



# Global Information Assurance Certification Paper

Copyright SANS Institute  
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permitted without express written permission.

## Interested in learning more?

Check out the list of upcoming events offering  
"Network Monitoring and Threat Detection In-Depth (Security 503)"  
at <http://www.giac.org/registration/gcia>

David J. Naelon  
GCIA Practical Version 4.0  
Submitted: November 3, 2004

## Intrusion Report For GIAC University

© SANS Institute 2004, Author retains full rights.

|  |    |
|--|----|
| Executive Overview .....                   | 3  |
| Detailed Analysis .....                    | 5  |
| Data Chosen for Analysis .....             | 5  |
| Relationships and Analysis Process .....   | 5  |
| Overview of Detects .....                  | 9  |
| Three Critical Detects.....                | 10 |
| BACKDOOR Q Access .....                    | 10 |
| Description .....                          | 10 |
| Reason for Selection.....                  | 11 |
| Detect Generation Source.....              | 11 |
| Source Spoofing Probability .....          | 12 |
| Attack Mechanism .....                     | 12 |
| Correlations .....                         | 13 |
| Evidence of Active Targeting .....         | 14 |
| Severity .....                             | 15 |
| Defensive Countermeasures.....             | 15 |
| NON-RFC HTTP DELIMITER .....               | 16 |
| Description .....                          | 16 |
| Reason for Selection.....                  | 16 |
| Detect Generation Source.....              | 16 |
| Source Spoofing Probability .....          | 17 |
| Attack Mechanism .....                     | 17 |
| Correlations .....                         | 19 |
| Evidence of Active Targeting .....         | 20 |
| Severity .....                             | 20 |
| Defensive Countermeasures.....             | 21 |
| Proxy Scanning .....                       | 22 |
| Description .....                          | 22 |
| Reason for Selection.....                  | 22 |
| Detect Generation Source.....              | 22 |
| Source Spoofing Probability .....          | 23 |
| Attack Mechanism .....                     | 23 |
| Correlations .....                         | 24 |
| Evidence of Active Targeting .....         | 24 |
| Severity .....                             | 24 |
| Defensive Countermeasures.....             | 25 |
| Network Statistics.....                    | 25 |
| Top Five Talkers .....                     | 25 |
| Top Five Target Source Ports .....         | 26 |
| Three Suspicious External Sources .....    | 26 |
| Analysis Process.....                      | 27 |
| Hardware and Software Configurations ..... | 27 |
| Process .....                              | 27 |
| Appendix A – Bibliography.....             | 29 |
| Appendix B – snort.conf.....               | 30 |

## Executive Overview

In response to GIAC University's request for an intrusion report as described in the GCIA Practical Assignment 4.0, I have prepared the following document. Data was provided to me by <http://isc.sans.org/logs/raw/2002.9.26> for analysis. Included in this analysis, the requestors have asked for relationships of attacked/attacking hosts, specific data regarding what was happening, correlated proof of validity and a rating of the severity according to accepted IDS practices. Finally, defensive recommendations should be made to prevent similar events from happening in the future.

After carefully reviewing the data presented, I found three specific incidents worthy of note. Each incident was unique and each incident is preventable going forward. The final decision for defensive mechanism implementation will be up to the Executive board.

In all cases, the generated incidents are based on Web traffic. It is Internet Web Browser/Web Application traffic that either triggers or causes the event. Stopping all Internet traffic is obviously not the answer, however, better protection at the Internet Connection level is highly recommended to limit the inbound traffic that is allowed.

The first incident uncovered called Backdoor Q Access was likely triggered by a user participating in an on-line chat session like AOL Instant Messenger or MSN Messenger. The chat activity for our users is common as these are excellent tools for collaboration and study groups. However the construction of the traffic executing the exploits should not have been allowed to enter the network. Common DMZ perimeter best-practices should block the trigger from entering in. In addition, the inbound trigger is searching for machines that are infected on our network. Whether the trigger is sent to a random target or a specific slave host has not been determined as further investigation is needed.

If we do have infected hosts, this has the potential to spread quite rapidly. The infected host will run processes or code (programs really) as the administrator or root. These privileges on a server or PC allow for the code to have full access to potentially destructive system commands and critical data.

The second incident is NON-RFC HTTP Delimiter. That is a fancy term for a known bug in one of Microsoft's web development products. The inbound traffic looks very normal and should be allowed in. For all intents and purposes, the data looks like normal web requests to our internal web servers.

However, the request pattern contains a directive to the server that exploits the bug. If properly executed, access to other network data and system passwords will become available to the user. In addition, variants of this problem can lead to web server defacement and catastrophic data loss.

The third and final incident is a very common problem. As many of you may know, we have a web security device/application called a proxy. This device acts on your behalf when browsing the Internet. It protects you the user from external hackers and protects vital information about your system when surfing. However, it can also be used by a hacker as an intermediary box; a sort of lauchpad if you will.

An “open” or “anonymous” proxy will allow anyone to connect through it, whether they are supposed to or not. Once hidden behind this proxy, the attacker is not really know, but due to the proxy technology, will receive all or most of the data they are trying to find. The proxy hides them from the attacked device.

From a defensive standpoint, many of the necessary devices are already in place. I have some concerns about the inbound access currently allowed; however, this can be remedied without incurring cost through additional configuration and patch management of the devices we already have. Additional security hardware can be purchased for added security measures; however, it is not entirely necessary. It is my overall assessment that increased awareness of border device configuration in conjunction with proper patch management will alleviate many of the intrusion issues this University is currently facing.

## Detailed Analysis

### Data Chosen for Analysis

Log file used: 2002.9.26

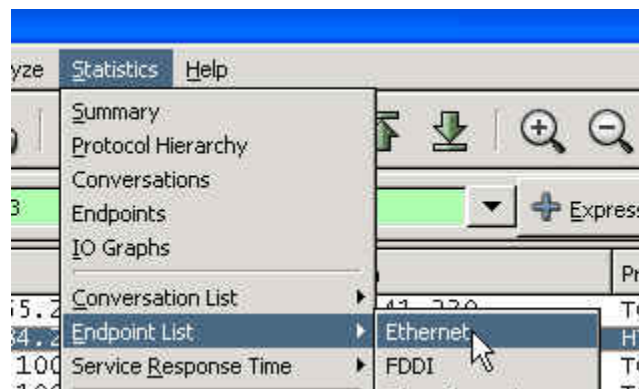
Link to file: <http://isc.sans.org/logs/raw/2002.9.26>

Although the timestamp on the file in the directory tree is September 26, the actual date of the data is October 25 and October 26.

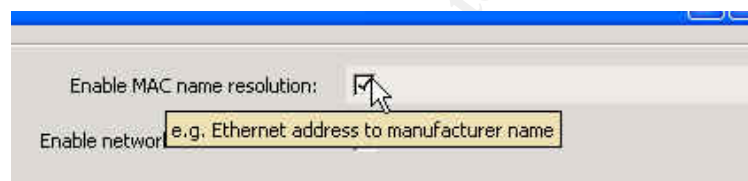
### Relationships and Analysis Process

To begin the analysis, it was important for me to determine how the network was laid out before relationships between addresses could be determined. I loaded the capture file into Ethereal to take a visual look at the network data in an easy to use interface. While there is some debate as to using command tools versus visual tools, my premise is to start basic and work out from there. A cursory view through the packet stream

showed only two different MAC addresses. Spot checking a number of packets continued to prove this. However, a more accurate view was necessary. Under the Statistics Menu in Ethereal there is an option to display Conversation Endpoints by Hardware Address as shown in the diagram.



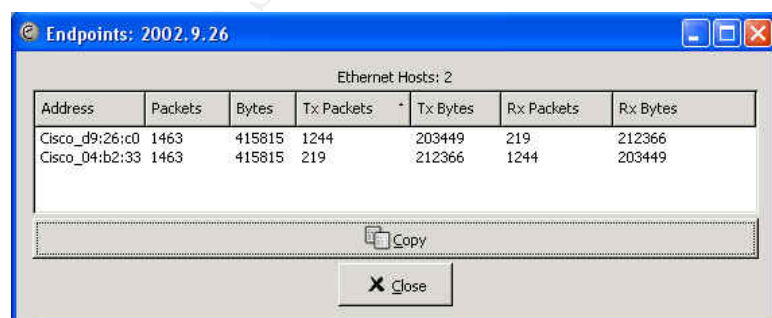
This confirmed only two hardware devices were picked up on the trace.



The only MAC Addresses found in the trace are:

1. 00:03:e3:d9:26:c0
2. 00:00:0c:04:b2:33

Within Ethereal, enabling MAC Address Name Resolution will show the hardware manufacturer in the display field of the packet as shown in the diagram. Both devices



Endpoints: 2002.9.26

Ethernet Hosts: 2

| Address        | Packets | Bytes  | Tx Packets | Tx Bytes | Rx Packets | Rx Bytes |
|----------------|---------|--------|------------|----------|------------|----------|
| Cisco_d9:26:c0 | 1463    | 415815 | 1244       | 203449   | 219        | 212366   |
| Cisco_04:b2:33 | 1463    | 415815 | 219        | 212366   | 1244       | 203449   |

Copy

Close

resolve to Cisco Systems, a common border router supplier for large corporations and ISPs. With these two settings in place, the output from the Endpoint List as shown in the diagram.

This was double checked

against the IEEE website (<http://standards.ieee.org/regauth/oui/index.shtml>) and confirmed.

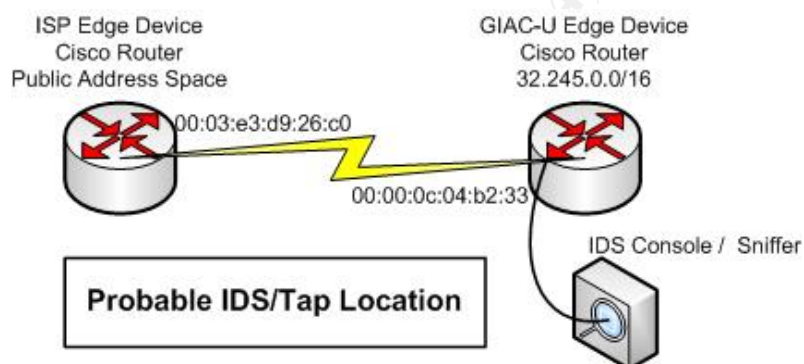
Another interesting note from the packet traces is that none the addresses are RFC 1918/Private space. The following Ethereal filter yields null values, proving that there are no endpoints in the RFC 1918 space.

```
ip.src == 192.168.0.0/16 and ip.src == 172.16.0.0/12 and ip.src == 10.0.0.0/8 and  
ip.dst == 192.168.0.0/16 and ip.dst == 172.16.0.0/12 and ip.dst == 10.0.0.0/8
```

Typically, some sort of NAT or Proxy is done at the edge of a network prior to leaving the trusted space. So if the address space is public, and all of the communication is sourced by two Cisco routers, then it is a safe assumption that this traffic has been pulled from an edge DMZ between the University border router and the ISP aggregation point.

All traffic associated with Cisco\_04:b2:33 (source or destination) is from the 32.245.x.x network. The next logical assumption is the assigned address space for the University is 32.245.0.0/16. While this may be a large space assignment, traffic on the GIAC-U MAC comes from a number of different /24 boundaries. Traffic from the ISP MAC never relates to the 32.245.0.0/16 space. This is proven with two display filters:

1. `ip.src == 32.245.0.0/16 and ip.dst == 32.245.0.0/16`
2. `ip.src == 32.245.0.0/16 and eth.src == 00:03:e3:d9:26:c0`



Both filters yield null values proving that all traffic on the 32.245.0.0/16 network is University traffic. More than likely, the network monitored would look like the diagram. The two Cisco devices are probably border routers for the University and ISP and the monitoring device (IDS or sniffer) providing the data would sit somewhere in-between. Typically, this device would be on the University premise. A second possibility is that both routers are owned by the university and represent the inbound/outbound DMZ prior to hitting anything external. In either case, the important design element is the same: traffic is pulled from a network between two routers and all the address space is public.

We now have the hardware (physical) relationship set based on the data. The next step is to determine the logical (IP based) relationships based on the trace.

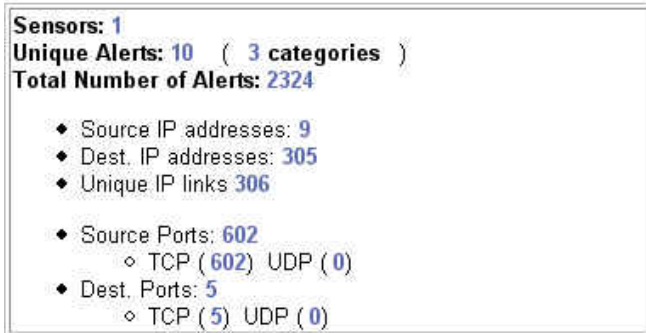
Since this is a University and typically IT funding is relatively low, I chose open source products for data analysis. So far, Ethereal has been used to pull statistics and perform preliminary traffic information. Now we need to move to more of a content based analysis.

On an older lab PC, I built a Fedora Core 2 Linux server with minimum packages, and then installed Snort 2.2.0 and ACID<sup>1</sup>. I set the home network for the snort.conf file to 32.245.0.0/16 (see Appendix B for the full configuration file) and the external network to any. I used an updated rule set<sup>2</sup> from 10/29/2004 and ran the file through Snort attached to a MYSQL Database and ACID Console with the following command:

```
snort -c /etc/snort/snort.conf -r 2002.9.26 -k none -dyev
```

- `-c /etc/snort/snort.conf` – tell snort which configuration file to use.
- `-r 2002.9.26` – load the trace file into snort.
- `-k none` – turns off checksum verification. Since the ip addresses have been changed, the checksums will not be correct. Having snort verify them is not needed in this analysis.
- `-dev`
  - `d` – dumps the application layer data to see what is in the packet
  - `e` – log the layer 2 packet header
  - `v` – verbose mode, although slower it will provide more data. Since this is not real-time monitoring, this will provide more data for analysis.

The ACID output page provides some excellent information here to begin the analysis. I



The screenshot shows the ACID console output with the following statistics:

- Sensors: 1
- Unique Alerts: 10 ( 3 categories )
- Total Number of Alerts: 2324
- Source IP addresses: 9
- Dest. IP addresses: 305
- Unique IP links: 306
- Source Ports: 602
  - TCP ( 602 ) UDP ( 0 )
- Dest. Ports: 5
  - TCP ( 5 ) UDP ( 0 )

already knew that I only had 1 sensor (my Snort box running on Fedora); however the rest of the data provides an excellent starting point for analysis. The nine source addresses is the logical place to begin since that is where our traffic originates. Each of the numbers in this web output is a link to more data. Clicking the “9” yields the following:

<sup>1</sup> Harper, Patrick. 15 Oct 2004 <[http://www.internetsecurityguru.com/documents/Snort\\_SSL\\_FC2.pdf](http://www.internetsecurityguru.com/documents/Snort_SSL_FC2.pdf)>.

<sup>2</sup> The Open Source Network Intrusion Detection System. Sourcefire. 29 Oct 2004 <<http://www.snort.org/dl/rules/>>.



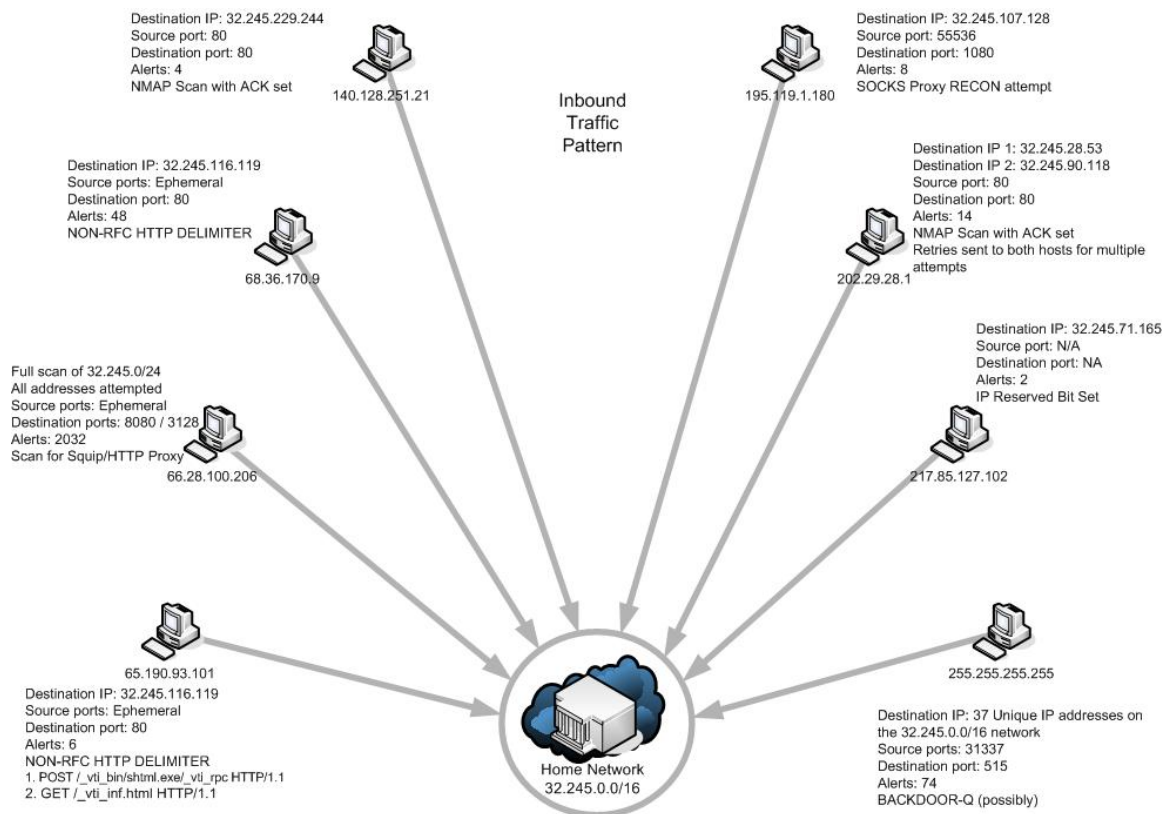
| < Src IP address > | FQDN                                   | Sensor # | Total # | Unique Alerts | Dest. Addr. |
|--------------------|--|----------|---------|---------------|-------------|
| 32.245.166.236     | Unable to resolve address              | 1        | 136     | 3             | 8           |
| 65.190.93.101      | Unable to resolve address              | 1        | 6       | 1             | 1           |
| 66.28.100.206      | Unable to resolve address              | 1        | 2032    | 2             | 254         |
| 68.36.170.9        | pcp02172463pcs.verona01.nj.comcast.net | 1        | 48      | 1             | 1           |
| 140.128.251.21     | Unable to resolve address              | 1        | 4       | 1             | 1           |
| 195.119.1.180      | Unable to resolve address              | 1        | 8       | 1             | 1           |
| 202.29.28.1        | Unable to resolve address              | 1        | 14      | 1             | 2           |
| 217.85.127.102     | pd9557f66.dip.t-dialin.net             | 1        | 2       | 1             | 1           |
| 255.255.255.255    | Unable to resolve address              | 1        | 74      | 1             | 37          |

We now have some preliminary information to begin further analysis. Each source IP address identified is listed, along with how many packets were generated, how many alerts Snort generated for the associated traffic and how many different destination IP addresses it hit. Some initial assumptions based on IP Source address stand out:

- 255.255.255.255 stands out as a problem as it is the all broadcast address. Typically, nothing good can come from this type of source as it must be spoofed.
- 66.28.100.206 is a very chatty host hitting 254 addresses (an entire class C). But the "254" destination addresses looks like it is probably a scan/recon effort for a network since it hits all the addresses in a /24.
- 68.36.170.9 is sourced from a very large cable modem network. The ARIN registration<sup>3</sup> confirms this. Could be a novice hacker doing recon or someone bouncing off of an "anonymous" proxy/unprotected PC.
- 32.245.166.236 is an internal address sending traffic out. It is either a web server or web proxy for internal users. All the outbound traffic is sent to port 80 sourced from normal ephemeral ports. It would wise to verify this traffic and do some cursory checks on the data based on the types of recon that has been detected elsewhere.

<sup>3</sup> "ARIN WHOIS Database Search." ARIN. 01 Nov 2004 <<http://ws.arin.net/cgi-bin/whois.pl>>.

An overview of the inbound traffic looks like the following:



## Overview of Detects

I identified 10 unique alerts. Nine of them were generated by inbound packets and one was outbound. A list of the detects:

| < Signature >   | < Classification > | < Total # > | Sensor # | < Src. Addr. > | < Dest. Addr. > |
|---|--------------------|-------------|----------|----------------|-----------------|
| [arachNIDS][snort] BACKDOOR Q access                  | misc-activity      | 74 (3%)     | 1        | 1              | 37              |
| [snort] SCAN Proxy Port 8080 attempt                  | attempted-recon    | 1016 (44%)  | 1        | 1              | 254             |
| [snort] SCAN Squid Proxy attempt                      | attempted-recon    | 1016 (44%)  | 1        | 1              | 254             |
| [snort] (http_inspect) DOUBLE DECODING ATTACK         | unclassified       | 6 (0%)      | 1        | 1              | 1               |
| [snort] (http_inspect) IIS UNICODE CODEPOINT ENCODING | unclassified       | 2 (0%)      | 1        | 1              | 1               |
| [snort] (http_inspect) BARE BYTE UNICODE ENCODING     | unclassified       | 128 (5%)    | 1        | 1              | 7               |
| [snort] BAD-TRAFFIC ip reserved bit set               | misc-activity      | 2 (0%)      | 1        | 1              | 1               |
| [arachNIDS][snort] SCAN nmap TCP                      | attempted-recon    | 18 (1%)     | 1        | 2              | 3               |
| [snort] (http_inspect) NON-RFC HTTP DELIMITER         | unclassified       | 54 (2%)     | 1        | 2              | 1               |
| url[snort] SCAN SOCKS Proxy attempt                   | attempted-recon    | 8 (0%)      | 1        | 1              | 1               |

- <Signature> – what the alert is
- <Classification> – general data about the activity
- <Total #> – number of instances matching that rule.
- <Sensor #> – which sensor fired off the alert.
- <Src. Addr.> – how many source addresses created the alert

- <Dest. Addr.> – how many targets

Just because an alert was logged does not necessarily mean the traffic was destructive or bad. It is important at this point to have some knowledge of what each alert is prior to determining the full impact.

In this case, the alert with the greatest number of hits is probably the least intrusive. The second and third alerts, (1) SCAN Proxy Port 8080 attempt and (2) SCAN Squid Proxy attempt generate 2032 alerts. But a look into the traffic shows that it is a rolling scan through the 32.245.157.0/24 network. The scanner tries to be a little bit stealthy, starting out at 32.245.157.117, then starting over at 32.245.157.1 after reaching the end of the subnet. However, overall this is not worth spending much time on.

Working down the list, the next three alerts (1) Double Decoding Attack, (2) IIS Unipoint Encoding Attack and (3) Bare Byte Encoding are generated mostly by an internal machine and are web requests. It is possible that this is either an outbound proxy or a user web browsing. They seem particularly interested in car shopping, mixed in with a little porn browsing and streaming media. Other than reviewing the Internet Usage Policies with the user, there does not seem to be much attack data here.

Although not enough data exists for a full analysis, the next alert BAD-TRAFFIC ip reserved bit set might be worth further investigation at a later date as a follow up. Currently, there is not a good reason to have the reserved bit set. Since this is a fragment, with no other fragmented packets to work with, further analysis will be difficult.

More recon is attempted in our next alert from two other source IP addresses. These are NMAP type scans and no data was sent back to the originating hosts. Since the packets are obviously crafted (source/destination ports are the same), and not crafted with much stealth in mind, this is probably a novice hacker. The last alert is relatively innocuous as well. The possible attacker seems to be searching for an open SOCKS proxy. Not many scan attempts are made and there doesn't seem to be data passing.

### ***Three Critical Detects***

1. BACKDOOR Q Access
2. NON-RFC HTTP Delimiter
3. Proxy Scans

## **BACKDOOR Q Access**

### ***Description***

BACKDOOR Q access – 74 alerts

Q is a Trojan Horse offering the attacker remote access to the victim host. This event is generated when raw TCP packets are sent to the victim server<sup>4</sup>. This description fits the fundamentals of the packets. Both the Snort reference and the Whitehats reference make mention of the payload not being part of the signature of attack<sup>5</sup>. Rather, the “raw” TCP packets act as a trigger. Any number of items in the packet could cause the trigger: source port, source address and possibly the payload. So it is possible that in this environment, that payload is the trigger. Also in this environment, there are multiple infected machines all receiving the same trigger.

### **Reason for Selection**

This detect was selected for a number of reasons. First, its data and patterns point to a number of infected machines on multiple subnets. In all, 37 destination hosts were targeted by the sending host. The Trojan is also known to spawn processes running as root on the infected machine. This is a system compromise that could potentially lead to additional infections throughout the network and complete loss of control and data on the target system.

### **Detect Generation Source**

The detect was generated by Snort 2.2.0 and ACID.

The following rule generated the alert:

```
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;)
```

Here is a printout of the packet using TCPDUMP

```
[root@homelab-snort dhaelon]# tcpdump -navvx -r 2002.9.26 src 255.255.255.255 and dst 32.245.41.230
reading from file 2002.9.26, link-type EN10MB (Ethernet)
19:53:05.436507 00:03:e3:d9:26:ce > 00:00:0c:04:b2:33, ethertype IPv4 (0x0800), length 60: IP (tos 0x0, ttl 15, id 0,
offset 0, flags [none], proto 6, length: 43, bad cksum 4dda (->60f3)!): 255.255.255.255.31337 > 32.245.41.230,printer: R
[bad tcp cksum 302 (->161b)!] 0:3(3) ack 0 win 0 [RST cko]
0x0000: 4500 002b 0000 0000 0f06 4dda ffff ffff E.,+.....M.....
0x0010: 20f5 29e6 7a69 0203 0000 0000 0000 0000 ..).z1.....
0x0020: 5014 0000 0302 0000 636b 6f00 0000 P.....cko...
[root@homelab-snort dhaelon]#
```

<sup>4</sup> Sourcefire. 01 Nov 2004 <<http://www.snort.org/snort-db/sid.html?sid=184>>.

<sup>5</sup> "SID203." WhiteHats. 01 Nov 2004 <<http://www.whitehats.com/info/ids203>>.

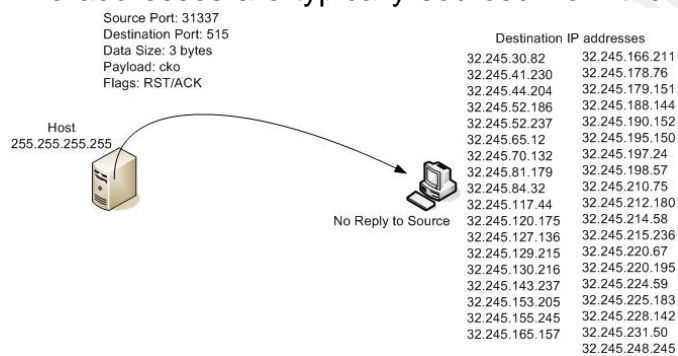
The rule looks for TCP packets sourced from 255.255.255.0/24 to the home network on any port (source or destination.) Referring back to our Snort.conf file (Appendix B) \$HOME\_NET is set to 32.245.0.0/16. Within the header, the 9<sup>th</sup> bite offset is 0x06 (TCP). The source address is 255.255.255.255 and the destination of 32.245.41.230 is within the \$HOME\_NET range. RST/ACK flags are set matching “flags:A+” in the rule.

### Source Spoofing Probability

It is absolutely certain the source address is spoofed. The address itself is not naturally occurring on a network: 255.255.255.255. In addition, the TTL of the packet is only 15. This is extremely low for any operating system, let alone a UNIX based one. Lastly, spoofing of the Q trigger is desired. Since the Trojan looks to spawn processes on other machines, return traffic could possibly lead to its discovery. Using the broadcast address should (in a normal routing configuration) prevent packets from leaving the local subnet in search of the originating host. Each packet has the same exact TTL which would lead to the same source for each packet. Each sequence number is the same (0) which is unusual. Finally, each of the packets have the RST/ACK flags set. This should further ensure that no replies are sent to the host.

### Attack Mechanism

The Q Trojan sits idle on a Unix platform then sends raw ip data out to infected hosts. The addresses are typically sourced from the broadcast address of a class C network.



The source port, source address or payload seems to trigger a process to run on the remote machine. In our alert, each of the packets had the same payload, cko. The diagram to the left lists the destination IP addresses along with the general packet structure.

Packets have a source port of 31337.

This spells “ELEET” in hacker slang and is a well known source port for backdoors and Trojans. The destination port is also telling: 515. This is a very common port to have opened on Unix systems as it is used for LPD (print) communication. The likelihood of this port being open is high.

The varied range of destination addresses leads to the assumption that there are a number of infected slave hosts throughout the network. Since packets sent on the local subnet to the sending server could lead to an arbitrary response since the MAC address would be know local, it searched out and found hosts on multiple networks. Broadcast traffic should not cross routers throughout the network and would stay local.

However, since this traffic has been pulled from our DMZ area, and there is no other local traffic inbound on the trace, it makes sense that these are external sources hitting hosts on the internal network. Remember, earlier I determined that no Home Network

32.245.0.0/16 addresses were associated with the external router MAC. This is consistent with the stealth of the backdoor and its desire to spawn processes on remote machines without detection.

There are several similar detects referencing IRC channels as a possible trigger (explained in detail under the Correlations section) prompting some further investigation. Since this type of traffic would not normally set off alerts, it was back to some more manual investigation. From Ethereal, I enable DNS resolution for the trace and found instances of 32.145.166.236 sending outbound traffic to Yahoo, referencing Instant Messenger. Not the following information pulled from the packet:

Internet Protocol, Src Addr: 32.245.166.236 (32.245.166.236), Dst Addr: f144.mail.yahoo.com (216.136.174.168)

<snip>

```
0180 72 69 61 6c 3e 48 69 2c 2b 79 6f 75 72 2b 66 72   rial>Hi,+your+fr
0190 69 65 6e 64 2b 3c 42 3e 6d 61 67 67 69 65 63 61   iend+<B>maggieca
01a0 6d 65 62 61 63 6b 2b 28 41 6e 6e 69 65 29 3c 2f   meback+(Annie)</
01b0 42 3e 2b 68 61 73 2b 69 6e 76 69 74 65 64 2b 79   B>+has+invited+y
01c0 6f 75 2b 28 76 65 72 6c 69 6e 75 58 58 58 40 79   ou+(verlinuXXX@y
01d0 61 68 6f 6f 2e 63 6f 6d 29 2b 74 6f 2b 73 69 67   ahoo.com)+to+sig
01e0 6e 2b 75 70 2b 66 6f 72 2b 59 61 68 6f 6f 21 2b   n+up+for+Yahoo!+
01f0 4d 65 73 73 65 6e 67 65 72 2e 3c 2f 46 4f 4e 54   Messenger.</FONT
```

Reading between the lines, we have our friend "Annie" requesting us to sign up for Instant Messenger.

### **Correlations**

Port 31337 is a well known source port for Back Fire, Back Orifice (Lm), Back Orifice russian, Baron Night, Beeone, BO client, BO Facil, BO spy, BO2, cron / crontab, Freak88, icmp\_pipe.c and Sockdmini<sup>6</sup>.

There are several correlations to support the backdoor theory, although without more data it is near impossible to obtain a full understanding of how far the infection has gone, but we can look to similar detects and reports to determine if we are on the right track.

---

<sup>6</sup> "Odd Ports." SANS. 01 Nov 2004 <<http://www.sans.org/resources/idfaq/oddports.php>>.



“Kraut” wrote an interesting post at the following url: <http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00112.html>. The traces referenced in his posting are remarkably similar to the ones presented here. The conclusions are similar as well: particularly noting the following statement:

“This traffic calls into question what expected response the attacker anticipates from this stimuli. Presumably the 'cko' string is some sort of direction for systems infected with the Q Trojan to perform some action.”

This explanation is in response to the following example packets:

```
14:42:01.334488 255.255.255.255.31337 > aaa.bbb.155.46.printer: R [bad
tcp cksum f7fa!] 0:3(3) ack 0 win 0 [RST cko] (ttl 14, id 0, len 43, bad
cksum e8a2!)
14:43:40.334488 255.255.255.255.31337 > aaa.bbb.74.158.printer: R [bad
tcp cksum faf7!] 0:3(3) ack 0 win 0 [RST cko] (ttl 14, id 0, len 43, bad
cksum 3c30!)
```

Notice the source address as broadcast on port 31137 destined for hosts using port 515 (printer). In addition the RST flag and payload of “cko” fit the profile of our detect and his explanation coincides with the detect and analysis done thus far. The Peterson article goes on to mention IRC channels as a possible trigger to the attacking host. This prompted me to search the packet trace a bit more. We do find evidence of packets from the internal proxy headed out to various Yahoo mail servers and launch servers, so this is a distinct possibility.

Jeff Peterson in his post, <http://lists.jammed.com/incidents/2001/05/0037.html>, seems to have the same type of traffic and even references the IRC channel as well. He writes:

“This seems to have something to do with IRC. Almost every time I connect to a certain IRC, I get this probe. Possibly somebody looking for a certain trojan?”

In reference to the following packet data:

```
04/22-05:54:25.295925 0:0:C:8:D5:6 -> 0:10:11:FF:E0:0 type:0x800 len:0x3C
255.255.255.255:31337 -> ***.***.106.102:515 TCP TTL:12 TOS:0x0 ID:0
IpLen:20
```

Again, we see source and destination ports of 31337 and 515 respectively. Packets come in from the all-broadcast to various addresses within the class b network (see the post for additional destination).

### ***Evidence of Active Targeting***

It would seem highly likely that the target hosts have been actively targeted based on the trace information. Looking at the Attack Mechanism diagram, we see 37 separate destination addresses, almost all on different subnets within the home network. This is

either wildly random, or actively targeted. I tend to err on the side of caution and assume the hosts are compromised. I would also recommend removing those hosts from the network. Additional information on the source IP addresses would be helpful to further investigation.

### **Severity**

*Criticality: 4* – While there is some evidence of active targeting, this can not be absolutely confirmed. The absence of source traffic detected from the destination hosts leads to this reduced criticality.

*Lethality: 4* – The Q Trojan runs as root on the infected host. This fact alone could lead to the assumption of data loss and/or system loss and full compromise. However, the lack of traffic from the infected hosts is the factor to drop this down from 5.

*System Countermeasures: 3* – Investigations on the destination hosts, patch levels and security stance is needed.

*Network Countermeasures: 2* – Inbound traffic on port 515 should not be allowed as it is not necessary in from the perimeter. In addition, source broadcast addresses should be blocked as there is no good reason for that traffic to enter.

*Overall Score: 3*

$(\text{Criticality} + \text{Lethality}) - (\text{Sys Countermeasures} + \text{Net Countermeasures}) /$

$(4+4) - (3+2) = 3$

### **Defensive Countermeasures**

Filtering at the external device is highly recommended. This can also be implemented with no additional cost since the border Cisco router can perform this function, and would require only minor configuration changes. on the Ingress Cisco device. Specifically, inbound hosts from the 255.255.255.0/24 network should be blocked, as well as inbound TCP/UDP on port 515. There is not a good reason to allow this type of traffic in. I recommend the following filters to block this inbound traffic:

```
Interface Serial X/X/X (00:00:0c:04:b2:33 interface)
```

```
IP access-group SECURITY in
```

```
IP access-list SECURITY
```

```
deny tcp any any eq 515
```

```
deny udp any any eq 515
```

```
deny ip 255.255.255.0 0.0.0.255 any
```

```
permit ip any any
```

These lines create an access list applied to the inbound interface that blocks inbound traffic to port 515 as well as any traffic from the 255.255.255.0/24 network. While additional filtering would be recommended for overall security posture, this recommendation is specific to the intrusion detected.



## NON-RFC HTTP DELIMITER

### *Description*

NON-RFC HTTP Delimiter – 54 alerts

This is an alert in response to various vulnerabilities in a number of web servers. While the alert may or may not be critical, it prompts for further investigation.

### *Reason for Selection*

Due to the known Translate: f bug within the WebDAV environment, and the known use of IIS throughout the University this is a potentially critical exploit. Defacement of web presence, access to network shares hosting files and possible password gleaning are all possible results of this exploit.

### *Detect Generation Source*

The detect was generated by Snort 2.2.0 and ACID.

The following default preprocessor generated the alert:

```
preprocessor http_inspect_server: server default \  
  profile all ports { 80 8080 8180 } oversized_dir_length 500
```

The HTTP Inspect pre-processor is a generic http decoder for user applications. Given a data buffer, it will decode the buffer, find http fields and normalize the fields.<sup>7</sup> Our traffic matches the rule for pre-processing of the packet. Here is a look at a snip of the packet data pulled from the print packet function of Ethereal:

```
Internet Protocol, Src Addr: 65.190.93.101 (65.190.93.101), Dst Addr:  
32.245.166.119 (32.245.166.119)  
  Transmission Control Protocol, Src Port: 65054 (65054), Dst Port: http  
    (80), Seq: 0, Ack: 0, Len: 265  
    Source port: 65054 (65054)  
    Destination port: http (80)  
    Version: 4  
    Hypertext Transfer Protocol  
    GET /_vti_inf.html HTTP/1.1\r\n
```

---

<sup>7</sup> Snort User Manual, page27

The destination port matches the preprocessor port profile for HTTP traffic. The var HOME\_NET variable (mentioned earlier) is set to 32.245.0.0 and the var HTTP\_SERVERS variable is set to \$HOME\_NET. The preprocessor processes destination web servers based on the HTTP\_SERVERS variable. So the destination address of 32.245.166.119 fulfills the requirement for packet reassembly. Finally, the HTTP/1.1\r\n fields are not normal. This tells the preprocessor to alert on the packet.

### Source Spoofing Probability

There are two source addresses that triggered the alert:

| < Src IP address > | FQDN                                   |
|--------------------|--|
| 65.190.93.101      | Unable to resolve address              |
| 68.36.170.9        | pcp02172463pcs.verona01.nj.comcast.net |

I would doubt these packets are spoofed. Using the above packet data as a reference, it seems rather normal. The TTL values of each packet (not mentioned above) are consistent for each request. The source ports are in the range you would expect: 60K+ range for 65.190.93.101 and 30K+ range for 68.36.170.9. The one possibility I do see is the source network for 68.36.170.9. That address resolves to a very large cable modem network where it would be common place to find unsuspected “users” with their PCs always on and unpatched. These novice user PCs are targets for hackers to use as anonymous proxies. However, this is not possible to prove from the data provided.

### Attack Mechanism

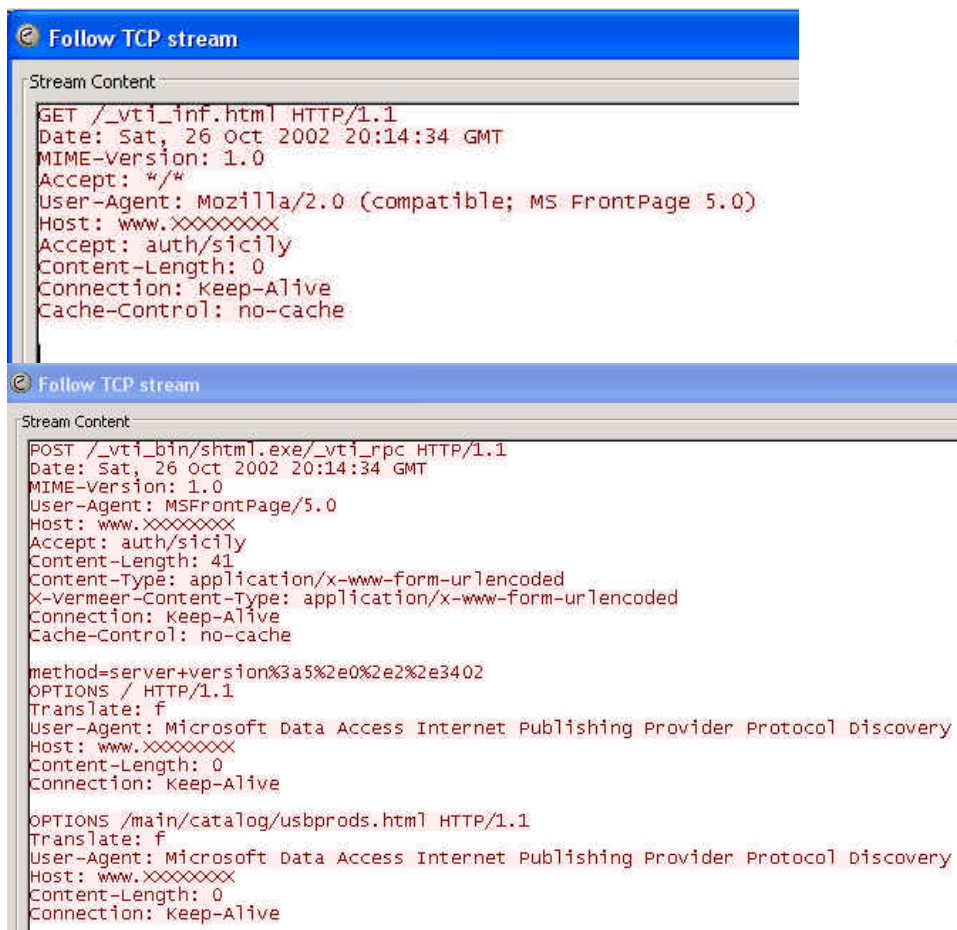
To pull this data, I issued the following filter into Ethereal:

```
ip.src == 68.36.170.9 or ip.src == 65.190.93.101
```

This pulled out all the packets associated with the alert. I saw a pattern of HTTP GET requests, aimed towards 32.245.166.119, followed by POST and OPTIONS requests as follows:

| Source                | Destination    | Protocol | Info                              |
|-----------------------|----------------|----------|-----------------------------------|
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | GET /_vti_inf.html HTTP/1.1       |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | POST /_vti_bin/shtml.exe/_vti_rpc |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | OPTIONS / HTTP/1.1                |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | OPTIONS /main/catalog/usbprods.ht |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | GET /_vti_inf.html HTTP/1.1       |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | POST /_vti_bin/shtml.exe/_vti_rpc |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | OPTIONS / HTTP/1.1                |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | OPTIONS /usb/usbprods.html HTTP/1 |
| pcp02172463pcs.verona | 32.245.166.119 | HTTP     | GET /_vti_inf.html HTTP/1.1       |

Using the “Follow TCP Stream” option within Ethereal, I pulled the following data from 4 consecutive packets:



The first packet in the list is a single stream, followed by the next three requests. Two points stick out: (1) the source browser is a Mozilla browser and (2) the “Translate: f” portion of the last two packets.

Translate: f is a known bug in certain versions of the Microsoft IIS Web Server with WebDAV installed. The vulnerability, referenced in KB256888 (<http://support.microsoft.com/kb/q256888/>) and the cert-advisory URL <http://cert.uni-stuttgart.de/archive/bugtraq/2000/08/msg00155.html>, allows for the attacking host to execute the following:

“When someone makes request for ASP/ASA (or any other scriptable page) and adds "Translate: f" into headers of HTTP GET request (headers are not part of URL, they are part of HTTP request), there is a serious security bug in Windows 2000 (unpatched by SP1) that in return gives complete ASP/ASA code instead of processed file (one has to add trailing slash "/" to end of requested url to have this really working).<sup>8</sup>”

The main problem in the vulnerability is the potential attacker could glean password information and access to other network shares, depending on where the files are hosted for the web server. Since “Translate: f” in the intended form is a normal WebDAV component, allowing for the retrieval of a source file for editing, it is commonly found on unpatched servers. The danger also becomes that defacement of websites is a common result since files can be pulled, edited and possibly posted. The keep alive.

The vulnerability, while not limited to, is expected to be attacked by a non-Microsoft browser. Mozilla is an open source browser specification.

### **Correlations**

Microsoft has a knowledge base article discussing and describing the bug in the WebDAV code. It can be referenced: <http://support.microsoft.com/kb/q256888/>. Specifically, note the following:

#### **SYMPTOMS**

If an Internet Information Services (IIS) server receives a file request that contains a specialized header as well as one of several particular characters at the end, the expected ISAPI extension processing may not occur. The result is that the source code of the file would be sent to the browser.

This statement is consistent with the packets shown since other non-Microsoft references make mention of the specific request for “Translate: f” as mentioned in the following quote:

“adds "Translate: f" into headers of HTTP GET request (headers are not part of URL, they are part of HTTP request), there is a serious security bug in Windows 2000 (unpatched by SP1) that in return gives complete ASP/ASA code instead of processed file”

---

<sup>8</sup> <http://cert.uni-stuttgart.de/archive/bugtraq/2000/08/msg00155.html>

This quote was pulled from the bugtrack in the URL, <http://cert.uni-stuttgart.de/archive/bugtraq/2000/08/msg00155.html>.

Another quote on this same fact:

"Translate:f" is legitimate header for WebDAV, it is used as it should be - adding this to HTTP GET is a signal for the WebDAV component to return the source code of the requested file and bypass processing. It is used in FrontPage2000 and any WebDAV compatible client to get a file for editing. It has to be accompanied by some other information, which should prevent unauthorized users from viewing the source. Unfortunately, a coding problem makes it possible to retrieve those files by simply adding "Translate:f" in the header, and placing "/" at end of request to the HTTP GET.

This quote from <http://www.securiteam.com/windowsntfocus/5VP0P0A2AS.html> shows that it is somewhat difficult to detect since the Translate: f field is fairly common and expected in normal traffic. Therefore it places increasing importance on host (target) machine verification for patch levels.

Further underscoring the possibility for risk is this quote from <http://www.securityspace.com/smysecure/catid.html?viewsrc=1&id=10491> referencing ODBC connections:

There is a serious vulnerability in Windows 2000 (unpatched by SP1) that allows an attacker to view ASP/ASA source code instead of a processed file. ASP source code can contain sensitive information such as username's and passwords for ODBC connections.

The reference here to passwords from ODBC connection is particularly scary since traditionally, these passwords are not unique among data sources within an enterprise.

### ***Evidence of Active Targeting***

It is likely the web server is actively targeted. We have two separate source IP addresses making multiple attempts to pull/post data from the 32.245.166.119 destination. The 48 alerts generated by 68.36.170.9 point to repeated attempts at access to our web server. Had this source host attempted the same packet types to multiple destinations, it would seem more likely they are doing recon. The addition of the second host, 65.190.93.101 attempting the same exact exploit affirms this conclusion.

### ***Severity***

**Criticality: 4** – There is evidence of active targeting by multiple source hosts. This leads to the assumption that there is a known vulnerability not patched running on the server.

*Lethality: 3* – The exploit looks for access to server shares and possibly gleans password data and secured data. The patches needed to fix it are relatively old, however, the possibility for compromise is high.

*System Countermeasures: 2* – Since multiple source hosts are attempting access to the same destination, it is likely this host is not patched.

*Network Countermeasures: 1* – Inbound traffic on port 80 is necessary for normal operations. Typically, this inbound access is allowed from any source. Therefore the network countermeasures cannot really be effective.

*Overall Score: 4*

$(\text{Criticality} + \text{Lethality}) - (\text{Sys Countermeasures} + \text{Net Countermeasures}) /$

$(4+3) - (2+1) = 4$

### ***Defensive Countermeasures***

There are two recommendations I have for this detect. First, the exploit research clearly points to a known bug in the Microsoft code that is fixed with a patch: Service Pack 1. I would recommend Service Pack 4 at this point, providing we prestage the change in a test environment to ensure the patch does not have unexpected impact to the overall operations of the box. Furthermore, an enterprise wide password change is in order as this bug has the possibility of leaked passwords.

We can't simply block inbound web-traffic at the router as with Backdoor Q as that would cause an extreme negative impact on operations and business. If patching the server is not an option, some sort of in-line device is required to protect the host. Snort has the ability to operate as an in-line IDS and would serve as an inexpensive option. A second option would be an in-line IPS style device. Teros makes an application/HTML gateway product<sup>9</sup>. Although a bit more costly, the Teros device is extremely popular among financial institutions for Windows based web hosts in the DMZ. The Teros gateway can create a set of rules that set out normal responses from the server host. Responses from the server that are outside the normal range are blocked.

---

<sup>9</sup> <http://www.teros.com/products/appliances/gateway/index.shtml>

## Proxy Scanning

### *Description*

SCAN Proxy Port 8080 – 1016 Alerts\*

SCAN Squid Proxy Attempt - 1016 Alerts\*

\*2032 total Alerts discussed as one event

Multiple inbound proxy attempts were made to the 32.245.157.0/24 network. All the addresses were hit within the subnet, however, no other addresses on the 32.245.0.0/16 were attempted. Trace data also shows no return traffic is sent. This event is a reconnaissance effort by an external host, possibly looking for an anonymous proxy to bounce attacks off of to other networks, or a proxy specific exploit.

### *Reason for Selection*

As far as attacks go, this is a fairly esoteric proxy scan and quite common. There are no specific threats uncovered by the scan, however, as we will see, there are some defensive postures worth taking as a result of the intrusion report as a whole.

### *Detect Generation Source*

This detect will combine two different alerts and group them together. Both of the detects are scans for open proxy servers. The rules generating the alerts are:

```
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;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 3128 (msg:"SCAN Squid Proxy attempt"; flags:S,12; flow:stateless; classtype:attempted-recon; sid:618; rev:9;)
```

In both cases of this rule, the same source address triggers the alert: 66.28.100.206. In all cases, the scans are destined for 32.245.157.0/24 and all addresses on the /24 subnet are attempted twice per port per address. That gives us 2 attempts times 2 ports times 2 address, or 8 total attempts. Multiply the 8 total attempts by 254 total addresses and you arrive at the 2032 alert number.

The source address of 66.28.100.206 falls in the \$EXTERNAL\_NET variable range while the 32.245.157.0/24 falls within the \$HOME\_NET variable range of 32.245.0.0/16 described earlier.

Looking at a snip of a representative packet:



```

▶ Frame 137 (74 bytes on wire, 74 bytes captured)
▶ Ethernet II, Src: 00:03:e3:d9:26:c0, Dst: 00:00:0c:04:b2:33
▼ Internet Protocol, Src Addr: 66.28.100.206 (66.28.100.206), Dst Addr: 32.245.1
  Version: 4
  Header length: 20 bytes
  ▶ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  Total Length: 60
  Identification: 0xac73 (44147)
  ▶ Flags: 0x04 (Don't Fragment)
  Fragment offset: 0
  Time to live: 43
  Protocol: TCP (0x06)
  Header checksum: 0x28bc (incorrect, should be 0x3dd4)
  Source: 66.28.100.206 (66.28.100.206)
  Destination: 32.245.157.149 (32.245.157.149)
▼ Transmission Control Protocol, Src Port: 56921 (56921), Dst Port: 3128 (3128),
  Source port: 56921 (56921)
  Destination port: 3128 (3128)
  Sequence number: 0 (relative sequence number)
  Header length: 40 bytes
  ▶ Flags: 0x0002 (SYN)
  Window size: 5840
  Checksum: 0x5798 (incorrect, should be 0x6cb0)
  ▶ Options: (20 bytes)

```

We notice the SYN flag set as required by both rules. In addition, the stateless specification in the rule looks for either initiating or reply packets, meaning its not necessarily looking for established connections. The destination port of 3128 matches the destination port for the rule.

### **Source Spoofing Probability**

It is not likely that the source address is spoofed. Since this is a reconnaissance effort, a reply is generally preferred. Keep in mind, the attacker is looking for something. On the wire, the only way an attacker can 'see' if something is listening is by a reply. A spoofed packet would send the reply elsewhere.

### **Attack Mechanism**

There are so many attack mechanisms that could generate this kind of scan. Nmap, scanlogd, SuperScan and a number of password cracking script kiddies can generate this type of traffic. The source address is registered to a large ISP in the U.S.A.<sup>10</sup> so it

<sup>10</sup> <http://ws.arin.net/cgi-bin/whois.pl>



is unlikely this is a hostile foreign entity or hacker. Also looking at the timestamps in the following diagram:

|    | Time        | Source        | Destination    |
|----|-------------|---------------|----------------|
| 7  | 6651.070000 | 66.28.100.206 | 32.245.157.117 |
| 8  | 6654.090000 | 66.28.100.206 | 32.245.157.117 |
| 9  | 6654.090000 | 66.28.100.206 | 32.245.157.117 |
| 10 | 6657.070000 | 66.28.100.206 | 32.245.157.117 |
| 11 | 6660.070000 | 66.28.100.206 | 32.245.157.117 |
| 12 | 6663.070000 | 66.28.100.206 | 32.245.157.117 |
| 13 | 6663.070000 | 66.28.100.206 | 32.245.157.117 |
| 14 | 6664.070000 | 66.28.100.206 | 32.245.157.117 |
| 15 | 6669.070000 | 66.28.100.206 | 32.245.157.117 |

We see they are not spread too far apart. The attacker either doesn't care if detected, or doesn't know enough to spread out the requests over a longer period of time.

### **Correlations**

Proxy scans are among the most common types of attacks. Typically, they are just users with script kiddies looking for open proxies to bounce off of. The Snort.Org rule page, <http://www.snort.org/snort-db/sid.html?sid=620>, for this activity states:

This event indicates that an attempt has been made to scan a host.

This may be the prelude to an attack. Scanners are used to ascertain which ports a host may be listening on, whether or not the ports are filtered by a firewall and if the host is vulnerable to a particular exploit.

Tools such as those found on <http://www.proxyblind.org/tut.shtml> readily advertise their ability to seek out and find open proxies.

### **Evidence of Active Targeting**

There is no specific evidence of active targeting. The attacker is not particularly stealthy, although starting at 32.245.157.117 is a nice touch, then resetting back to 32.245.157.117 when the end of the /24 is reached ensures completion of the network. Overall, this attempt seems rather random and there are no other packets to or from this network.

### **Severity**

*Criticality:* 3 – The scan in and of itself allowed in is a problem. Since University networks are notoriously “open” it is important to be on the lookout for possible attackers using our network as a launch pad..

*Lethality:* 3 – Other than being annoying, if the attacker keeps up this activity, a possible DOS condition could exist..

*System Countermeasures:* 4 – There is no evidence of proxy response traffic therefore it is probably that most of the systems are patched properly and/or there are no open proxies on that network.

*Network Countermeasures: 1* – Why is inbound traffic allowed on port 8080 and 3128? There is no good need for this. Inbound web services are provided on port 80 (in most cases) and there is no reason to allow inbound squid proxy attempts..

*Overall Score: 1*

(Criticality + Lethality) – (Sys Countermeasures + Net Countermeasures) /  
(3+3) – (4+1) = 1

### **Defensive Countermeasures**

Port scans are commonplace throughout the Internet and defending against them can be more trouble than it is worth. However, there are some simple steps that can be taken to ensure the reconnaissance efforts are fruitless.

First, our browser proxy servers should be password protected. This ensures that they are more difficult to compromise and are not “open” for general use. In addition, remember the porn browsing mentioned earlier in the paper? Requiring passwords for access also allows us a better tracking mechanism for inappropriate use.

As with Backdoor, we can do some external filtering. Traffic inbound to port 3128 or 8080 is really not necessary and can be filtered out. Consider adding the following lines to the Cisco border device:

```
Interface Serial X/X/X (00:00:0c:04:b2:33 interface)
IP access-group SECURITY in

IP access-list SECURITY
deny tcp any any eq. 8080
deny up any any eq. 3128
permit ip any
```

This configuration blocks inbound attempts for proxy access. However, since the ports are above 1028, there is a slim chance we could block traffic we want. Source ports are typically above 1028 and both of these filters would block that response traffic. The likelihood is low, it does however exist.

A second and more costly option is to install a CheckPoint firewall at the perimeter with SmartDefense enabled. The Smart Defense component has some interesting “IPS” style features. It can be setup to detect a port scan, then reject packets from the source. In addition, the SmartDefense component can work directly with the ISC to generate packet filters based on known hostile IP addresses.

### **Network Statistics**

#### **Top Five Talkers**

This top five talkers list is based on alert's detected.

1. 66.28.100.206 – As described in the third alert of the intrusion report, this host scanned the entire 32.245.157.0/24 network looking for open proxies in a classic reconnaissance attempt. Monitoring for additional traffic from this source is recommended.
2. 32.245.166.236 – This is an internal host issuing HTTP requests outbound to various sites, including porn sites. Review of the internet usage policy is recommended.
3. 255.255.255.255 – A spoofed source possibly sending Backdoor Q triggers based on IRC requests. Should be filtered at the ingress interface of the external router.
4. 68.36.170.9 – Possibly a spoofed source packet looking to actively target a web server in an attempt to steal data, passwords or deface a website. Review of the destination host patch levels is recommended.
5. 202.29.28.1 – A host randomly sending crafted pings (source/destination ports both 80) looking for open services.

### **Top Five Target Source Ports**

Only five target ports triggered alerts

1. 8080 – common web proxy
2. 3128 – common squid proxy
3. 80 – typically open port for most external firewalls/filters
4. 515 – port known for Backdoor Q activity based on correlation analysis
5. 1080 – common SOCKS proxy

### **Three Suspicious External Sources**

Based on alerts, and type of alerts, the following three external sources are worthy of noting:

1. 255.255.255.255- Any spoofed address making its way into the network needs further attention. In addition, targeted hosts should be removed from the network for forensic analysis.
2. 66.208.100.206 – User is attempting reconnaissance for a reason. Worthy to watch for additional traffic from this source.
3. 68.36.170.9 – Source host actively targeted an internal resource. Forensics analysis of system compromise and data loss should be executed.

# Analysis Process

## ***Hardware and Software Configurations***

1. IDS Sensor
  - a. Hardware: Intel i386
  - b. Operating System: Fedora Core 2
  - c. Software: Snort v2.2.0 build 30
  - d. Software: ACID v0.9.6b23
2. Analysis/Client
  - a. Hardware: Intel i386
  - b. Operating System: Windows XP
  - c. Software: Ethereal v0.10.7(c)

## ***Process***

My analysis process was done in phases.

- Phase 1: Determine the network configuration
- Phase 2: Find suspicious traffic
- Phase 3: Research similar traffic and detects

Note:

Pages 4 through 6 and 6 through 7 have descriptions of most of the detail of the exact process I worked through for the analysis. Please refer back to those pages as referenced below in order to fully understand the process. Due to space limitations, I could not repeat them here.

During my analysis process, I made heavy use of graphical/automated tools. While conventional tools such as TCPDUMP are great to use, tools such as ethereal automate much of the process, and in my mind, reduce a great deal of error. Filters for Ethereal are much easier to generate and the graphic representations are familiar to most users.

As described earlier, I began the process with MAC address discovery using Ethereal in an effort to determine the physical layout. Knowing the physical layout provides insights into how traffic flows while hardware manufacturers can provide an insight into device selection probability. Then I associated network addresses with those MAC addresses. This showed the flow of traffic and made it much easier to determine inbound vs. outbound flows. A detailed discussion of this process is found on pages 4 through 6.

After determining what networks we were trying to protect, I created a Snort IDS installation to study the traffic. Since this is a University scenario, I wanted to use open source, inexpensive products where possible. I added ACID and MySQL to the Snort implementation (described on pages 6 -7) again to use a graphical tool to view the results. This configuration allowed for the dissemination of suspicious traffic. The ACID

interface provides hyperlinks to relevant data. Rather than “ | grep “ for information, the backend database works to provide ease of access to data.

In each case, I used the alert generated, along with the source/destination IP address combinations to find more information. I hand drew a number of link graphs and traffic flows in order to determine what was happening. Then, using Ethereal filters (described numerous times above) I pulled representative packet data, header information and payload decodes to get a closer look at what was happening. Looking at packet payload and data was crucial in the dissemination of the Backdoor Q correlation as well as the NON-RFC HTTP Delimiter attack. In each case the subtle clue to the detect was in the payload. The data was so random in the search, that writing filters for it would have been near impossible. However, since Ethereal easily prints out payload data, packet review was easier.

The ACID interface provides hyperlinks to relevant data about the detect. Rather than “ | grep “ for information or read through cryptic rule definitions, the backend database works to provide ease of access to data and links to other sources of information. From these categories, I was able to group source and destination addresses and glean some basic detect statistics. Visual representations of each packet triggering an alert are also available further aiding in the effort for more data.

One of the overriding themes in my Track 3 class was, “Google is your friend.” I can remember Mike Poor saying that at least once every day. He was right. Using odd payload contents, flag combinations and key words pulled from ACID and Ethereal, I was able to find large amounts of data explaining and describing what I was seeing.

## Appendix A – Bibliography

- Baker, Andrew R., Brian Caswell and Mike Poor. *Snort 2.1 Intrusion Detection: Second Edition*. Rockland, MA: Syngress, 2004.
- Cox, Kerry and Christopher Gerg. *Managing Security with Snort and IDS Tools*. Sebastopol, CA: O'Reilly Media, Inc., 2004.
- Hall, Eric A. *Internet Core Protocols: The Definitive Guide*. Sebastopol, CA: O'Reilly & Associates, 2000.
- Harper, Patrick. 15 Oct 2004  
<[http://www.internetsecurityguru.com/documents/Snort\\_SSL\\_FC2.pdf](http://www.internetsecurityguru.com/documents/Snort_SSL_FC2.pdf)>.
- Hunt, Craig. *TCP/IP Network Administration*. Sebastopol, CA: O'Reilly & Associates, 1998.
- Koziol, Jack. *Intrusion Detection with Snort*. Indianapolis: Sams Publishing, 2003.
- Marcell, Albert J., and Robert S. Greenfield, eds. *Cyber Forensics: A Field Manual for Collecting, Examining, and Preserving Evidence of Computer Crimes*. Boca Raton: Auerback Publications, 2002.
- McClure, Stuart, Joel Scambray, and George Kurtz. *Hacking Exposed: Network Security Secrets and Solutions, Fourth Edition*. Berkeley: McGraw-Hill/Osbourne, 2003.
- Northcutt, Stephen, et al., *Inside Network Perimeter Security*. Indianapolis: New Riders, 2003.
- Northcutt, Stephen and Judy Novak. *Network Intrusion Detection: Third Edition*. Indianapolis: New Riders, 2003.
- Orebaugh, Angela, Greg Morris, Ed Warnicke, Gilbert Ramirez, ed. *Ethereal Packet Sniffing*. Rockland, MA: Syngress, 2004.
- Shimonksi, Robert J., ed, Will Schmied, Dr. Thomas W. Shinder, Victor Chang, Drew Simonis and Damiano Imperatore. *Building DMZs for Enterprise Networks*. Rockland, Maryland: Syngress, 2003.
- Stevens, W. Richard. *TCP/IP Illustrated, Volume 1: The Protocols*. Boston: Addison-Wesley, 1994.

## Appendix B – snort.conf

```
var HOME_NET 32.245.0.0/16
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var SNMP_SERVERS $HOME_NET
var HTTP_PORTS 80
var SHELLCODE_PORTS !80
var ORACLE_PORTS 1521
var AIM_SERVERS
[64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0/24,64.12.161.0/24,64.12.163.0/24,
205.188.5.0/24,205.188.9.0/24]

var RULE_PATH /etc/snort/rules
preprocessor flow: stats_interval 0 hash 2
preprocessor frag2
preprocessor stream4: disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252

preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 } oversize_dir_length 500

preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode
output database: log, mysql, user=snort password=snort dbname=snort host=localhost
include classification.config
include reference.config

include $RULE_PATH/attack-responses.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/chat.rules
include $RULE_PATH/classification.config
include $RULE_PATH/ddos.rules
include $RULE_PATH/deleted.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/icmp-info.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/info.rules
include $RULE_PATH/local.rules
```

```
include $RULE_PATH/misc.rules
include $RULE_PATH/multimedia.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/p2p.rules
include $RULE_PATH/policy.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/porn.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/shellcode.rules
include $RULE_PATH/smtp.rules
include $RULE_PATH/snmp.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/x11.rules
```