CSE 523S: Systems Security

Co

Systems

Spring 2018

Jon Shidal

Plan for Today

- Announcements
  - HW3 due 1pm 3/21
  - Get started early. It is harder than 1 & 2.

- Security ne

- Assignment

- Stack buffer overflows

Security News

Memcrashed: amplification attack using Memcached

Memcached servers speedup loading of dynamic web pages by caching objects.

Recently found vulnerable to n attacks:

1, src_ip = target ~51,000X

attacker ----> Memcached Server ----> target

Used on Wednesday for largest DDoS attack ever, target was github(~1.3 Tbps)

Assignment

- For Wednesday, 3/7
  - Readings
    - HTAOE:

Today: Lecture and Exercises

- Many of today's slides come from CSE361
  - from an old offering
  - they use
  - Based on Computer , by Bryant and O'Hallaron

Stack Reminders

- Stack grows down from high address
- Each procedure has its own stack frame
- Stack frame contents:
  - return address
  - frame poi
  - local storage
  - arguments to callee d)
  - temporary space (if needed)
- Set-up code at beginning of procedure
- Clean-up code before return
- For ‘C’ code, managed by the compiler

String Library Code

- Implementation of Unix function `gets()`
  - No way to specify limit on number of characters to read

```
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getc();
    char
    while
        *p++ = c;
        c = getc();
    }
    *p = '\0';
    return dest;
}
```


{

- Similar
  - `strcpy`: Copies string of arbitrary length
  - `scanf`, `fscanf`, `sscanf`, when given `%s` conversion specification

Vulnerable Buffer Code

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```



```
int main()
{
  printf("Type a string:");
  echo();
  return 0;
}
```

Buffer Overflow Executions

```
unix>./bufdemo
Type a string:123
123
```

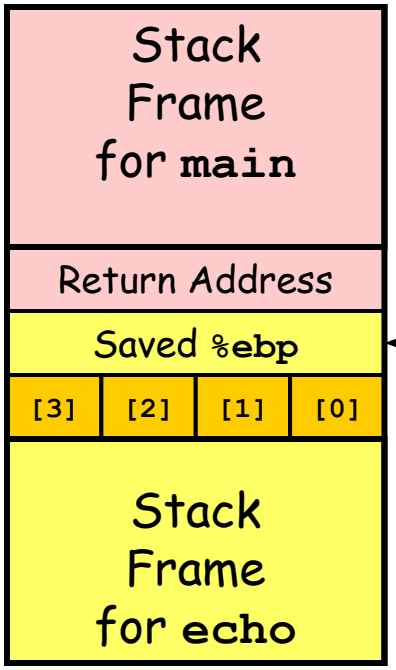


```
unix>.
Type a
Segmen
```



```
unix>./bufdemo
Type a string:12345678
Segmentation Fault
```

Buffer Overflow Stack



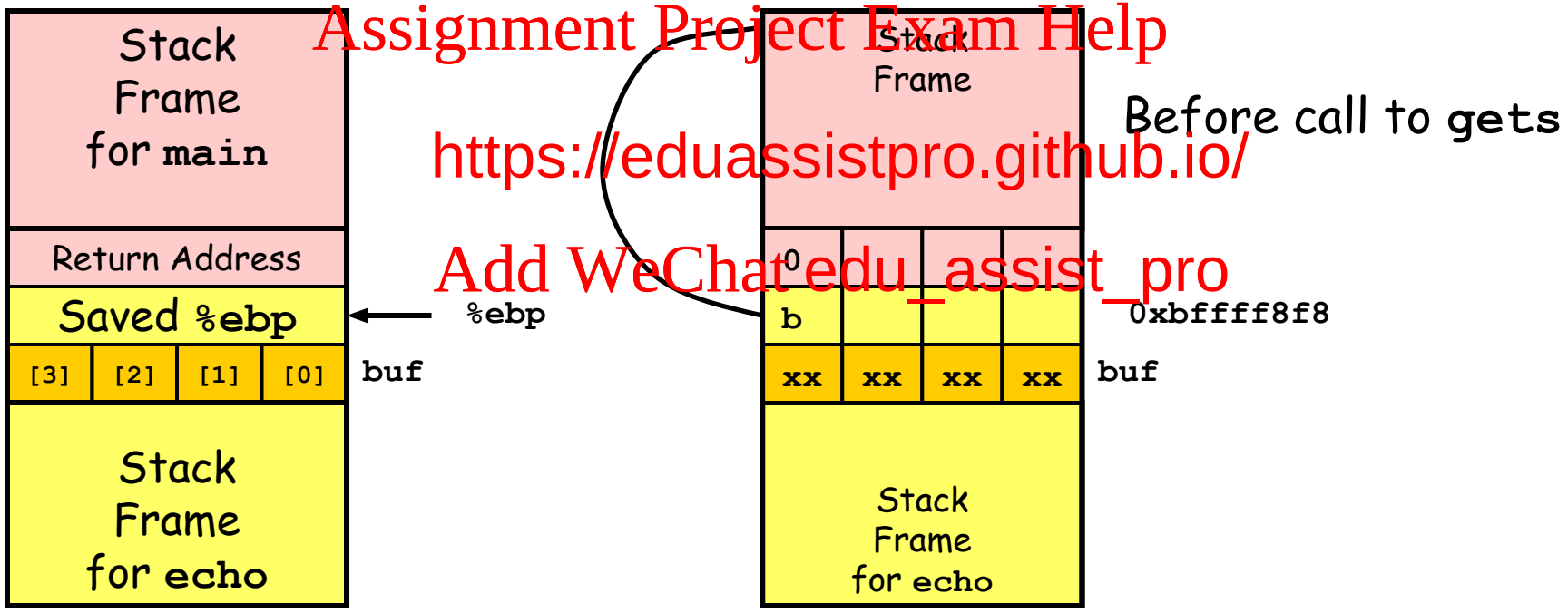

```
/* Echo Line */
void echo()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    puts(buf);
}
```



```
ec
    pushl %ebp              on stack
    movl %esp,%eb
    subl $20,%esp # Allocate stack space
    pushl %ebx    # Save %ebx
    addl $-12,%esp   # Allocate stack space
    leal -4(%ebp),%ebx  # Compute buf as %ebp-4
    pushl %ebx    # Push buf on stack
    call gets # Call gets
    . . .
```

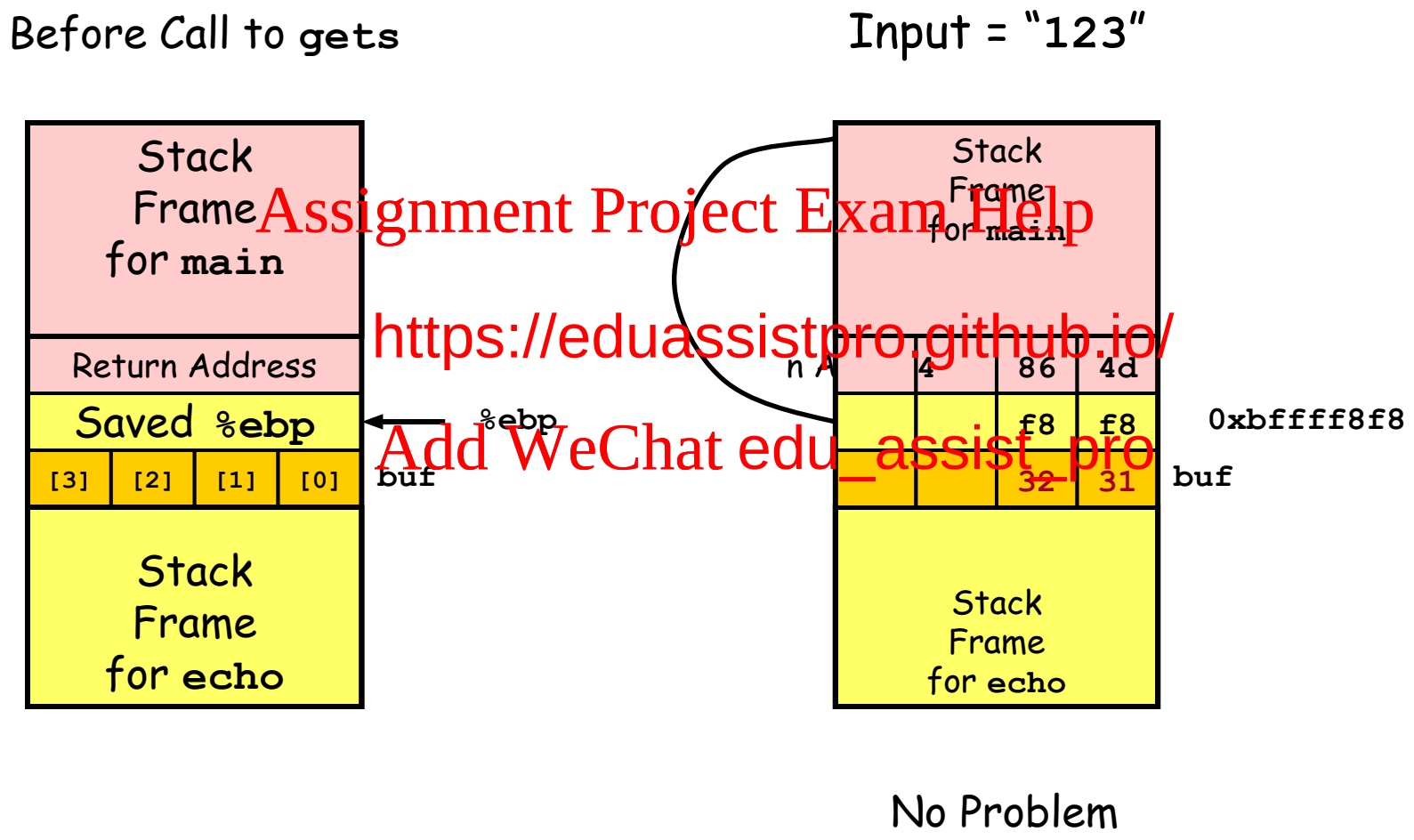
Buffer Overflow Stack Example

```
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbffff8f8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
```

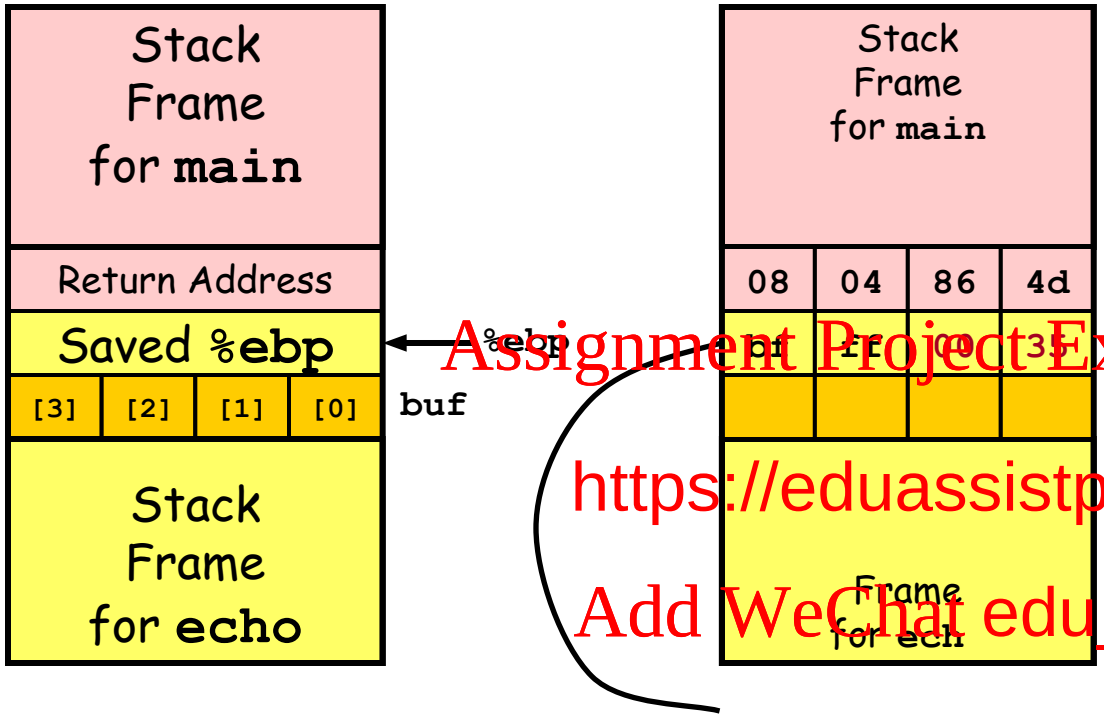




```
8048648:   call 804857c <echo>
804864d:   mov  0xffffffe8(%ebp),%ebx # Return Point
```

Buffer Overflow Example #1



Buffer Overflow Stack Example #2



Input = "12345"

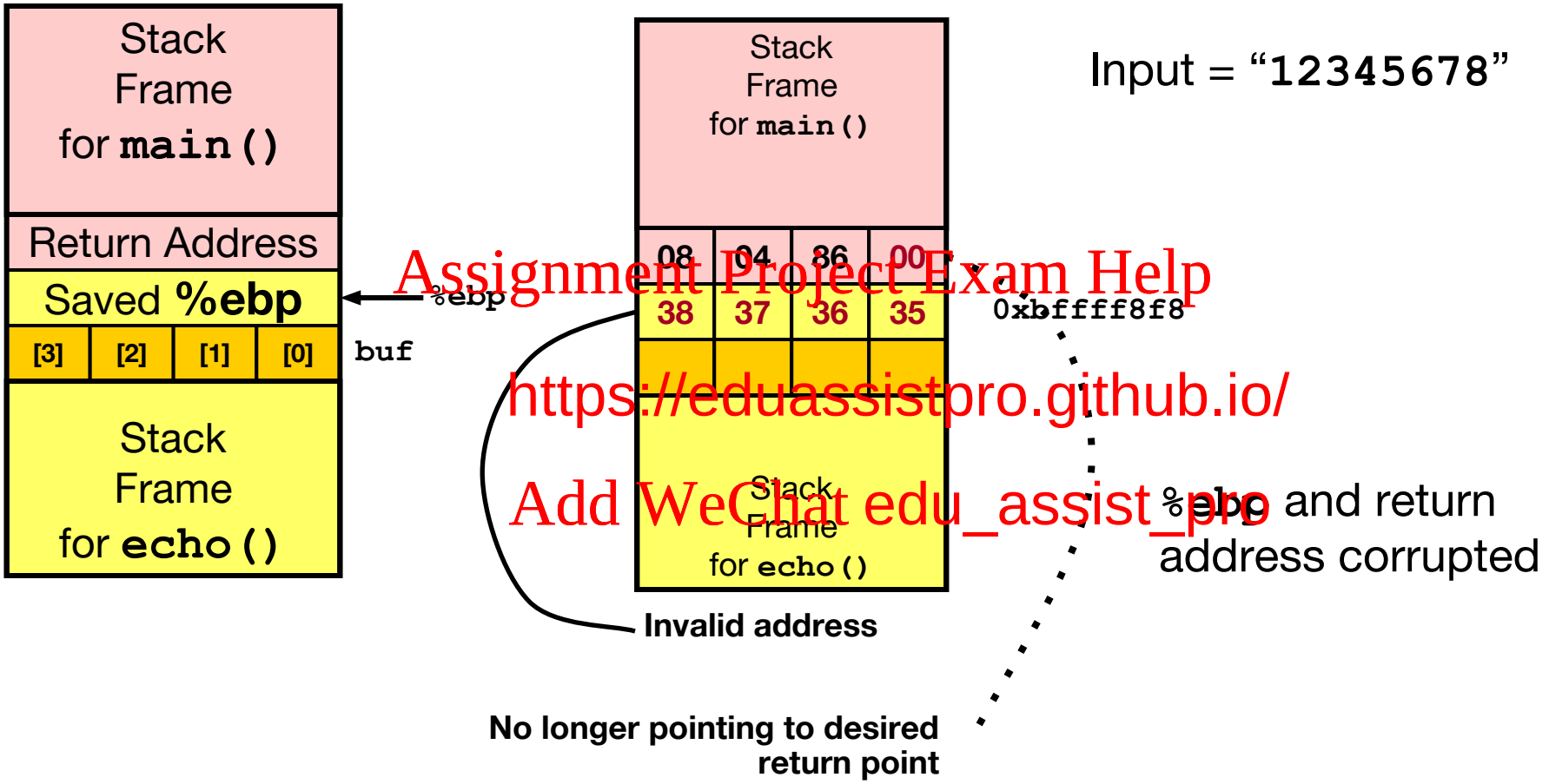
avgd value of %ebp set to fff0035

news when later attempt to restore %ebp

echo code:

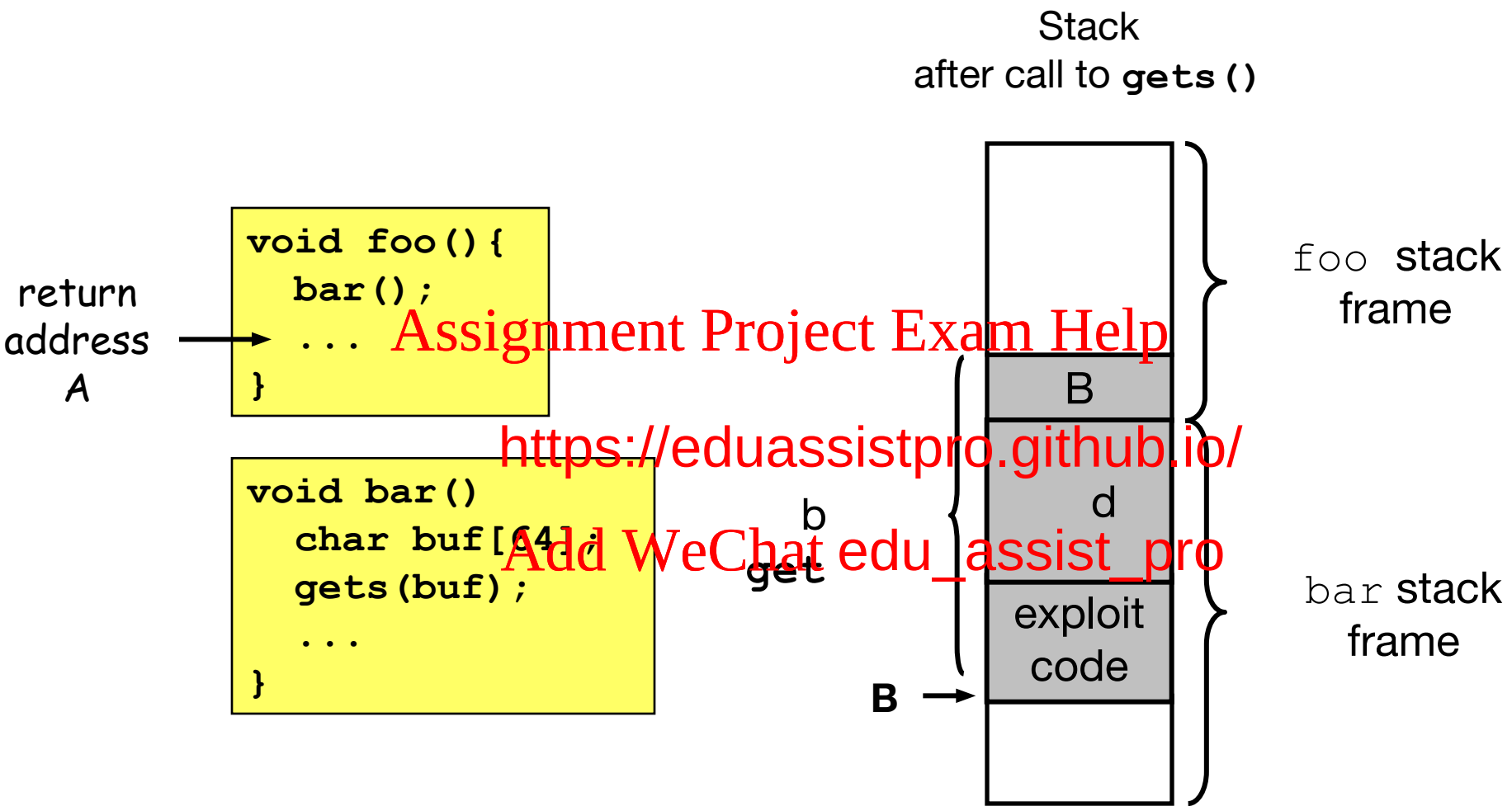
```
8048592:   push   %ebx
8048593:   call   80483e4 <_init+0x50>  # gets
8048598:   mov    0xfffffff8(%ebp),%ebx
804859b:   mov    %ebp,%esp
804859d:   pop    %ebp # %ebp gets set to invalid value
804859e:   ret
```

Buffer Overflow Stack Example #3




```
8048648:   call 804857c <echo>
804864d:   mov  0xffffffe8(%ebp),%ebx # Return Point
```

Malicious Use of Buffer Overflow



– Input string contains byte representation of executable code
– Overwrite return address with address of buffer
– When `bar()` executes `ret`, will jump to exploit code

Let's get to work!

- See exploring-stack-overflow-notes in Google Docs

er, d<s>o</s> Important: us
the browser i

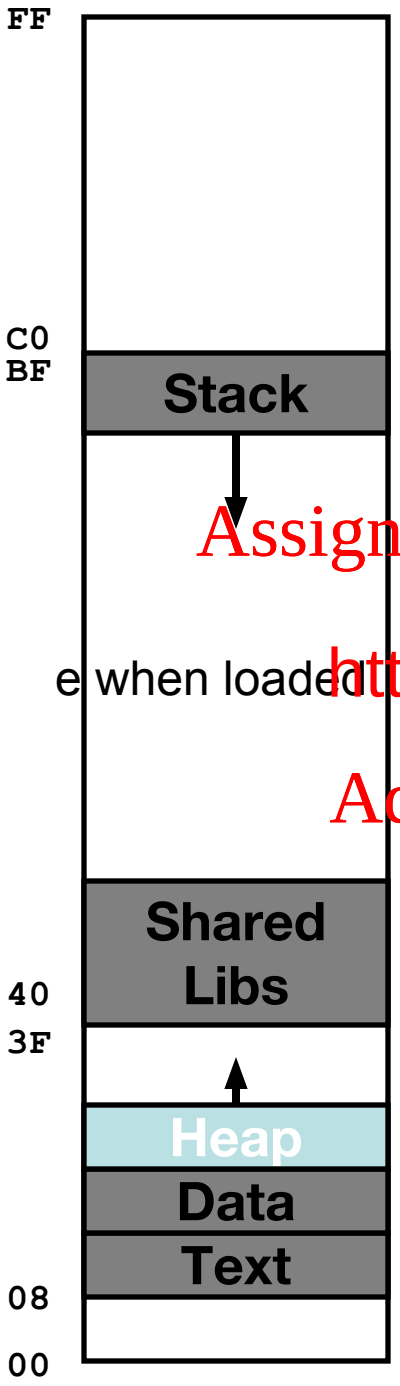
- Also use "Tracking Progress 3/5/2018" to indicate when you have reached a gate

- Additional background slides follow!

Linux Memory Layout

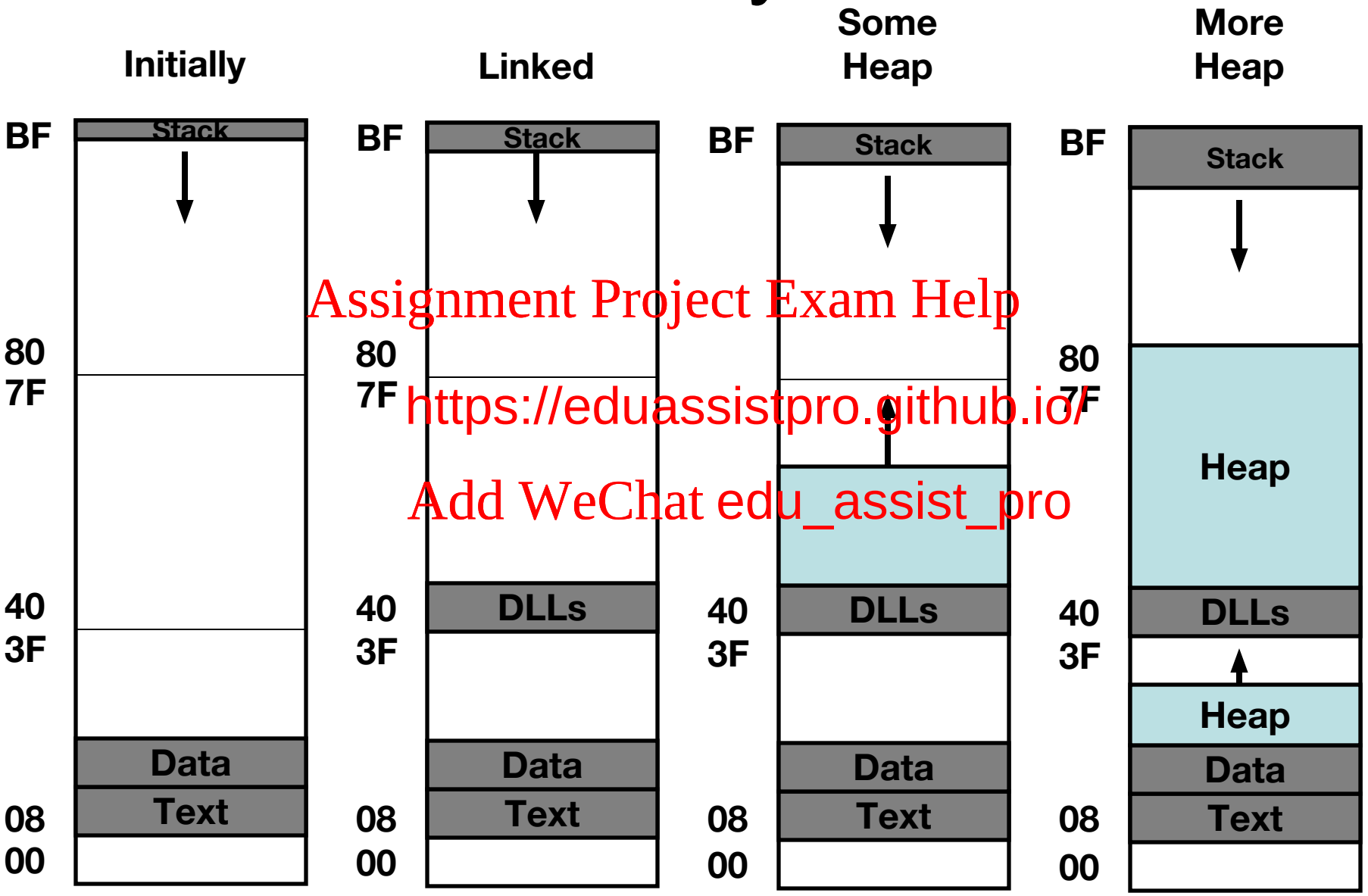
Red Hat v. 6.2 ~1920MB memory limit

Upper 2 hex digits of address



- Stack
  - Runtime stack (8MB limit)
- Heap
  - Dynamically allocated storage
  - When call `malloc(), calloc(), new()`
- Shared Libraries
  - Dynamically Linked Libraries (`printf(), malloc()`)
- Data
  - Statically
  - E.g., arrays & strings declared in code
- Text
  - Executable machine instructions
  - Read-only

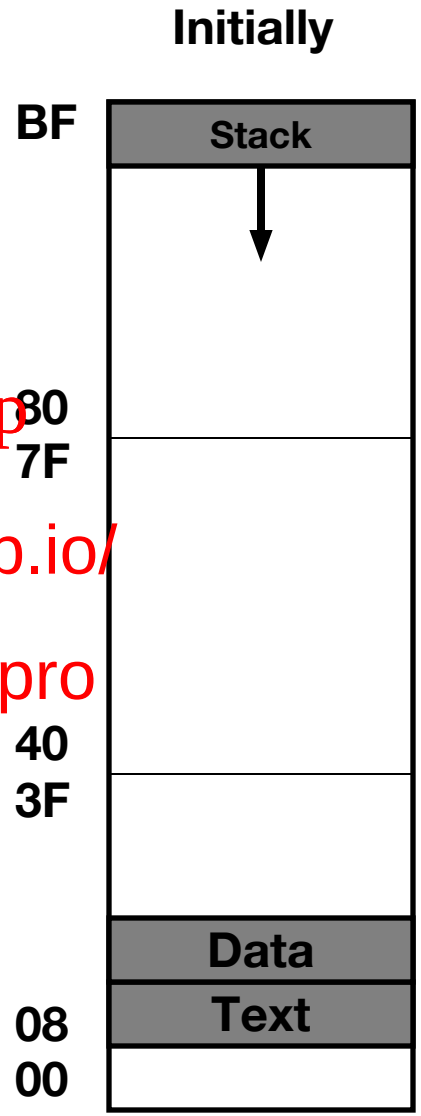
Linux Memory Allocation



Text & Stack Example

```
(gdb) break main
(gdb) run
  Breakpoint 1, 0x804856f in main ()
(gdb) print $esp
  $3 = (void *) 0xbffffc78
```

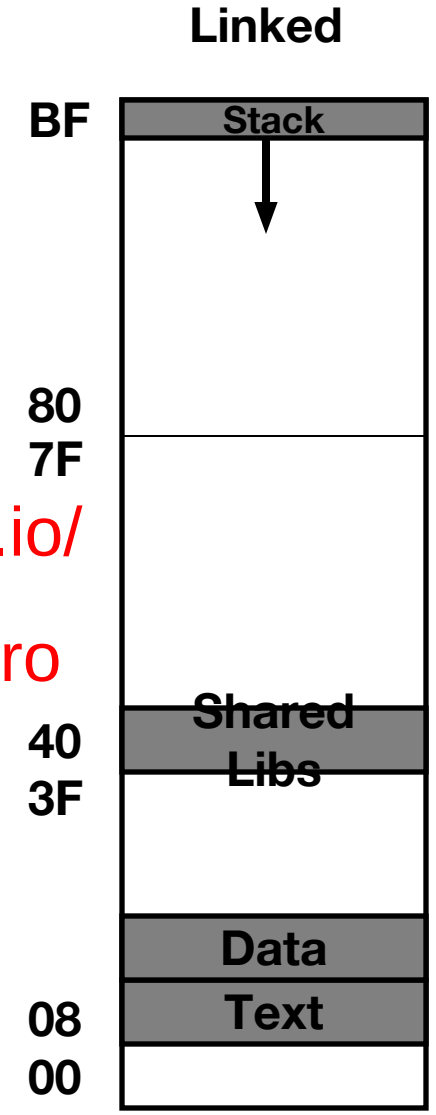

- Main
  - Address `0x804856f (0x0804856f)`
- Stack
  - Address `0xbffffc78`



Dynamic Linking Example

```
(gdb) print malloc
  $1 = {<text variable, no debug info>}
    0x8048454 <malloc>
(gdb) run
  Program exited normally.
(gdb) print malloc
  $2 = {void *(u
    0x40006240 <
```


- Initially
  - Code in text segment that invokes dynamic linker
  - Address `0x8048454` (should be read `0x08048454`)
- Final
  - Code in shared library region



Memory Allocation Example

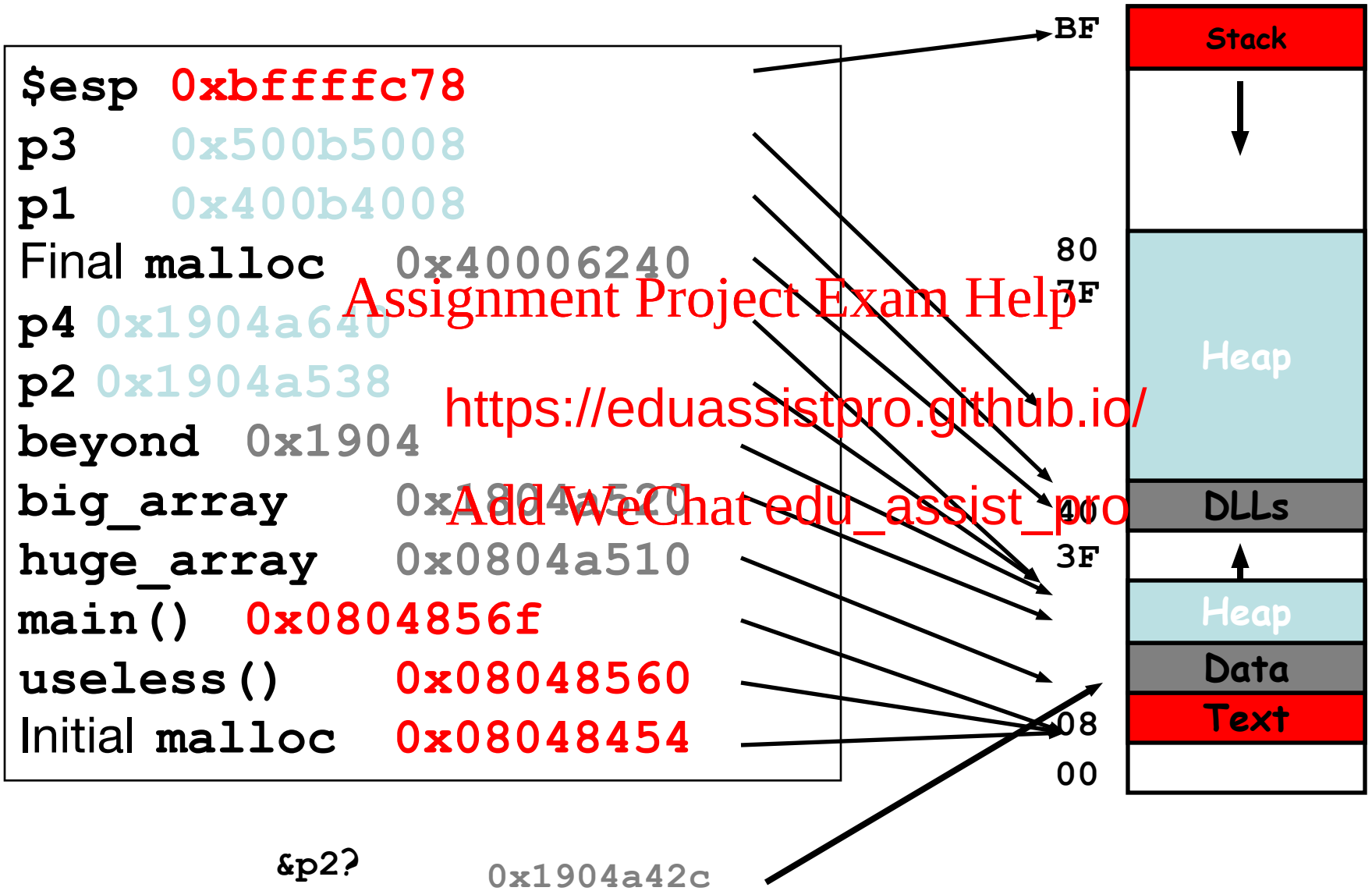
```
char big_array[1<<24];  /*  16 MB */
char huge_array[1<<28]; /* 256 MB */

int beyond;
char *p1, *p2, *p3, *p4;

int useless(

int main()
{
 p1 = malloc(1 <<28);  /* 256 MB */
 p2 = malloc(1 << 8);  /* 256 B  */
 p3 = malloc(1 <<28);  /* 256 MB */
 p4 = malloc(1 << 8);  /* 256 B  */
 /* Some print statements ... */
}
```

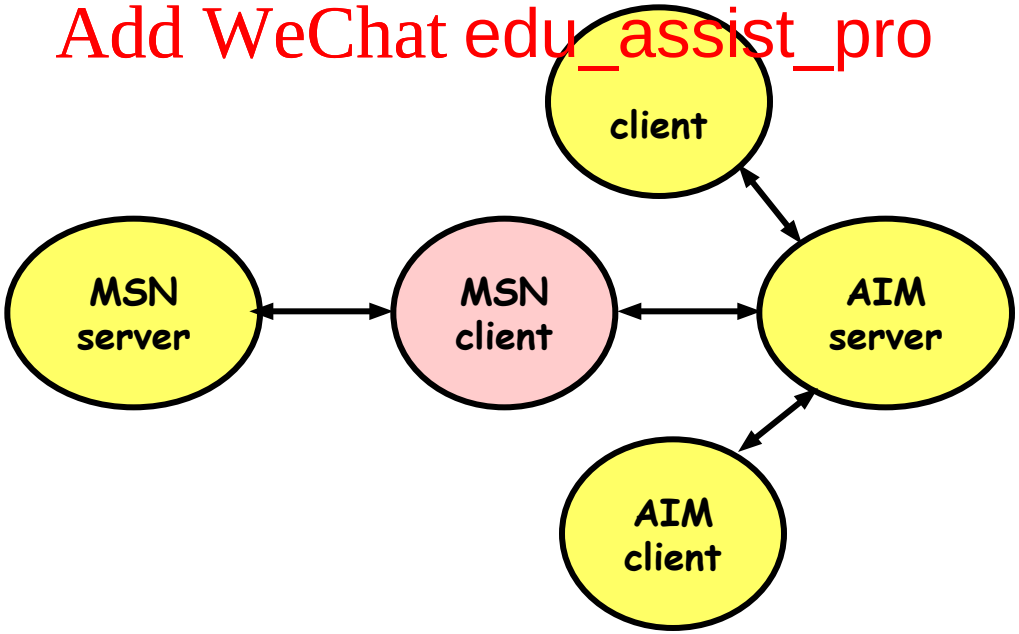
Example Addresses



Internet Worm and IM War

- November, 1988
  - Internet Worm attacks thousands of Internet hosts.
  - How did it happen?

- July, 1999
  - Microsoft launches MSN Messenger (instant messaging system).
  - Messenger clients can access popular AOL Instant Messaging Service (AIM) servers



Internet Worm and IM War (cont.)

August 1999

- Mysteriously, Messenger clients can no longer access AIM servers.
- Microsoft and AOL begin the IM war:
  - AOL changes server to disallow Messenger clients
  - Microsoft mak
  - At least 13 su
- How did it happen?

The Internet Worm and AOL/Microsoft War were both based on *stack buffer overflow* exploits!

- many Unix functions do not check argument sizes.
- allows target buffers to overflow.

Exploits Based on Buffer Overflows

Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.

Internet worm

– Exploited versions of `fingerd` running on the client's `gets()` to re

- `finger joe@cse.wustl.`

– Worm attacked fingerd serv ing phony argument:

- `finger "exploit-code padding new-return-address"`
- exploit code: executed a root shell on the victim machine with a direct TCP connection to the attacker.

The Internet Worm

11/2	18:24 first west coast computer infected
19:04	ucb gateway infected
20:00	mit attacked
20:49	cs.utah.edu infected
21:21	load avg reaches 5 on cs.utah.edu
21:41	load avg re
22:01	load avg re
22:20	worm killed
22:41	cs.utah.edu reinfected, load
22:49	cs.utah.edu shut down
23:31	reinfected, load reaches 37

Exploits Based on Buffer Overflows

Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.

IM War

- AOL exploited bug in AIM clients
  - exploit code: returned 4 bytes (the bytes at some location in the AIM client.
  - Server would only respond to clients that sent the right signature
  - When Microsoft changed code to match signature, AOL changed signature location.

Date: Wed, 11 Aug 1999 11:30:57 -0700 (PDT)
From: Phil Bucking <philbucking@yahoo.com>
Subject: AOL exploiting buffer overrun bug in their own software!
To: rms@pharlap.com

Mr. Smith,

I am writing you because I have discovered something that I think you might find interesting because you are an Internet security expert with experience in this area. I have also tried to contact AOL but received no response.

ewIan an developer who has
messaging client that sho

...
It appears that the AIM client has a buffer By itself
this might not be the end of the world, as had its share.
But AOL is now *exploiting their own buffer to help in
its efforts to block MS Instant Messenger.

....
Since you have significant credibility with the press I hope that you can use this information to help inform people that behind AOL's friendly exterior they are nefariously compromising peoples' security.

Sincerely,
Phil Bucking
Founder, Bucking Consulting
philbucking@yahoo.com

It was later determined that this email originated from within Microsoft!

Code Red Worm

History

- June 18, 2001. Microsoft announces buffer overflow vulnerability in IIS Internet server
- July 19, 2001. over 250,000 machines infected by new virus in 9 hours
- White house must change its IP address. Pentagon shut down public W

Still in the wild, today

- Web servers receive strings of form (contains the virus 'boot sequence')

```
GET
  /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN....NNNNNNNNNNNN
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%
  ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u
  8b00%u531b%u53ff%u0078%u0000%u00=a
HTTP/1.0" 400 325 "-" "-"
```

Code Red Exploit Code

- Starts 100 threads running
- Spread self
  - Generate random IP addresses & send attack string
  - Between 1st & 19th of month
- Attack www.w
  - Send 98,304
    - Denial of service attack
  - Between 21st & 27th of month
- Deface server's home page
  - After waiting 2 hours

ours; repeat

Avoiding Overflow Vulnerability

```
/* Echo Line */
void echo()
{
    char buf[4]; /* Way too small! */
    fgets(buf, 4, stdin);
    puts(
}
```


Use Library Routines that Li Lengths

- `fgets` instead of `gets`
- strncpy instead of `strcpy`
- Don't use `scanf` with `%s` conversion specification
  - Use `fgets` to read the string
  - Or use `%ns` where *n* is a suitable integer

System-Level Protections

- Randomized stack offsets
  - At start of program, allocate random amount of space on stack
  - Makes it difficult for hacker to predict begi
- Nonexecutable code segm
  - In traditional x86, can mark region of memory as either “read-only” or “writeable”
    - Can execute anything readable
  - Add explicit “execute” permission