

Let us try an existing shell code and check how it is working

- [Linux/x86 - Password Authentication portbind port 64713/tcp - 166 bytes](#) by [Gotfault Security](#)
- [Linux/x86 - portbind port 64713 - 86 bytes](#) by [Gotfault Security](#)
- [Linux/x86 - setreuid\(0,0\) + execve\(/bin/sh, \[/bin/sh, NULL\]\) - 33 bytes](#) by [Gotfault Security](#)
- [Linux/x86 - setuid\(0\) setgid\(0\) execve\("/bin/sh", \["/bin/sh", NULL\]\) - 37 bytes](#) by [Gotfault Security](#)
- [Linux/x86 - Force Reboot shellcode 36 bytes](#) by [Hamza Megahed](#)
- [Linux/x86 - Remote Port forwarding - 87 bytes](#) by [Hamza Megahed](#)
- [Linux/x86 - execve /bin/sh shellcode - 23 bytes](#) by [Hamza Megahed](#)
- [Linux/x86 - execve-chmod 0777 /etc/shadow - 57 bytes](#) by [Hamza Megahed](#)
- [Linux/x86 - iptables --flush - 43 bytes](#) by [Hamza Megahed](#)
- [Linux/x86 - ASLR deactivation - 83 bytes](#) by [Jean Pascal Pereira](#)
- [Linux/x86 - chmod 666 /etc/passwd & /etc/shadow - 57 bytes](#) by [Jean Pascal Pereira](#)
- [Linux/x86 - execve\(/bin/sh\) - 28 bytes](#) by [Jean Pascal Pereira](#)
- [Linux/x86 - ///sbin/iptables -POUTPUT DROP - 60 bytes](#) by [John Babio](#)
- [Linux/x86 - /etc/init.d/apparmor teardown - 53 bytes](#) by [John Babio](#)

```

*****
*                               *
*           Author: Hamza Megahed           *
*                               *
*****
*                               *
*           Twitter: @Hamza_Mega           *
*                               *
*****
*           blog: hamza-mega[dot]blogspot[dot]com           *
*                               *
*****
*           E-mail: hamza[dot]megahed[at]gmail[dot]com           *
*                               *
*****

xor    %eax,%eax
push   %eax
push   $0x68732f2f
push   $0x6e69622f
mov     %esp,%ebx
push   %eax
push   %ebx
mov     %esp,%ecx
mov     $0xb,%al
int     $0x80

*****
#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}

```

```

root@kali:~# sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.5).
build-essential set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 228 not upgraded.
root@kali:~# gedit test.c

```

```

sd_execve[] =
99"          /* cdq          */
52"          /* push %edx     */
68\x6e\x2f\x73\x68" /* push $0x68732f6e */
68\x2f\x2f\x62\x69" /* push $0x69622f2f */
89\xe3"        /* movl %esp,%ebx */
53"          /* push %eax     or %edx */

```

```

#include <stdio.h>
#include <string.h>

unsigned char code[] = \
"SHELLCODE";

main()
{
    printf("Shellcode Length: %d\n", strlen(code));
    int (*ret)() = (int (*)())code;
    ret();
}

```

```

#include <stdio.h>
#include <string.h>

/* FreeBSD 23 byte execve code. Greetz to anathema, the first who published */
/* unsigned char code[] = "\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x62\x69" */
/* "SHELLCODE"; */
/*      editor      marcetam      */
/*      admin@marcetam.net      */
/* ***** */
main()
{
char fbsd_execve[] = "\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x62\x69\x68\x2f\x2f\x62\x69\x89\xe3\x51\x52\x53\x53\x6a\x3b\x58\xcd\x80";
printf("Shellcode Length: %d\n", strlen(code));
int (*ret)() = (int (*)())code;
ret();
/* push %edx */
/* push $0x68732f6e */
/* push $0x69622f2f */
/* movl %esp,%ebx */
/* push %ecx - or %edx :) */
/* push %edx - or %ecx :) */
/* push %ebx */
/* push %ebx */
/* push $0x3b */
/* pop %eax */
/* int $0x80 */

~
~
int main() {
void (*run)()=(void *)fbsd_execve;
printf("%d bytes \n",strlen(fbsd_execve));
}
"test.c" 15L, 183C

```

```

root@kali:~# vim test.c
root@kali:~# gcc test.c -o shellcode
root@kali:~#

```

```

#include <stdio.h>
#include <string.h>

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)

```

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

unsigned char code[] = \
    "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
    "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main()
{
    printf("Shellcode Length: %d\n", strlen(code));

    int (*ret)() = (int (*)())code;

    ret();
}
```

```
root@kali:~# gcc -fno-stack-protector -z execstack test.c -o shellcode
root@kali:~#
```

```
root@kali:~# ./shellcode
Shellcode Length: 23
Segmentation fault
root@kali:~#
```

```

root@kali:~# gdb -q ./shellcode
Reading symbols from ./shellcode...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(gdb) break main
Breakpoint 1 at 0x1149
(gdb) run
Starting program: /root/shellcode

Breakpoint 1, 0x000055555555149 in main ()
(gdb) disassemble
Dump of assembler code for function main:
   0x000055555555145 <+0>:    push    rbp
   0x000055555555146 <+1>:    mov     rbp, rsp
=> 0x000055555555149 <+4>:    sub     rsp, 0x20
   0x00005555555514d <+8>:    mov     DWORD PTR [rbp-0x14], edi
   0x000055555555150 <+11>:   mov     QWORD PTR [rbp-0x20], rsi
   0x000055555555154 <+15>:   lea     rdi, [rip+0x2ee5]          # 0x555555558040 <code>
>
   0x00005555555515b <+22>:   call    0x55555555030 <strlen@plt>
   0x000055555555160 <+27>:   mov     rsi, rax
   0x000055555555163 <+30>:   lea     rdi, [rip+0xe9a]          # 0x555555556004
   0x00005555555516a <+37>:   mov     eax, 0x0
   0x00005555555516f <+42>:   call    0x55555555040 <printf@plt>
   0x000055555555174 <+47>:   lea     rax, [rip+0x2ec5]          # 0x555555558040 <code>
>
   0x00005555555517b <+54>:   mov     QWORD PTR [rbp-0x8], rax
   0x00005555555517f <+58>:   mov     rdx, QWORD PTR [rbp-0x8]
   0x000055555555183 <+62>:   mov     eax, 0x0
   0x000055555555188 <+67>:   call    rdx
   0x00005555555518a <+69>:   mov     eax, 0x0
   0x00005555555518f <+74>:   leave
   0x000055555555190 <+75>:   ret
End of assembler dump.
(gdb) █

```

```

(gdb) shell cat test.c
#include <stdio.h>
#include <string.h>

unsigned char code[] = \
    "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
    "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";
int main(int argc, char *argv[])
{
    printf("Shellcode Length: %d\n", strlen(code));

    int (*ret)() = (int (*)())code;

    ret();
}

(gdb) █

```



```
(gdb) print /x code
'code' has unknown type; cast it to its declared type
(gdb) print /x &code
$1 = 0x555555558040
```

Dump of assembler code for function main:

```
0x000055555555145 <+0>: push    rbp
0x000055555555146 <+1>: mov     rbp, rsp
=> 0x000055555555149 <+4>: sub     rsp, 0x20
0x00005555555514d <+8>: mov     DWORD PTR [rbp-0x14], edi
0x000055555555150 <+11>: mov     QWORD PTR [rbp-0x20], rsi
0x000055555555154 <+15>: lea     rdi, [rip+0x2ee5]          # 0x555555558040 <code>
>
0x00005555555515b <+22>: call    0x55555555030 <strlen@plt>
0x000055555555160 <+27>: mov     rsi, rax
0x000055555555163 <+30>: lea     rdi, [rip+0xe9a]          # 0x555555556004
0x00005555555516a <+37>: mov     eax, 0x0
0x00005555555516f <+42>: call    0x55555555040 <printf@plt>
0x000055555555174 <+47>: lea     rax, [rip+0x2ec5]          # 0x555555558040 <code>
>
0x00005555555517b <+54>: mov     QWORD PTR [rbp-0x8], rax
0x00005555555517f <+58>: mov     rdx, QWORD PTR [rbp-0x8]
0x000055555555183 <+62>: mov     eax, 0x0
0x000055555555188 <+67>: call    rdx
0x00005555555518a <+69>: mov     eax, 0x0
0x00005555555518f <+74>: leave
0x000055555555190 <+75>: ret
```

```
(gdb) print /x code
'code' has unknown type; cast it to its declared type
(gdb) print /x &code
$1 = 0x555555558040
(gdb) x/xw 0x555555558040
0x555555558040 <code>: 0x6850c031
(gdb) x/xb 0x555555558040
0x555555558040 <code>: 0x31
(gdb) x/23xb 0x555555558040
0x555555558040 <code>: 0x31    0xc0    0x50    0x68    0x2f    0x2f    0x73    0x68
0x555555558048 <code+8>: 0x68    0x2f    0x62    0x69    0x6e    0x89    0xe30
x50
0x555555558050 <code+16>: 0x53    0x89    0xe1    0xb0    0x0b    0xcd    0x80
(gdb)
```

```
>
0x00005555555515b <+22>:  call    0x555555555030 <strlen@plt>
0x000055555555160 <+27>:  mov     rsi, rax
0x000055555555163 <+30>:  lea     rdi, [rip+0xe9a]          # 0x5555555556004
0x00005555555516a <+37>:  mov     eax, 0x0
0x00005555555516f <+42>:  call    0x555555555040 <printf@plt>
0x000055555555174 <+47>:  lea     rax, [rip+0x2ec5]        # 0x5555555558040 <code
>
0x00005555555517b <+54>:  mov     QWORD PTR [rbp-0x8], rax
0x00005555555517f <+58>:  mov     rdx, QWORD PTR [rbp-0x8]
0x000055555555183 <+62>:  mov     eax, 0x0
0x000055555555188 <+67>:  call    rdx
0x00005555555518a <+69>:  mov     eax, 0x0
0x00005555555518f <+74>:  leave
0x000055555555190 <+75>:  ret
End of assembler dump.
(gdb) shell cat shellcode.c
cat: shellcode.c: No such file or directory
(gdb) shell cat test.c
#include <stdio.h>
#include <string.h>
```

```

(gdb) print /x eax
No symbol "eax" in current context.
(gdb) print /x $eax
$2 = 0xffffffff2
(gdb) break *0xffffffff2
Breakpoint 3 at 0xffffffff2
(gdb) c
Continuing.
Warning:
Cannot insert breakpoint 3.
Cannot access memory at address 0xffffffff2

```

Command aborted.

```
(gdb) disassemble
```

Dump of assembler code for function code:

0x000055555558040 <+0>:	xor	eax, eax
0x000055555558042 <+2>:	push	rax
0x000055555558043 <+3>:	push	0x68732f2f
0x000055555558048 <+8>:	push	0x6e69622f
0x00005555555804d <+13>:	mov	ebx, esp
0x00005555555804f <+15>:	push	rax
0x000055555558050 <+16>:	push	rbx
0x000055555558051 <+17>:	mov	ecx, esp
0x000055555558053 <+19>:	mov	al, 0xb
0x000055555558055 <+21>:	int	0x80
=> 0x000055555558057 <+23>:	add	BYTE PTR [rax], al

End of assembler dump.
(gdb)

```

xor    %eax,%eax
push   %eax
push   $0x68732f2f
push   $0x6e69622f
mov     %esp,%ebx
push   %eax
push   %ebx
mov     %esp,%ecx
mov     $0xb,%al
int     $0x80

```

Now let's try our own shell code.

```
root@kali:~# uname -a
Linux kali 4.18.0-kali2-amd64 #1 SMP Debian 4.18.10-2kali1 (2018-10-09) x86_64 GNU/Linux
root@kali:~#
```

```
Linux kali 4.18.0-kali2-amd64 #1 SMP Debian 4.18.10-2kali1 (2018-10-09) x86_64 GNU/Linux
root@kali:~# sudo apt-get install nasm
Reading package lists... Done
Building dependency tree
Reading state information... Done
nasm is already the newest version (2.13.03-2).
nasm set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 228 not upgraded.
root@kali:~#
```

```
root@kali:~# cd Desktop
root@kali:~/Desktop# touch hello.asm
root@kali:~/Desktop# ls
hello.asm
root@kali:~/Desktop#
```

```
root@kali:~/Desktop# nano hello.asm
```

IT17138482

```
File Edit View Search Terminal Help
GNU nano 3.1 hello.asm

[ Read 0 lines ]
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

```
GNU nano 3.1                                hello.asm

section .data
    text db "Hello Hello",10
section .text
    global _start

_start:
    mov rax, 1
    mov rdi, 1
    mov rsi, text
    mov rdx, 14
    syscall

    mov rax, 60
    mov rdi, 0
    syscall

[ Wrote 16 lines ]
```

```

root@kali:~# cd Desktop
root@kali:~/Desktop# touch hello.asm
root@kali:~/Desktop# ls
hello.asm
root@kali:~/Desktop# nano hello.asm
root@kali:~/Desktop# nano hello.asm
root@kali:~/Desktop# ls
hello.asm
root@kali:~/Desktop# nasm -f elf64 -o hello.o hello.asm
root@kali:~/Desktop# ls
hello.asm  hello.o
root@kali:~/Desktop# man ld
root@kali:~/Desktop#

```

```

LD(1)                                GNU Development Tools                                LD(1)

NAME
    ld - The GNU linker

SYNOPSIS
    ld [options] objfile ...

DESCRIPTION
    ld combines a number of object and archive files, relocates their data and
    ties up symbol references. Usually the last step in compiling a program is
    to run ld.

    ld accepts Linker Command Language files written in a superset of AT&T's
    Link Editor Command Language syntax, to provide explicit and total control
    over the linking process.

    This man page does not describe the command language; see the ld entry in
    "info" for full details on the command language and on other aspects of the
    GNU linker.

    This version of ld uses the general purpose BFD libraries to operate on
    object files. This allows ld to read, combine, and write object files in
    many different formats---for example, COFF or "a.out". Different formats
    may be linked together to produce any available kind of object file.

    Aside from its flexibility, the GNU linker is more helpful than other
    linkers in providing diagnostic information. Many linkers abandon
    execution immediately upon encountering an error; whenever possible, ld
    continues executing, allowing you to identify other errors (or, in some
    cases, to get an output file in spite of the error).

    The GNU linker ld is meant to cover a broad range of situations, and to be
    as compatible as possible with other linkers. As a result, you have many
    Manual page ld(1) line 1 (press h for help or q to quit)

```



```
root@kali:~/Desktop# man ld
root@kali:~/Desktop# ld hello.o -o hello
root@kali:~/Desktop# ls
hello  hello.asm  hello.o
root@kali:~/Desktop# ./hello
Hello Hello
root@kali:~/Desktop#
```