CS590 LBS PROJECT

Exploiting Apps under Linux

Looking for Vulnerabilities, Developing Exploits

Denis Ulybyshev, Servio Palacios

10/11/2014

Setting Applications Environment, Compiling Apps, Crafting Exploits

Contents

Preparing Virtual Machine to exploit applications:

Brief introduction

We began with selection of many very well known applications that could be exploited. For instance, we chose: Apache, MySQL, Gzip, Squid, ProFTPD, Nginx, Samba, and many others.

We looked for vulnerabilities of different applications using CVE (http://www.cvedetails.com/).

Then we tried to set up the environment to trigger the bug. The hard part is that not all the applications triggered the bug exactly as described in CVE or other bug reports. For instance, we ran some exploits on Debian and FreeBSD, but those only worked on RedHat, so we also installed RedHat 9.0 and tried to compile the application. After some time, we struggled to add modules and libraries to create an exploit and see if we could trigger the bug in that particular application. In some applications like MySQL, Apache, Squid, simulating environment was harder because they have many libraries and modules, some of them were difficult to configure, and eventually bugs weren't triggered as expected.

We started exploiting Apache versions 1.3.22, 1.3.26, 1.3.27, 1.3.31, 2.0.47, 2.4.7 in four different Operating Systems (FreeBSD 4.8, Ubuntu, Debian 5.0 and RedHat 9.0 using VMWare virtual machines).

We have installed MySQL(3.23.56, 4.1.1), Apache(All mentioned above), proftpd-1.3.0a, gzip-1.2.4, squid-2.3.STABLE5, from scratch (using **configure**, **make**, **make install**).

We realized that it would be better if we started with some smaller Linux applications. We started looking for smaller applications like gzip, bc, man, ncompress, cvs, and polymorph. With many of these we were able to simulate the exact bug as described.

Operating System

We were able to compile most of those applications under Debian GNU/Linux 5.0.

Gcc version

gcc version 4.3.2

Choosing one application to develop exploit that triggered Shell

We analyzed the source code of these applications and chose **ncompress** to start with..

We chose ncompress based on **CVE-2001-1413:**

ncompress: is a utility that will do fast compression and decompression compatible with the original Unix compress utility (.Z extension). I will not handle gzipped (.gz) images.

We looked for older versions, and we found version **ncompress-4.2.4** to be a perfect candidate for our tests:

- http://www.redhat.com/archives/enterprise-watch-list/2004-December/msg00005.html
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1168
- http://www.securityfocus.com/advisories/7626

CVE-2001-1413

Description

Stack-based buffer overflow in the comprexx function for ncompress 4.2.4 and earlier, when used in situations that cross security boundaries (such as FTP server), may allow remote attackers to execute arbitrary code via a long filename argument.

Impact

By supplying carefully crafted filename or other option, an attacker could execute arbitrary code on the system. A local attacker could only execute code with his own rights, but since compress and uncompress are called by various daemon programs, this might also allow a remote attacker to execute code with the right of the daemon making use of **ncompress**.

In order to have it running on our system we did the next steps.

Steps to prepare ncompress to be able to be exploitable

Download ncompress 4.2.4 source code

- http://sourcecodebrowser.com/ncompress/4.2.4.0/files.html
- http://sourcecodebrowser.com/ncompress/4.2.4.0/compress42_8c_source.html

Compiling ncompress 4.2.4 on Debian Linux

The original Makefile includes some options that were not present in our gcc version: We read some reports of these two flags and it was safe to disable them.

They threw:

```
cc1: error: unrecognized command line option "-freduce-all-givs"
cc1: error: unrecognized command line option "-fmove-all-movables"
make: *** [compress42.o] Error 1
```

We disable or comment in our Makefile: -freduce-all-givs -fmove-all-movables, because these options became extinct in gcc version 4.x.

Source Code errors and fixes:

Then, we encounter that our compress42.c did not compile due to few errors:

```
compress42.c:652: error: conflicting types for ‘rindex’
compress42.c: In function ‘main’:
```

```
compress42.c:705: warning: return type of ‘main’ is not ‘int’
compress42.c: At top level:
compress42.c:1817: error: conflicting types for ‘rindex’
compress42.c: In function ‘rindex’:
compress42.c:1820: error: argument ‘s’ doesn’t match prototype
/usr/include/string.h:313: error: prototype declaration
```

We proceed to look "string.h" prototypes declarations, we encounter:

```
312 /* Find the last occurrence of C in S (same as strrchr).  */
313 extern char *rindex (__const char *__s, int __c)
314      __THROW __attribute_pure__ __nonnull ((1));
```

Figure 1. string.h prototypes declaration.

After that we compare the code in compress42.c to this prototype:

```
652 char    *rindex      ARGS((char *,int));
```

Figure 2. Line errors in compress42.c

```
1816 char *
1817 rindex(s, c)
1818   REG1 char*s;
1819   REG2 int   c;
```

Figure 3. Line errors in compress42.c

According to the prototype and the errors thrown, these two definitions should be like this:

```
652 char    *rindex      ARGS((const char *,int));
```

Figure 4. rindex definition fixed.

```
1816 char *
1817 rindex(s, c)
1818   REG1 const char *s;
1819   REG2 int   c;
```

Figure 5. Some definitions fixed.

Understanding source code

At this point we had a workable version of our tool. Then, We started to understand the code and look for the exact line that triggers the buffer overflow.

```
891  void
892  comprexx(fileptr)
893    char **fileptr;
894    {
895      int    fdin;
896      int    fdout;
897      char tempname[MAXPATHLEN];
898
899      strcpy(tempname,*fileptr);
900      //590 -- memory corruption happens here
901      errno = 0;
```

Figure 6. Line of code that triggers the bug.

Developing a workable exploit

We have inserted code to realize System address and shell address:

Figure 7. Inserting code and compile new code.