Global Information Assurance Certification Paper

Dameware Mini-Remote Control Buffer Overflow

Mike Rabinowitz
April 7 2004
SANS GCIA v3.4

Abstract: While working on Assignment 3 of this paper, Analyze This, in my analysis of "MY.NET.30.3 activity" in section V, I came across a buffer overflow for an application called Dameware. I had never heard of Dameware before, nor the exploit, which is still only a few months old at the writing of this paper in March of 2004. Also, I was interested to find that there was no existing Snort signature, so my goal for this assignment will be to analyze the exploit in enough to detail to write a rock-solid signature (or signatures) to positively detect the exploit and keep false positives to an absolute minimum.

Assignment 1: Dameware Mini-Remote Control Buffer Overflow

Introduction

Dameware is a "lightweight remote control intended primarily for administrators and help desks for quick and easy deployment without external dependencies and machine reboot. Developed specifically for the 32 bit Windows environment (Windows 95/98/Me/NT/2000/XP/2003), DameWare Mini Remote Control is capable of using the Windows challenge/response authentication and is able to be run both as an application and a service. Some additional features include View Only, Cursor control, Remote Clipboard, Performance Settings, Inactivity control, TCP only, Service Installation and Ping."

http://www.dameware.com/products/

In short it's similar to such tools as PCAnywhere or VNC Viewer with the added capabilities of something like the Microsoft Management Console. All in all, it's a handy suite of tools.

As of the writing of this paper in March 2004, Incidents.org's Internet Storm Center lists the port most commonly assocated with Dameware, port 6129, among the Top Ten Most Attacked Ports.

http://isc.incidents.org/

As visibility of the exploit rises, the number of exploit attempts has lowered, particularly since the end of February. This is presumably the result of network administrators upgrading their version of Dameware. With less exploitable systems available, the exploit itself becomes far more difficult to rely on for results from the point of view of the attacker.

http://isc.incidents.org/port_details.html?port=6129

An Overview Of The Exploit In Action

The earliest mention I could find of the exploit was from December 14 2003, by the discoverer of the vulnerability:

http://sh0dan.org/files/dwmrcs372.txt

The description and technical details that follow here are quoted directly from the link above:

“A buffer overflow vulnerability can be exploited remotely by an unauthenticated attacker who can access the DameWare Mini Remote Control Server. By default (DameWare Remote Control Server) DWRCS listens on port 6129 TCP. By constructing fake communication packets pretending to be a client, we can cause a buffer overflow due to insecure calls to the strcpy (lstrcpyA) functions inside of DWRCS.exe. This overflow is caused after the client finishes sending all pre-authentication information. This includes local username, remote username, local NetBIOS name, Company Name, Registration Name, Registration Key, Date & time, lower case NetBIOS name, IP Address(s) of the client, and Version of the remote client. After this initial packet is sent, the client sends the requested authentication type (in this case NTLMSSP.) If the username is incorrect, the server will respond and then return from the vulnerable function.

When first communicating with the DWRCS, packet dumps showed the server responds with the current Windows Service Pack level, as well as the Operating System Version in the second response packet. The OS can be identified by 16th and 17th bytes of this packet. This information can be used to find valid addresses for our op codes which we can change at will depending on how the server responds. Next if we send all of the variables listed in the description portion of this advisory, the server will respond whether or not authentication succeeded, or if there was an error. During the process of reading in these variables, the server copies these values using strcpy. Since no bounds checking is done, when the authentication fails (or possibly even succeeds), we can overwrite the return address on the stack and have the process call our code.” [1]

For reference, here is one version of the C code:

http://www.securityfocus.com/data/vulnerabilities/exploits/DameWare-MRC-Remote.c

When the code is run, you can watch the exploit unfold across a session such as this, generated on my own closed LAN.

```
23:20:52.329146 10.31.63.203.4364 > 10.31.63.216.6129: S 1532267702:1532267702(0) win 65535 <mss
1460,nop,nop,sackOK> (DF)
23:20:52.329222 10.31.63.216.6129 > 10.31.63.203.4364: S 1850600403:1850600403(0) ack 1532267703 win 65535
<mss 1460,nop,nop,sackOK> (DF)
23:20:52.329345 10.31.63.203.4364 > 10.31.63.216.6129: . ack 1 win 65535 (DF)
23:20:52.333634 10.31.63.216.6129 > 10.31.63.203.4364: P 1:41(40) ack 1 win 65535 (DF)
23:20:52.333843 10.31.63.203.4364 > 10.31.63.216.6129: P 1:41(40) ack 41 win 65495 (DF)
23:20:52.334027 10.31.63.216.6129 > 10.31.63.203.4364: . 41:1501(1460) ack 41 win 65495 (DF)
23:20:52.334045 10.31.63.216.6129 > 10.31.63.203.4364: . 1501:2961(1460) ack 41 win 65495 (DF)
23:20:52.334064 10.31.63.216.6129 > 10.31.63.203.4364: P 2961:4137(1176) ack 41 win 65495 (DF)
23:20:52.334720 10.31.63.203.4364 > 10.31.63.216.6129: . 41:1501(1460) ack 1501 win 65535 (DF)
23:20:52.334944 10.31.63.203.4364 > 10.31.63.216.6129: . 1501:2961(1460) ack 1501 win 65535 (DF)
23:20:52.334975 10.31.63.216.6129 > 10.31.63.203.4364: . ack 2961 win 65535 (DF)
23:20:52.335046 10.31.63.203.4364 > 10.31.63.216.6129: P 2961:4176(1215) ack 1501 win 65535 (DF)
23:20:52.335081 10.31.63.203.4364 > 10.31.63.216.6129: . ack 4137 win 65535 (DF)
23:20:52.335099 10.31.63.216.6129 > 10.31.63.203.4364: . ack 4176 win 64320 (DF)
23:20:52.335150 10.31.63.216.6129 > 10.31.63.203.4364: P 4137:4221(84) ack 4176 win 64320 (DF)
```

```
23:20:52.335680 10.31.63.203.4364 > 10.31.63.216.6129: R 1532271878:1532271878(0) win 0 (DF)
```

As suggested in the technical description above, there are a few crucial packets that are key to the operation of this exploit.

1. The return of the Service Pack Level of Windows

By this packet, the exploit has already delivered as mentioned above all of its "pre-authentication" information.

In the sixth packet, the Service Pack Level of Windows is returned in the data payload. This packet is abbreviated in its padding for purposes of brevity:

```
23:20:52.319621 10.31.63.216.6129 > 10.31.63.203.4363: . 41:1501(1460) ack 41 win 65495 (DF)
0x0000    4500 05dc 58ba 4000 8006 0000 0a1f 3fd8        E...X.@.......?.
0x0010    0a1f 3fcb 17f1 110b 6e4d 39eb 5b53 56f7        ..?.....nM9.[SV.
0x0020    5010 ffd7 99af 0000 1027 0000 9400 0000        P........'......
0x0030    0500 0000 0000 0000 9308 0000 0200 0000        ................
0x0040    5365 7276 6963 6520 5061 636b 2034 0000        Service.Pack.4..
0x0050    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0070    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0080    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0090    0000 0000 0000 0000 0000 0000 0000 0000        ................
```

As mentioned previously, this is revealed in the 16th and 17th bytes of the payload.

2. The overflow of the memory space

You can see in the ninth packet, the client sends the |9090 9090….| string, for purposes of overflowing the memory stacks. Such strings are used regularly for buffer overflows. This is the hexadecimal equivalent for "No operation". These are typically called NOP Sleds, which are "a series of no-operation instructions in the machine code of the target architecture. This series is often used as part of a buffer overflow technique, where the return address in the call stack is modified to point to exploit code. By using a NOP sled, the precise address of the exploit code need not be known — instead, an address in the middle of the NOPs is chosed, causing execution to *slide* into the exploit code." [2]

An example of such usage can be found at this link, for an lprng vulnerability:

http://www.kb.cert.org/vuls/id/382365

You can see this here in the ninth packet. The packet below is quite large and necessary to our further investigation. The packet is only abbreviated slightly at the end where some zero padding is removed:

```
23:20:52.334720 10.31.63.203.4364 > 10.31.63.216.6129: . 41:1501(1460) ack 1501 win 65535 (DF)
0x0000    4500 05dc b367 4000 8006 add3 0a1f 3fcb        E....g@.......?.
0x0010    0a1f 3fd8 110c 17f1 5b54 8cdf 6e4d f1b0        ..?.....[T..nM..
0x0020    5010 ffff 5399 0000 1027 0000 0000 0000        P...S....'......
0x0030    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0040    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0050    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0060    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0070    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0080    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x0090    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00a0    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00b0    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00c0    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00d0    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00e0    0000 0000 0000 0000 0000 0000 0000 0000        ................
0x00f0    0000 0000 0090 9090 9090 9090 9090 9090        ................
0x0100    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0110    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0120    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0130    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0140    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0150    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0160    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0170    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0180    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0190    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01a0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01b0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01c0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01d0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01e0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x01f0    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0200    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0210    9090 9090 9090 9090 9090 9090 9090 9090        ................
0x0220    9090 9090 9090 9090 9090 9090
```

As mentioned at http://sh0dan.org/files/dwmrcs372.txt, the buffer overflow is caused due to insecure calls to the strcpy (lstrcpyA) functions inside of DWRCS.exe.

3. The code to be executed following the stack overflow:

Following the buffer overrun above, the exploit code is delivered

```
0x0220                                   6320 377c        ............c.7|
0x0230    eb03 5deb 05e8 f8ff ffff 8bc5 83c0 1133..]............3
0x0240    c966 b9a2 0180 3088 40e2 fadd 0364 037c        .f....0.@....d.|
0x0250    ee09 6408 8860 ae89 8888 01ce 7477 fe74        ..d..`......tw.t
0x0260    e006 c686 6460 a389 8888 01ce 64e0 bbba        ....d`......d...
0x0270    8888 e0ff fbba d7dc 77de 6401 ce70 77fe        ........w.d..pw.
0x0280    74e0 2551 8d46 6082 8988 8801 ce56 77fe        t.%Q.F`......Vw.
0x0290    74e0 fa76 3b9e 6072 8888 8801 ce52 77fe        t..v;.`r.....Rw.
0x02a0    74e0 6746 68e8 6062 8888 8801 ce5e 77fe        t.gFh.`b.....^w.
0x02b0    70e0 4365 74b3 6052 8888 8801 ce7c 77fe        p.Cet.`R.....|w.
0x02c0    70e0 5181 7d25 6042 8888 8801 ce78 77fe        p.Q.}%`B.....xw.
0x02d0    70e0 6471 22e8 6032 8888 8801 ce60 77fe        p.dq".`2.....`w.
0x02e0    70e0 6ff1 4ef1 6022 8888 8801 ce6a bb77        p.o.N.`".....j.w
0x02f0    0964 7c89 8888 dce0 8989 8888 77de 7cd8        .d|.........w.|.
0x0300    d8d8 d8c8 d8c8 d877 de78 0350 e082 97b7        .......w.x.P....
0x0310    43e0 8a88 8c5a 0344 e298 d9db 77de 600d        C....Z.D....w.`.
0x0320    48fd d2e0 ebe5 ec88 01ee 5a0b 4c24 05b4        H.........Z.L$..
0x0330    acbb 48bb 4108 499d 236a 754e ccac 98cc        ..H.A.I.#juN....
```

```
0x0340   76cc acb5 76cc acb6 01d4 acc0 01d4 acc4   v...v.....
0x0350   01d4 acd8 05cc ac98 dcd8 d9d9 d94e ccac   .............N..
0x0360   8b80 c9d9 c1d9 d977 fe5a d977 de52 0344   .......w.Z.w.R.D
0x0370   e277 77b9 77de 5603 40db 77de 6a77 de5e   .ww.w.V.@.w.jw.^
0x0380   deec 29b8 8888 8803 c884 03f8 9425 03c8   ..).........%..
0x0390   80d6 4a8c 88db ddde df03 e4ac 9003 cdb4   ..J.............
0x03a0   03dc 8df0 8b5d 03c2 9003 d2a8 8b55 6bba   .....].......Uk.
0x03b0   c103 bc03 8b7d bb77 74bb 4824 b24c fc8f   .....}.wt.H$.L..
0x03c0   4947 858b 7063 7ab3 f4ac 9cfd 6903 d2ac   IG..pcz.....i...
0x03d0   8b55 ee03 84c3 03d2 948b 5503 8c03 8b4d   .U........U....M
0x03e0   638a bb48 035d d7d6 d5d3 4a8c 8800 0000   c..H.]....J.....
0x03f0   0000 0000 0000 0000 6e65 546d 614e 6961   ........neTmaNia
0x0400   6300 0000 0000 0000 0000 0000 0000 0000   c...............
```

As pointed out by the author, the problem lies in the insecure calls to the memory made by the Dameware server. After this, any kind of code can be executed with the privilege of the user of Dameware, the Windows Administrator.

Writing A Snort Signature To Detect Dameware Mini Remote Control Buffer Overflow

Snort rules include several possibilities for detecting this exploit. We could make a rule triggering solely off of the Dameware Mini Remote Control server port, using nothing more than the option in the rule header:

alert tcp any any -> any 6129 (msg:"Dameware Mini-Remote Control Buffer Overflow;)

Unfortunately all that does is trigger on any traffic towards tcp port 6129, which of course will generate false positives, as it would trigger any time tcp port 6129 was accessed, whether the destination was running Dameware or not.

Further, Dameware has the built-in capability to run its server from any tcp port the user chooses, thereby rendering such a simple signature useless.

Instead we are forced to examine other ways to create a good signature for this exploit. There are several fields we will use to write this signature.

1. Content

Content allows us to search for specific content in the packet payload. As mentioned in the last section, we are concerned primarily with three patterns for our content

- The return of the Service Pack Level of Windows

```
content: "5365 7276 6963 6520 5061 636b ";
```

  As mentioned previously, this will be delivered from client to server following the pre-authentication, in the sixth packet.

- The overflow of the memory space

content: “90 90 90 90 90 90 90 90 90 90 90 90 90 90”

- The code to be executed following the stack overflow

content: “6320 377c eb03 5deb 05e8 f8ff ffff”

2. Offset

Offset sets the place in the packet from which to begin checking for the pattern. Used in tandem with Content, we want to try to limit the possibilities of a false positive as much as possible.

- The return of the Service Pack Level of Windows

  Despite the fact that in the aforementioned sections, we explained that the Service Pack is returned in the 16th and 17th bytes (of the sixth packet), we are forced to look for something slightly different. What we really need rather than the version of the Service Pack (e.g. 4) is actually the hex code for the words “Service Pack” which begin before the 16th and 17th bytes. This is in fact at the 13th byte offset from the beginning of the TCP packet.

offset: 13

- The overflow of the memory space

  The NOP Sled begins at the 116th byte of the ninth packet

offset: 116

- The code to be executed following the stack overflow

  The actual exploit code is also in the ninth packet and follows the NOP sled at the 271st byte.

  However, we can use the “within” keyword, rather than an offset here. See below.

3. Depth

Depths, which limit the number of bytes from the starting initial offset to be searched, should always be used to not impact performance

4. Flow

Flow can be used to again reduce overhead in only examining certain packets. Since we know that the this buffer overflow can only occur in a tcp session which is already established, we can use this:

```
flow: established;
```

5. Within

As mentioned under offset, to reduce some processing, we can use the Within feature, which we can search for the exploit code "within" a certain amount of bytes after the start of the NOP Sled.

```
within: 256
```

However, we run into a slight problem in using all of these together. The return of the Service Pack happens in a different packet than the NOP Sled and exploit, and not just an adjacent packet. As I discovered with investigating the writing of this rule the stream4 preprocessor (written to do tcp stream assembly, i.e. do a kind of stateful inspection, will not process properly a session like the one for this exploit).

As explained by Matt Kettler here:
http://sourceforge.net/mailarchive/message.php?msg_id=7732577

The stream4 preprocessor will flush its memory following each acknowledgement in a session, which does indeed happen between the sixth and ninth packets. Therefore, we are forced to exclude our use of the Service Pack in detecting this exploit.

Putting together the options we have left brings us to a rule that looks like this:

```
alert tcp any any -> any any (msg:"Dameware Mini-Remote Control Buffer Overflow "; flow: established; content:"|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; offset:116; depth:14; content:" 6320 377c eb03 5deb 05e8 f8ff ffff|"; within: 256;)
```

This was added to the end of the shellcode.rules and the following command was run:

```
$snort –c /etc/snort/snort.conf –A full –P 1460 dameware-dump –l .
```

The following alert was successfully generated:

[**] [1:0:0] Dameware Mini-Remote Control Buffer Overflow [**]
[Priority: 0]
02/22-23:20:52.334720 10.31.63.203:4364 -> 10.31.63.216:6129
TCP TTL:128 TOS:0x0 ID:45927 IpLen:20 DgmLen:1500 DF
A* Seq: 0x5B548CDF Ack: 0x6E4DF1B0 Win: 0xFFFF TcpLen: 20

Assignment 2 Network Detects

Detect 1: Protocol Independent Multicast / Cisco IOS Denial of Service

1. Source of Trace

The trace for this came from a posting on the intrusions mailing list from January of this year: http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00065.html As you can see, Lois Marais asked the basic question “Should I be worried about this alert?” A brief answer was given by Donald Smith, a frequent contributor to the list. Donald’s response seems absolutely correct at first glance, but I thought that the topic warranted a little more analysis.

2. Detect generated by

Unfortunately, Lois does not say precisely what version of Snort she is running. She does however provide in great detail a picture of the alert generated and the traffic related to it:

```
I have these packets coming from a Cisco router in our network. can any one in the list
tell me if I should be worried about these alerts?
the snort web site description indicates that there are no known false positives reported
for this signature


[**] [1:2189:1] BAD-TRAFFIC IP Proto 103 (PIM) [**]
[Classification: Detection of a non-standard protocol or event] [Priority: 2]
12/15-17:04:26.365787 cisco.router -> 224.0.0.13
PIM TTL:1 TOS:0xC0 ID:15438 IpLen:20 DgmLen:38
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0567][Xref =>
http://www.securityfocus.com/bid/8211]

The alert has been generated by the following signature:

"alert ip any any -> any any (msg:"BAD-TRAFFIC IP Proto 103 (PIM)"; ip_proto:103;
reference:bugtraq,8211; reference:cve,CAN-2003-0567; classtype:non-standard-protocol;
sid:2189; rev:1;)"


0000  01 00 5e 00 00 0d 00 30 7b 94 7e 84 08 00 45 c0   ..^....0{.~...E.
0010  00 26 3c 4e 00 00 01 67 cb ac 9e a6 32 03 e0 00   .&<N...g....2...
0020  00 0d 20 00 cd a5 00 01 00 02 00 69 00 14 00 04   .. ........i....
0030  00 00 11 d6 00 00 00 00 00 00 00 00               ............

No. Time Source Destination Protocol Info
```

```
71 21.617334 cisco.router a 224.0.0.13 PIMv2 Hello
76 25.729810 cisco.router b 224.0.0.13 PIMv2 Hello
153 43.037489 cisco.router b 224.0.0.13 PIMv2 Bootstrap
154 43.038547 cisco.router a 224.0.0.13 PIMv2 Bootstrap
169 51.493821 cisco.router a 224.0.0.13 PIMv2 Hello
177 55.427327 cisco.router b 224.0.0.13 PIMv2 Hello
253 81.260530 cisco.router a 224.0.0.13 PIMv2 Hello
258 84.554113 cisco.router b 224.0.0.13 PIMv2 Hello
301 103.041772 cisco.router a 224.0.0.13 PIMv2 Bootstrap
```

Let's examine the rule that this alert triggered on:

"alert ip any any -> any any (msg:"BAD-TRAFFIC IP Proto 103 (PIM)"; ip_proto:103; reference:bugtraq,8211; reference:cve,CAN-2003-0567; classtype:non-standard-protocol; sid:2189; rev:1;)"

This rule doesn't make necessary too many specifics for the alert to trigger. The addresses and ports can be 'any', the IP protocol must be protocol number 103, there are no special content requirements or much of anything else. Basically, there is just one situation that triggers this alert: Anytime that protocol 103 is seen on the wire, regardless of source, destination, or any other field.

This rule, SID 2189 (http://www.snort.org/snort-db/sid.html?id=2189), was written to detect one part of a widespread Cisco IOS Denial of Service Vulnerability for almost any Cisco Operating System that supported IP version 4.

As we will see further on, this rule will certainly trigger for a true positive, but also for false positives.

See section 7 for an improved signature.

3. Probability the source address was spoofed:

Very high. As we'll see in Sections 4 and 5, this packet can easily have a spoofed source address as we are potentially examining a Denial of Service in which no reply to the Source from the Destination would be necessary to complete the execution of the exploit.

4.Description of the attack:

In the third week of July 2003 the internet community at large was made aware by multiple avenues of a serious flaw in all current versions of Cisco IOS (the operating system that runs many routers and other network devices on the internet) that allowed a potential Denial of Service, with no authentication and via very simple packet crafting with already existing tools (in fact, it could barely be considered 'packet crafting'; we'll see below that anyone who can figure out to use the correct switches with hping could execute the attack).

Cisco put the description of the attack in very brief and scary terms: "Cisco routers and switches running Cisco IOS® software and configured to process Internet Protocol version 4 (IPv4) packets are vulnerable to a Denial of Service (DoS) attack."

http://www.cisco.com/warp/public/707/cisco-sa-20030717-blocked.shtml

Also, the CVE number is CAN-2003-0567.

To be a little more specific however, it is in the poor handling of a limited set of IP protocols that a Denial of Service can be executed. Specifically, those protocols are:

53 (SWIPE)
55 (IP Mobility)
77 (Sun ND)
103 (Protocol Independent Multicast - PIM)

5. Attack mechanism:

Since the alert we are analyzing specifically involves PIM, we will examine only scenarios involving PIM, and not IP protocols 53, 55, or 77.

Just to first give an overview on what Protocol Independent Multicast is: IP Multicasting is simply intended to be a bandwidth saving technology, which sends single streams of information to multiple hosts (but not all hosts, as we would call a broadcast). There are quite a few different applications for this, the most common of which are certain routing protocols and even videoconferencing. PIM, or Protocol Independent Multicast is intended to be an improvement on the first generation of Multicast technology, typically DVMRP.

Interestingly, Cisco IOS devices that actually have PIM enabled are not vulnerable to this exploit. It is the devices that do not have PIM enabled that are vulnerable. PIM is typically enabled via one of the following commands:

ip pim dense-mode, ip pim sparse-mode, or ip pim sparse-dense-mode

The actual attack mechanism is very simple:

Any IPv4 packet, with headers set to send IP protocol 103, or PIM, packets can effectively cause a Denial of Service towards the Cisco IOS device. The Cisco advisory makes a big deal of specifying that a TTL of any value must be set, but of course, a TTL will always be set.

So for example one could run the following very simple hping2 command to cause this Denial of Service:

#hping2 -103

This will set the protocol field of the IP header to 103.

Such a packet will cause the device to incorrectly flag the input queue on an interface as full. The input queue can be thought of as the connection table. A full connection table stops the device from accepting further connections, resulting in an effective Denial of Service.

On a device where the PIM processes are already running, the packets are handled correctly.

With the description and the mechanism of the attack now more clear, it is important to point out that in the case of this network detect, we are very likely looking at a false positive. Herein are some reasons backing up this idea:

- As demonstrated by Lois by her question and by calling the source 'cisco.router', and as further pointed out by Donald Smith at http://cert.uni-stuttgart.de/archive/intrusions/2004/01/msg00069.html regarding the MAC address of the source being one that is OEM by Cisco devices, the source is almost certainly an actual Cisco router. Nowhere could I find any suggestion that the Cisco IOS Denial of Service could be executed intentionally from the Cisco IOS
- The destination is the appropriate multicast address that is associated with the normal operation of the PIM multicasting. For evidence of a more active targeting we would expect to see a particular target, but in this case there is none.
- There are multiple types of PIM multicast packets sent. Not only is a PIM 'Hello' packet sent, but also the 'Bootstrap' packets are sent. To execute the Denial of Service, neither a Hello nor a Bootstrap is necessary, only that the IP protocol is set to protocol number 103. It's always possible we have a very sophisticated hacker on our hands thinking far enough ahead to do IDS evasion, but it seems like an unlikely circumstance.
- PIM is a perfectly valid protocol. We are looking at an alert generated by by a very loose signature. As mentioned in the second section of this detect, any packet using the PIM protocol will trigger an alert. We could probably improve this signature:

As mentioned above, any normal use of the protocol should have a multicast address as destination. By doing a negate in the rule, one should be able to eliminate valid uses of the PIM protocol. The result would be a rule as follows:

alert ip any any -> !224.0.0.13 any (msg:"Cisco IOS DoS with PIM"; ip_proto:103; reference:bugtraq,8211; reference:cve,CAN-2003-0567; classtype:non-standard-protocol)

6. Correlations:

First, there are links already cited describing the vulnerability:

- http://www.counterpane.com/alert-v20030718-001.html
- http://www.cisco.com/warp/public/707/cisco-sa-20030717-blocked.shtml
- http://www.securityfocus.com/bid/8211

Someone's SnortSnarf output

- http://www.elilabs.com/~rj/today/snfout.alert/sig/sigsid-2189.html

7. Evidence of active targeting:

There is no evidence of active targeting based on the information we're provided with. In fact, there is a good deal of evidence, as presented in the sections above, that this is not an attempted Denial of Service but normal network traffic.

8 Severity:

Criticality 3
We know little about Lois' network, but the destination is a multicast address. Therefore it stands to reason that those devices that may answer such a multicast could be anything, as little in significance is an unused switch, to something as important as a core router. Therefore we'll take the middle road and assign a three.

Lethality 1
In the case where this was a positive attack, it would certainly warrant giving this alert the highest possible lethality of five, since it would mean a Denial of Service for the destination. However, as discussed above this is most assuredly a normal use of the PIM protocol.

System Countermeasures 3
It's difficult to determine the system countermeasures in place. Are other routers on the network running a patched IOS for this exploit? If not, do they have PIM

enabled, thereby making them not vulnerable? See **9 Defensive Recommendations** for details

System countermeasures cannot be determined.

Network Countermeasures 3
Presumably, there are some network countermeasures in place, but these are unknown. There is a very simple network countermeasure that can be taken. See **9 Defensive Recommendations** for details

Severity -2
(criticality 3 + lethality 1) -(system countermeasures 3 + network countermeasures 3) = -2

9 Defensive Recommendations

There are several recommended defensive suggestions:

- An obvious countermeasure is to upgrade your IOS to the latest usable operating version. A full list of the affected versions is listed at http://www.securityfocus.com/bid/8211/info/. The matrix for upgrades is also listed here: http://www.cisco.com/warp/public/707/cisco-sa-20030717-blocked.shtml#fixes
- Enabling PIM on vulnerable devices is not a perfect fix, but will work in a pinch. On the other hand, enabling services which are not needed can always have other unwanted side effects. For example, any existing or future PIM exploits leave the device in question in danger.
- As detailed at http://www.securityfocus.com/bid/8211/solution/, you could also create access lists denying any particular IP protocol.

10 Multiple choice test question:

How can you do negation of a source or destination in Snort?

A) Use the keyword 'not', e.g. not 192.168.0.0/24
B) Use the symbol '!=', e.g. != 192.168.0.0.24
C) Use the keyword 'ne', e.g. ne 192.168.0.0.24
D) Use the symbol '!', e.g. ! 192.168.0.0.24

Answer is D.

Detect 2: DNS named version attempt

Pease note that this detect was posted to http://cert.uni-stuttgart.de/archive/intrusions/2003/12/msg00049.html, but no questions to respond to were received.

1. Source of Trace
The trace for this came from <http://www.incidents.org/logs/Raw/2002.6.1>
Data from the trace was analyzed further with the use of Snort 2.0.4 and Ethereal 0.9.13a, by running the trace through Snort's default detect rules to find an alert that would have triggered and then further decoding the detected alert with the help of Ethereal.

Please note that protocol checksums are incorrect in the packet traces due to obfuscation of the original IP addresses of the monitored network.

Little information can be gleaned about the network this packet dump was taken from based only on data from the triggered alert.

2. Detect generated by
As mentioned above, the alert was generated by Snort 2.0.4, with the following command:

snort –r 2002.6.1 –c /etc/snort/snort.conf –l 2002.6.1 –Xde –k none

A quick note on options used:

r -read and process tcpdump file
c -use specified rules file
l -alerts will be generated into specified directory
X -display hexadecimal data
d -dumps application layer data
e -displays mac address information
k -checksum mode

*An interesting note on checksum mode. As a first time user of Snort 2.0.4, but having used previous versions in the past, I had never used the k option. However, I found that the logs from incidents.org would not generate alerts for me via snort. However, I found the following post on incidents.org which explained the a change in behavior in later versions of Snort.
http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00215.html

An example of the alert triggered is printed below. A total of eleven such alerts occurred in total when considering by source IP address. The eleven alerts occurred slowly over a few hours. This is ostensibly for the purpose of concealing the attacker's actions or making it at least slightly more stealthy. The

other triggered alerts contain the same source IP address, but a different destination IP address.

[**] DNS named version attempt [**]
07/01-06:10:02.764488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x48
203.122.47.137:11122 -> 46.5.179.78:53 UDP TTL:42 TOS:0x0 ID:5896
IpLen:20 DgmLen:38 Len: 30

An examination of the generated alert gives the following information, some of which is in hexadecimal form above, but is converted below in decimal or other more easily readable format:

Signature Name: DNS named version attempt
Timestamp: 07/01-06:10:02.764488
Source Mac: 0:3:E3:D9:26:C0
Destination Mac: 0:0:C:4:B2:33
Protocol in frame: IP
Packet length: 72 bytes
Source IP Address: 203.122.47.137
Source Port: 11122
Destination IP Address: 46.5.179.78
Destination Port: 53
IP Protocol: UDP (User Datagram Protocol)
Time To Live: 42
Type Of Service: None
IP Identification Number: 5896
IP Header Length: 20 bytes
IP Total Length: 58 bytes
UDP Datagram Length: 38

Unlike the reconnaissance in the first detect, this reconnaissance takes place further up the OSI model, so in fact in this packet there is also a so-called "payload"; in other words, the packet we are examining contains not only the data necessary for communication, but also data that should be transmitted.

For this we'll examine the hexadecimal output, which was taken with tcpdump, via the following command:

#tcpdump –vvv –X –r 2002.6.1 host 203.122.47.137

A quick note on options used:

v -for verbose, extra v for more detail
X -for hexadecimal output
R -read from file

Host -for pattern matching by IP address

06:10:02.764488 203.122.47.137.11122 > 46.5.179.78.domain: [bad udp cksum f7fa!] 4660 [b2&3=0x80] TXT CHAOS)? version.bind. [|domain] (ttl 42, id 5896, len 58, bad cksum a25c!)

```
0x0000    4500 003a 1708 0000 2a11 a25c cb7a 2f89    E..:....*..\.z/.
0x0010    2e05 b34e 2b72 0035 0026 c852 1234 0080    ...N+r.5.&.R.4..
0x0020    0001 0000 0000 0000 0776 6572 7369 6f6e    .........version
0x0030    0462 696e 6400 0010 0003
```

Our examination above of the triggered Snort alert was an examination essentially of only the bytes up until the payload. As mentioned above, the total frame length was 72 bytes, with a 20 byte IP header, and an 8 byte UDP header. And, as specified in the UDP header, since the total UDP length is 38 bytes, the total data payload should then be 30 bytes. So, we will examine those 30 bytes, which are:

```
                                    1234 0080    ...N+r.5.&.R.4..
0x0020    0001 0000 0000 0000 0776 6572 7369 6f6e    .........version
0x0030    0462 696e 6400 0010 0003
```

To take them in order:

1234: The first two bytes are the DNS Transaction ID.
0080: Specifies this transaction will be for a 'Standard query'
0001: Specifies that there is one query
0000: Answer resource records
0000: Authority resource records
0000: Additional resource records

The final 18 bytes will be the most interesting to us, since they are the actual query.

The first set of 14 bytes specifies that this is a query for the version of BIND:

0776 6572 7369 6f6e 0462 696e 6400

The second to last set of two bytes specifies that the type retrieved is text

0010

The last set set of two bytes specifies that the class is class chaos.

To put these together, the source address is sending a request for the text record for the version of BIND using the CHAOS class.

The rule that triggered this alert is from the standard default Snort rulebase and is SID 1616:

dns.rules:alert udp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS named version attempt"; content:"|07|version"; nocase; offset:12; content:"|04|bind"; nocase; offset: 12; reference:nessus,10028; reference:arachnids,278; classtype:attempted-recon; sid:1616; rev:4;)

The rule triggers when traffic towards UDP port 53 to a host on the Snort $HOME_NET when, from 12 bytes offset into the UDP section of the total packet is detected the hexadecimal value of 0x07 and the string 'version' (case-insensitive) and the hexadecimal value of 0x04 and the string 'bind' (case-insensitive).

3. Probability the source address was spoofed:
Low, but certainly not impossble. It's unlikely that the source address is spoofed, since for such reconnaissance and exploitation to be successful, the server would have to return its version to the source.

However, there are at least two possibilities in which the source address of these alerts could be spoofed and the attacker still is able to get the information desired:

- The attacker is able to sniff the replies. Imagine the scenario where the attacker is physically close to a place where the replies to the queries must pass through. He could send his spoofed query from anywhere. As long as it's an address that will be routed past where the attacker has his "sniffing" activated.
- The other possibility is that this alert is just one alert in a larger scan from multiple sources and the attacker hopes to decoy the security analyst into deciding that this particular alert isn't important, when in fact it's a different source to which the necessary information in delivered. However, I checked the logs and there is no other DNS traffic which would match this situation.

If the source address is spoofed of course, and the server has answered, the server will only send its response back to the spoofed address.

4. Description of the attack:
Necessary to the function of the internet's domain name service is the idea of querying. There are many different kinds of DNS queries. The query in this network detect is a query to find out which version of BIND is running on the destination.

In fact, as mentioned earlier in our analysis of this detect, in fact we're looking at a source address running such a query against multiple systems on the $HOME_NET. A query is made by the source to a number of different targets, and the hope of the attacker is that at least one of the destinations will answer, revealing the version of BIND that it's running.

A picture of these logs in brief will lend to the paragraph above visually:

```
02:25:14.194488 203.122.47.137.18949 > 46.5.18.213.domain
05:16:49.844488 203.122.47.137.26914 > 46.5.5.210.domain
05:18:05.804488 203.122.47.137.28115 > 46.5.187.208.domain
06:10:02.764488 203.122.47.137.11122 > 46.5.179.78.domain
06:53:37.054488 203.122.47.137.28442 > 46.5.80.178.domain
07:16:07.304488 203.122.47.137.29521 > 46.5.234.18.domain
09:18:44.864488 203.122.47.137.13053 > 46.5.191.201.domain
09:19:35.304488 203.122.47.137.13825 > 46.5.145.25.domain
09:34:40.224488 203.122.47.137.28010 > 46.5.20.91.domain
12:07:22.384488 203.122.47.137.18056 > 46.5.172.48.domain
12:38:03.664488 203.122.47.137.24950 > 46.5.149.47.domain
```

"BIND (Berkeley Internet Name Domain) is an implementation of the Domain Name System (DNS)…The BIND DNS Server is used on the vast majority of name serving machines on the Internet, providing a robust and stable architecture on top of which an organization's naming architecture can be built. The resolver library included in the BIND distribution provides the standard APIs for translation between domain names and Internet addresses and is intended to be linked with applications requiring name service." [3]

5. Attack mechanism:

There are, ostensibly, two parts to the attack mechanism of this network detect. On the one hand we are primarily examining the DNS query for a BIND version. Such a query is easily made via a standard command like 'nslookup':

```
#nslookup
> set q=txt
> set class=CHAOS
> version.bind
Server:         a.b.c.d
Address:        a.b.c.d#53

VERSION.BIND    text = "8.2.2-P5"
```

And, if such a query is allowed, the version of BIND is returned as above.

On the other hand, gathering the version of BIND off a DNS server (or several DNS servers) gives the attacker the information he needs to now launch something beyond simple reconnaissance.

Listed in the next section are some actual specific exploits, but just imagine some of the possibilities once an attacker knows the version of BIND that is running. The following examples are taken from http://www.isc.org/index.pl?/sw/bind/bind-security.php

Name: "BIND: Remote Execution of Code"
[Added 11.12.2002]
Versions affected: BIND 4.9.5 to 4.9.10
BIND 8.1, 8.2 to 8.2.6, 8.3.0 to 8.3.3
Severity: SERIOUS
Exploitable: Remotely
Type: Possibility to execute arbitrary code.

Name: "DoS internal consistency check"
Versions affected: All BIND 9 version prior to 9.2.1
Severity: SERIOUS
Exploitable: Remotely
Type: Denial of Service

Name: "BIND: Multiple Denial of Service"
[Added 11.12.2002]
Versions affected: 1. BIND 8.3.0 - 8.3.3
2. BIND 8.2 - 8.2.6, BIND 8.3.0 - 8.3.3
Severity: SERIOUS
Exploitable: Remotely
Type: Possibility to execute arbitrary code.

Such exploits allow for many variations of attack: Is the attacker angry with owner of the DNS server? If he creates a Denial of Service towards the nameservers of this owner, and the owner's nameserver can't accept any more requests, then it may not be just the nameserver that is stuck with a Denial of Service, but also the owner's other servers, such as webservers or database servers that suddenly become unreachable via their Fully Qualified Domain Names, and only via IP address (which are not what the "average Joe" types into his web browser)

Other variations could include a scenario where the attacker can execute code, for example a rootkit, that allows him to make changes to the server. He could point DNS requests for the owner's domain in any direction he wanted, or perhaps use the DNS server as a bounce point for further attacks towards the owner or towards somewhere else on the internet.

While DNS (and along with it, BIND) have been an intergral part of everything surrounding the operation of the internet, BIND has been almost notorious for the amount of exploits its provided.

Under the correlations section, you can find some interesting links about the disclosures surrounding BIND.

6 Correlations:

- In regards to the source of 203.122.47.137: This source is noticed in at least two other SANS practicals doing the same type of DNS querying:

  http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00083.html

  http://216.239.39.104/search?q=cache:WymDwr2vpj8J:www.giac.org/practical/GCIA/Ewen_Fung_GCIA.pdf+%22203.122.47.137%22&hl=en&ie=UTF-8

  And 203.122.47.137 is also recorded doing some DNS probing in some logs belonging to the University of Vermont:

  https://rdweb.cns.vt.edu/~lat/log_archives/020518.txt

  All of this takes place in the same general time period of late spring and early summer of 2002.

- SANS lists BIND as its number one vulnerability for Unix systems in its Top Twenty:

  http://www.sans.org/top20/#u1

  These are a few of the more major disclosures regarding various vulnerabilities in BIND over the years that ISC lists on its' site:

  CERT Advisory CA-2002-19 - 06/28/2002
  CERT Advisory CA-2001-02 - 01/29/2001
  CERT Advisory CA-99-14 - 11/10/1999
  CERT Advisory - CA-98.05 - 04/08/1998, revised 11/16/1998

  The CVE that relates directly to this network detect is CVE-1999-0009, but here is a more complete list of the entries from CVE relating to BIND. CVE-1999-0010, CVE-1999-0011, CVE-1999-0024, CVE-1999-0184, CVE-1999-0833, CVE-1999-0835, CVE-1999-0837, CVE-1999-0848, CVE-1999-0849, CVE-1999-0851

  And, for anyone who may feel that using information gleaned from an DNS BIND query can only result in something trivial, try the following search on Google:

  http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=hack+dns+news

This will return many thousands of news articles, groups, and individuals discussing actual hacks or attempted hacks resulting from BIND vulnerability.

7 Evidence of active targeting:

There is indeed some evidence of active targeting present here. As mentioned above, this was not a single query against a single server that we could pass off as a "curious" internet user. Instead, we have one source address attempting to query 11 different hosts in the same /16 segment over a relatively long amount of time (almost 10 hours). This suggests one of a few possibilities:

- The attacker has done already some previous reconnaissance and has an idea of which servers he may be able to query
- The attacker is doing a slow steady scan of the entire /16 network

It is also possible that the attacker is doing a slow scan of an even larger network than this /16, and it is both incidental and inevitable that the scan eventually hits this particular /16 this network.

In the end, it's difficult to determine with certainty.

8 Severity:

Criticality 5
DNS servers are typically at the top of list in importance at any organization. They usually provide a service in common for all of an organization's publicly accessible services, such as mail or webserving. In the case that DNS fails, it is likely that all of an organization's assets will fail.

Lethality 2
The lethality of the the reconnaissance itself is very low, but the lethality of a correctly returned version should be rather high, depending on the version returned. Since according to our packet traces there is no version returned (nor is there any response at all), the lethality of these network detects is low.

System Countermeasures 4
It's difficult to determine the system countermeasures in place, but no reply is ever sent from the targeted destination.

It's very likely that the DNS server is configured in such a way that no answer to this query will be given. See **9 Defensive Recommendations** for details

Network Countermeasures 3
Presumably, there are network countermeasures in place. As mentioned above, there was never a reply from the server to the client, so in all likelihood 'Any' traffic towards these destinations is dropped by the firewall. In the end we actually can't say with any certainty what network countermeasures are in place, so we'll go the middle road and assign a three.

Severity 5
(criticality 5 + lethality 2) -(system countermeasures 4 + network countermeasures 3) = 0

9 Defensive Recommendations

There is a very good set of recommendations specifically regarding DNS from SANS at http://www.sans.org/top20/#u1

A device that can do packet filtering, even application filtering, such as a firewall should always be used. However, sometimes, with a service that must be publicly accessible to the internet, for example, a webserver with port 80 being served, there is no longer the possibility to count on a firewall, or even the ACLs on a border router, as such traffic must be allowed.

When you cannot limit at layers lower down on the OSI model, you must implement stronger security on the upper layers of the OSI model.

In the case of this network detect, it is necessary to secure the DNS server itself, as port 53 is presumably going to be served to the internet. Obviously, running the most recent version of bind is necessary.

"Configure access control via the 'allow-query'directive in the /etc/named.boot file. Add the line 'query-source address * port 5*;*'under the options block of the /etc/named.conf file to force both servers to use UDP port 53 in a server to server DNS query.

Subscribe to BIND security mailing lists to stay current on DNS vulnerabilities, such as the "bind-annouce" list maintained by ISC. Go to: http://www.isc.org/services/public/lists/bind-lists.html." [4]

10 Multiple choice test question:

What type of class is used in a DNS query for the version of BIND?

A. CHAOS
B. Named.ver

C. Version.named
D. Order_bind

Answer to this question: A

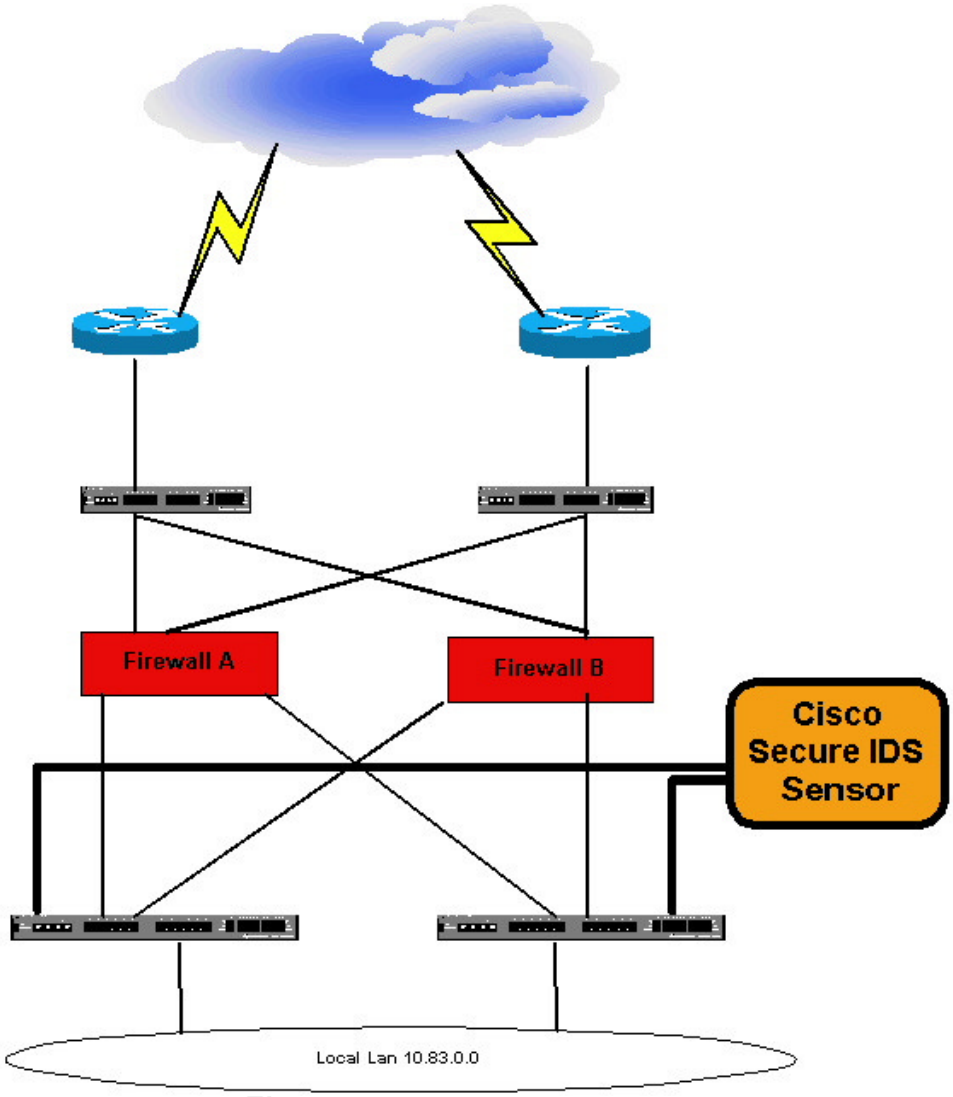
Detect 3: IP Fragmentation

1. Source of Trace
The trace for this came from the production networks of the Managed Security Service Provider that I am employed by and were taken on December 06 2003. The alert was originally triggered with the Cisco Intrusion Detection System.

Data from the packet trace taken in parallel with the generation of the alert was analyzed further with the use of Snort 2.0.4 and Ethereal 0.9.13a, by running the trace through Snort's default detect rules to find if an alert that would trigger and then further decoding the detected alert with the help of Ethereal.

Please note that protocol checksums are incorrect in the packet traces due to obfuscation of the original IP addresses of the monitored network.

The sensor that this alert was generated from was an internal sensor, i.e. it operates on the clean side of the customer's firewall, and receives traffic via switches on the customer's internal LAN. Thus, the RFC 1918 network addresses from the 10/8 network appear legitimately in these packets. The setup in brief looks something like this:



2. Detect generated by

The original detect, generated by the Cisco Intrusion Detection System, looks like this:

csids.1,Sat Dec 6 17:38:41 2003,Sat Dec 6 17:38:41 2003,10008,10001,10005,10006,2,100,SRC_OUTSIDE_PROT_NETWORK,DST_OUTSIDE_PROT_NETWORK,2150,0,211.13.231.126,0,10.83.64.158,0,0.0.0.0,,,,
csids.1,Sat Dec 6 17:39:11 2003,Sat Dec 6 17:39:11 2003,10008,10001,10005,10006,2,100,SRC_OUTSIDE_PROT_NETWORK,DST_OUTSIDE_PROT_NETWORK,2150,0,211.13.231.126,0,10.83.64.158,0,0.0.0.0,- Interval Summary: 3 of total 3 alarms,,

This detect provides some basic information, such as timestamp, source address and source port, destination address and destination port, the signature triggered (2150-0), along with some information that is only necessary to the operation of the sensors in a managed and monitored environment: The string '10008, 10001, 10005, 10006' tells the Cisco IDS console that received the alert details on what sensor the alert came from. This is unrelated to our examination of the network detect.

Notice that there are a total of two alerts generated, however, the second is an "Interval Summary". This is a function of the Cisco IDS, probably for brevity and bandwidth purposes. It means that the same alert triggered three times. So the second alert says that including the first alert, you have a total of three of the same alerts.

The Interval Summary is a condition that can be tuned to different parameters, for example, "how many" consecutive alerts over "a certain time period" should alerts trigger together to create an Interval Summary? You can customize each signature for such parameters.

The signature 2150-0 is listed at http://www.cisco.com/cgi-bin/front.x/csec/idsHome.pl (login required) with the description of Fragmented ICMP Traffic.

Cisco lists the following description of the signature: Triggers when a IP datagram is received with the protocol field of the IP header set to 1 (ICMP) and either the more fragments flag is set to 1 (ICMP) or there is an offset indicated in the offset field. The Boolean equation that describes this is: ICMP AND (MFFLAG OR OFFSET). Fragmented ICMP traffic may be indicative of a denial of service attempt.

While Cisco IDS doesn't natively take a packet trace and save it for the end user, I set one up for this assignment. I then ran the packet trace through the same tools in my first two network detects. I ran the tcpdump-fomatted file through Snort 2.0.4 with the following command:

```
# snort –r icmp-frag –c /etc/snort/snort.conf –l icmp-frag.snort –Xde –k none
```

A quick note on options used:

r	-read and process tcpdump file
c	-use specified rules file
l	-alerts will be generated into specified directory
X	-display hexadecimal data
d	-dumps application layer data
e	-displays mac address information
k	-checksum mode

This result of this command was the following Snort IDS alert:

```
[**] MISC Tiny Fragments [**]
12/06-17:38:41.110304 0:3:FE:3A:87:FC -> 8:0:20:D9:1B:90 type:0x800
len:0x3C
211.13.231.126 -> 10.83.64.158 ICMP TTL:46 TOS:0x0 ID:12631 IpLen:20
DgmLen:28 MF
Frag Offset: 0x0000   Frag Size: 0x0008
0x0000: 08 00 20 D9 1B 90 00 03 FE 3A 87 FC 08 00 45 00  .. ......:....E.
0x0010: 00 1C 31 57 20 00 2E 01 A0 0C D3 0D E7 7E A0 53  ..1W ........~.S
0x0020: 40 9E 08 00 4B 3B 31 57 01 00 00 00 00 00 00 00  @...K;1W........
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00              ............

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

[**] MISC Tiny Fragments [**]
12/06-17:38:41.110707 0:3:FE:3A:87:FC -> 8:0:20:D9:1B:90 type:0x800
len:0x3C
211.13.231.126 -> 10.83.64.158 ICMP TTL:46 TOS:0x0 ID:12631 IpLen:20
DgmLen:28 MF
Frag Offset: 0x0000   Frag Size: 0x0008
0x0000: 08 00 20 D9 1B 90 00 03 FE 3A 87 FC 08 00 45 00  .. ......:....E.
0x0010: 00 1C 31 57 20 00 2E 01 A0 0C D3 0D E7 7E A0 53  ..1W ........~.S
0x0020: 40 9E 08 00 15 3B 31 57 02 00 00 00 00 00 00 00  @....;1W........
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00              ............

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

[**] MISC Tiny Fragments [**]
12/06-17:38:41.111484 0:3:FE:3A:87:FC -> 8:0:20:D9:1B:90 type:0x800
len:0x3C
211.13.231.126 -> 10.83.64.158 ICMP TTL:46 TOS:0x0 ID:12631 IpLen:20
DgmLen:28 MF
Frag Offset: 0x0000   Frag Size: 0x0008
0x0000: 08 00 20 D9 1B 90 00 03 FE 3A 87 FC 08 00 45 00  .. ......:....E.
0x0010: 00 1C 31 57 20 00 2E 01 A0 0C D3 0D E7 7E A0 53  ..1W ........~.S
0x0020: 40 9E 08 00 FD 3A 31 57 03 00 00 00 00 00 00 00  @....:1W........
0x0030: 00 00 00 00 00 00 00 00 00 00 00 00              ............

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

Again, we arrive to the same conclusion from the detects generated by Snort: Three triggered alerts.

Examining the alerts gives some common information among all of them:

Signature Name: MISC Tiny Fragments
Source Mac: 0:3:FE:3A:87:FC
Destination Mac: 8:0:20:D9:1B:90
Protocol in frame: IP
Packet length: 60 bytes
Source IP Address: 211.13.231.126
Destination IP Address: 10.83.64.158
IP Protocol: ICMP
Time To Live: 46
Type Of Service: None
IP Identification Number: 12631
IP Header Length: 20 bytes
IP Total Length: 28
MF: More fragment bit is set (means another fragment will come)
Frag Offset: 0 (fragment starts counting at zero)
Frag Size: 8 bytes

Further what should serve to make this even more interesting is the other traffic that this occurs with, as in the tcpdump here:

```
17:38:41.110304 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.110324 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.110427 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
17:38:41.110707 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.110725 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.110784 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
17:38:41.111484 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.111502 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.111563 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
```

Cisco:
Rule: CSIDS-2150-0: ICMP Fragmentation
Triggered because the IP datagram traced had the protocol field of the IP header set to 1 for ICMP and also because the MF (More Fragment) bit was set to 1.

Snort:
Rule: MISC Tiny Fragments, SID 522
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC Tiny Fragments";
fragbits:M; dsize: < 25; classtype:bad-unknown; sid:522; rev:1;)
Triggered because the MF (More Fragment) bit was set and the size of the data encapsulated was less than 25

3. Probability the source address was spoofed:
There is a definite possibility that the source was spoofed. IP fragmentation can be used for many things and one of them is a denial of service. Since no reply would be necessary in the case of a denial of service, the source address could be spoofed.
In the case that this fragmentation is being used for another purpose, for example, an attack on the destination host where the fragmentation is used only to pass along data "hidden" from the IDS sensor, probably a reply would be needed and the source address could not be spoofed.

4.Description of the attack:

IP Fragmentation can be used for evasion of IDS sensors and other packet filtering devices as well as possible Denial of Service Attacks.

Some examples:

- An attacker who wishes to evade an IDS device might craft a set of packets fragmented in such a way that only the cumulative payload contains the exploit. In the simplest possible example, someone doing an http GET for cmd.exe, could intentionally divide up the GET request into three separate payloads. Instead of one GET for "cmd.exe", he could do a GET over three packets for "cm", d.e", and "xe". Certain IDS devices may not be able to reassemble this into "cmd.exe", and thus the GET is successfully done without alerting anyone.

- Since the assembly of fragmented packets is only done at the destination host, it is possible, such as in the 'Ping of Death' attack, to send fragmented packets across the sum of which the payload of data is large enough to crash the destination host when all packets are reassembled.

- A device that does packet filtering which may not be stateful, like certain routers, may be tricked into allowing past them packets which should have been blocked by an access list. This is because the IP protocol field (e.g. tcp, udp) is only carried in the first fragmented packet. Other fragments of this packet do not have such a field and the device may allow the packet.

In the case of fragmenting ICMP, a packet filtering device may drop the initial inbound echo request, however, will not drop the second packet or any after that with the continuing fragments, as in ICMP, the packet with the continuing fragmented data does not have an ICMP header which specifies type and code (whether it is an echo request, reply, etc.) Arriving fragmented packets at the destination host can cause a denial of service since the destination will spend too

much cpu time on trying to reassemble all the fragments it is keeping in memory, and for which the original packet will never arrive.

Looking back to the whole packet train in at the end of Section 2, there is an echo request with MF (more fragmentation set), the expected fragmented packet arriving, and the echo reply. This happens a total of three times:

```
17:38:41.110304 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.110324 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.110427 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
17:38:41.110707 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.110725 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.110784 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
17:38:41.111484 211.13.231.126 > 10.83.64.158: icmp: echo request (frag 12631:8@0+)
17:38:41.111502 211.13.231.126 > 10.83.64.158: (frag 12631:56@8)
17:38:41.111563 10.83.64.158 > 211.13.231.126: icmp: echo reply (DF)
```

Now, with a further, more detailed analysis we can come to some very interesting conclusions:

First of all, without even examining the issues surrounding the fragmentation, let's look at the entire nine packets. There were in fact, three echo requests (with MF set), the three continuations of that packet, and three echo replies, all in the order expected. However, they as a whole have one very suspicious common factor: The IP Identification number stays the same, not across just one echo-request-fragment-reply (as would be expected), but across all nine packets! Such a thing cannot occur naturally, and is absolutely the result of a crafted packet.

Secondly we should examine the case of the fragmentation itself. The triggered alert itself is in fact triggered on the first packet (for the explanation of how both Cisco and Snort trigger for the alert, see the end of this section).

The first packet, an ICMP echo request, has the MF, or 'more fragments' bit set, and indicates that the packet has been fragmented, and another packet is on the way with the rest of the data.

IP datagrams are sometimes fragmented naturally. MTU, or maximum transmission unit, is a setting on a network device, such as a router that may restrict that the maximum transmitted amount of bytes. This is typically something like 1500 MTU. If the size of an IP datagram is greater than this limit, then it must be fragmented.

5. Attack Mechanism

Looking at our packet at a whole, we have a suspicious problem:

The first packet, with the MF bit set to indicate another packet will come with the rest of the fragment, looks like this: (frag 12631:8@0+). The first number indicates the fragment ID number (taken from the IP ID number), the second number denotes that there will be 8 bytes of data in this fragment, and the third, is a numerical representation of the MF bit. It indicates that more fragments will follow.

The second packet's fragmentation looks like this: (frag 12631:56@8). The first number is again 12631, and it's normal that this should be the same as the first packet's fragment number. This is what allows the destination host to reassemble these two fragments into one packet. The second number is 56, meaning there are 56 bytes of data in the packet, and the 8 is the offset into which this fragment falls (i.e. the place in the last packet from which to continue for reassembly). Also the second packet does not have the MF bit set which indicates this is the last of the fragmented packets.

There are two basic things that are wrong here:

First, if you reassemble the data in bytes from these two packets, including the IP headers, you reach a total of only 104 bytes. There shouldn't be a network device anywhere in the world with such an MTU set, so we can again guess that the packets are crafted.

Secondly, the typical ICMP echo request should contain 40 bytes of data: 8 bytes for the ICMP header, and then 32 bytes of ICMP echo request data. Here we have 8 bytes of ICMP header, followed by 56 bytes of data. This is also abnormal, and especially so since the packets shouldn't have even been fragmented in the first place.

As it turns out, our analysis is somewhat correct. These packet are indeed crafted by someone, however, the purpose turns out to not be malicious, but completely benign. As this analyst searched out the internet for correlations to this alert, the following link was discovered:

http://www.incidents.org/archives/intrusions/msg02850.html

It clearly states that in fact this alert is a false positive, with the following explanation:

"Owner of 211.13.231.126 is J-Stream, who are one of the major streaming company in Japan. They are using their own CDN to distribute their streaming contents. 211.13.231.126 is the IP address of their global traffic balancer.

Once user accesses their streaming contents, their global traffic balancer probes name server of user by sending tiny packets (ping, echo

etc). Your logs are their probing log to your name server. This means that someone in your network access their streaming contents and their global traffic balancer sent probing packets to your name server. These packets are part of their services, not an attack."

6 Correlations:

There is a lot of information out there on IP fragmentation, and particularly on ICMP fragmentation:

There's a discussion of the ICMP fragmentation via PTMU, originally posted to Bugtraq by Anitrez. This is the link to the ISS signature:

http://xforce.iss.net/xforce/xfdb/5975

A good discussion of fragmentation in ICMP on Jim Parker's Firewall-1 website:

http://firewall-1.jimparker.co.uk/icmp_dos_attacks.html

A good overview of the basics is at:

http://www.hackinthebox.org/article.php?sid=4005

7 Evidence of active targeting:

As just demonstrated further above, this appears completely to be a false positive.

8 Severity:

Criticality 3

Importance of the destination host is unknown.

Lethality 1

As mentioned above, this is a false positive.

System Countermeasures 2

What type of host defenses are employed is unknown, but considering the relative technical competence of my contacts at this customer, I presume that at

least generally accepted countermeasures are in place, such as anti-spoofing, and not allowing unnecessary services in and out of the Local LAN.

Network Countermeasures 2

Certainly, the customer has firewalls in place on the outside of the network. What type of defenses are employed is unknown, but considering the relative technical competence of my contacts at this customer, I presume that at least generally accepted countermeasures are in place, such as anti-spoofing, and not allowing unnecessary services in and out of the Local LAN.

Severity 5
(criticality 3 + lethality 1) -(system countermeasures 2 + network countermeasures 2) = 0

9 Defensive Recommendations

Most important defense in the case of fragmentation are stateful devices. Firewalls and Intrusion Detection systems which are fully stateful should be able to detect and even deny packets which are crafted for evasion or denial of service.

At the packet filtering level, a good firewall policy should be created. ICMP is often misunderstood and too great a stress is put upon it as *the* diagnostic tool. Firewall administrators are all too often likely to allow the entire suite of ICMP types and codes in their firewall rule bases when only a certain type and code is necessary. Also limitation by IP addresses should be done.

Further keeping up to date your operating systems is necessary since fragmented IP packets will be reassembled by the host, so the burden in the case of IP fragmentation lays heavily with the task of making sure that systems which are reachable to the outside world via certain ICMP types and codes are not vulnerable to any exploits using those types and codes.

SANS Top Twenty recommends the following ICMP permissions:

“ICMP: block incoming echo request (ping and Windows traceroute), block outgoing echo replies, time exceeded, and destination unreachable messages except "packet too big" messages (type 3, code 4). (This item assumes that you are willing to forego the legitimate uses of ICMP echo request in order to block some known malicious uses.)

In addition to these ports, block spoofed addresses: packets coming from outside your company sourced from internal addresses, private addresses (RFC1918) and IANA reserved addresses. It is also suggested that you block packets bound

for broadcast or multicast addresses. Specifically blocking source routed packets or any packets with IP options set will be advantageous as well.

You should also apply egress filters on your border routers to block spoofed packets from originating from your network. Only allow packets sourced from your assigned addresses to be routed out of your organization."

10 Multiple choice test question:

When a packet of 1580 bytes traverses hops across the internet and reaches a device with an MTU of 1500, what happens?

A. The router sends a MF (Must Fragment) packet back to the sender, so it resends a packet with an appropriate MTU
B. The router disassembles the packet into a two packets less than the size of its MTU and sends them along to the destination. The first packet has the MF (More Fragment) bit set, and the second has the DF (Dual Fragment) bit set
C. The router sets the DF (Don't Fragment) bit and sends the packet back one hop and the packet takes another route with the proper MTU
D. The router disassembles the packet into two packets less than the size of its MTU and send them along to its destination. The first packet has the MF (More Fragment) bit set, and the second has neither the MF nor the DF (Don't Fragment) bits set.

Proper answer is D.

Assignment #3 Analyze This

I. Executive Summary

In giving you an executive level summary, we want to provide for you a "helicopter view" of those incidents and events, in your network, which require action from your teams.

In creating such a summary, we will also point out the roadblocks to making such a summary effective and accurate.

Effectiveness Of Sensor Configuration

- In investigating the alerting produced by your IDS sensor, we noticed an important pattern: No traffic is ever seen between two hosts on your internal network of 130.85.0.0/16. It seems likely that the sensor is placed physically at a chokepoint between the internal network and the internet since only traffic to and from the 130.85/16 network is detected. That is, there is never any traffic seen between addresses from your internal networks, only inbound and outbound alerts.

  A better placement of the sensor might be after the chokepoint. Far more correlation and analysis would be possible if the relationships between your many systems was visible.

  In theory, there are separated networks inside. For example there might be an internal network and a DMZ network. These would be good locations for placement of one or more sensors. For example, a sensor on the DMZ would have let us determine which systems are on the DMZ as well as relationships between the machines on the DMZ, as well as relationships between systems on the DMZ and other networks, including the internet.

- At the moment, only 130.85.0.0/16 is configured as an internal network for Snort. However, as demonstrated in several analyses below, there are other networks besides this one that can be considered as internal to the University. You can find some of these networks in the detailed analyses below, for example in the analysis of "ICMP SRC and DST outside network".

  This can have lead to a massive skewing of the analyses in this paper. For example, the presumptions made in the analyses of MY.NET.30.3 and MY.NET.30.4 are that connections to the Netware services on these hosts are made from the outside, thus heightening the

worry that important systems are accessed by users on the internet. If in fact, these connections are coming from the inside and no connections are made to these systems from the outside, there is far less to worry about.

- A full review of your customized signatures is in order as well. For example, the signatures for the Adore, or Red Worm, i.e. “High port 65535 tcp - possible Red Worm – traffic” are creating far too many false positives based on port.

Effectiveness Of Network Configuration

- Following further on the first point above in the “Effectiveness of Sensor Configuration”: We cannot determine solely from the logs we have that the configuration and location of the networks at the University. From the point-of-view of your Snort sensor, there are only two networks: Your home network of 130.85.0.0/16 and the rest of the world.

  If this is actually the case, you should begin investing an effort in designing a more robust network configuration. Your first goal would be to create a DMZ, a network for “public” access, used exclusively for servers that are accessed by external clients on the Internet, such as Web, FTP and E-Mail servers. By placing these public access servers on a separate isolated network, you provide an extra measure of security for your internal network**.** No connections then should be allowed from the outside towards anywhere on your inside except for the DMZ.

- Following this last point, a review of the IP routing and IP addressing used at the University may be in order

Investigating the Highest Risk Events

- **Assorted traffic to and from 130.85.30.3 and 130.85.30.4** appear to run Netware 5.1 and are among the highest trafficked servers on your network. The many connections to these from the outside of your network to port 524 and others suggest they may be used as either a central login point to the rest of your network or as a directory for your network’s services, or both. The presence of a host management tool known as Dameware on these servers also raises suspicion.

- **Scanning from 66.149.34.140** Full details regarding this IP address are available further down in this report. We suggest you do further investigation in host logs of targets for anything that may have happened “under the radar.” Further, the University should alert Mindspring to the abuse from this system, which consisted at the very least of a massive amount of scanning.

- **Review of Sun RPC Network and Host Security** As you can see in the Brief Analysis in section VII, there are quite a few instances of attempted connections on SunRPC ports. A search of the CVE (Common Vulnerabilities and Exposures) Dictionary for RPC reveals a total of 66 entries or candidates. http://www.cve.mitre.org/cgi-bin/cvekey.cgi?keyword=rpc. Sun RPC has been a regular and popular target for successful expoiting of the years and we recommend a full review of the the addresses listed in the Sun RPC analysis in Section VII for proper patching as well as proper network protection (i.e. not letting outside or other hosts make connections on services which are not necessary).

II. Time of Analysis:

The time period used for this analysis was the Monday through Friday of the week of December 18 2003. The files provided for this week were the following:

Alerts: These are the signature based IDS alerts generated by the University's Snort system
alert.031218
alert.031219
alert.031220
alert.031221
alert.031222

Scans: These are the portscans generated by the University's Snort system
scans.031218
scans.031219
scans.031220
scans.031221
scans.031222

OOS: These are the Out of Spec packets triggered by the University's Snort system
oos_report_031218
oos_report_031219
oos_report_031220
oos_report_031221
oos_report_031222

III. Procedure for Security Analysis

A security incident analysis of the scope and breadth necessary in a network as large as the University's can be difficult purely because of the sheer volume of traffic detected at the sensor. For instance, excepting even the logs generated by the portscan and out-of-spec preprocessors, we still wind up with a whopping 1.5 million events for just five days of logs.

In the interest of presenting meaningful analysis and especially a means for investigating and resolving security related traffic, we start with examining the data culled from the Alerts:

Type and Volume by Alert Type
Top Ten Type and Volume by Source Address
Top Ten Type and Volume by Destination Address

As we investigate these instances we will use for correlation the port scans and OOS data as well as data culled from previous reports and similar instances reported to the general security community on the internet.

In tackling the largest volumes of traffic first, we hope to be able to drill down to highlight those incidents and events which require action from the University's administrative teams.

IV. Network Definitions

The internal network is defined with Snort as MY_NET, which seems to be set to 130.85.0.0/16

Snort's other default network is EXTERNAL_NET which should represent everything else.

V. Tabled Alerts Data

The tables below were derived with variations on the commands following and all types of tables and data within this paper were also derived from similar combinations of commands used in bash and scripting in perl:

```
# cat alert.0312* | grep -v portscan | awk -F** '{print $2}' | sort -rn | uniq -c | sort -rn
```

Type and Volume by Alert Type

No. of Alerts	Signature	% of Total Alerts
24954	MY.NET.30.3 activity	26.21%
24650	MY.NET.30.4 activity	25.89%

13260	Incomplete Packet Fragments Discarded	13.93%
8557	EXPLOIT x86 NOOP	8.99%
5047	TFTP - Internal TCP connection to external tftp server	5.30%
4651	SMB Name Wildcard	4.89%
4484	connect to 515 from inside	4.71%
2314	High port 65535 udp - possible Red Worm - traffic	2.43%
1726	NMAP TCP ping!	1.81%
1512	ICMP SRC and DST outside network	1.59%
1198	High port 65535 tcp - possible Red Worm - traffic	1.26%
542	External RPC call	0.57%
330	Possible trojan server activity	0.35%
256	TCP SRC and DST outside network	0.27%
241	SUNRPC highport access!	0.25%
190	UMBC NIDS IRC Alert IRC user /kill detected, possible trojan.	0.20%
127	FTP passwd attempt	0.13%
108	SMB C access	0.11%
81	UMBC NIDS External MiMail alert	0.09%
50	EXPLOIT x86 setuid 0	0.05%
38	FTP DoS ftpd globbing	0.04%
35	RFB - Possible WinVNC - 010708-1	0.04%
32	EXPLOIT x86 setgid 0	0.03%
32	DDOS shaft client to handler	0.03%
26	TCP SMTP Source Port traffic	0.03%
20	TFTP - External TCP connection to internal tftp server	0.02%
14	EXPLOIT NTPDX buffer overflow	0.01%
10	UMBC NIDS IRC Alert Possible sdbot floodnet detected attempting to IRC	0.01%
9	TFTP - Internal UDP connection to external tftp server	0.01%
9	EXPLOIT x86 NOPS	0.01%
7	TFTP - External UDP connection to internal tftp server	0.01%
7	IRC evil - running XDCC	0.01%
7	EXPLOIT x86 stealth noop	0.01%
7	Attempted Sun RPC high port access	0.01%
6	External FTP to HelpDesk MY.NET.70.50	0.01%
5	External FTP to HelpDesk MY.NET.70.49	0.01%
5	DDOS mstream client to handler	0.01%

4	UMBC NIDS IRC Alert User joining Warez channel detected. Possible XDCC bot	0.00%
4	UMBC NIDS IRC Alert K\:line'd user detected, possible trojan.	0.00%
4	External FTP to HelpDesk MY.NET.53.29	0.00%
3	UMBC NIDS IRC Alert User joining XDCC channel detected. Possible XDCC bot	0.00%
2	UMBC NIDS IRC Alert XDCC client detected attempting to IRC	0.00%
2	SYN-FIN scan!	0.00%
2	Probable NMAP fingerprint attempt	0.00%
2	EXPLOIT identd overflow	0.00%
1	Traffic from port 53 to port 123	0.00%
1	Possible wu-ftpd exploit - GIAC000623	0.00%
1	PHF attempt	0.00%
1	NIMDA - Attempt to execute cmd from campus host	0.00%
1	Happy 99 Virus	0.00%
1	Bugbear@MM virus in SMTP	0.00%

Top Ten Type and Volume by Source Address

No. of Alerts	Signature	% of Total Alerts
12102	68.50.114.89	12.71%
4953	66.149.34.140	5.20%
4567	68.55.241.230	4.80%
4470	MY.NET.162.41	4.70%
3207	66.68.62.250	3.37%
3040	68.57.90.146	3.19%
2696	MY.NET.21.67	2.83%
2575	MY.NET.21.92	2.70%
2512	68.55.62.244	2.64%
2280	151.196.239.212	2.39%

Top Ten Type and Volume by Destination Address

No. of Alerts	Signature	% of Total Alerts
24952	MY.NET.30.3	26.21%
24650	MY.NET.30.4	25.89%
4469	128.183.110.242	4.69%
2565	69.10.132.121	2.69%

2143	169.254.0.0	2.25%
1410	MY.NET.42.3	1.48%
1346	MY.NET.5.92	1.41%
1294	MY.NET.163.76	1.36%
1080	169.254.45.176	1.13%
693	MY.NET.80.232	0.73%

VI. Detailed Analysis

MY.NET.30.3 activity

Introduction

The first “alert” we will examine is not exactly a signature based IDS alert, but the result of a customized rule in Snort approximating the idea of anomaly-based intrusion detection. In anomaly based intrusion detection, the idea is define those sources, destinations, and services which are normal. That which does not match such a definition can then be considered an anomaly and investigated as a security incident.

Example Snort Rule

In this case we have a similar idea, though it may not be used precisely for anomaly based intrusion detection. In any case, the customized Snort rule probably looks something like this:

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

This is a generic rule that will essentially catch any traffic at all from external hosts towards host MY.NET.30.3.

Detailed Analysis

With the large scope of traffic that this rule may trigger on, it is necessary to breakdown further the activity triggering on this host. Below follows the volume and service by port of the top five alerts triggering for “MY.NET.30.3 activity” :

24304 MY.NET.30.3:524
293 MY.NET.30.3:2200
176 MY.NET.30.3:2036
66 MY.NET.30.3:80
41 MY.NET.30.3:6129

With the full breadth and depth of software available now, it is not possible based solely on this information to determine exactly what services MY.NET.30.3 is offering, but we will work based on the most well known service for the port where possible.

Further the above is not actually a complete and total listing of every service that is available on MY.NET.30.3. This is only the top five accessed services.

MY.NET.30.3:524
This accounts for 97% of the traffic triggering this alert. Both TCP and UDP ports 524 are typically associated with Netware server and clients. Netware is an application that can be used for a multitude of applications by operating within active directory type solutions. So administration of services, sharing of files, and other applications across this directory are all possible depending on the configuration of this Netware box.

http://www.novell.com/coolsolutions/netware/features/a_ports_nw5_nw.html

MY.NET.30.3:2200
While this, along with all other traffic towards this host accounts for less than one percent of the traffic, port 2200 is further evidence that this host is a Netware server. According to Novell, this is the port that serves their Web Manager software, with which you can administer even more privileges and software. For example, you can manage user authentication or install servers such as Apache or SSH.

http://www.novell.com/documentation/lg/nw65/pdfdoc/admin_ovw/admin_ovw.pdf

MY.NET.30.3:2036
Port 2036 can also be associated with the Novell Netware suite of applications. The RConsoleJ Agent is served on this port, and according to Novell can provide remote console access to this host, using the RconsoleJ client.

http://www.novell.com/coolsolutions/netware/features/a_ports_nw5_nw.html

MY.NET.30.3:80
Port 80 is well known for serving up HTTP. It's possible that any make and version of HTTP server is running on this machine, and we will cover this further in the next set of sections below.

MY.NET.30.3:6129
Finally, we found also that port 6129 is listening on this host, which may be a bad sign. Port 6129 is typically associated with Dameware Mini Remote Control, an

application similar to things such as VNC or PCAnywhere, allowing from very little to a maximum amount of remote control. While it's a security risk in itself to be serving this port to the whole internet, Dameware was also linked in just December of 2003 to a buffer overflow exploit that could compromise the entire machine. As of the writing of this document, it is number five on incidents.org's Top Attacked Ports. http://isc.incidents.org/. There is more discussion of this in the next sections.

Ramifications

Presumably, the Novell Netware software on this host was installed by the university with the original intent of centralizing Active Directory services. Leaving these services available to the internet was probably not the intention, but from the many varied source addresses in the logs, this appears to be the case in reality.

With this being the case, we have to make a point that it's possible this host has already been compromised. The fact is, while it's certainly possible that the services at the other ports mentioned above were also installed by university administrators, there's also the distinct possibility an existing service (such as Netware) was first compromised, and then the other services were installed by malicious users. This host could currently be serving any type of webserver or FTP server. It could be used for bounce attacks, as a spam relay, or simply as a gateway into further services at the university.

Defensive recommendation:

As mentioned above, the traffic that is allowed to this box should be limited immediately. First, if at all possible, cut off any inbound access to this host from the internet. If that is not possible, try limiting this by IP address. If that is also not possible, try setting up a second type of authentication, like a Radius server to tie into the Netware directory services.

In any of the cases above, this box should first be taken offline for a maintenance. It should be thoroughly inspected for signs compromise or any type of malicious or strange activity. Check with the administrator of this box exactly what its purposes should be. Should it be running an HTTP server? What about Dameware?

Further, after thoroughly cleaning this host up, proper care and patch servicing of the box is necessary not just for Novell, but for all services which must run on it. For example, it is noted in the logs that http and ftp also are served from this box. Are you running the most current versions of the software used? Check the vendor websites for possible security flaws in the versions of software you are using.

Correlations

- MY.NET.30.3 is the Number 1 Talker on the network when sorted by Destination address. You can see this in the "Type and Volume by Destination Address" Table
- This alert is examined by others in their own analyses of your network. Here is an example: http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf
- Here is an interesting link on several different types of exploits for Novell servers: http://www.infosyssec.net/infosyssec/novsec1.htm

MY.NET.30.4 activity

Introduction

This analysis will be very similar to our first analysis. This is again a customized Snort rule, triggering not for specific exploits, but for any traffic towards MY.NET.30.4.

Example Snort Rule

As with our analysis of the "MY.NET.30.3" alerts, it's more than likely that the Snort rule in use for the "MY.NET.30.4" alerts looks something like this:

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

This is a generic rule that will essentially catch any traffic at all from external hosts towards host MY.NET.30.4.

Detailed Analysis

Like our first examination, since this is a customized rule, we will examine the specific traffic directed at this machine. This is a list of the top five services in traffic triggering the "MY.NET.30.4" rule

14118	*MY.NET.30.4:51443*
7467	*MY.NET.30.4:80*
2827	*MY.NET.30.4:524*
125	*MY.NET.30.4:2036*
45	*MY.NET.30.4:6129*

We will discuss herein in a little bit more detail each of these services

MY.NET.30.4:51443
Interestingly, this port is most likely again related to Netware. If you'll note in the table above, this host is again serving a Netware related port, 524. A quick search on our favorite search engine right away reveals that port 51443 is the SSL enabled user interface towards Novell NetStorage

"Novell NetStorage provides simple Internet-based access to file storage. It acts as a bridge between a company's protected Novell network and the Internet, giving users secure file access from any Internet location. Files and folders on a NetWare server can be accessed using either a browser or via Network Neighborhood and Microsoft Web Folders. No Novell Client software is required. Users can securely access files from any IP-enabled machine via SSL (Secure Socket Layers) and HTTPS (Secure HyperText Transfer Protocol)."
http://developer.novell.com/research/appnotes/2002/june/03/a0206033.htm

MY.NET.30.4:80
This is served for HTTP traffic. It can be any kind of webserver running there, and can be related to Netware as the next service is, or can be completely independent.

MY.NET.30.4:524
As with MY.NET.30.3, it seems now that this is host is running Novell Netware server. Netware is an application that can be used for a multitude of applications by operating within active directory type solutions. So, administration of services, sharing of files, and other applications across this directory are all possible depending on the configuration of this Netware box.

MY.NET.30.4:2036
Again, we see another port in common between these hosts: Port 2036 can also be associated with the Novell Netware suite of applications. The RConsoleJ Agent is served on this port, and according to Novell can provide remote console access to this host, using the RconsoleJ client.

MY.NET.30.4:6129
Finally, we found also that port 6129 is listening on this host, which may be a bad sign. Detail on this service was already covered for the "MY.NET.30.3 activity" section.

Ramifications

Like the ramifications mentioned in the report on "MY.NET.30.3 activity", we have to make a point that it's possible this host has already been compromised. The fact is, while it's certainly possible that the services at the other ports mentioned above were also installed by university administrators, there's also the distinct

possibility an existing service (such as Netware) was first compromised, and then the other services were installed by malicious users.
Defensive recommendation:

Again, as with the report on "MY.NET.30.3 activity", important here are several levels of security: Security first at the network level, whether it's via packet level or application level security as well as security on a host based level.

Correlations

- MY.NET.30.4 is the Number 2 Talker on the network when sorted by Destination address. You can see this in the "Type and Volume by Destination Address" Table

- This alert is examined by others in their own analyses of your network. Here are two examples:

  http://www.giac.org/practical/GCIA/Shakeel_Akhter_GCIA.pdf
  http://www.giac.org/practical/GCIA/Holger_van_Lengerich_GCIA.pdf

Incomplete Packet Fragments Discarded

Introduction

The "Incomplete Packet Fragments Discarded" alert is generated by Snort's fragmentation preprocessor. IP Fragmentation can occur both naturally or as a result of deliberate packet crafting. Snort will generate this alert when it detects that it cannot reassemble one or more packets with the fragment fields set into a full, legitimate packet. There is more discussion of fragmentation below.

A link towards the end of the this analysis provides a link to SANS that has a detailed explanation of IP Fragmentation.

Example Snort Rule

As just mentioned, there is not an actual Snort rule that generates this alert, but a Snort preprocessor, which in this case can be either spp_defrag or spp_defrag2.

Detailed Analysis

IP Fragmentation should sometimes occur naturally in IP communications. As a packet traverses hops on a network, each device checks its length. In the case where that packet's length exceeds the MTU (Maximum Transmission Unit) a certain device, that device will fragment the packet, sending the smaller

fragments along to the final destination, where they will be reassembled at the destination host.

However, IP Fragmentation can also be used for evasion of IDS sensors and other packet filtering devices as well as possible Denial of Service Attacks.

Some examples:

- A device that does packet filtering which may not be stateful, like certain routers, may be tricked into allowing past them packets which should have been blocked by an access list. This is because the IP protocol field (e.g. tcp, udp) is only carried in the first fragmented packet. Other fragments of this packet do not have such a field and the device may allow the packet.

- Since the assembly of fragmented packets is only done at the destination host, it is possible, such as in the 'Ping of Death' attack, to send fragmented packets across the sum of which the payload of data is large enough to crash the destination host when all packets are reassembled.

Below is presented a table providing the amount of times the 'Incomplete Packet Fragments Discarded' to groups of sources and destinations:

Vol	Src	Dst
629	MY.NET.21.67	82.129.16.7
548	MY.NET.21.79	82.129.16.7
511	MY.NET.21.92	82.129.16.7
504	MY.NET.21.68	82.129.16.7
455	MY.NET.21.69	82.129.16.7
423	MY.NET.21.67	69.93.3.154
384	MY.NET.21.67	208.155.109.75
383	MY.NET.21.69	208.155.109.75
357	MY.NET.21.79	208.155.109.75
348	MY.NET.21.92	69.93.3.154
347	MY.NET.21.68	69.93.3.154
342	MY.NET.21.69	69.93.3.154
338	MY.NET.21.92	208.155.109.75
336	MY.NET.21.79	69.93.3.154
322	MY.NET.21.68	208.155.109.75
297	MY.NET.21.92	68.163.165.81
278	MY.NET.21.89	81.196.81.105
270	MY.NET.21.89	194.126.143.33
244	MY.NET.21.67	68.163.165.81
239	MY.NET.21.67	69.50.176.215
237	MY.NET.21.67	69.65.7.25
230	MY.NET.21.89	142.177.200.150
229	MY.NET.21.92	213.175.160.223
224	MY.NET.21.92	69.50.176.215
223	MY.NET.21.68	68.163.165.81
212	MY.NET.21.79	69.50.176.215
203	MY.NET.21.79	68.163.165.81
202	MY.NET.21.68	69.50.176.215
179	MY.NET.21.67	213.175.160.223
163	MY.NET.21.79	213.175.160.223
162	MY.NET.21.69	69.65.7.25
161	MY.NET.21.79	69.65.7.25
161	MY.NET.21.68	69.65.7.25
154	MY.NET.21.89	202.155.98.167
149	MY.NET.21.92	146.201.6.10
139	MY.NET.21.89	193.230.240.28
137	MY.NET.21.89	199.232.0.24
122	MY.NET.97.64	64.85.73.31
116	MY.NET.21.89	80.19.60.209
107	MY.NET.21.67	146.201.6.10
100	MY.NET.21.68	146.201.6.10
99	MY.NET.21.89	64.172.65.98
94	MY.NET.21.92	66.205.110.174
91	213.250.62.133:0	MY.NET.153.33:0
90	MY.NET.21.79	146.201.6.10
82	MY.NET.21.92	213.112.70.27
82	MY.NET.21.67	66.205.110.174
71	MY.NET.21.68	213.112.70.27
69	MY.NET.21.79	213.112.70.27
68	MY.NET.21.92	24.132.112.123
65	MY.NET.21.67	213.112.70.27
58	MY.NET.21.79	24.132.112.123
58	MY.NET.21.68	66.205.110.174
54	MY.NET.21.89	66.150.49.250

The purpose of showing this table is to demonstrate the low volume of alerts triggered for any one source and destination combination and further the low amount of source addresses.

The source addresses are all from the University's network. Further, more than 90% of the sources in this table are comprised of just a handful of sources:

MY.NET.21.67
MY.NET.21.68
MY.NET.21.69
MY.NET.21.79
MY.NET.21.89
MY.NET.21.92

This points to a fairly simple explanation, based on the source addresses' close "proximity" to each other. The most likely situation is that their outbound traffic is routed through a device with a lower MTU then the packets are being sent with. This could easily account for normal IP fragmentation. As for Snort not being able to reassemble the packets completely, it's always possible that there is some benign packet loss. In other words, Snort is probably reassembling a lot of fragmentation for these devices and we are looking at a picture of only that traffic which is fragmented and could not be reassembled.

For example, here is a log entry, with MY.NET.21.89, as a source address, triggering a Snort alert:

12/18-16:21:49.500834 [**] [UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC [**] MY.NET.21.89:27649 -> 199.184.165.133:6667

From this we can presume that our theory above may be true: Snort is able to reassemble some fragmentation correctly from this segment.

The other, darker possibility assuming the theory put forth above is incorrect, is that these 'Incomplete Packet Fragments Discarded' alerts are in fact malicious, which has several implications:

a) the machines are already compromised by outsiders
b) mischievous users on your network are misusing your resources

However, this seems very unlikely. The reason behind this is that if the machines were being used for malicious purposes, we would expect to see a lot more intrusion alerts surrounding these boxes. In fact, there's very little mischievous traffic during the week from these hosts There's very little traffic for them aside from outgoing IRC traffic, and even those alerts can be normal false positives.

Ramifications

As mentioned in the above sections, we don't think there is any real security threat here.

To reduce the appearance of these alerts in the logs and to thus be able to focus on real security matters, we would recommend simply making a short network investigation of this traffic.

Find out the path of the route out to the internet for these machines. There is almost certainly a device on your networks that has an MTU that may be shorter than is necessary.

Take a packet trace of the traffic leaving the source addresses mentioned in this section and compare their MTU with the MTU of the hops on the way to the internet.

It may be possible to increase the MTU at one of these hops, thereby limiting the appearance of this alert in your logs.

Defensive recommendation:

In the case where these alerts are in fact malicious or even just mischievous, your investigation should start on the source addresses themselves.

As mentioned, there is little in the way of detected alerts to or from them aside from the the 'Incomplete Packet Fragments Discarded' and some IRC traffic, so it is unlikely that these machines are compromised.

Nevertheless, check the machines for signs of compromise of mischief. Check the logs for who has been logging and what they have been doing. Check what services are running. Update software and patch levels as necessary.

Correlations

- This was examined also in the SANS paper below: http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html
- There's some nice discussion of the possible false positives resulting from this from the author of the preprocessor: http://www.geocrawler.com/archives/3/4890/2001/2/350/5151528/
- Two of the sources mentioned above also appear in the "Type and Volume by Source Address" Table

EXPLOIT x86 NOOP

Introduction

Though we couldn't find any rule in the default Snort rulebase with this exact signature name, it's likely the case that this is just a modified version of an alert like SID 648, 'SHELLCODE x86 NOOP'.

Like this and the other NOOP (or 'No Operation') based signatures, Snort is checking for the presence in the hex dump of each packet for something like this: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90|". This is a perfect example of how a buffer overflow may be executed. By using the NOOP, a clever coder may take advantage of insecure usage of memory in poorly coded programs and insert malicious code of his own in the insecured memory space.

Example Snort Rule

The rule probably looks something like this:

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

Detailed Analysis

Based on that, this alert should trigger anytime we see a series of NOOPs in a row in a certain packet. Unfortunately, not knowing the content of the exact rule, we cannot determine how many NOOPs it takes in a packet to trigger one of these alerts. Nor do we know if there is any other content in addition that is necessary to trigger this alert. Further, since NOOPs can occur in legitimate traffic (for example, in the simple downloading of images or other binary data over http), it's not going to be easy for us to say what traffic is legitimate and what traffic should be considered dangerous.

There are more than 8,000 individual occurrences of this signatures in the logs, however more than 80% of it can be attributed to three source addresses:

Vol	Source
4953	66.149.34.140
1627	210.183.217.72
606	212.202.57.13

Further it's always towards port 80, which is a good sign. This points towards the better possibility that it's false positive, as mentioned above, possibly the simple downloading of binary image files.

The problem with a detailed analysis on this signature from our side is that unfortunately it's difficult to determine without the actual URLs involved.

We can tell you that the sources above reach many different destinations on your network.

Ramifications

Of course, the problem here (and with all such shellcode exploits) is the difficulty in determining whether the alert is a false positive or not.

In the case that it is a false positive, there's not much to worry about, save for preventing the amount of false positives triggering, which we'll cover below.

In the case that it is an actual attack, a full forensics investigation of the system and the events will certainly be necessary.

Defensive recommendation:

The first step towards our defensive recommendations is determining whether these are false positives or not.

The easiest way to check this is the logs of the destination systems. Check the logs that record which URLs are accessed and correlate these against the timestamps in the 'SHELLCODE x86 NOOP' alerts. If we see normal GETs at these timestamps for such binary files as jpeg or gif image files, then we likely have a situation of pure false positives.

The other possibility is that these are actual buffer overflow attempts. In this case the first step that should be taken is an investigation in full of the destination webservers to make sure that they have not been compromised.

In the case that these are actual buffer overflow attempts, we have included the public information regarding the three top offending sources above so that you can contact their administrators:

66.149.34.140
==========

```
140.34.149.66.in-addr.arpa      name = user-119a8kc.biz.mindspring.com.
OrgName:    EARTHLINK, INC
OrgID:      ERSD
Address:    1375 PEACHTREE ST, LEVEL A
City:       ATLANTA
StateProv:  GA
PostalCode: 30309
Country:    US

ReferralServer: rwhois://rwhois.admin.atl.earthlink.net:4321/

NetRange:   66.149.0.0 - 66.149.255.255
CIDR:       66.149.0.0/16
NetName:    EARTHLINK-3-SDSL
NetHandle:  NET-66-149-0-0-1
Parent:     NET-66-0-0-0-0
NetType:    Direct Allocation
NameServer: ITCHY.MINDSPRING.NET
NameServer: SCRATCHY.MINDSPRING.NET
Comment:    ADDRESSES WITHIN THIS BLOCK ARE NON-PORTABLE
Comment:    **********************************************************
Comment:    Reassignment information for this block is
Comment:    available at rwhois.admin.atl.earthlink.net port 4321
Comment:    **********************************************************
RegDate:    2001-06-28
```

Updated: 2003-06-09

TechHandle: DAE4-ARIN
TechName: Domain Administrator, Administrator
TechPhone: +1-404-815-0770
TechEmail: arinpoc@corp.earthlink.net

OrgAbuseHandle: ABUSE60-ARIN
OrgAbuseName: ABUSE TEAM
OrgAbusePhone: +1-404-815-0770
OrgAbuseEmail: abuse@abuse.earthlink.net

OrgTechHandle: ELNK-ORG-ARIN
OrgTechName: EarthLink, Inc.
OrgTechPhone: +1-404-815-0770
OrgTechEmail: arin_tech@lists.corp.earthlink.net

ARIN WHOIS database, last updated 2004-02-21 19:15
Enter ? for additional hints on searching ARIN's WHOIS database.

210.183.217.72

==========

** server can't find 72.217.183.210.in-addr.arpa.: NXDOMAIN
inetnum: 210.178.0.0 - 210.183.255.255
netname: KRNIC-KR
descr: KRNIC
descr: Korea Network Information Center
country: KR
admin-c: HM127-AP
tech-c: HM127-AP
remarks: **
remarks: KRNIC is the National Internet Registry
remarks: in Korea under APNIC. If you would like to
remarks: find assignment information in detail
remarks: please refer to the KRNIC Whois DB
remarks: http://whois.nic.or.kr/english/index.html
remarks: **
mnt-by: APNIC-HM
mnt-lower: MNT-KRNIC-AP
changed: hostmaster@apnic.net 19981124
changed: hostmaster@apnic.net 20010606
status: ALLOCATED PORTABLE
source: APNIC

person: Host Master
address: 11F, KTF B/D, 1321-11, Seocho2-Dong, Seocho-Gu,
address: Seoul, Korea, 137-857
country: KR
phone: +82-2-2186-4500
fax-no: +82-2-2186-4496
e-mail: hostmaster@nic.or.kr
nic-hdl: HM127-AP
mnt-by: MNT-KRNIC-AP
changed: hostmaster@nic.or.kr 20020507
source: APNIC

212.202.57.13

==========

13.57.202.212.in-addr.arpa name = port-212-202-57-13.reverse.qsc.de.
inetnum: 212.202.34.0 - 212.202.62.255
netname: HOME-DYNAMIC-NET
descr: QSC AG Dynamic IP Addresses
country: DE
admin-c: QSC1-RIPE
tech-c: QSC1-RIPE
status: ASSIGNED PA
mnt-by: QSC-NOC

```
mnt-lower:    QSC-NOC
remarks:      ************************************
remarks:      * For spam, portscans, hacks, ... *
remarks:      * please contact to abuse@qsc.de  *
remarks:      ************************************
changed:      roland.haenel@NOSPAM.qsc.de 20030728
source:       RIPE
route:        212.202.0.0/17
descr:        QSC AG
origin:       AS20676
mnt-by:       QSC-NOC
mnt-lower:    QSC-NOC
changed:      oliver.schueten@NOSPAM.qsc.de 20030612
source:       RIPE
role:         QSC Internet Services
address:      QSC AG
address:      Mathias-Brueggen-Str. 55
address:      D-50829 Koeln
address:      Germany
phone:        +49 221 66 98 000
fax-no:       +49 221 66 98 009
e-mail:       abuse@qsc.de
remarks:      ********************************************
remarks:      QSC AG - Network Design Department
remarks:
remarks:      Fuer Fragen zu SPAM, Portscans, Trojanern
remarks:      usw. wenden Sie sich bitte an abuse@qsc.de
remarks:
remarks:      To report SPAM/UCE/Portscans/Hacks please
remarks:      contact abuse@qsc.de.
remarks:
remarks:      For peering requests, BGP policy changes
remarks:      etc. contact peering@NOSPAM.qsc.de. For
remarks:      Routing issues noc-ip@NOSPAM.qsc.de. Please
remarks:      remove NOSPAM. from email address.
remarks:      ********************************************
admin-c:      RH168-RIPE
tech-c:       RH168-RIPE
tech-c:       OS101-RIPE
tech-c:       RW590-RIPE
tech-c:       BF359-RIPE
tech-c:       MD1900-RIPE
nic-hdl:      QSC1-RIPE
mnt-by:       QSC-NOC
changed:      rha@NOSPAM.qsc.de 20040127
source:       RIPE
```

In any event, it would be recommended to take some further steps for defense:

Lock down the webserver:
http://www.lokboxsoftware.com/SecureWin2K/
http://www.apachefreaks.com/apache_tutorials.php

Consider application-level filtering and control:
http://www.ubizen.com/c_products_services/3_ubizen_dmzshield/c3312.html
http://www.checkpoint.com/products/protect/smartdefense.html

Correlations:

- http://www.snort.org/snort-db/sid.html?sid=648

- The top source involved here, 66.149.34.140, was noticed by several people to be involved in suspicious activity during December 2003 http://www.google.com/search?hl=en&lr=&ie=UTF-8&oe=UTF-8&q=%2266.149.34.140%22
- Also, our top source appear in Section VI, as the the number two top talker by source IP address.

TFTP - Internal TCP connection to external tftp server

Introduction

This alert triggers on a customized rule, for any TFTP traffic from the internal "MY.NET" towards external hosts.

TFTP, or Trivial File Transfer Protocol, is a simple form of the File Transfer Protocol (FTP). TFTP uses the User Datagram Protocol (UDP) and provides no security features. It is often used by servers to boot diskless workstations, X-terminals, and routers.

http://www.webopedia.com/TERM/T/TFTP.html

Example Snort Rule

alert UDP $MY_NET any <> $EXTERNAL_NET 69 (msg: "Internal TCP connection to external tftp server";)

Note the "<>" which watches for the ephemeral, or high-port responses of such traffic.

Detailed Analysis

A more detailed parsing of the alerts, shows two main occurrences in which TFTP is being used from MY.NET towards hosts on the internet serving up TFTP:

1. MY.NET.42.3 -> 69.10.132.121:69

   What follows here is some publicly available information about the destination address:

   [mrabinowitz@replica mrabinowitz]$ whois 69.10.132.121@whois.arin.net
   RackForce Hosting Inc. RACKFORCE-1 (NET-69-10-128-0-1)
   69.10.128.0 - 69.10.159.255
   Memset Ltd. MEMSET-MAINNET (NET-69-10-132-0-1)
   69.10.132.0 - 69.10.132.255

[mrabinowitz@replica mrabinowitz]$ nslookup 69.10.132.121
Non-authoritative answer:
121.132.10.69.in-addr.arpa name = martiab1.miniserver.com.

A quick look at the names as well as the URL http://www.miniserver.com shows that this is the IP address of a hosting service. martiab1.miniserver.com is probably the dedicated server of one of their customers (quick guess: Martia B.).

2. MY.NET.70.225 -> 68.61.18.36:69

[mrabinowitz@replica mrabinowitz]$ whois 68.61.18.36@whois.arin.net
[whois.arin.net]
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. MICHIGAN-B-6 (NET-68-61-0-0-1)
68.61.0.0 - 68.61.255.255

[mrabinowitz@replica mrabinowitz]$ nslookup 68.61.18.36
36.18.61.68.in-addr.arpa name =
pcp03529920pcs.pthurn01.mi.comcast.net.

Best guess for this one is simply that it is a dial-up or broadband account belonging to a Comcast internet service subscriber.

The next question that follows is “why”? See Ramifications section for possible reasons.

As mentioned above, TFTP can be used for several different things, such as diskless booting. As the name suggests it can also simply transfer files.

Ramifications

The ramifications of such traffic can be varied.

The sources can potentially be misconfigured hosts on your network, currently set to boot off an image on a network host (in this case the destinations). It could also be the pointed actions of a user on your network downloading files that may or may not be malicious.

Defensive recommendation:

Investigate the source hosts in question as well as the user accounts logging into these machines and the actions they are taking.

Are these legitimate, expected actions?

Are these misconfigured hosts?

Recommendations for this alert would have to be based upon your further investigation. However, if TFTP outbound is not absolutely necessary it should be blocked at your perimeters. If such file transfers are definitely necessary, certainly consider use of Antivirus software on the source hosts.

Correlations

- Such alerts seem to be regularly seen on your network, as referenced in the following papers:
  http://216.239.41.104/search?q=cache:ZFis7CRgvOQJ:www.giac.org/practical/GCIA/Jamell_Creque_GCIA.pdf+%22TFTP+-+Internal+TCP+connection+to+external+tftp+server%22&hl=en&ie=UTF-8
  http://216.239.41.104/search?q=cache:he1C4xV6qt0J:www.giac.org/practical/GCIA/Holger_van_Lengerich_GCIA.pdf+%22TFTP+-+Internal+TCP+connection+to+external+tftp+server%22&hl=en&ie=UTF-8
  http://216.239.41.104/search?q=cache:Abyd6Gss2e8J:www.whitehats.ca/main/members/Herc_Man/Files/Al_Williams_GCIAPractical.pdf+%22TFTP+-+Internal+TCP+connection+to+external+tftp+server%22&hl=en&ie=UTF-8
  http://216.239.41.104/search?q=cache:lILwdCP5tYIJ:thing.fwsystems.com/build/sburns/Kyle_Haugsness_GCIA.doc+%22TFTP+-+Internal+TCP+connection+to+external+tftp+server%22&hl=en&ie=UTF-8
- Also the source address examined above, MY.NET.42.3, appears as the sixth highest talker by source address in Section VII.

SMB Name Wildcard

Introduction

The alert triggered here is an alert to describe the function of certain Windows systems to do the “NetBIOS name query” which is completely normal but can return sensitive information. The general rule of thumb for such a signature to let it trigger only inbound requests. Since it is normal occurring traffic for Windows machines we want to reduce the amount of false positives just to alert us when there is a potential for external information leakage.

http://www.snort.org/docs/faq.html#4.15

Example Snort Rule

The rule itself may or may not be customized. An older standard version of this rule looks like the following:

alert UDP $EXTERNAL any ->$MY_NET 137 (msg:
"IDS177/netbios-name-query"; content:
"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00
00|";)

http://archives.neohapsis.com/archives/snort/2000-01/0222.html

However, all the alerts resulting in this report show traffic in the direction of MY_NET -> EXTERNAL (and it should be noted, not MY_NET <> MY_NET).

So in all likelihood, the alert triggers for MY_NET <> EXTERNAL, like this:

alert UDP $MY_NET 137 <> $EXTERNAL 137 (msg:
"SMB Name Wildcard"; content:
"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAA|00
00|";)

Detailed Analysis

There are over five hundred unique source and destination combinations. The majority constitute the following scenario:

Hosts on MY_NET -> Hosts on the 169.154/16 network

The one 169.154/16 range is a range reserved by IANA. Further detail can be gleaned from RFC 3330.

http://www.faqs.org/rfcs/rfc3330.html

Just to summarize however, it's a range which can be used by machines that are configured to do DHCP for IP address assignment but that assignment fails. This behavior is seen on several different flavors of Microsoft Windows default installations, which is significant to our analysis here since SMB is the protocol used. SMB, or Server Message Blocks, is very typically used for Windows networks for purposes of file sharing and navigation of Windows domains.

To return for a moment to examination of the 'SMB Wildcard Name' alert: As mentioned in the Introduction, this alert is triggered by Windows traffic that is actually quite normal. When a Windows machine has the IP address, but not the NetBIOS name, of the host it wants to connect with, it runs the NetBIOS name query.

http://archives.neohapsis.com/archives/snort/2000-01/0222.html

Seen the combination of these two factors, the most likely assessment of this traffic is that the destination hosts are most likely just boxes located within MY_NET, that is, internally at your organization, and they are hosts that were never correctly addressed by your administrators. There is no major insecurity to this, particularly since this range of IP addresses won't be routed over the internet. Further comments on this are in the Ramifications section.

There is also traffic from MY_NET towards some single hosts on the internet. There is nothing that suggests any maliciousness in such traffic, but if such connections are successful, it is possible that harmful files could be downloaded into your network. Again, there is further discussion relating to this in the Ramifications section.

Ramifications

Regarding the first scenario in which MY_NET hosts are doing NetBIOS name queries towards the 169.154/16 network: Presumably, the 169.154/16 network, despite not being a part of Snort's MY_NET, must be internal to your organization, since such a range will not be routed over the internet. There aren't great security ramifications to this, but you may correlate this with some other unwanted effects on your network, for instance, a lot of noise from 169.154/16 hosts.

For the second scenario, the most crucial factor is whether these requests are actually reaching their destinations and making successful connections. In that case, there is the possibility for the user on your network to perhaps download malicious or dangerous files into your network.

And to assess the NetBIOS name query's overall possible dangers: By accessing system name table information, individuals can obtain information which can be used to launch an attack. Information available includes: 1. The NetBIOS name of the server. 2. The Windows NT workgroup domain name. 3. Login names of users who are logged into the server. 4. The name of the administrator account if they are logged into the server.

http://archives.neohapsis.com/archives/snort/2000-01/0222.html

As mentioned earlier, the greatest danger resulting from this is when NetBIOS connections are allowed inbound, not outbound.

Defensive recommendation:

Only outbound traffic is noticed. We presumed in assessing your 'SMB Name Wildcard' rule that a $MY_NET 137 <> $EXTERNAL 137 access was in place. However, if a '->' was used instead in this rule instead of '<>' then we may be missing inbound requests. This should be corrected immediately.

Do not allow such requests outbound unless absolutely necessary, and do not allow inbound requests at all. As described in the Ramifications section, quite a lot of information can be gleaned from the NetBIOS name query.

Correlations

- There is along history to both this signature and it's differently named equivalents outside of Snort:
  http://www.sans.org/resources/idfaq/port_137.php
  http://www.incidents.org/archives/intrusions/msg09242.html
- And for more information just about NetBios Name queries:
  http://www.ubiqx.org/cifs/

Connect to 515 from inside

Introduction

This is again, a rule customized by your organization. It most likely triggers on requests from MY_NET towards 'Any' for port 515.

Port 515 is most typically associated with printing services in a Unix environment, and is known as registered as 'spooler' with IANA.

http://www.iana.org/assignments/port-numbers

It is typically associated with the Unix flavored binaries lpd or lprng.

Example Snort Rule

The most likely used Snort rule should like something like this, based on the traffic in the logs:

alert tcp $MY_NET any <> $EXTERNAL 515 (msg: "connect to 515 from inside";)

It's also possible that the destination could be 'Any' rather than $EXTERNAL, but based on the fact that we didn't find in the logs any connections to hosts on MY_NET, we made the above assumption.

Detailed Analysis

Except for a fraction of less than a percent, there is just one unique traffic pattern that causes this signature to appear in the Top Ten:

MY.NET.162.41 -> 128.183.110.252:515

Some extra information on the destination host:

[mrabinowitz@replica mrabinowitz]$ nslookup 128.183.110.252
Non-authoritative answer:
252.110.183.128.in-addr.arpa name = forest.gsfc.nasa.gov.

[mrabinowitz@replica mrabinowitz]$ whois 128.183.110.252@whois.arin.net
[whois.arin.net]

OrgName: National Aeronautics and Space Administration
OrgID: NASA
Address: AD33/Office of the Chief Information Officer
City: MSFC
StateProv: AL
PostalCode: 35812
Country: US

NetRange: 128.183.0.0 - 128.183.255.255
CIDR: 128.183.0.0/16

A quick examination of just via web browsing of http:// forest.gsfc.nasa.gov shows a page seemingly dedicated to a NASA project on forests. There is, interestingly enough, information via one of the links about an application, which seems to be affiliated with folks on this website, called the Ecosystem Modeling Interface. There is even a sort of tutorial page and a 'File' Menu with 'Print' in it.

So, in all likelihood, the traffic towards the destination in these alerts is benign.

The best guess is that one of the University's students is making use of this application or possibly another NASA application, that has the potential to print to NASA's forest.gsfc.nasa.gov server.

There is quite a lot of this traffic (it spans the entire breadth of the five days worth of log files constantly), so unless there is a constant print job going to the NASA server, probably the firewall blocks an already started print job, and the application itself doesn't allow its spooled print job to ever time out.

Ramifications

There are quite a few exploits surrounding port 515 and the lpd printing daemon. For example, the LPRng User-Supplied Format String Vulnerability:

http://www.securityfocus.com/bid/1712/info/
In any case, the exploitable service is on the destination host, not the MY_NET source.

This particular vulnerability was also used in the more well-known Ramen worm.

http://www.whitehats.com/library/worms/ramen/

There is no possibility that the source is infected with Ramen since traffic from the host would be pointed towards many more services and destinations.

Defensive recommendation:

If it all possible, it's recommended that you shouldn't be allowing printing outbound to the internet. However, as mentioned above, based on the repeated and constant traffic matching the one pattern, this is probably already the case.

In this case, you should probably just contact the user and investigate the source machine current state, particularly what software is installed and forcing such a printing job.

Correlations

- Certainly, there is plenty of information out there about Ramen, however, as determined above, this is most likely not Ramen traffic.
- Both source and addresses involved with this particular alert appear respectively in our Sections VI and VII, as Top Talkers by Source and Destination Addresses.

High port 65535 udp - possible Red Worm - traffic
High port 65535 tcp - possible Red Worm - traffic

Introduction

The two signatures above actually account for the seventh and tenth spots in the Top Ten Signatures, but will be discussed as a whole, since the basic rules, events, and analysis can be examined the same way.

The Red Worm, more popularly known as the Adore Worm, is a worm that spreads in Linux systems by use of any of a combinations of already existing vulnerabilities. These vulnerabilities concern BIND named, wu-ftpd, rpc.statd and lpd services.

Significant to our investigation here will be the fact that the worm is associated with port 65535, which is used as a backdoor. The backdoor activates when it receives a ping packet with correct size, and opens a shell in the port 65535.

http://www.f-secure.com/v-descs/adore.shtml

Example Snort Rule

This is most likely a custom rule.

Based on the fact that there is not a single instance of this alert in which port 65535 is not either a source or destination port, we can assume that it is critical to the rule triggering.

Also alerts seem to trigger in any direction.

From these factors, we can determine that the rules should look something like this:

alert tcp any any <> any 65535 (msg: “High port 65535 tcp - possible Red Worm - traffic ";)

alert tcp any 65535 <> any any (msg: “High port 65535 tcp - possible Red Worm - traffic ";)

alert udp any any <> any 65535 (msg: “High port 65535 udp - possible Red Worm - traffic ";)

alert udp any 65535 <> any any (msg: “High port 65535 udp - possible Red Worm - traffic ";)

Detailed Analysis

The main problem with this analysis is that a lot of the alerts we see are clearly false positives. For example, take this connection:

MY.NET.25.68:65535 -> 209.133.120.80:25

What we have here is a very simple mail communication. The host on your network is only sending a mail to 209.133.120.80, on the normal smtpd port of 25. But notice the source port of 65535 from which the source address is sending its mail. It’s port 65535, which as a ‘high port’ (almost the highest port actually), can be randomly used as a source port in all kinds of legitmate connections.

Unfortunately, based on alerts like this that are received we have a lot of false positives to wade through in searching for possible real alerts.

So for this investigation, we have to take a slightly different tactic in investigating further. We can examine log files where we have already filtered out alerts that contain a “well-known” service port (such as http or smtp) and make the assumption that only the hosts leftover need to be investigated. Unfortunately,

that is still an investigation of several hundred hosts, and further can still contain false positives.

For this reason we will provide you first we the type of log files we mention above. See the Ramifications and Defensive Recommendations sections for your continuance of the investigation.

Ramifications

As mentioned above, even eliminating those alerts which seem like obvious false positives due to their communication on well-known ports, we still have possible ramifications that must be investigated.

Take these two examples, where no “well-known” port is seen:

24.231.229.43:6349 -> MY.NET.97.62:65535

In this first example, either of two situations is possible: First, 65535 is in fact the open control port on a host on your network infected with the Adore worm, and an outsider is already connected to the control channel.

The other possibility is that despite that fact that it’s not a well-known port, port 6349 is the service on the internet host of 24.231.229.43. In fact, a few internet searches will suggest that port 6349 is one of the service ports of BearShare, a P2P file sharing application.

http://www.ldc.lu.se/security/P2P-list.shtml

MY.NET.153.37:1750 -> 24.169.185.58:65535

In this second example, we again have two possible situations. You can see that this time port 65535 is listening on a host outside your network. This raises the possibility that someone in your network is actually controlling a Red Worm backdoor on an internet host

The other possibility is again the chance of a false positive. It’s possible that the internet host is simply accessing the reachable service sitting at port 1750 on MY.NET.153.37, which could be whatever the administrators of that box have setup.

Defensive recommendation:

The first step you can take is to make sure via your firewall or other stateful packet filtering devices that no connections initiated on port 65535 either leave or enter your network.

Secondly, along with the log file we provide you, investigate the state of each machine.

The tool at the link below can be easily run on systems that are suspected to be infected. The tool will even clean the system in question.
http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/adorefind.htm

You can also easily eliminate certain systems from the investigation that are not vulnerable. Mostly only certain versions of Linux are vulnerable. See the list at the following link.

http://www.sans.org/y2k/adore.htm

Correlations

- This paper makes the same assumptions about traffic similar to the above traffic analyzed here: http://216.239.41.104/search?q=cache:4NhPok8madYJ:www.giac.org/practical/GCIA/Marcus_Wu_GCIA.pdf+false+positive+65535+adore&hl=en&ie=UTF-8

NMAP TCP ping!

Introduction

Nmap is a very well known network tool that can be used for a host of different purposes ranging from simple scanning of networks for reachable systems to scanning of hosts for listening services to OS fingerprinting. There are several existing Snort signatures to detect the various ways in which in Nmap is used.

http://www.insecure.org/nmap/index.html

Example Snort Rule

The 'NMAP TCP ping!' is generally set to check that the bit for the Acknowledge flag is set while the Acknowledge field is set to 0. The rule should look something like this

alert tcp any any -> $MY_NET any (flags: A; ack: 0; msg:"NMAP TCP ping!";)

Detailed Analysis

Sending such a packet has only one purpose: To determine certain information about the destination.

Take the example where client X wants to determine if port 25 is reachable (i.e. not behind a stateful packet filtering device) and listening (i.e. has service smtpd started) on server Y. Sending a "tcp ping" via nmap sends a packet as described above to the server.

If the server is reachable and listening, it should return a packet with the RST, or reset flag set, since any normal TCP connection should first begin with the SYN-SYN/ACK-ACK handshake. Since a packet arriving has the ACK flag set (and naturally, a new IP ID), the server rejects it as it doesn't see it as part of an already existing session. The way it rejects it is with the RST flag.

The other possibility is that the server doesn't return anything at all, in which case the port (or destination host) is filtered beforehand.

Interestingly, more than a third of the total 1,726 alerts triggered most certainly result from a false positive:

67.20.173.236 -> MY.NET.5.92:25

While it can still be a real Nmap "tcp ping", there are a couple of factors that make this unlikely. For one, there are more than 600 of these connections, with a constantly incrementing source port, suggesting it's a connection being continuously attempted. Secondly, the desired result would be returned in less than a second, i.e. there is no reason to make so many attempts. In all likelihood, this is a networking problem between the source and client, resulting in a false positive.

For the rest of the alerts, they certainly look like legitimate NMAP 'tcp pings' or other similar uses of the TCP/IP stack for knowledge gain. For example:

195.6.62.30:80 -> MY.NET.12.6:25

Note the source port. No such traffic should exist in naturally.

Ramifications

Nmap is only a reconnaissance tool and none of these alerts are the result of harmful attacks.

It is possible that outsiders have already gained insight into what services are running on certain systems.

Defensive recommendation:
The best defense here is to put at the chokepoint of your networks packet filtering devices that are completely stateful, i.e. can track the state of connections and have the functionality to drop packets that do not appear to be part of an already existing connection or are not attempting to properly initiate a new connection themselves.

Correlations

- There are other existing signatures for Nmap within Snort's rulebase: http://www.snort.org/cgi-bin/sigs-search.cgi?sid=nmap
- It is analyzed in several other papers:
  http://cert.uni-stuttgart.de/archive/intrusions/2003/11/msg00209.html
  http://www.giac.org/practical/Wei-Chieh_Lim_GCIA.doc

ICMP SRC and DST outside network

Introduction

This is most likely a custom made rule and it doesn't seem to be as much related to security and intrusion detection as it is to possible network diagnostics and discovery.

Example Snort Rule

Based on both its own very telling naming along with the logs we've retrieved, this rule most likely just triggers on any ICMP traffic with $EXTERNAL as both source and destination:

alert icmp $EXTERNAL <> $EXTERNAL (msg: "ICMP SRC and DST outside network";)

Detailed Analysis

Here the issue comes down to one in which the important issue is the way in which the University defines which network belong to its MY_NET, and conversely, everything else, which will then belong to EXTERNAL.

We presumed earlier in the paper that the MY_NET variable in Snort is defined as 130.85.0.0/16. Therefore everything else in the world, that is everything from 65.173.218.106 (www.sans.org) to 10.0.0.0/8 and other RFC 1918 networks that aren't even routable over the internet become defined, as understood by Snort, to be EXTERNAL.

So, some examples of alerts we can see matching this signature are the following:

172.169.103.233 -> 172.171.226.214

Clearly, the above IP addresses match Snort as being part of EXTERNAL. However, are they really two hosts on the internet? Probably not, since if they both were, the traffic would never have been routed inside your network and past your sensor in the first place. These are probably in fact two hosts inside your network using addresses that at least don't match the IP addressing schemes in Snort, and may even not match the University's overall IP addressing schemes.

This isn't necessarily harmful in any way, but it may make administering and controlling your network more difficult. Further, these addresses are technically registered to AOL. It's most likely the case that the ISP you use towards the internet wouldn't route those addresses for you anyway.

192.168.0.25 -> 211.150.211.6

Here we have a slightly different situation. The source address is an RFC1918 address, i.e. an address block reserved for use on private LANs. Such an address will not be routed over the internet. Of course you may have a firewall or other device that will NAT this source towards the internet.

Again, Snort sees this address as being EXTERNAL even though clearly it is internal to the University.

0.0.0.0 -> 169.252.135.16
0.0.0.0 -> 169.252.135.17
0.0.0.0 -> 169.252.135.18
0.0.0.0 -> 169.252.135.19
0.0.0.0 -> 169.252.135.20
0.0.0.0 -> 169.252.135.21
0.0.0.0 -> 169.252.135.22
0.0.0.0 -> 169.252.135.23

Finally, this last set of events can probably be attributed to a host attempting to contact a DHCP server scanning the and RFC3330 address block.

Ramifications

The security threat here is difficult to determine, mainly due to the lack of information on such addresses and the determination of which direction the traffic may be flowing in certain situations.

Obviously, if ICMP is allowed inbound, you certainly open yourself up to various threats. Even allowing ICMP echo replies inbound only is a bit more than necessary in our opinion. ICMP was written as a diagnostic protocol and can be used in malicious or at least mischievous ways, particularly as a network discovery device.

Defensive recommendation:

The first recommendation, certainly defensive in nature, is to block ICMP from being used as much as possible. Don't allow it inbound and don't allow it outbound. If this is not possible, limit this types of access as much as possible. Don't allow such access to and from entire networks. Limit it on a host by host basis at the very least.

The second recommendation, not completely security related, is to do an audit of your network for systems that are not integrated properly into your network, for example, the 172.171.226.214 address mentioned above.

Finally, update Snort to be aware of all networks that are not considered to be EXTERNAL.

Correlations

- Such traffic has been noticed in other papers for your networks:
  http://www.giac.org/practical/michael_wilkinson_gcia.doc
  http://www.giac.org/practical/esperanza_lopez-wilkin_gcia.doc

VII. Brief Analysis

External RPC call and SUNRPC highport access

There are more than 500 alerts towards hosts across the MY.NET networks on port 111, always from the same source address of 211.98.224.34, which doesn't resolve to anything. Port 111 is the Sun RPC portmapper.

There are also multiple connections towards port 32771, one of the Sun Remote Procedure Call High ports, to hosts on the MY.NET nets from the outside from multiple sources.

MY.NET hosts should be investigated for proper system and network security, proper patching, and failing those, attempted intrusion. Here are the top ten hosts accessed for Sun RPC services.

Volume	Host
70	MY.NET.97.93

50	MY.NET.97.98
21	MY.NET.70.37
14	MY.NET.162.22
12	MY.NET.97.186
12	MY.NET.163.142
9	MY.NET.97.64
9	MY.NET.97.63
9	MY.NET.111.168
6	MY.NET.97.167

Possible trojan server activity

This is, like some other rules, apparently customized only to trigger on port 27374, commonly associated with the SubSeven Trojan. Under no circumstance does there appear to be a MY.NET hosts with SubSeven installed.

TCP SRC and DST outside network

See 'ICMP SRC and DST outside network' in Detailed Analysis section. The sensor could be configured to correctly identify all networks which are internal to the University, but this is not the case.

Multiple UMBC NIDS IRC Alerts

There are many customized IRC alerts generated by the sensor, triggered by several different situations, such as /kill detected.

.

FTP passwd attempt

Multiple failed login attempts, perhaps triggering on FTP sessions with the word 'failed'. These are all towards your FTP server at MY.NET.24.47, ragnarok.umbc.edu.

SMB C access

Traffic towards hosts on the MY.NET.190/24 network seems to be allowed. In this case, the alert most likely triggers. The signature possibly customized, may look something like one of the ones here: http://www.digitaltrust.it/arachnids/IDS339/signatures.html. In any case, inbound SMB traffic should not be allowed. Further In any such case, it's best not to allow unadulterated access to the C$ of your Windows machines.

UMBC NIDS External MiMail alert

This alert is seen more than 80 times during the week on inbound mails towards MY.NET.12.6. It most likely detects the virus known as MiMail.

EXPLOIT x86 setuid 0
EXPLOIT x86 setgid 0
EXPLOIT x86 NOPS
EXPLOIT x86 stealth noop

FTP DoS ftpd globbing

Such alerts are always inbound to the ragnarok.umbc.edu FTP server.
This event indicates that the source may be attempting to crash the ftpd server software by sending a wildcard request to create a denial of service on vulnerable ftp servers. However, this can happen in legitimate network traffic.

RFB - Possible WinVNC - 010708-1

There are several hosts on the network that appear to run WinVNC, which is an application that, like Dameware or PCAnywhere, allow potential remote control of the system. This alert triggers 35 times towards the hosts below from outside sources:

MY.NET.111.51
MY.NET.162.91
MY.NET.70.156
MY.NET.70.210
MY.NET.97.63
MY.NET.97.81
MY.NET.111.46

DDOS shaft client to handler

This always triggered on outbound HTTP traffic, with source port of tcp 20432, which is commonly known to a control port for the Shaftnode DdoS tool.

TCP SMTP Source Port traffic

The main connection that triggered this alert was:

216.87.56.33 (sitemail.fanball.com):25 -> MY.NET.12.6:25

While it's not impossible for this to be a legitimate mail transfer, it is somewhat unlikely. A source port of 25 is highly suspicious. This may have been a scan for mailservers, or a possible attack attempt.

TFTP - External TCP connection to internal tftp server

TFTP - External UDP connection to internal tftp server

There were several connections made towards port 69 on several MY.NET hosts. Whether these were scans or actual connections is not possible to say. Nevertheless you should insure that tftp traffic towards host on your inside network is not allowed unless it is intended.

VIII. Relational Analysis of Network Services

Based on our detailed analyses so far, we can make certain assumptions about the network, its hosts, and the purposes of some of these hosts:

Name Services: MY.NET.1.3, MY.NET.1.4, MY.NET.1.5

As can be seen when looking up any MY.NET address publicly, these are the name servers for the University.

85.130.in-addr.arpa nameserver = UMBC3.umbc.edu
85.130.in-addr.arpa nameserver = UMBC4.umbc.edu
85.130.in-addr.arpa nameserver = UMBC5.umbc.edu
UMBC3.umbc.edu internet address = 130.85.1.3
UMBC4.umbc.edu internet address = 130.85.1.4
UMBC5.umbc.edu internet address = 130.85.1.5

Mail services: Hosts on MY.NET.25/24 and MY.NET.12/24

As we can see from multiple types of alerts, hosts such as MY.NET.12.2 and MY.NET.12.6 receive mails while hosts on the MY.NET.25/24 network seem to receive mails. For example:

Inbound
216.93.213.147 -> MY.NET.12.2:25 for “Possible Trojan Server Activity”
(a port-based false positive for SubSeven)
MY.NET.12.2 resolves to smtp.umbc.edu

Outbound
MY.NET.25.66 -> 64.201.107.242:25 for “Possible Red Worm Traffic”
(a port-based false positive for Red Worm, or Adore Worm)
MY.NET.25.66 resolves to mxin1.umbc.edu

Web services

From the logs there appear to web services running on many hosts across many of the University’s networks. One of the Top Ten Alerts analyzed above, for the NOP x86 Exploits, mentioned IP address 66.149.34.140 as having generated the

largest amount of these alerts. Its scanning presents what is probably a nice mapping of the University's web services: Here are the unique destination IP's with resulting from that scan. One can presume from such a differentiated variety of IP address that these are the packets allowed through by a network filtering device such as a router or firewall, rather than a specific list of targets

Conns	Host
655	MY.NET.80.232:80
642	MY.NET.83.70:80
635	MY.NET.112.216:80
634	MY.NET.83.98:80
333	MY.NET.29.18:80
325	MY.NET.150.101:80
322	MY.NET.150.44:80
321	MY.NET.111.72:80
315	MY.NET.29.12:80
307	MY.NET.75.13:80
151	MY.NET.112.153:80
40	MY.NET.5.45:80
35	MY.NET.5.20:80
34	MY.NET.5.67:80
34	MY.NET.17.45:80
27	MY.NET.5.44:80
23	MY.NET.29.8:80
21	MY.NET.5.95:80
21	MY.NET.29.19:80
20	MY.NET.5.46:80
19	MY.NET.5.92:80
16	MY.NET.5.25:80
12	MY.NET.112.226:80
11	MY.NET.150.6:80

FTP Server: MY.NET.24.47

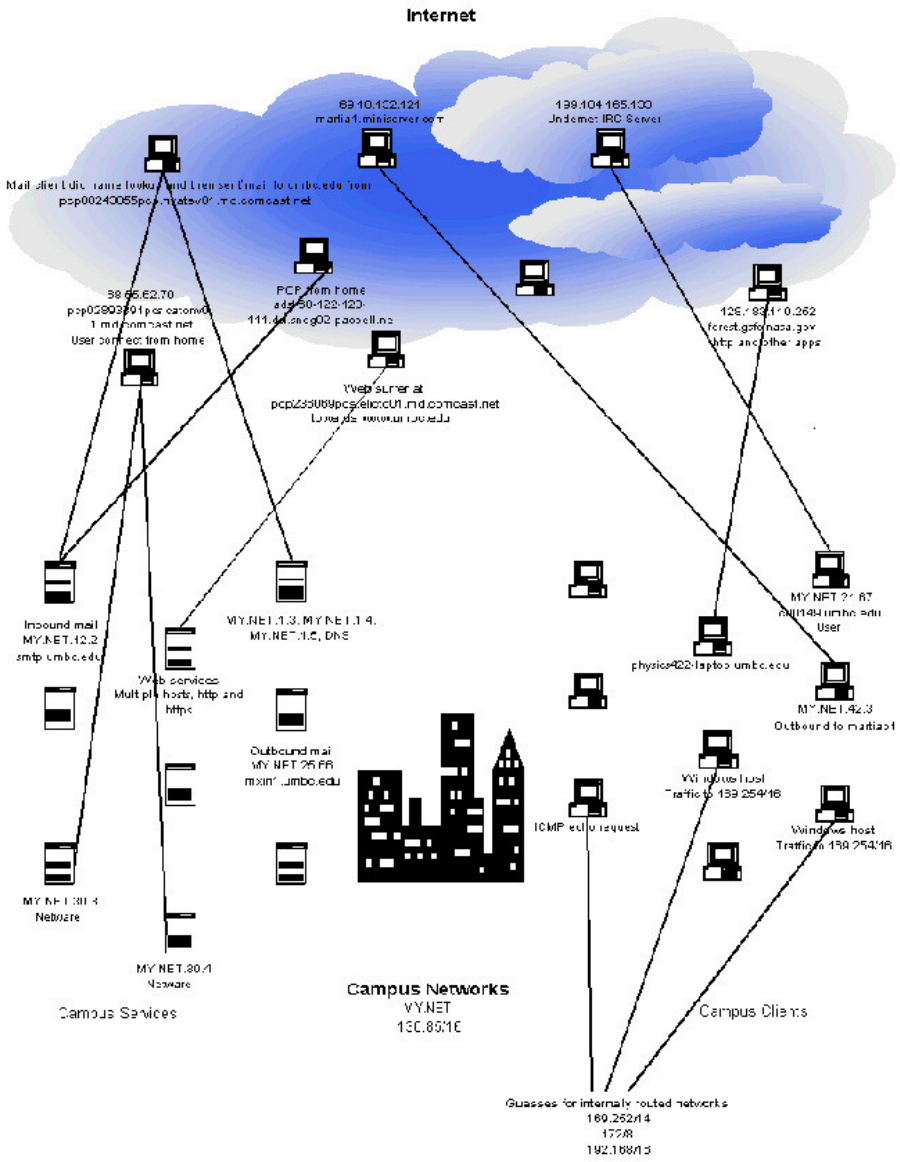
This server, with FQDN of ragnarok.umbc.edu, was the destination in several hundred "FTP Password" attempt" alerts, though there were rarely more than one from each unique source address, and can probably all be attributed to mistyping.

User Authentication: MY.NET.30.3, MY.NET.30.4

As seen in multiple analyses above, these systems are clearly very important to the University's networks. They are very likely the main Netware servers for the University, managing and monitoring all user authentication to the campus networks.

Link Analysis Diagram

Working now from both the network services we discovered above along with our detailed alert analyses we can make some interesting visual links:



IX References

[1] Real name unknown, goes by sh0dan, DameWare Mini Remote Control <= 3.72.0.0, 14 December 2003, http://sh0dan.org/files/dwmrcs372.txt

[2] Author unknown, though may be Lance Spitzer, Scan Challenge Answers, date unknown, http://project.honeynet.org/scans/scan20/sol/30/x369.html

[3] Author and date unknown http://www.isc.org/products/BIND/

[4] Holland, Jeff, DNS Security, 23 July 2000, http://www.whitehats.ca/main/members/Jeff/jeff_dns_security/jeff_dns_security.html