Return-to-libc Attack
and
Return-oriented Programming (ROP)

SUSTech CS 315 Computer Security 2023

Outline

- Recall NX/DEP countermeasure
- Defeat the countermeasure
- Understanding the process's stack layout
  - Function arguments
  - Functions prologue and epilogue
  - Multiple function call
- Design ROP chain
- Modern protections
- Summary

Recall NX/DEP countermeasure

- Marks memory regions as non-executable
  - Remove executable flag (x) i.e. rwx -> rw-
- Implemented by OS
- Hardware support(fast)

```
0x7ffffffde000     0x7ffffffff000 rw-p    21000 0          [stack]
```

stack memory marked as not executable

```
$gcc –z execstack shellcode.c
$ ./a.out
Good_Job!$
```

```
$gcc –z noexecstack shellcode.c
$ ./a.out
Segmentation fault (core dumped)
```

Bypass the countermeasure

- Return-oriented programming (ROP)
  - can be Turing complete

    Hovav Shacham. 2007. 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 (CCS '07). Association for Computing Machinery, New York, NY, USA, 552–561. https://doi.org/10.1145/1315245.1315313
  - not inject malicious instructions
  - uses **instruction sequences(gadgets)** already present in executable memory
  - exploit by manipulating return addresses
    - control registers:

```
.text:0804850A                 pop     edi
.text:0804850B                 pop     ebp
.text:0804850C                 retn
```

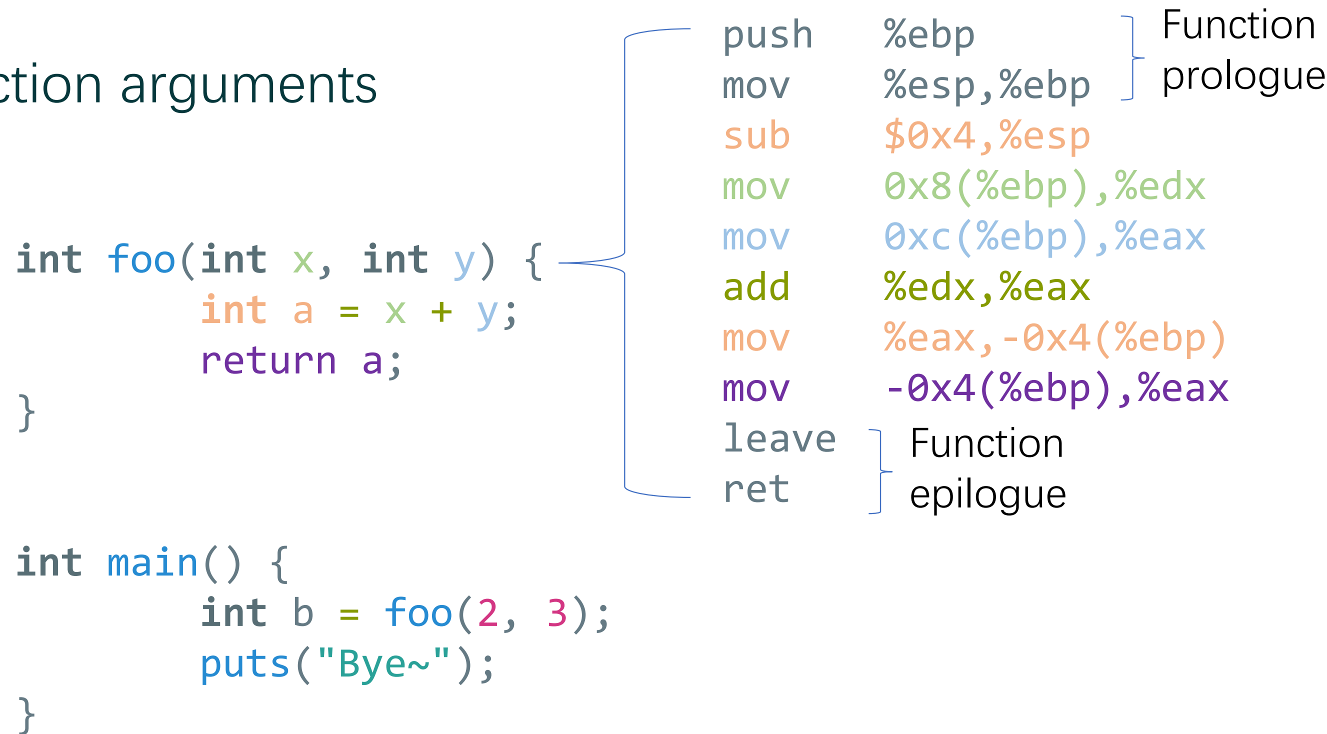
- Data only exploitation

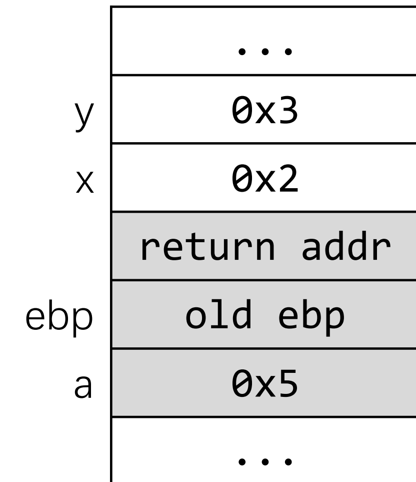
Bypass the countermeasure

- Return-oriented programming (ROP)
  - Let’s begin our trip bypass NX protection!

Understanding the process's stack layout

- Function arguments





调用约定(calling conventions)

*Note that we take x86 architecture as an example
*Also note in AT&T format, "mov %esp, %ebp" means set $ebp = $esp

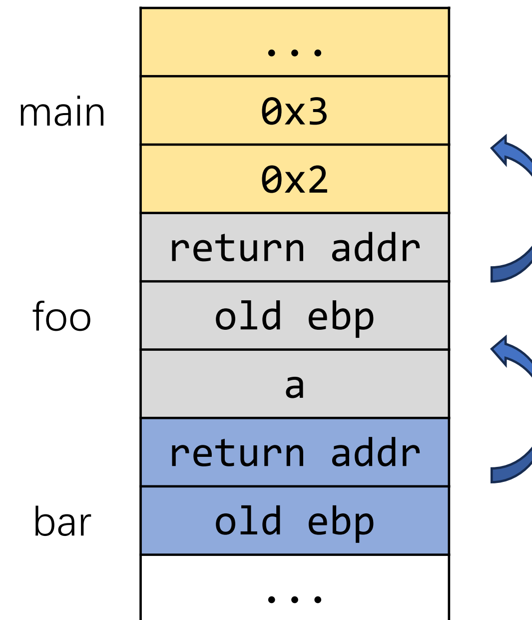
Understanding the process's stack layout

- Function call chain

```
int bar(){
        return 4;
}

int foo(int x, int y) {
        int a = bar();
        a += x + y;
        return a;
}

int main() {
        int b = foo(2, 3);
        puts("Bye~");
}
```



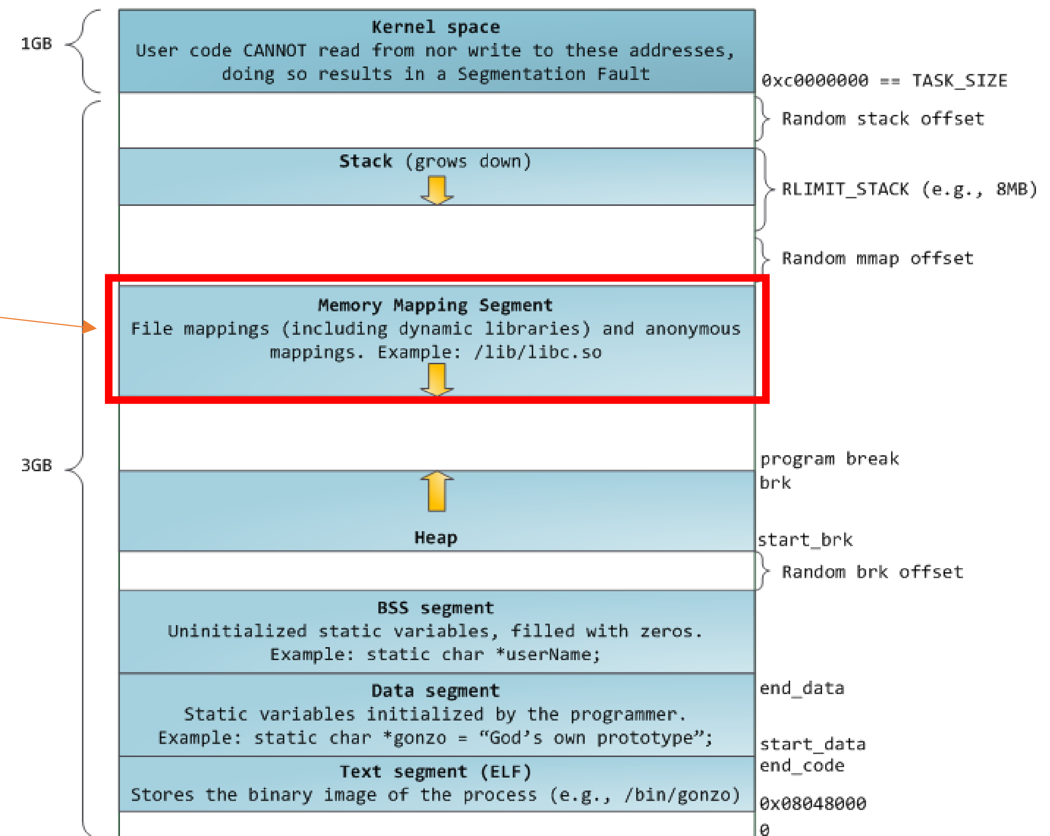
Bypass the countermeasure

- Return-oriented programming (ROP)
  - Let’s begin our trip bypass NX protection!
  - We can continually jump to many places(as long as it marked as executable)
  - So we can reuse many code gadget, call many functions: **set a call chain**.

  - BUT… where to find those gadgets/functions?

Recall: Program Memory, deeper view

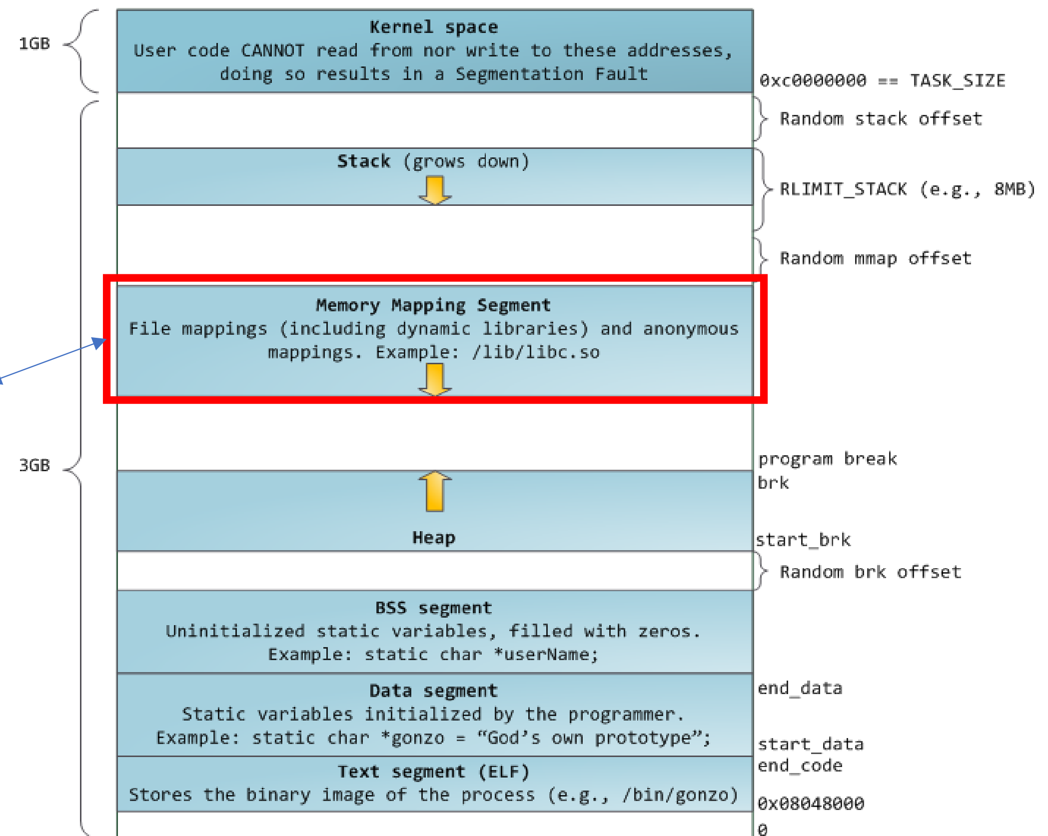
- Most modern programs are dynamically linked, this means they can use functions defined in shared libs(e.g. glibc)
- When program is loaded, the shared libs also loaded in program memory
- View by cat /proc/[fd]/maps (cmdline) or vmmap (gdb)



https://i.stack.imgur.com/epGfE.png

Recall: Program Memory, deeper view

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
     Start        End Perm     Size Offset File
 0x8048000  0x8049000 r-xp     1000      0 /home/student/Desktop/lab5/ret2libc
 0x8049000  0x804a000 rw-p     1000      0 /home/student/Desktop/lab5/ret2libc
 0x804a000  0x806c000 rw-p    22000      0 [heap]
0xf7dc7000 0xf7de0000 r--p    19000      0 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7de0000 0xf7f3b000 r-xp   15b000  19000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7f3b000 0xf7faf000 r--p    74000 174000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7faf000 0xf7fb0000 ---p     1000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb0000 0xf7fb2000 r--p     2000 1e8000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb2000 0xf7fb3000 rw-p     1000 1ea000 /usr/lib/i386-linux-gnu/libc-2.31.so
0xf7fb3000 0xf7fb6000 rw-p     3000      0 [anon_f7fb3]
0xf7fc9000 0xf7fcb000 rw-p     2000      0 [anon_f7fc9]
0xf7fcb000 0xf7fcf000 r--p     4000      0 [vvar]
0xf7fcf000 0xf7fd1000 r-xp     2000      0 [vdso]
0xf7fd1000 0xf7fd2000 r--p     1000      0 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7fd2000 0xf7ff0000 r-xp    1e000   1000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ff0000 0xf7ffb000 r--p     b000  1f000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffc000 0xf7ffd000 r--p     1000  2a000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xf7ffd000 0xf7ffe000 rw-p     1000  2b000 /usr/lib/i386-linux-gnu/ld-2.31.so
0xfffdd000 0xffffe000 rw-p    21000      0 [stack]
```



https://i.stack.imgur.com/epGfE.png

Bypass the countermeasure

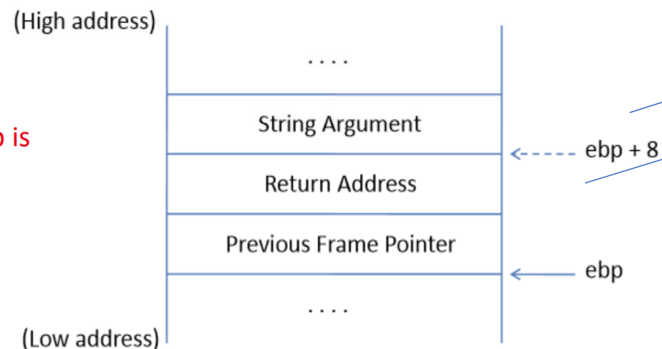
- Return-oriented programming (ROP)
  - Let’s begin our trip bypass NX protection!
  - We can continually jump to many places(as long as it marked as executable)
  - So we can reuse many code gadget, call many functions: **set a call chain**.
  - GNU C Library provides many frequently used functions and we **can reuse those function and gadgets.**
    - https://sourceware.org/git/?p=glibc.git;a=summary
    - The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). Hovav Shacham. In CCS'07. https://cseweb.ucsd.edu/~hovav/dist/geometry.pdf

Design ROP chain-A Quick view

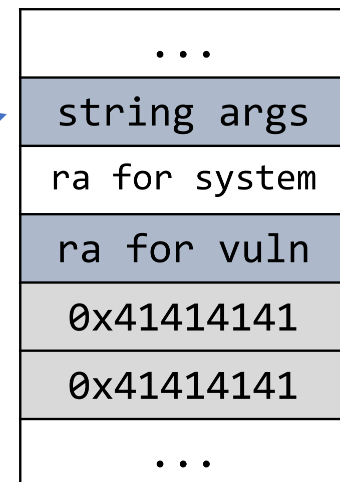
- Overwrite return address, arguments, and return address

  - Arguments are accessed with respect to `ebp`.
  - Argument for `system()` needs to be on the stack.

Need to know where exactly ebp is after we have “returned” to `system()`, so we can put the argument at `ebp + 8`.

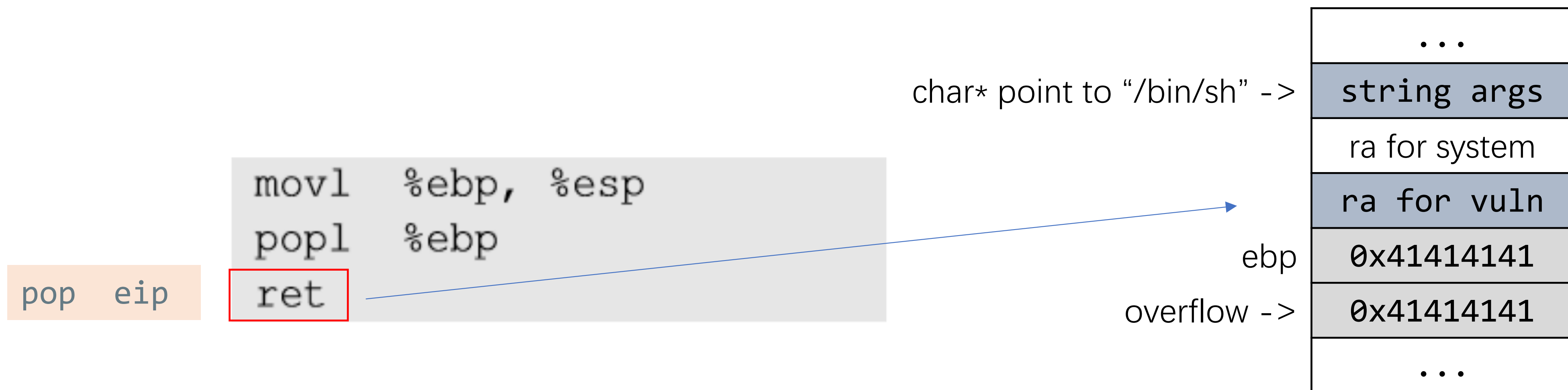


Frame for the system() function



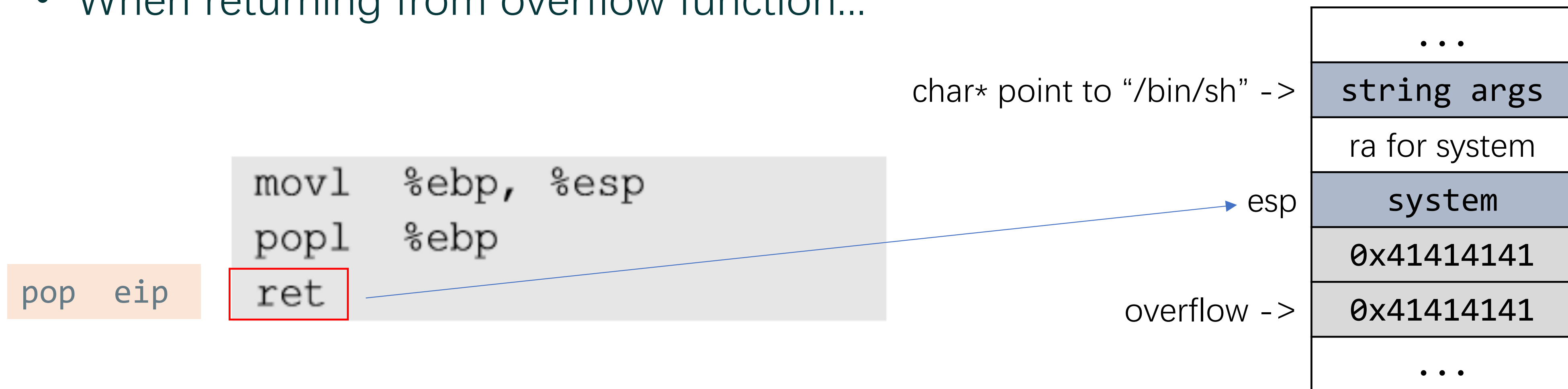
Design ROP chain-A Quick view

- Overwrite return address, arguments, and return address



Design ROP chain-A Quick view

- Overwrite return address, arguments, and return address
- When returning from overflow function…



Design ROP chain-A Quick view

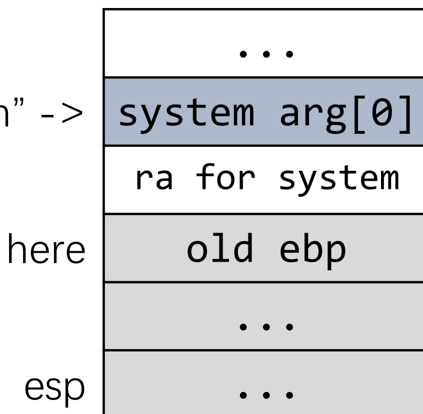
- Overwrite return address, arguments, and return address
- When returning from overflow function
- When returned (entering) to system in libc...
  - https://github.com/bminor/glibc/blob/master/sysdeps/posix/system.c

```
pushl   %ebp
movl    %esp, %ebp
subl    $N, %esp
```

char* point to “/bin/sh” ->

old ebp saved here

esp

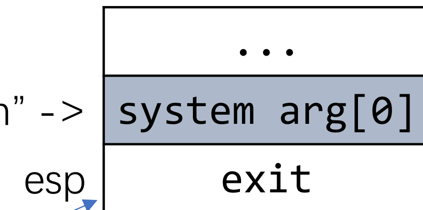


Design ROP chain-A Quick view

- Overwrite return address, arguments, and return address
- When returning from overflow function
- When returned to system in libc
  - https://github.com/bminor/glibc/blob/master/sysdeps/posix/system.c#L189
- When returning from system in libc...

char* point to "/bin/sh" ->

esp




```
pop   eip
```

```
movl   %ebp, %esp
popl   %ebp
ret
```

Bypass the countermeasure

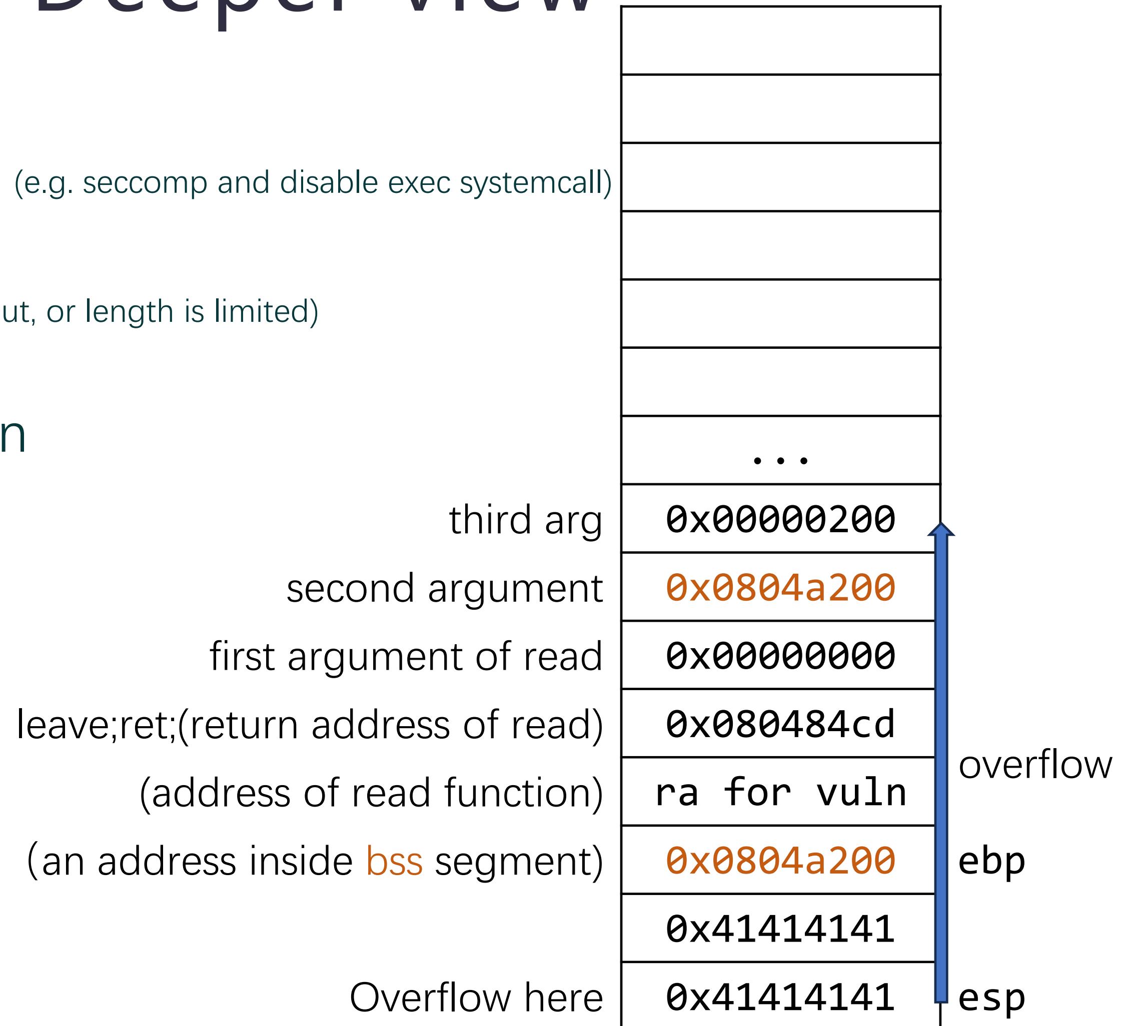
- Return-oriented programming (ROP)
  - Let’s begin our trip bypass NX protection!
  - We can continually jump to many places(as long as it marked as executable)
  - So we can reuse many code gadget, call many functions: **set a call chain**.
  - where to find those gadgets/functions?
  - GNU C Library provides many frequently used functions
  - **We can reuse function and gadgets in glibc. (more detail on later lab)**
  - ROP can be used to bypass more strong protections

Design ROP chain-A Deeper view

- If program enabled extra protections (e.g. seccomp and disable exec systemcall)
- If overflow is limited(e.g. scanf('%s') will chunk input, or length is limited)
- Use ROP to overcome these limitation

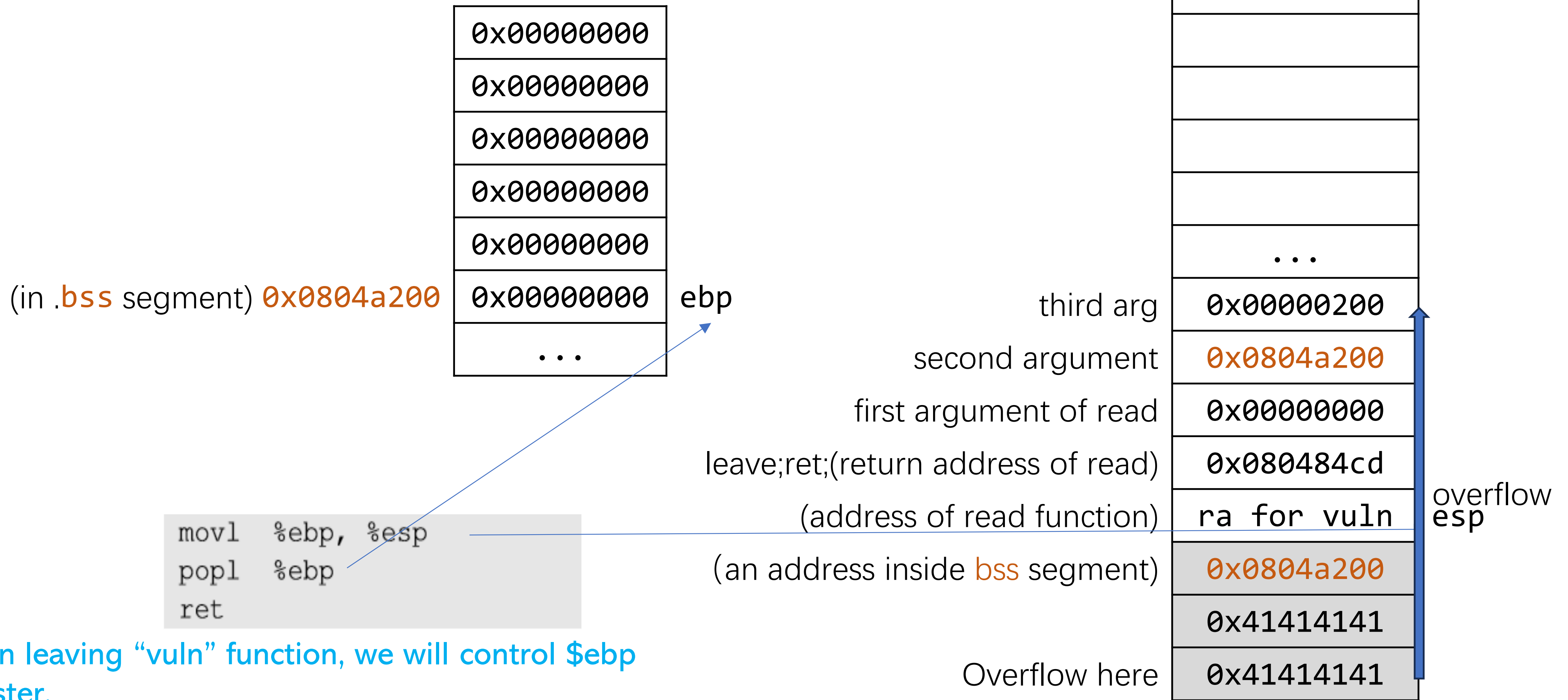
Design ROP chain-A Deeper view

- If program enabled extra protections (e.g. seccomp and disable exec systemcall)
- If overflow is limited(e.g. scanf('%s') will chunk input, or length is limited)
- Use ROP to overcome these limitation



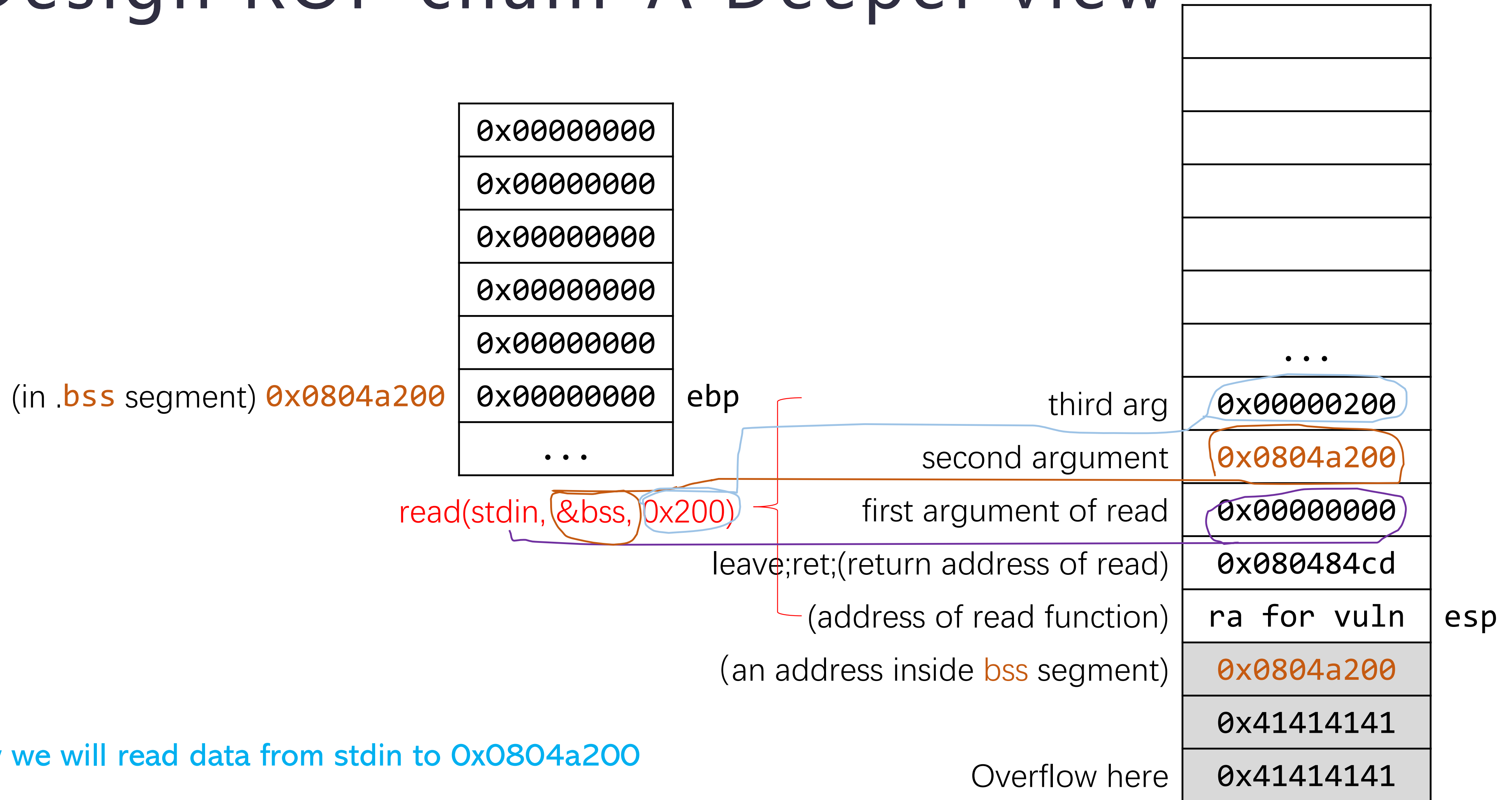
Let's consider a more powerful example(stack pivot)
With overflow, we set up our first ROP chain:

Design ROP chain-A Deeper view



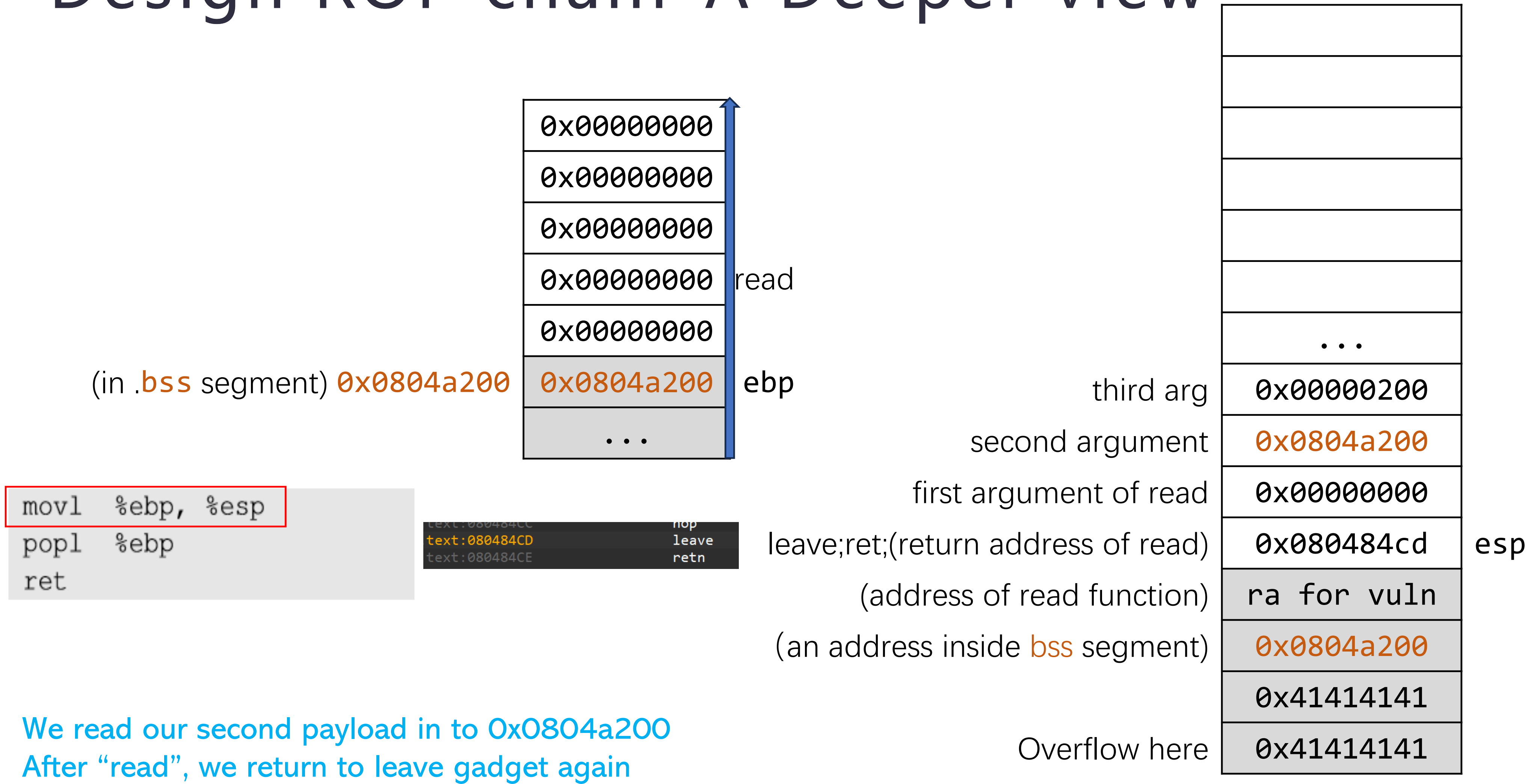
When leaving "vuln" function, we will control $ebp register.

Design ROP chain-A Deeper view



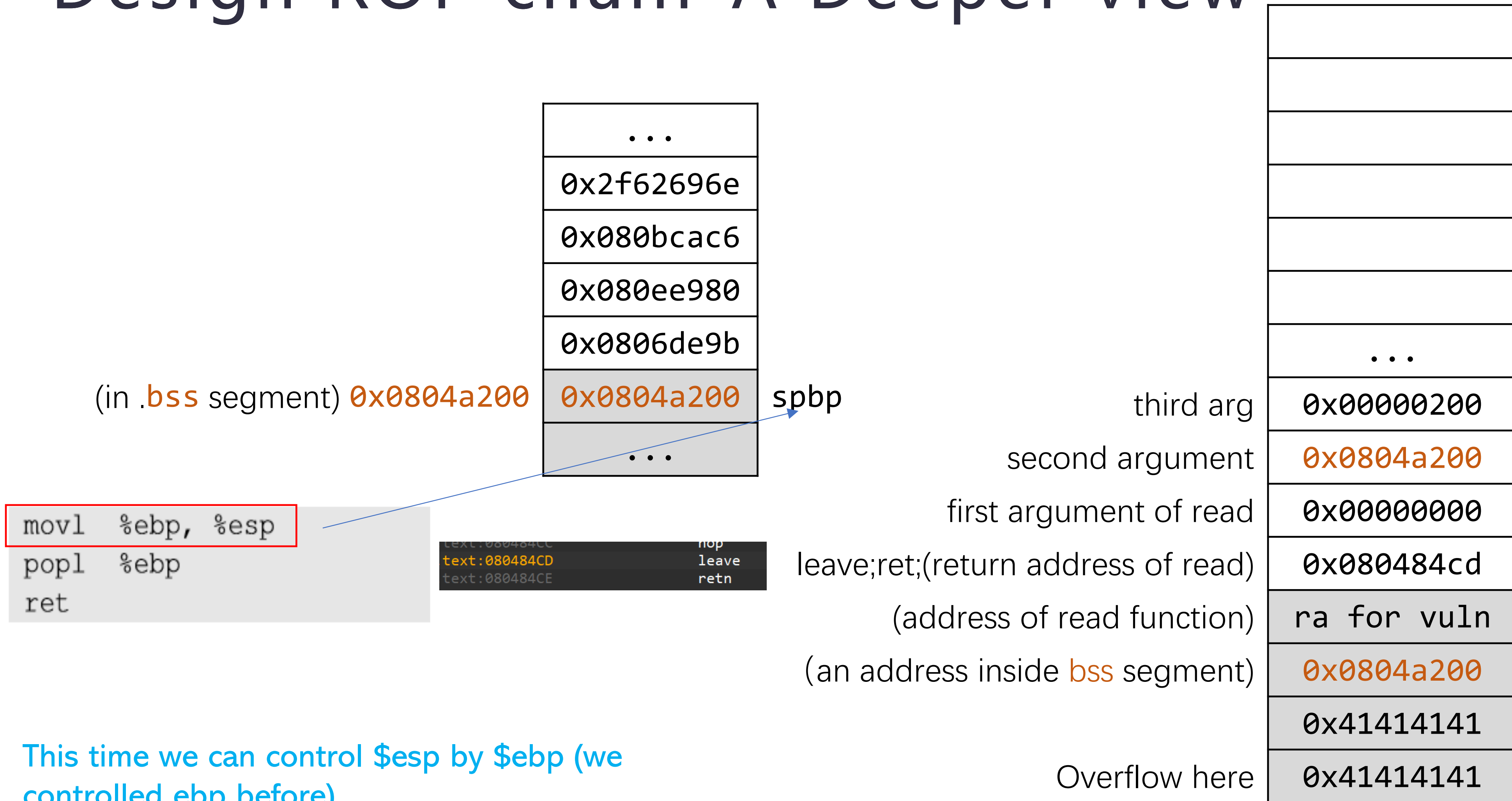
Now we will read data from stdin to 0x0804a200

Design ROP chain-A Deeper view



We read our second payload in to 0x0804a200
After “read”, we return to leave gadget again

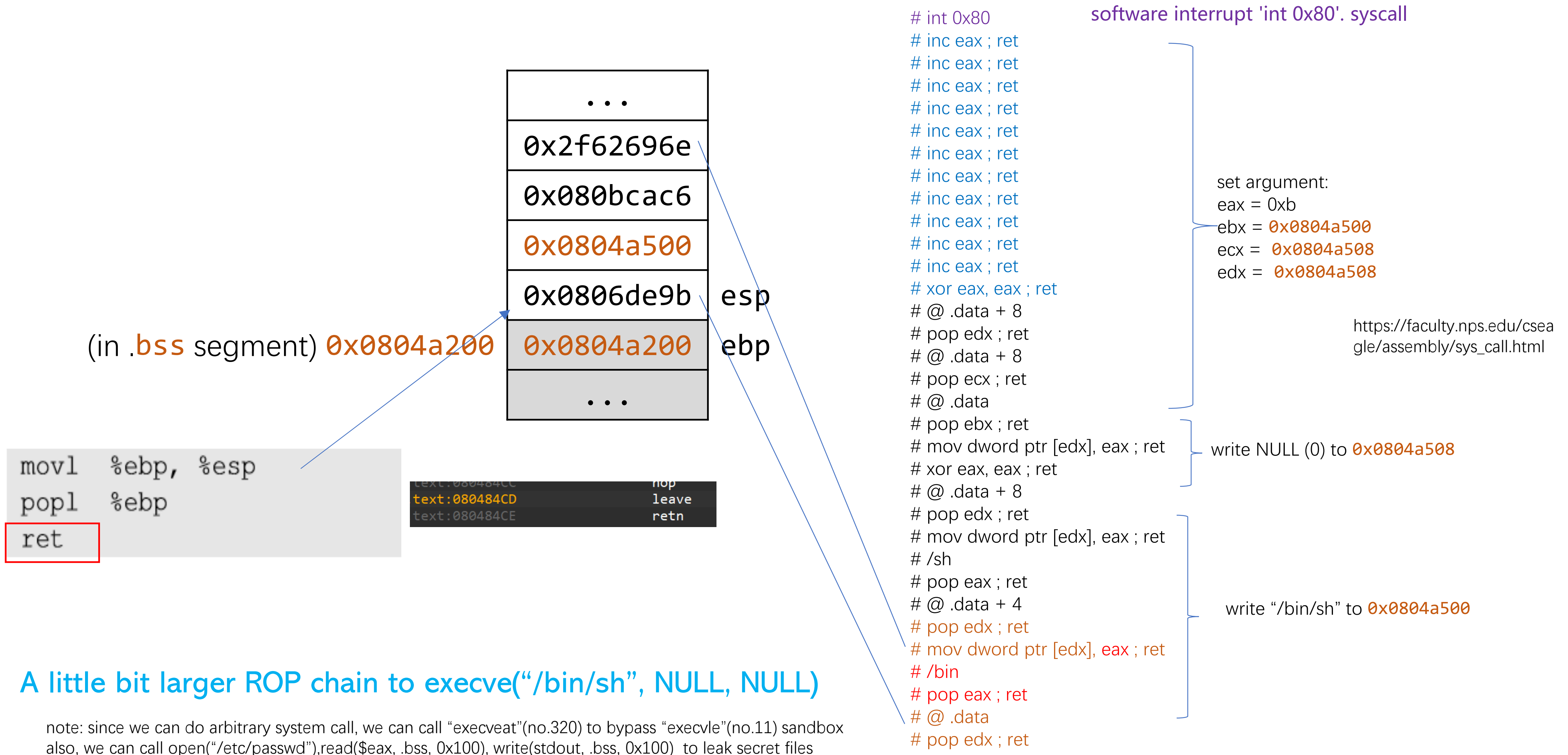
Design ROP chain-A Deeper view



Design ROP chain-A Deeper view

Now we can begin a new trip of ROP!
(but without length and chunk limitation)

Design ROP chain-A Deeper view



A little bit larger ROP chain to execve("/bin/sh", NULL, NULL)

note: since we can do arbitrary system call, we can call "execveat"(no.320) to bypass "execvle"(no.11) sandbox
also, we can call open("/etc/passwd"),read($eax, .bss, 0x100), write(stdout, .bss, 0x100) to leak secret files

Protections

- ROP is powerful, but there still more powerful protections invented to mitigate ROP attack.
- Like a infinite cat-and-mouse game.

Protection: Canary/Cookie Protection

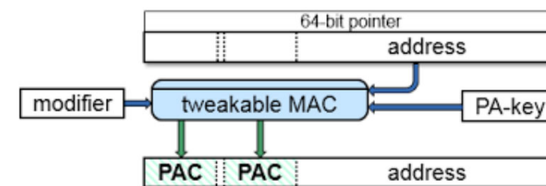
- (Canary/Cookie) can detect stack buffer overflow vulnerability when attacker overwrites the function return address in the stack frame
- Insert by compiler
- Defeat Canary:
    - Overwriting the Canary with the same value
    - – Brute force attack (e.g., DynaGuard in ACSAC'15)

More Protection: Control-flow integrity (CFI)

- include code-pointer separation (CPS), code-pointer integrity (CPI), stack canaries, shadow stacks, and vtable pointer verification.
- Widely used in Android

More Protection: Control-flow integrity (CFI)

- include code-pointer separation (CPS), code-pointer integrity (CPI), stack canaries, shadow stacks, and vtable pointer verification.
- Widely used in Android
- Other CFI implementation:
  - pointer authentication code PAC

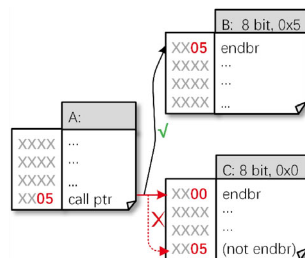


http://blog.ssg.aalto.fi/2019/06/protecting-against-run-time-attacks.html

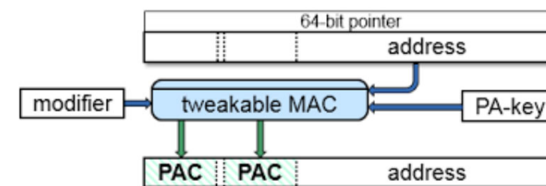
Explore PAC implementation in Apple:
https://www.usenix.org/system/files/usenixsecurity23-cai-zechao.pdf

More Protection:
Control-flow integrity (CFI)

- include code-pointer separation (CPS), code-pointer integrity (CPI), stack canaries, shadow stacks, and vtable pointer verification.
- Widely used in Android
- Other CFI implementation:
  - pointer authentication code PAC
  - AddressSanitizer
  - ENDBR



https://www.semanticscholar.org/paper/ABCFI%3A-Fast-and-Lightweight-Fine-Grained-Integrity-Li-Chen/1ddadfb44e66352a72550f3fc657be738858259e



http://blog.ssg.aalto.fi/2019/06/protecting-against-run-time-attacks.html

Explore PAC implementation in Apple:
https://www.usenix.org/system/files/usenixsecurity23-cai-zechao.pdf

Summary:

- The Non-executable-stack mechanism can be bypassed
- To conduct the attack, we need to understand lowlevel details about function invocation
- The technique can be further generalized to Return Oriented Programming (ROP)
- ROP can be mitigated by CFI check, the war between attack and defense never ends!
- We will try return-to-libc attack in lab exercise

About heap vulnerability:

- Use-After-Free
- Double-Free
- Unlink
- Heap Feng Shui
- Heap spray
- HeapOverflow

Those vulnerability and exploitation will not be included in class, Recommend link if you are interested:

https://heap-exploitation.dhavalkapil.com/

https://github.com/shellphish/how2heap

https://firmianay.gitbook.io/ctf-all-in-one/3_topics/pwn/3.1.6_heap_exploit_1