. This could tell you that the attack wasn't random in nature and that the attacker targeted this specific inside host. However, as we believe this is a false positive, we don't believe that there is evidence of active targeting.

Severity = 7
Severity is defined by the following equation. (criticality + lethality) - (system countermeasures + network countermeasures)

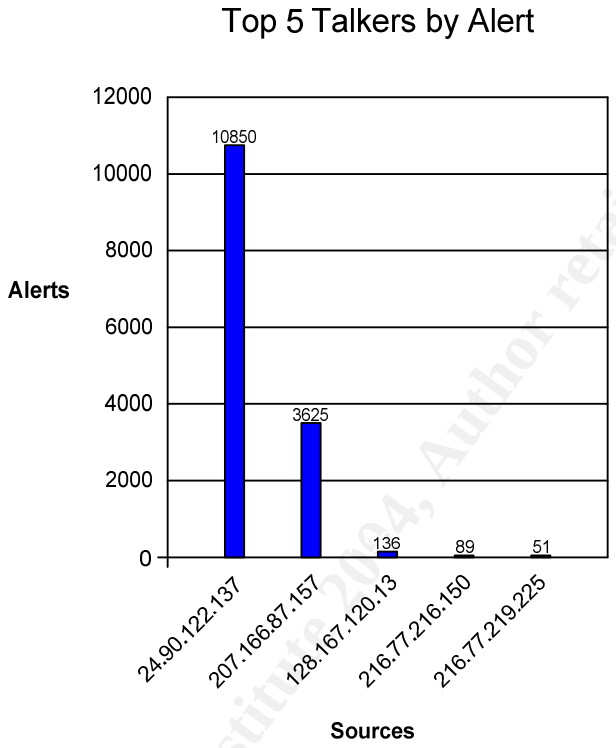
Criticality = 5
It is possible that the internal host 207.166.87.157 is a DNS server. Therefore we have assigned criticality a value of 5. Once further information about this inside host is obtained, this value may be adjusted.

Lethality = 4
If the attack was successful and didn't just crash the service, it's likely that a root shell could be obtained. Due to this, the lethality of this attack has been set to 4. We've assigned this detect a slightly lower value then the previous detect as it is possible that this attack would not be lethal.

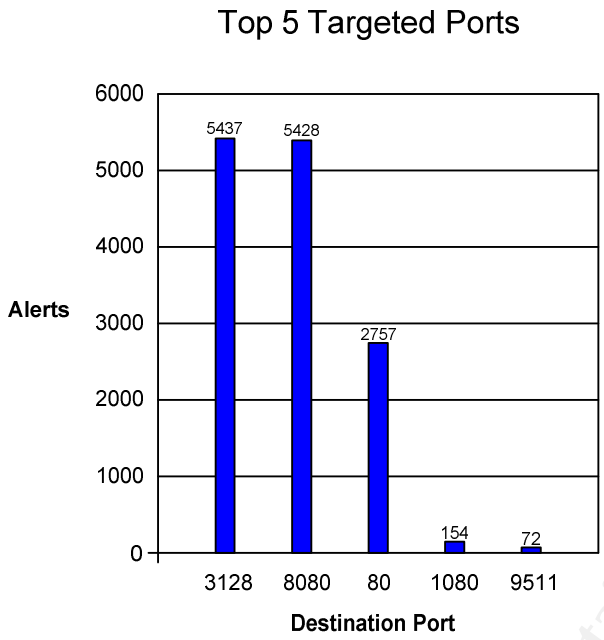
System Countermeasures = 1
We don't know if the target is running a host-based firewall but we do know that the host is generating a considerable amount of malicious traffic and believe that the system is either compromised or that the user is up to no good. Based on this, we have assigned a value of 1 to system countermeasures.

Network Countermeasures = 1
We know the perimeter router is letting in just about everything but we have limited visibility into the rule set of the inside router/firewall. We have to assign a 1 to network countermeasures until we can obtain further information about the rule set. This is one of the most critical things that needs to be determined.

4. Network Statistics



The top talker in the log (based on the total number of alerts generated) was an external host scanning for an open proxy. Internal host 207.166.87.157 was the second noisiest and also generated the most different types of alerts. The other hosts in the log didn't generate anywhere near this many events, many only generated only one or two.



The two top ports were proxy ports with web traffic coming in third. Another proxy port came in fourth while GNUtella came in fifth.

We decided to select the following three IP addresses as the most suspicious sources.

Source IP	# of Alerts	[1:937:9]	[1:962:12]	[1:990:8]	[1:1288:7]	[119:13:1]
61.218.105.219	12	3	3	2	3	1
61.222.191.98	5	1	1	1	1	1
61.170.244.77	4	1	1	1	1	

These boxes appear to be working together in an attempt to compromise Microsoft FrontPage which may or may not exist on 207.166.87.157. We selected these hosts as being the most suspicious as we believe the shell code attacks are false positives and there's no point looking into source IP 255.255.255.255. Registration information is not provided herein as we understand that the IP addresses in the packet log have been sanitized.

5. Correlations

We found the following three papers to be very enlightening. Pete's paper was referenced time and again because it was so informative. Pete's analysis of Q and references to other papers about Q was crucial to our understanding of the backdoor. We found a reference in Pete's paper to Les Gordon's paper on how Q works, which was also very helpful. We needed help in understanding how the setgid 0 exploit works and read several articles about buffer overflows including what appear to be a classic paper, Smashing the Stack for Fun and Profit.

Pete Storm. “GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 3.3.”
November 15, 2003. URL:
http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf

Les Gordon. “What is the Q Trojan?” Unknown Date.
URL: http://www.sans.org/resources/idfaq/qtrojan.php

Aleph One. “Smashing the Stack for Fun and Profit.” Unknown Date.
URL: http://www.insecure.org/stf/smashstack.txt

6. Internal Insights

We believe that 207.166.87.40 is a web server, while 207.166.87.157 and 207.166.87.159 may be DNS servers. We aren’t certain why 207.166.87.53 is being targeted as being a Windows server. It’s possible that the 207.166.87.0/24 net is a subnet with servers on it but it could also be the only active net inside.

7. Defensive Recommendations

The firewall rule set needs to be examined in order to determine if the attacks detected in the packet log were able to reach the inside network. In fact, we suggest confirming that the firewall is blocking the port that it is configured to block by doing some scanning and sniffing. As always, the firewall should only allow the necessary ports and block everything else. Further analysis is required if some of the attacks actually made it inside. We suspect that the FrontPage attacks made it to 207.166.87.157 so this host probably needs attention.

We suggest that the rules on the perimeter router are reviewed. We suggest blocking 255.255.255.255 for example. Additionally, only allow traffic out that originates from your internal network. This will prevent any of your hosts from spoofing other hosts. We also recommend internal and external penetration testing against the mission critical applications and infrastructure components. This will likely provide some good feedback for your IT staff.

We also suggest that you validate that all of your hosts are up to date on their patches, including applications and the OS. We don’t have any reason to believe that you aren’t up to date but we want to stress the importance of this. It’s an important part of your security architecture that must be managed effectively as it’s a never ending process.

Part III - Analysis Process

As part of the analysis, Linux and Windows platforms were used. Linux was used as the primary machine to analyze the packet log. We used Tcpdump, Ethereal and Snort on Linux. Windows was only used for the purposes of running Microsoft Office including Word, Excel and Visio.

We found the discrepancies between the original packet log and the log created during our Snort run was the biggest hurdle to overcome. Many alerts were not detected during our Snort run so we needed to figure out why. Also, some of the alerts that our Snort run generated were redundant. We found that making a spreadsheet of the source IP addresses found in the packet log and filling in details such as source port, destination port, number of alerts originally detected, number of alerts currently being detected and so on was the only way to make sense of it all. Once we had done that, things started falling into place. We found a huge number of discrepancies regarding source IP 207.166.87.157 and were forced to break out the alerts for this source by protocol and also by destination. This gave us more insight into what the host was doing.

Note that we tried several things that would allow us to compare the original packet log with the packet log that our Snort run generated but this didn't work too good. We managed to get both files to be in the same output format and did a sort and a diff on them, but the output wasn't what we expected, including discrepancies. It appeared as if some of the packets were duplicates, even though we thought the timestamps on the packets would have been unique. We also tried using Perl to put one of the files in a hash, then read in the other file line by line, checking each line to see if it was in the hash and if it wasn't, writing that line to a file. This didn't work too good either as we received some unexpected results. Suffice it to say that the spreadsheet based on the source IP worked best for us. Beyond the issues with 207.166.87.157, other duplicate alerts were readily apparent.

Considerable effort was put into determining why alerts were originally created but weren't being created during our Snort run. By analyzing the sources, destinations, ports and portions of the headers, we were able to determine why many of the alerts originally fired. We think the IDS that generated the original packet was operating with a customized rule set and that some of the signatures that fired no longer exist. More research was done in order to determine who the malicious sources were and what the most severe alerts were. We felt that Q and the shell code attacks were the worst. The more we dug into this though, it became apparent that the shell code attacks were very likely to be false positives. We decided to stick with these though as the only other really malicious activity was attempts to compromise a FrontPage server and then there were some crafted packets that appeared to be reconnaissance activity.

Appendix

1. References

Andrew Wagoner. “GIAC Certified Intrusion Analysts (GCIA) v. 3.4”
May 17, 2004. URL: http://www.giac.org/practical/GCIA/Andrew_J_Wagoner_GCIA.pdf

Daniel Wesemann. “LOGS: GIAC GCIA Version 3.3 Practical (Daniel Wesemann).” Jan. 11, 2003.
URL: http://www.dshield.org/pipermail/intrusions/2003-January/006363.php

Les Gordon. “What is the Q Trojan?” Unknown Date.
URL: http://www.sans.org/resources/idfaq/qtrojan.php

Microsoft. “Microsoft Security Bulletin MS02-053.” Sept. 25, 2002.
URL: http://www.microsoft.com/technet/security/bulletin/MS02-053.mspx

Pete Storm. “GIAC Certified Intrusion Analyst (GCIA) Practical Assignment Version 3.3.”
November 15, 2003. URL: http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf

Rich Helton. “LOGS: GIAC GCIA Version 3.4 Practical Detect 2 Rich Helton.” Jan. 26, 2004. URL:
http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00137.html

James Affeld. “GIAC Certified Intrusion Analyst – GCIA Practical Assignment Version 3.5.”
June 3, 2004. URL: http://www.giac.org/practical/GCIA/James_Affeld_GCIA.pdf

Aleph One. “Smashing the Stack for Fun and Profit.” Unknown Date.
URL: http://www.insecure.org/stf/smashstack.txt

2. Snort Alert Information

[1:184:6] BACKDOOR Q access
This Trojan affects UNIX operating systems. The Trojan is controlled by sending raw packets (TCP/UDP/ICMP) to the victim host containing commands to be run as root.
255.255.255.255 -> many (18)
Total (18)

[1:504:6] MISC source port 53 to <1024
Traffic from TCP port 53 is used by DNS servers for zone transfers. Normal DNS traffic uses the UDP protocol. An attacker could use a TCP source port of 53 to pass through a poorly configured firewall. DNS traffic from port 53 using either UDP or TCP should be to a port above 1023. Ports 1023 and below are privileged.
12.5.225.66 -> 207.166.87.159 (1)
Total (1)

[1:523:5] BAD-TRAFFIC ip reserved bit set
Under normal circumstances IP packets do not use the reserved bit. This may be an indicator of the use of the reserved bit by a malicious user to instigate covert channel communications, an indicator of unauthorized network use, reconnaisance activity or system compromise. These rules may also generate an event due to improperly configured network devices.
217.36.23.34 -> 207.166.135.150 (1)
Total (1)

[1:540:11] CHAT MSN message

Instant Messaging (IM) and other chat related client software can allow users to transfer files directly between hosts. This can allow malicious users to circumvent the protection offered by a network firewall. Vulnerabilities in these clients may also allow remote attackers to gain unauthorized access to a host.

207.166.87.157 -> 64.4.12.192 (21)
207.166.87.157 -> 64.4.12.172 (3)
Total **(24)**

[1:556:5] P2P Outbound GNUTella client request

GNUTella is a P2P (Peer-to-Peer) protocol for exchanging arbitrary files. Depending on your site's policies, using it may be a policy violation. If not properly configured, GNUTella clients may accidentally share out confidential files. GNUTella worms (which use deceptive names to encourage download) and viruses may also be accidentally downloaded by a client. This rule being triggered means that a GNUTella client has been detected on your network.

207.166.87.157 -> many (918)
Total **(918)**

[1:648:7] SHELLCODE x86 NOOP

The NOP allows an attacker to fill an address space with a large number of NOPs followed by his or her code of choice. This allows "sledding" into the attackers shellcode.

203.66.215.25 -> 207.166.87.157 (1)
Total **(1)**

[1:649:8] SHELLCODE x86 setgid 0

Snort detected data resembling the x86 assembly code to change the group identity to 0.

63.250.205.44 -> 207.166.87.157 (1)
Total **(1)**

[1:882:5] WEB-CGI calendar access

An open source calendar perl script by Matt Kruse, Allows commands to be executed without input verification using the perl open() function. ie /cgi-bin/calendar_admin.pl place the string "|ping 127.0.0.1|" in the configuration file field, this executes the command "ping 127.0.0.1"

207.166.87.157 -> 207.68.176.250 (1)
Total **(1)**

[1:884:14] WEB-CGI formmail access

This event is generated when an attempt is made to access the perl cgi script Formmail. Early versions (1.6 and prior) had several vulnerabilities (Spam engine, ability to run commands under server id and set environment variables) and should be upgraded immediately. Newer versions can still be used by spammers for anonymizing email and defeating email relay controls.

203.167.97.19 -> 207.166.87.40 (4)
68.129.127.199 -> 207.166.87.40 (1)
Total **(5)**

[1:895:7] WEB-CGI redirect access

This event is generated when an attempt is made to gain unauthorized access to a CGI application running ona web server. Some applications do not perform stringent checks when validating the credentials of a client host connecting to the services offered on a host server. This can lead to unauthorized access and possibly escalated privileges to that of the administrator. Data stored on the machine can be compromised and trust relationships between the victim server and other hosts can be exploited by the attacker. If stringent input checks are not performed by the CGI application, it may also be possible for an attacker to execute system binaries or malicious code of the attackers choosing.

207.166.87.157 -> 207.68.185.58 (2)
Total **(2)**

[1:937:9] WEB-FRONTPAGE _vti_rpc access

This event is generated when an attempt is made to compromise a host running Microsoft FrontPage Server Extensions. Many known vulnerabilities exist for this platform and the attack scenarios are legion.

65.193.39.163 -> 207.166.87.40	(1)
61.170.244.77 -> 207.166.87.40	(1)
61.218.105.219 -> 207.166.87.40	(3)
61.222.191.98 -> 207.166.87.40	(1)
Total	**(6)**

[1:962:12] WEB-FRONTPAGE shtml.exe access

This event is generated when an attempt is made to compromise a host running Microsoft FrontPage Server Extensions. Many known vulnerabilities exist for this platform and the attack scenarios are legion.

65.193.39.163 -> 207.166.87.40	(1)
61.170.244.77 -> 207.166.87.40	(1)
61.218.105.219 -> 207.166.87.40	(3)
61.222.191.98 -> 207.166.87.40	(1)
Total	**(6)**

[1:972:8] WEB-IIS %2E-asp access

Microsoft Internet Information Service (IIS) uses Active Server Page to supply HTML and server-side scripting. ASP files use a .asp extension. When the period of the .asp is hex-encoded with a "%2e" to reference an ASP file, the contents of the file are disclosed.

207.166.87.157 -> 207.68.176.190	(5)
207.166.87.157 -> 207.68.176.250	(2)
207.166.87.157 -> 207.68.185.58	(6)
207.166.87.157 -> 63.115.140.26	(3)
Total	**(16)**

[1:990:8] WEB-FRONTPAGE _vti_inf.html access

Microsoft FrontPage provides software for web designers to generate and administer web pages. The file '_vti_inf.html' contains FrontPage configuration information of version number and scripting paths that is normally used by a FrontPage client to communicate with the server. An attacker can craft a URL to access this file to disclose the version number and scripting paths.

65.193.39.163 -> 207.166.87.40	(1)
61.170.244.77 -> 207.166.87.40	(1)
61.218.105.219 -> 207.166.87.40	(2)
61.222.191.98 -> 207.166.87.40	(1)
Total	**(5)**

[1:1042:8] WEB-IIS view source via translate header

Microsoft Internet Information Services (IIS) 5.0 contains scripting engines to support various advanced files types such as .ASP and .HTR files. This permits the execution of server-side processing. IIS determines which scripting engine is appropriate to use depending on the file extension. If an attacker crafts a URL request ending in 'Translate: f' and followed by a slash '/', IIS fails to send the file to the appropriate scripting engine for processing. Instead, it returns the source code of the referenced file to the browser.

66.166.10.224 -> 207.166.87.40	(7)
Total	**(7)**

[1:1113:5] WEB-MISC http directory traversal

Directory traversal attacks usually target web, web applications and ftp servers that do not correctly check the path to a file when requested by the client. This can lead to the disclosure of sensitive system information which may be used by an attacker to further compromise the system.

207.166.87.157 -> 216.136.232.84	(1)
Total	**(1)**

[1:1288:7] WEB-FRONTPAGE /_vti_bin/ access

This event is generated when an attempt is made to compromise a host running Microsoft FrontPage Server Extensions. Many known vulnerabilities exist for this platform and the attack scenarios are legion. In particular this rule generates events when the directory _vti_bin is accessed. This directory contains sensitive files that may be utilized in an attack against the server.

65.193.39.163 -> 207.166.87.40	(1)
61.170.244.77 -> 207.166.87.40	(1)
61.218.105.219 -> 207.166.87.40	(3)
24.52.142.6 -> 207.166.87.40	(1)
61.222.191.98 -> 207.166.87.40	(1)
Total	**(7)**

[1:1390:5] SHELLCODE x86 inc ebx NOOP

This is the x86 opcode for 'inc ebx'. This can be used as a NOOP in an x86 architecture, however as with all shellcode rules, this can cause false positives. Check to see if you are ignoring shellcode rules on web ports, as this will reduce false positives.

143.166.224.204 -> 207.166.87.157	(1)
Total	**(1)**

[1:1432:6] P2P GNUTella client request

This event indicates that use of a p2p client has been detected. This may be against corporate policy. p2p clients connect to other p2p clients to share files, commonly music and video files but can be configured to share any file on the local machine. This activity may not only use bandwidth but may also be used to transfer company confidential information to unauthorized hosts external to the protected network bypassing other security measures in place. This rule detects activity from Gnutella p2p client applications.

207.166.87.157 -> many	(918)
Total	**(918)**

[1:1610:11] WEB-CGI formmail arbitrary command execution attempt

This could be an attempt to gain intelligence about the web-server that might be used to further exploit the machine. The environment variables of the web-server might be retrieved and sent via email to an address of the attackers choosing. More importantly this could be an attempt to execute commands on the web-server. Should this be successful, the commands would execute with the privileges of the user owning the httpd daemon.

203.167.97.19 -> 207.166.87.40	(4)
Total	**(4)**

[1:1653:5] WEB-CGI campus access

This event is generated when an attempt is made to gain unauthorized access to a CGI application running ona web server. Some applications do not perform stringent checks when validating the credentials of a client host connecting to the services offered on a host server. This can lead to unauthorized access and possibly escalated privileges to that of the administrator. Data stored on the machine can be compromised and trust relationships between the victim server and other hosts can be exploited by the attacker. If stringent input checks are not performed by the CGI application, it may also be possible for an attacker to execute system binaries or malicious code of the attackers choosing.

207.166.87.157 -> 207.68.176.250	(1)
207.166.87.157 -> 207.68.185.58	(3)
Total	**(4)**

[1:2570:6] WEB-MISC Invalid HTTP Version String

This event is generated when an attempt is made to compromise a host running a Web server or a vulnerable application on a web server. In particular this rule generates events when a non-standard HTTP request is made to a server. Some applications do not handle this exception in an acceptable manner and may present an attacker with the opportunity to exploit the application and server becasue of this. Some applications do not perform stringent checks when validating the credentials of a client host connecting to the services offered on a host server. This can lead to unauthorized access and possibly escalated privileges to that of the administrator. Data stored on the machine can be compromised and trust relationships between the victim server and other hosts can be exploited by the attacker.

68.129.127.199 -> 207.166.87.40	(1)
Total	**(1)**

[119:2:1] (http_inspect) DOUBLE DECODING ATTACK

This event is generated when double encoded characters are detected in web traffic. This is abnormal behavior and may be an indicator of a possible attack against a vulnerable system. This may also be an attempt to evade IDS.

207.166.87.157 -> 144.81.82.80	(4)
207.166.87.157 -> 205.188.144.241	(2)
207.166.87.157 -> 207.68.176.190	(9)
Total	**(15)**

[119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING

Microsoft IIS servers are able to use non-ASCII characters as values when decoding UTF-8 values. This is non-standard behavior for a webserver and violates RFC recommendations. All non-ASCII values should be encoded with a %. This event may indicate an attack against a web server or at the least an attempt to evade an IDS. No web clients encode UTF-8 characters in this way. This is most likely a malicious request.

207.166.87.157 -> many	(500)
Total	**(500)**

[119:7:1] (http_inspect) IIS UNICODE CODEPOINT ENCODING

This event is generated when the pre-processor http_inspect detects Unicode encoded web requests. This may be an indicator of an obfuscated attack against a server as well as an attempt to evade an IDS. The Unicode map for the target servers can be generated for specific servers. Refer to the documentation for http_inspect for instructions.

207.166.87.157 -> 209.11.34.129	(60)
207.166.87.157 -> 209.11.34.136	(16)
Total	**(76)**

[119:12:1] (http_inspect) APACHE WHITESPACE (TAB)

This event is generated by the http_inspect pre-processor when a tab character is detected in a web request. This is non-standard, but Apache web servers may use this character as a space delimeter.

207.166.87.157 -> many	(65)
Total	**(65)**

[119:13:1] (http_inspect) NON-RFC HTTP DELIMITER

This event is generated when the http_inspect pre-processor detects the use of a newline "\n" character as a delimeter. This is non-standard but is accepted by both Apache and IIS web servers.

207.166.87.157 -> 216.136.232.84	(6)
207.166.87.157 -> 209.11.34.129	(2)
61.218.105.219 -> 207.166.87.40	(1)
61.222.191.98 -> 207.166.87.40	(1)
Total	**(10)**

[119:15:1] (http_inspect) OVERSIZE REQUEST-URI DIRECTORY

This event is generated when the http_inspect pre-processor detects a request for a URL that is longer than a specified length. This may indicate an attack or an attempt to evade an IDS. The maximum expected length of the URL is user configured.

207.166.87.157 -> many	(116)
Total	**(116)**