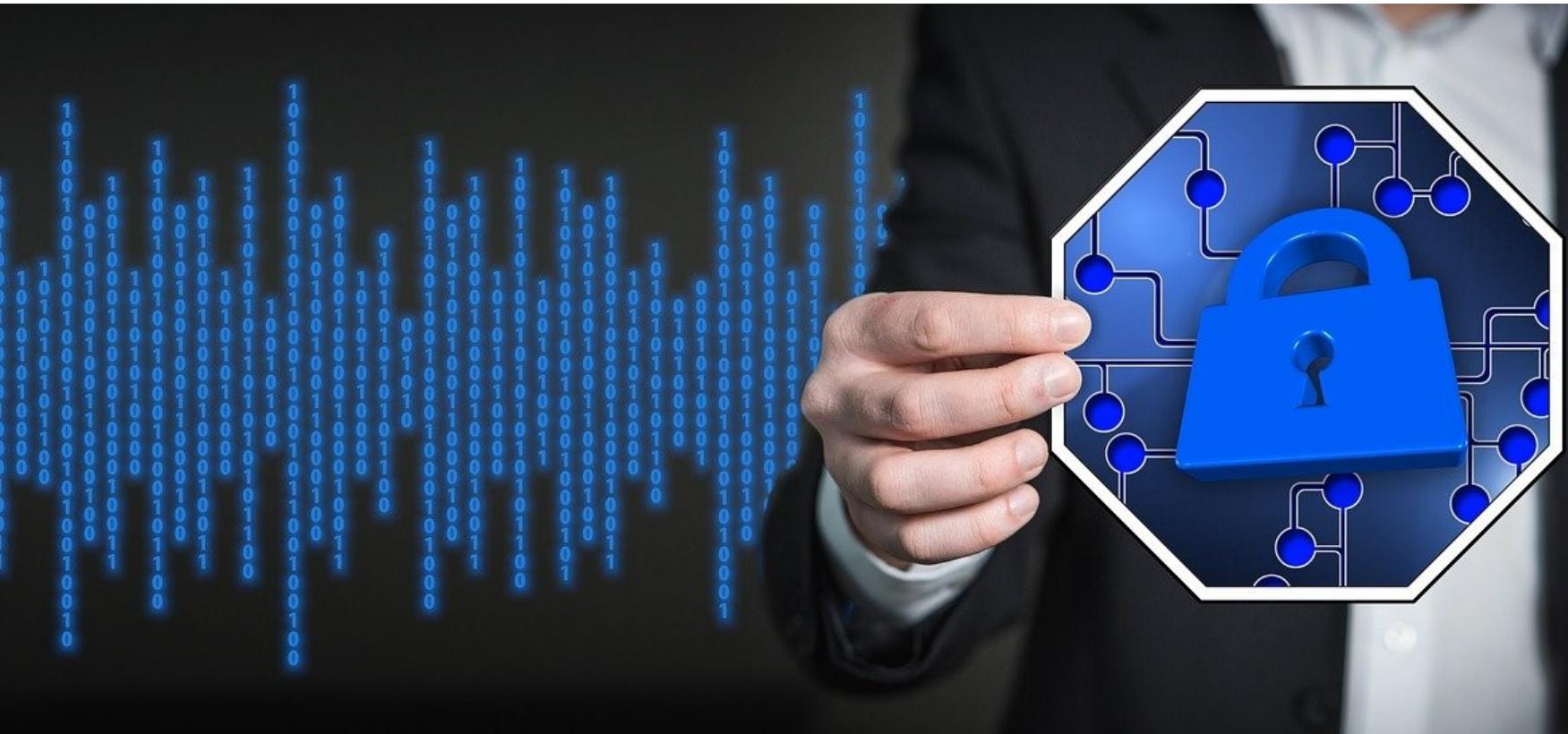




Trường ĐH CNTT – ĐHQG TP. HCM

NT521 - Lập trình an toàn & Khai thác lỗ hổng phần mềm

ROP & Heap overflow





Nội dung

- Tấn công ROP (Return Oriented Programming)
- Khai thác Lỗ hổng trên bộ nhớ Heap
 - **Khái quát về Heap**
 - **Lỗ hổng Heap Overflow**
 - Use-After-Free
 - Heap Spraying
 - Metadata Corruption





Tân công ROP (Return Oriented Programming)



Nhắc lại

- Khai thác lỗ hổng tràn bộ đệm - **buffer overflow**:
 - Trường hợp lý tưởng:
 - Có khả năng kiểm soát return addr: không dùng stack canary, có thể ghi đè đến vị trí return addr
 - Có thể truyền shellcode vào stack và thực thi: -z execstack
 - Các trường hợp ràng buộc hơn?
 - **Off-by-one**
 - *Giới hạn kích thước buffer có thể bị ghi đè, không đủ để ghi đè return addr*
 - **Return-to-libc**
 - *Shellcode cần viết quá phức tạp hoặc quá dài, stack không cho phép thực thi code*
 - **ROP (Return Oriented Programming)**
 - *Stack không cho phép thực thi*





Tấn công ROP: Vì sao?

- Tấn công tràn bộ đệm trên stack đôi khi không cho phép thực thi code trên stack.

```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
```

- ***Non-executable stack - DEP***
- Biên dịch với ***-z noexecstack*** (mặc định)





Tấn công ROP: Vì sao? (2)

• DEP – Data Execution Prevention

- Một kỹ thuật giảm thiểu tấn công bằng cách đảm bảo chỉ có **section chứa code** mới được gán nhãn là **thực thi được**.
- Giảm thiểu code injection hay payload chứa shellcode.
- Một số ký hiệu: DEP, NX, W^X.

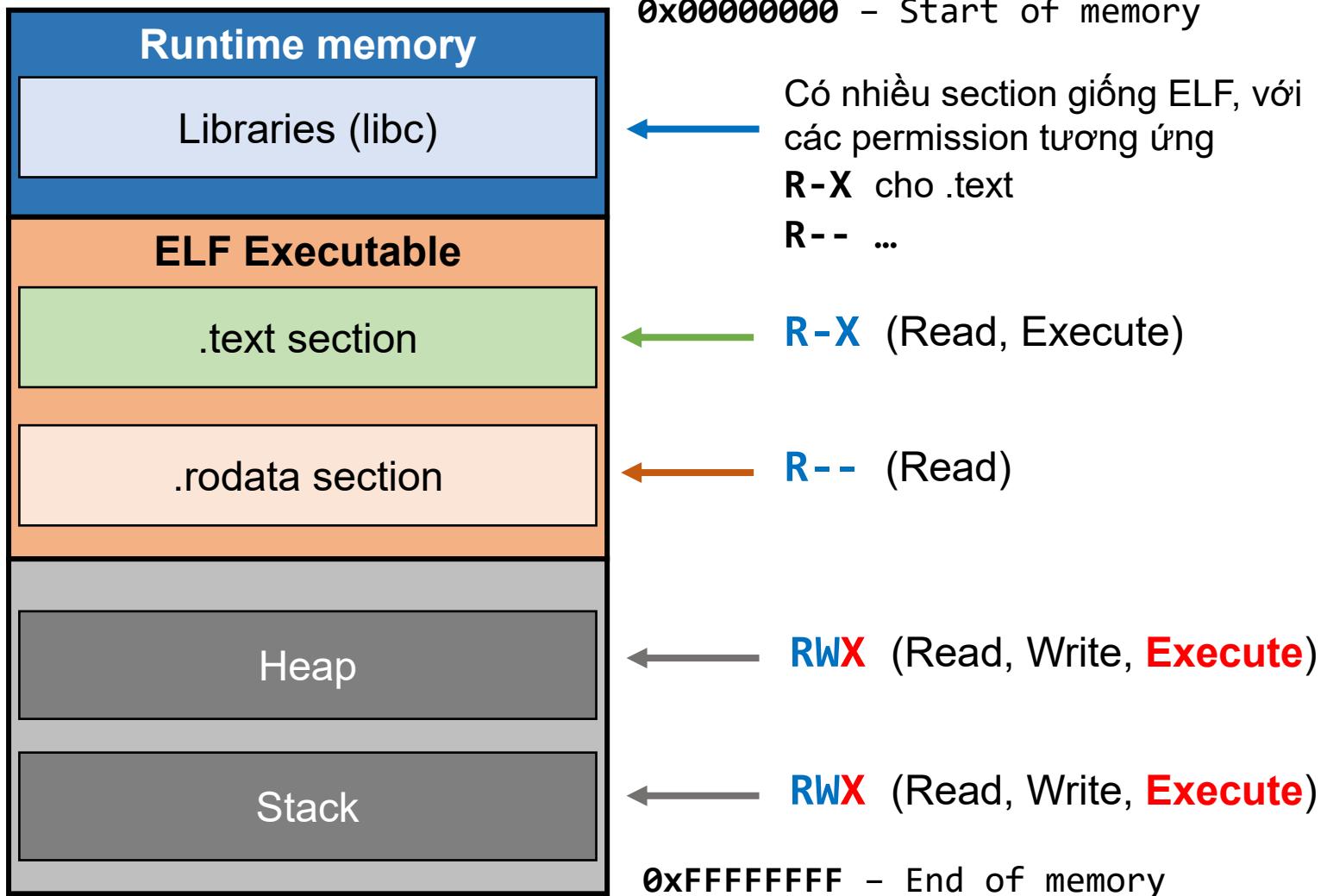
```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
```

- Cơ bản về DEP:
 - **Không có** section/segment bộ nhớ nào được phép **cùng lúc** cho phép **Ghi và Thực thi: W^X**.
 - Các section dữ liệu (data): stack, heap, .bss, .ro, .data
 - Các section chứa code: .text, .plt



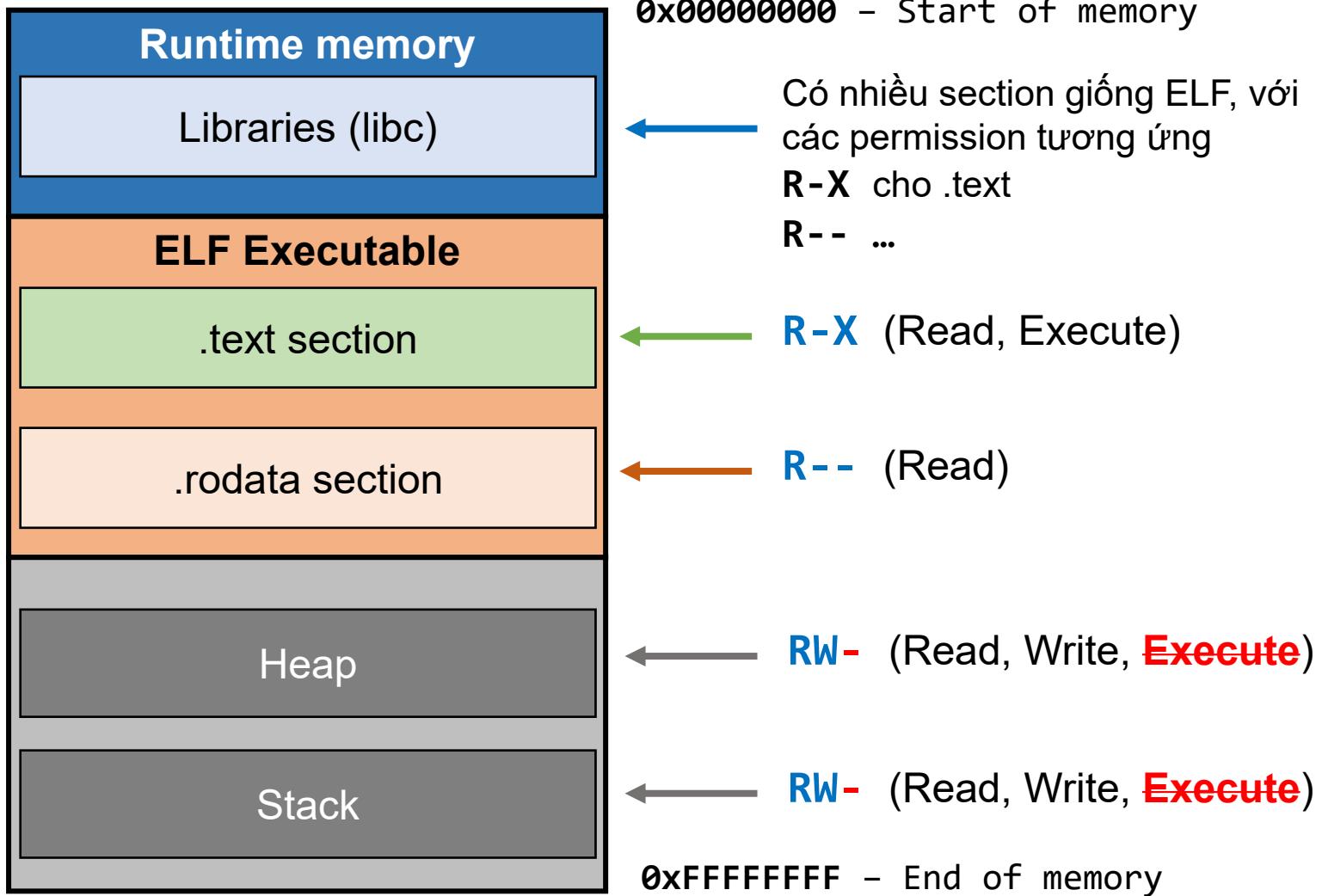
Tấn công ROP: Vì sao? (3)

- Nếu không dùng DEP?



Tấn công ROP: Vì sao? (4)

- Nếu dùng DEP?





Tấn công ROP: Vì sao? (5)

- Truyền shellcode vào stack để thực thi khi có sử dụng DEP?

→ **Segmentation Fault**





Tấn công ROP: Bypass DEP

- **Ý tưởng:**
 - Tận dụng code có sẵn trong chương trình.
- **Vì sao không tấn công return-to-libc?**
 - Cần có hàm libc
 - Thực thi các hàm libc có thể có tác dụng phụ
 - Address randomization có thể khiến các địa chỉ hàm libc khó đoán.
- **ROP – Return Oriented Programming**
 - Điểm khác biệt: không cần dùng toàn bộ hàm, chỉ sử dụng các đoạn code “đặc biệt” có các lệnh cần thiết – **gadget**.
 - Có thể xem là các *hàm nhỏ*
 - *Luôn kết thúc bằng lệnh ret*
 - Nếu có thể tìm đủ các *hàm nhỏ* như vậy, attacker có thể thực hiện hành vi mong muốn.



Tấn công ROP: Gadgets

- **Gadgets**

- là một chuỗi liên tục các lệnh assembly có ý nghĩa.
- kết thúc bằng lệnh **ret**.

```
pop eax  
ret
```

```
mov [ecx], ebx  
ret
```

```
mov edx, eax  
mov ebx, ecx  
add eax, 20  
ret
```

```
lea [esp + 12]  
push edx  
ret
```

```
mov [ecx], 1  
ret
```

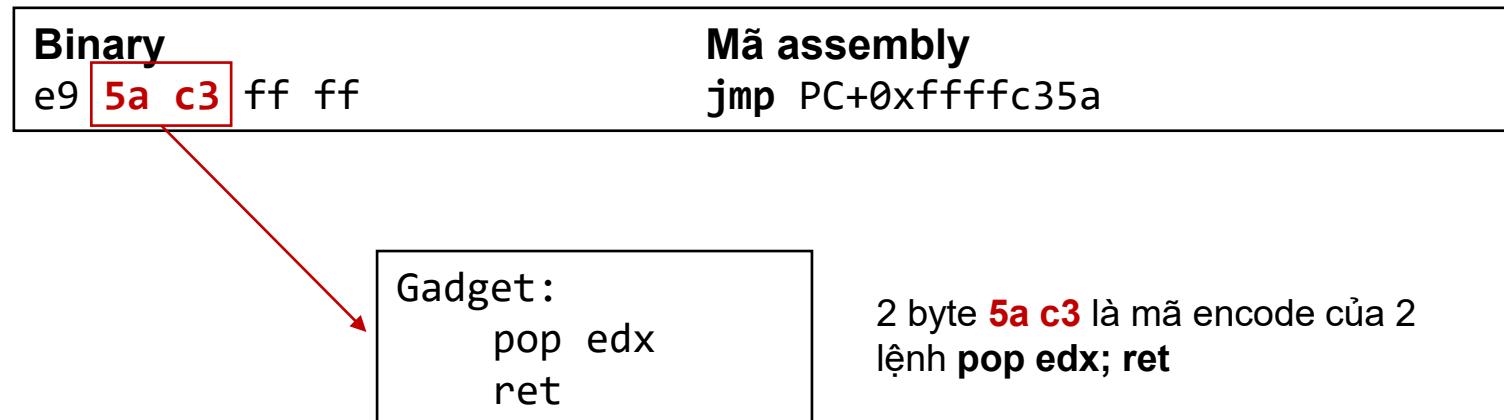
- Nhiều gadgets được nối (chain) với nhau để tạo thành 1 chuỗi lệnh hiện thực hành vi của attacker tương tự như shellcode.
 - **ROP Chain**



Tấn công ROP: Gadgets

- **Tìm gadget như thế nào?**

- Gadget có thể là các lệnh trong các hàm.
- Gadget có thể được giấu bên trong các byte của các lệnh assembly khác.



- ROPgadget

```
$ ./ROPgadget ./smashme.bin -att -asm "popl %edx; ret"

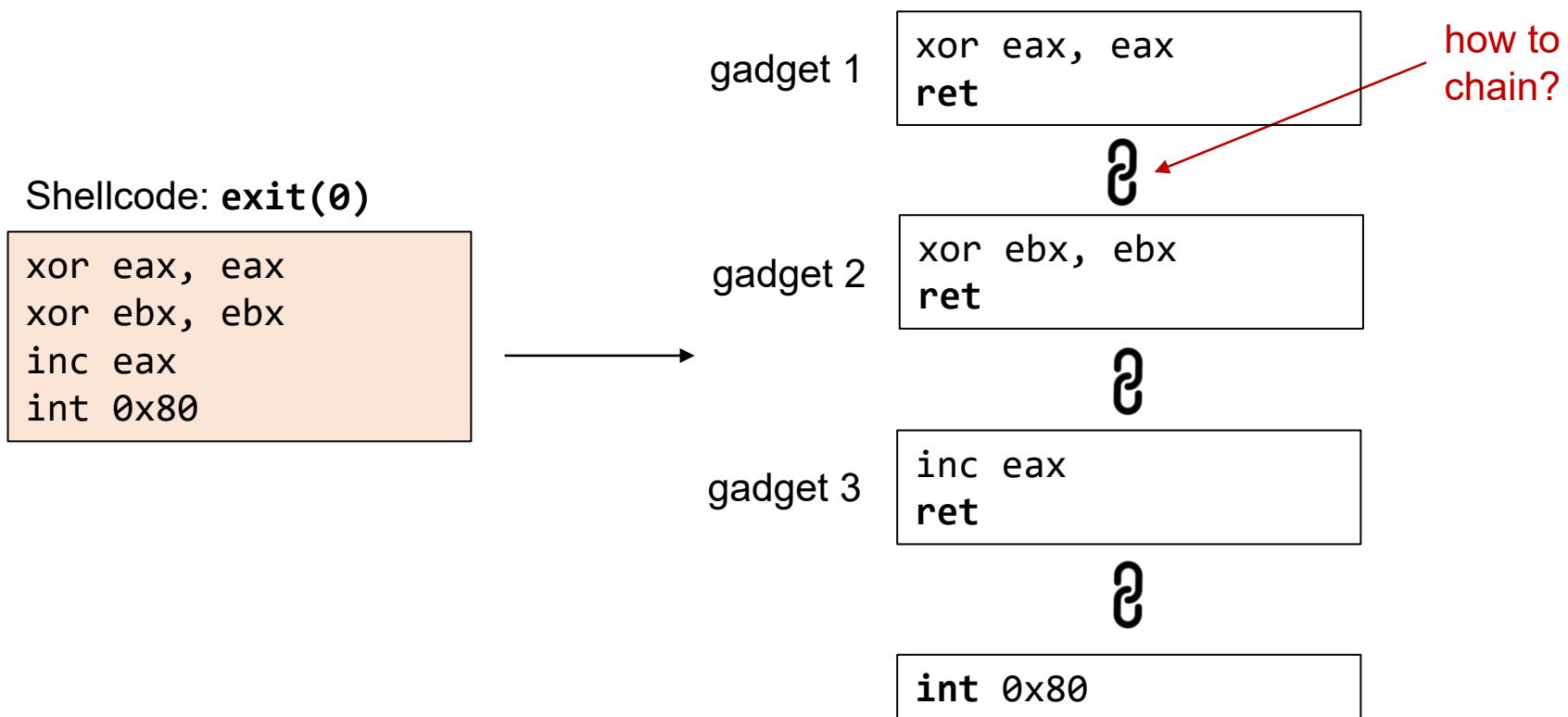
Gadgets information
=====
0x0806eb0a: "\x5a\xc3 <=> popl %edx; ret"
0x08081736: "\x5a\xc3 <=> popl %edx; ret"

Total opcodes found: 2
```



Tấn công ROP: ROP chain

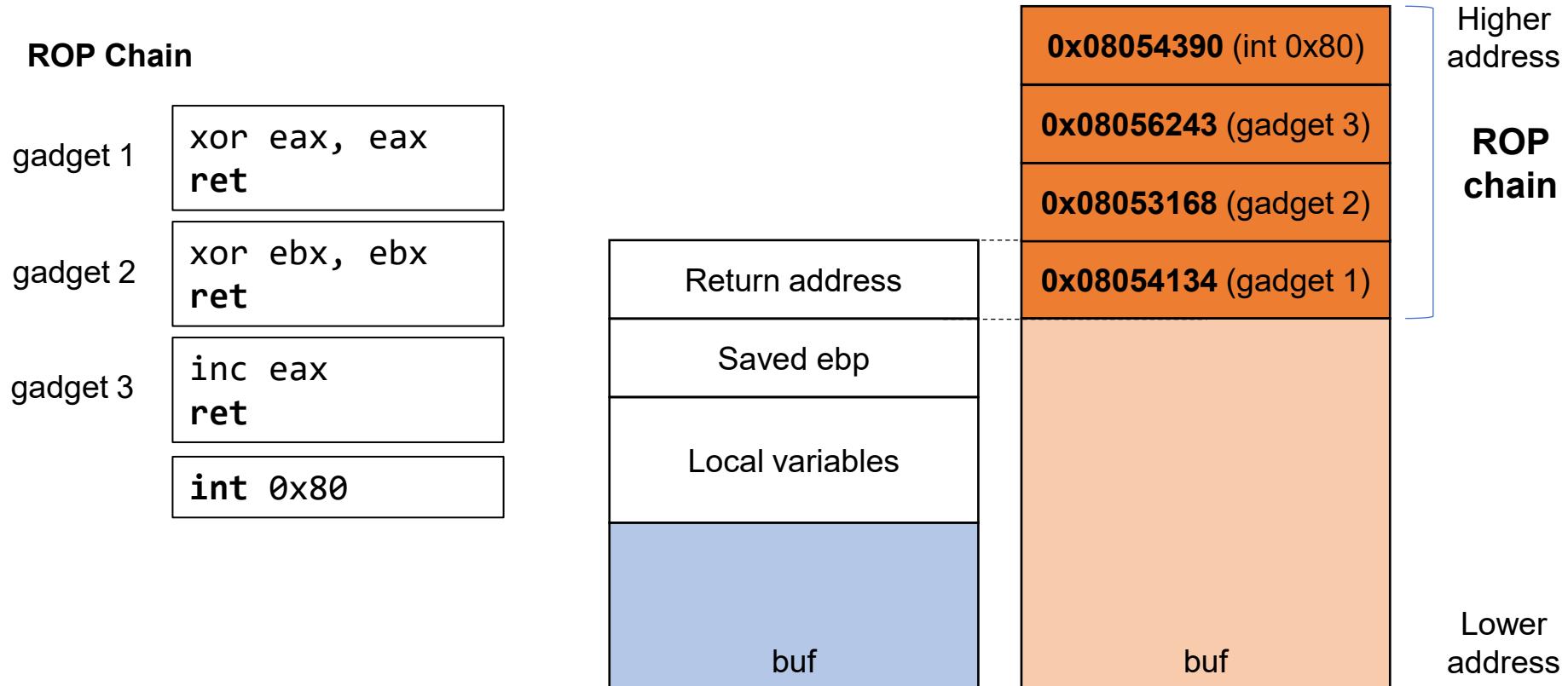
- ROP chain hoạt động như thế nào?
 - Làm sao để chain các gadget?



Tấn công ROP: ROP chain (2)

- **ROP chain hoạt động như thế nào?**

- Chain bằng cách sắp xếp các address của chúng ở các vị trí thích hợp trong stack.
- Address của gadget 1 sẽ ghi đè lên return address của chương trình.
- Address của gadget thực thi sau sẽ là return address của gadget thực thi trước.



Tấn công ROP: ROP chain (3)

- ROP chain hoạt động như thế nào?

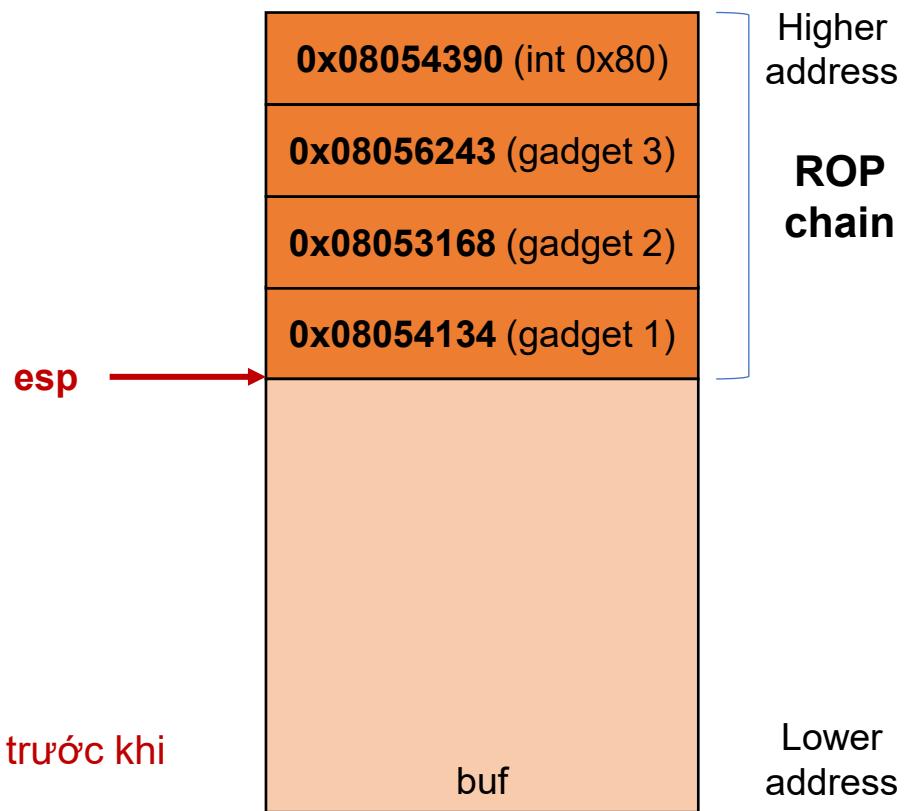
ROP Chain

gadget 1
 xor eax, eax
 ret

gadget 2
 xor ebx, ebx
 ret

gadget 3
 inc eax
 ret

int 0x80



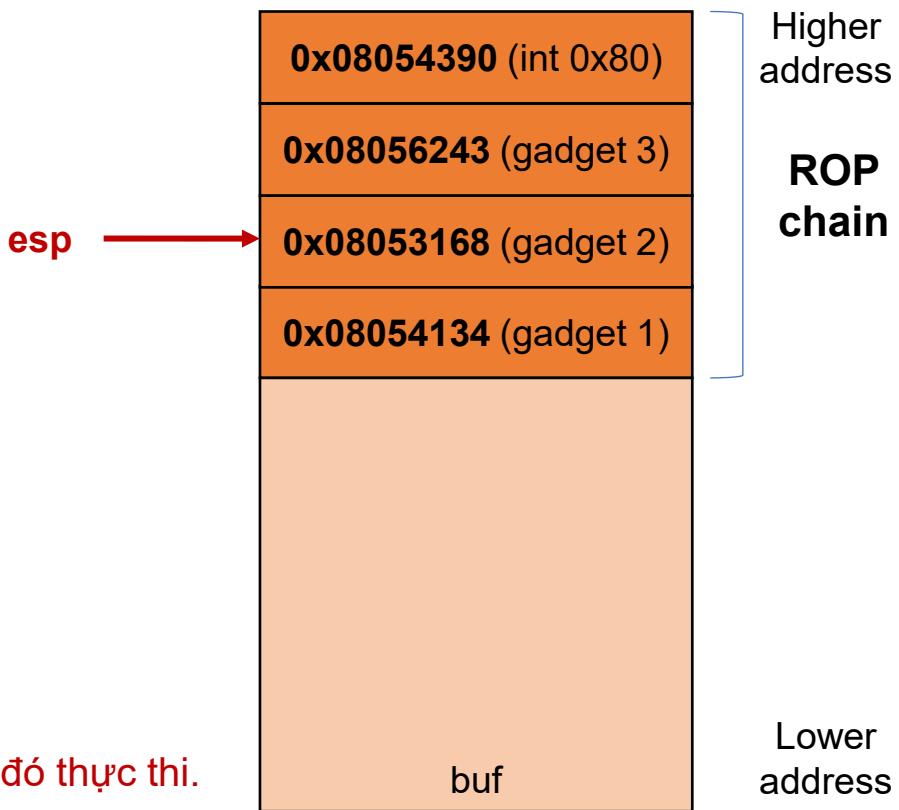
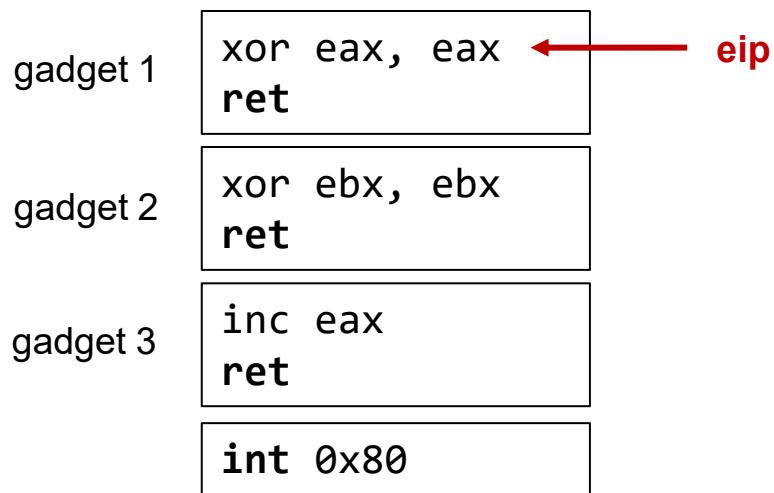
Sau khi nhập chuỗi **buf** để ghi đè return address và trước khi hàm bị khai thác trở về với lệnh **ret**.



Tấn công ROP: ROP chain (4)

- ROP chain hoạt động như thế nào?

ROP Chain



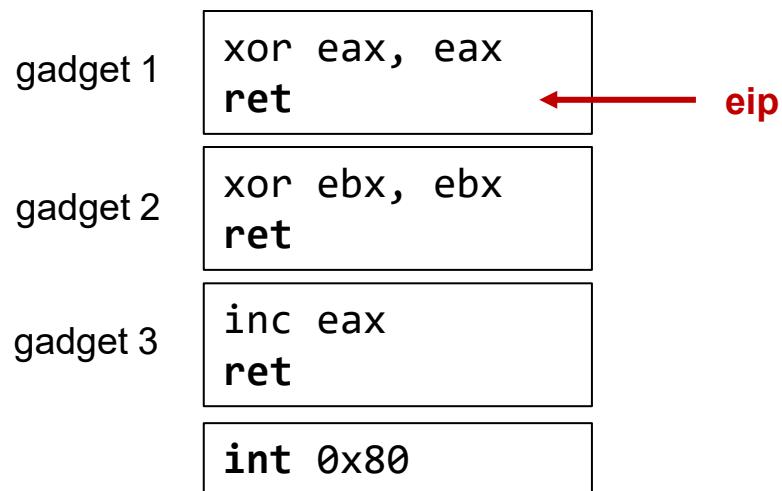
Khi hàm bị khai thác trở về:

- Address của gadget 1 được lấy ra và eip đi đến đó thực thi.
- esp tăng lên 4 trỏ đến address của gadget 2

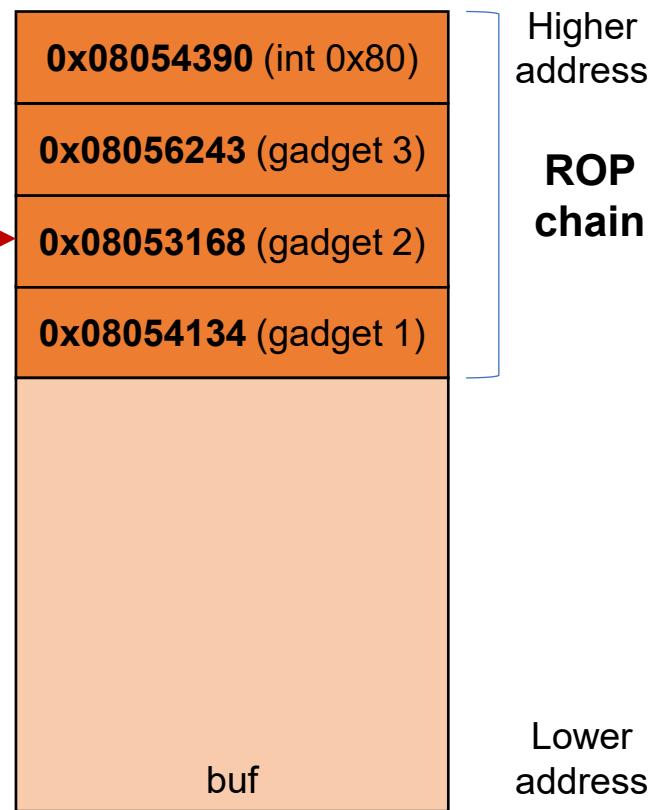
Tấn công ROP: ROP chain (4)

- ROP chain hoạt động như thế nào?

ROP Chain



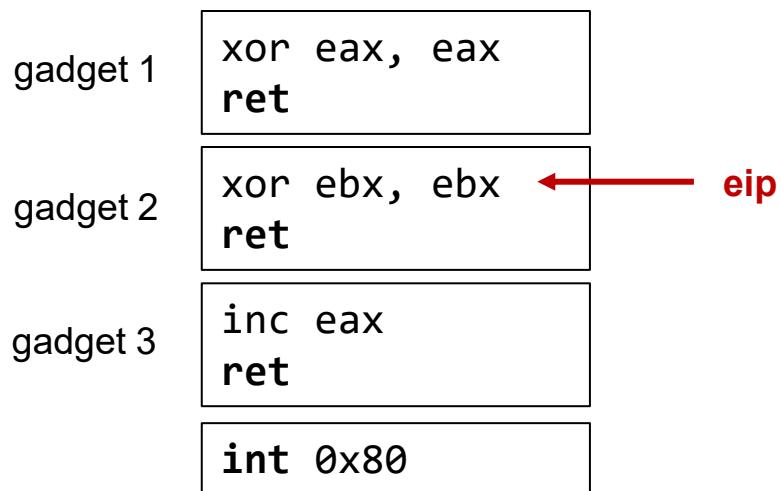
Stack trước khi gadget 1 thực thi lệnh ret



Tấn công ROP: ROP chain (5)

- ROP chain hoạt động như thế nào?

ROP Chain



Higher address

**ROP
chain**

Lower address

Khi gadget 1 thực thi ret để trở về:

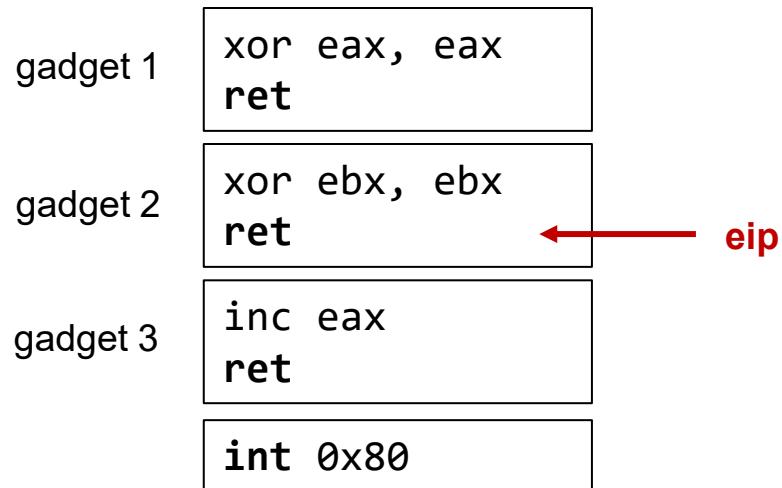
- Address của gadget 2 được lấy ra (từ vị trí esp đang trả đến) và eip đi đến đó thực thi.
- esp tăng lên 4 trả đến address của gadget 3



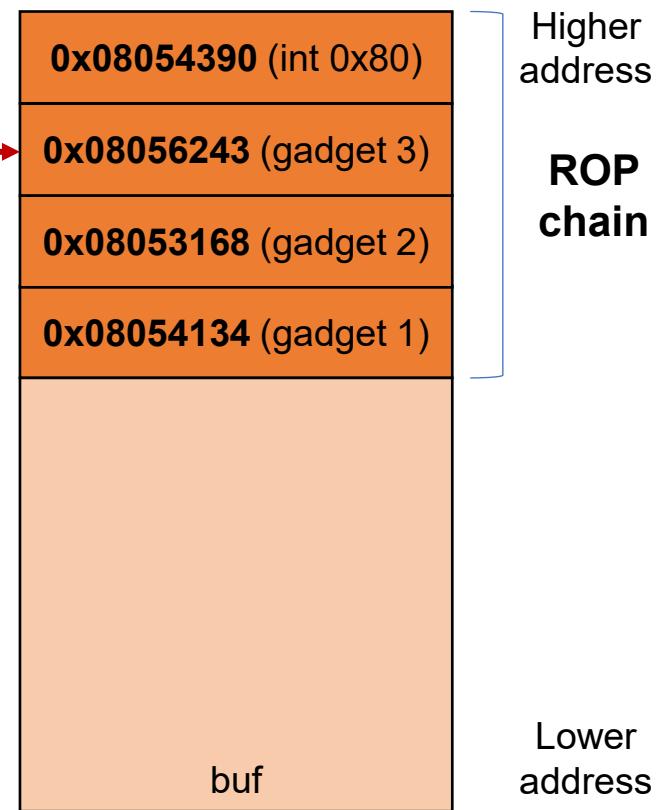
Tấn công ROP: ROP chain (6)

- ROP chain hoạt động như thế nào?

ROP Chain



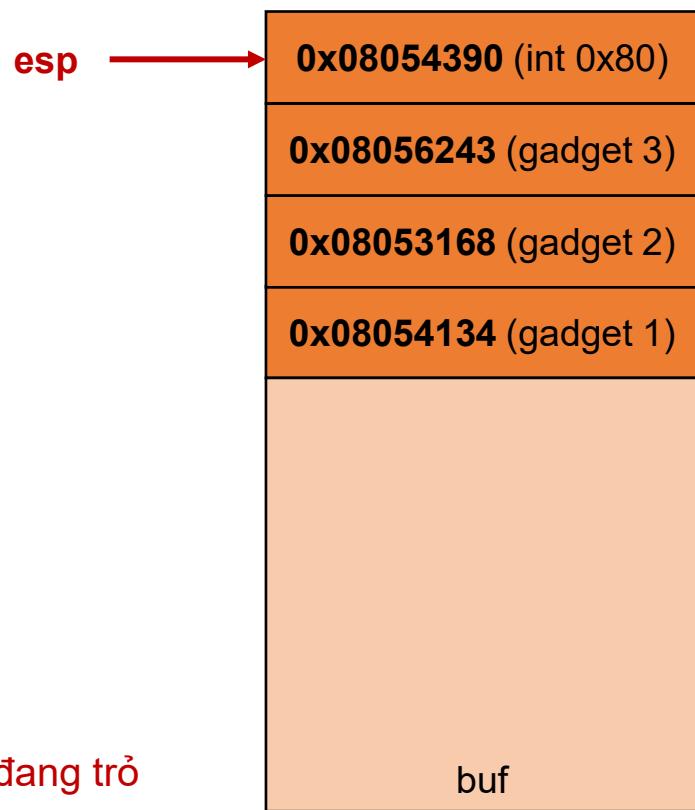
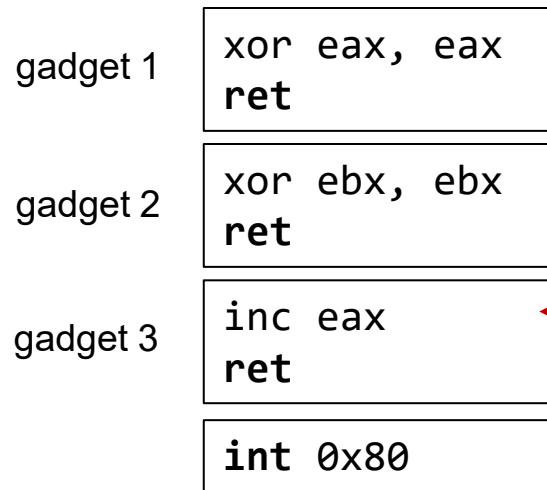
Stack trước khi gadget 2 thực thi lệnh ret



Tấn công ROP: ROP chain (7)

- ROP chain hoạt động như thế nào?

ROP Chain



Khi gadget 2 thực thi ret để trở về:

- Address của gadget 3 được lấy ra (từ vị trí esp đang trả đến) và eip đi đến đó thực thi.
- esp tăng lên 4 trả đến address của lệnh **int 0x80**



Tấn công ROP: ROP chain (8)

- ROP chain hoạt động như thế nào?

ROP Chain

gadget 1
xor eax, eax
ret

gadget 2
xor ebx, ebx
ret

gadget 3
inc eax
ret

int 0x80

esp



0x08054390 (int 0x80)

0x08056243 (gadget 3)

0x08053168 (gadget 2)

0x08054134 (gadget 1)

Higher address

**ROP
chain**

eip



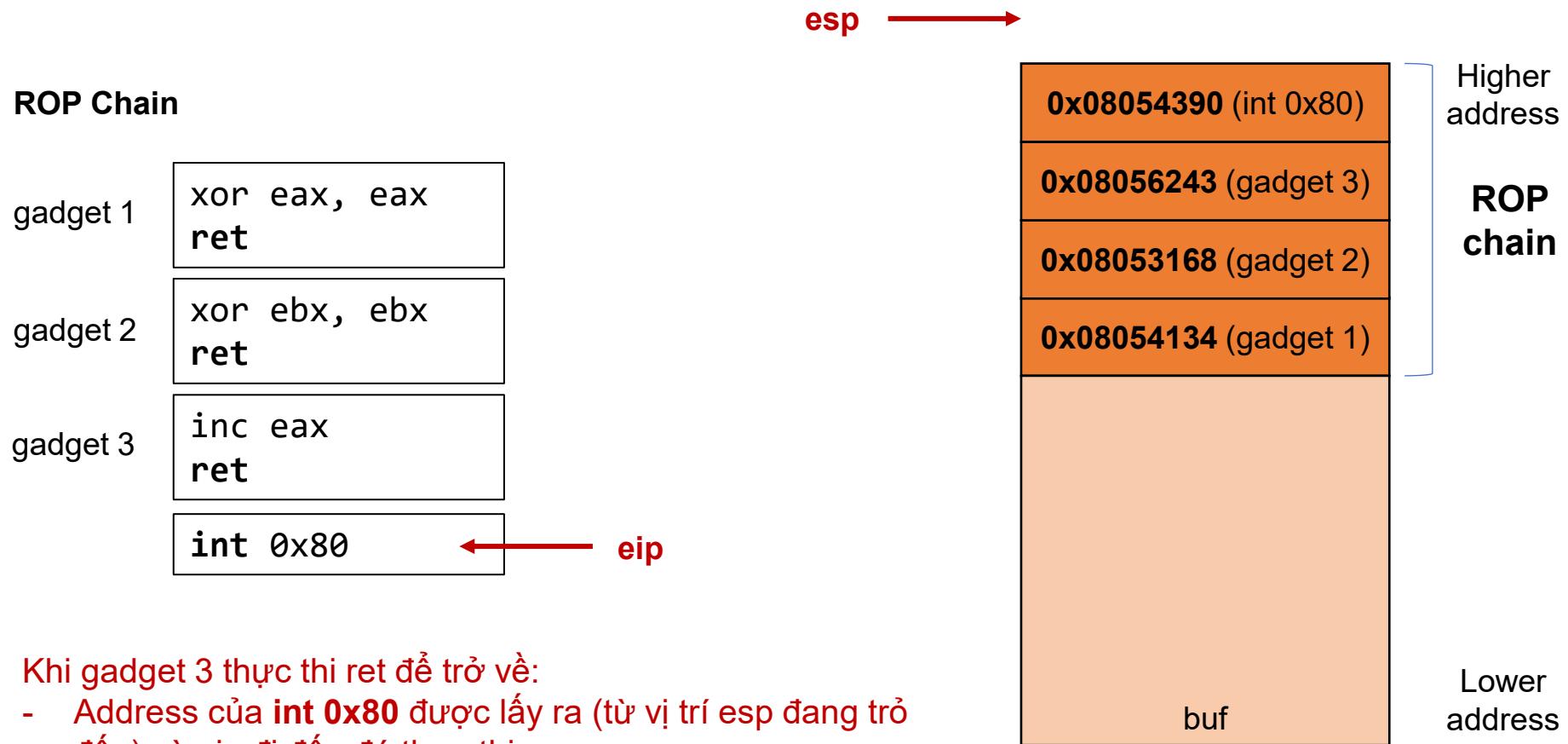
Stack trước khi gadget 3 thực thi lệnh ret

buf

Lower address

Tấn công ROP: ROP chain (9)

- ROP chain hoạt động như thế nào?





Tấn công ROP: Ví dụ 2

- Shellcode cần thực hiện

```
mov eax, 4          # eax = 4 (sys_write)
mov ebx, 1          # ebx = 1 (stdout)
mov ecx, str_addr   # ecx = address of string to print
mov edx, 13         # edx = length of string to print
int 0x80            # system call
```

- Các gadget tìm được

```
pop eax  
ret
```

```
leal ecx, [esp+12]  
ret
```

```
int 0x80  
ret
```

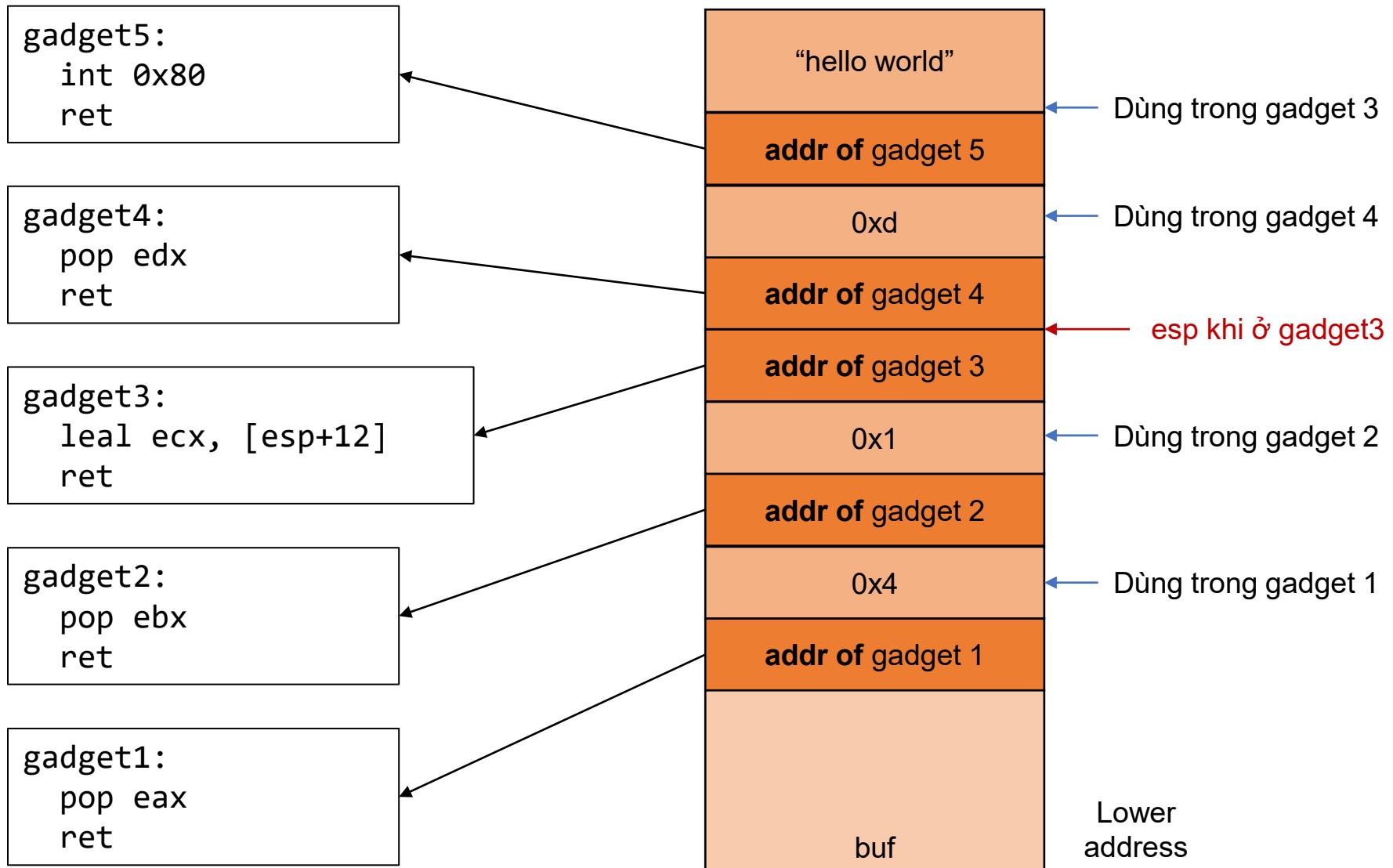
```
pop ebx  
ret
```

```
pop edx  
ret
```

- Có thể không tìm thấy các gadget để trực tiếp gán các giá trị như 4 hay 13 cho các thanh ghi
 - Các gadget trên lấy 1 giá trị nào đó trong stack gán cho các thanh ghi
- Có thể đặt sẵn các giá trị cần gán trong stack để gadget dùng.



Tấn công ROP: Ví dụ 2





Tấn công ROP: Stack pivoting

- *Sinh viên tự tìm hiểu Stack pivoting*



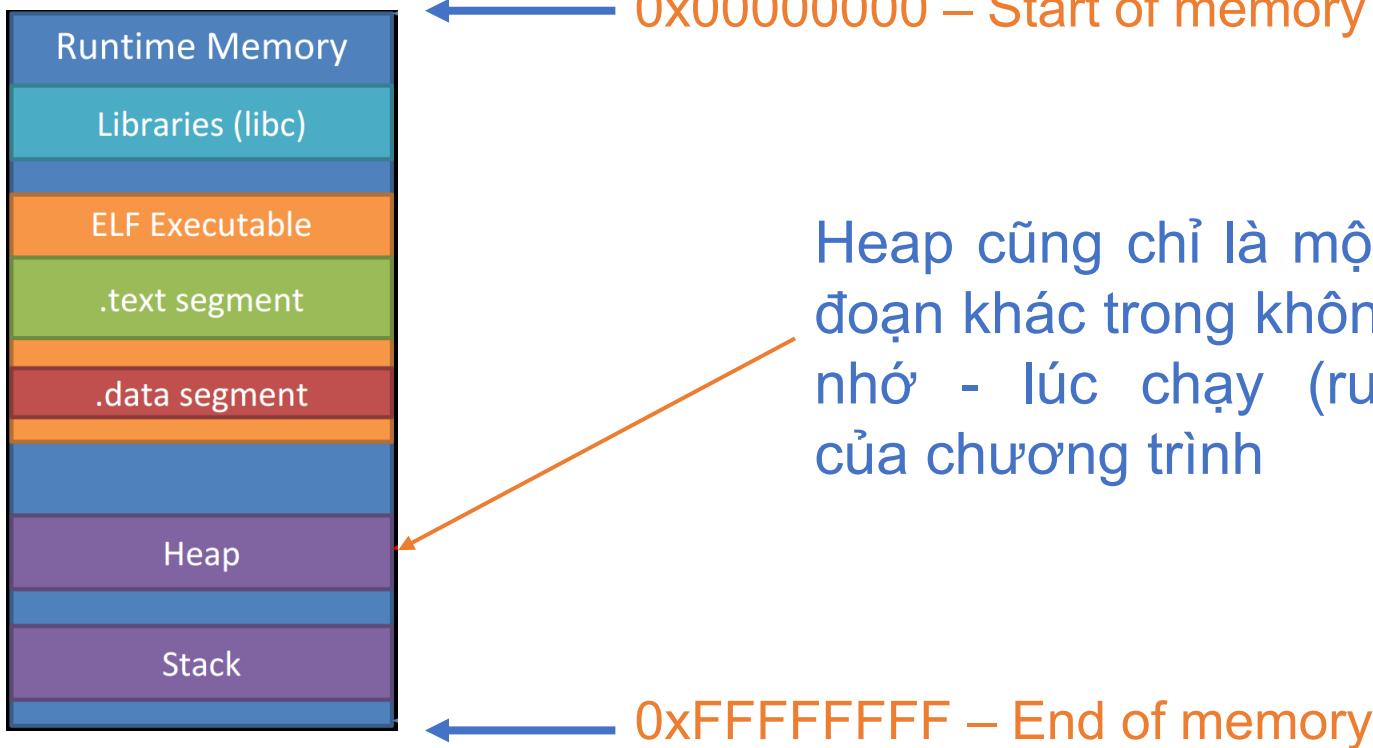
Khai thác lỗ hổng trên bộ nhớ Heap

- Cơ bản về bộ nhớ Heap
- Heap Overflow
- Use-After-Free
- Heap Spraying
- Metadata Corruption

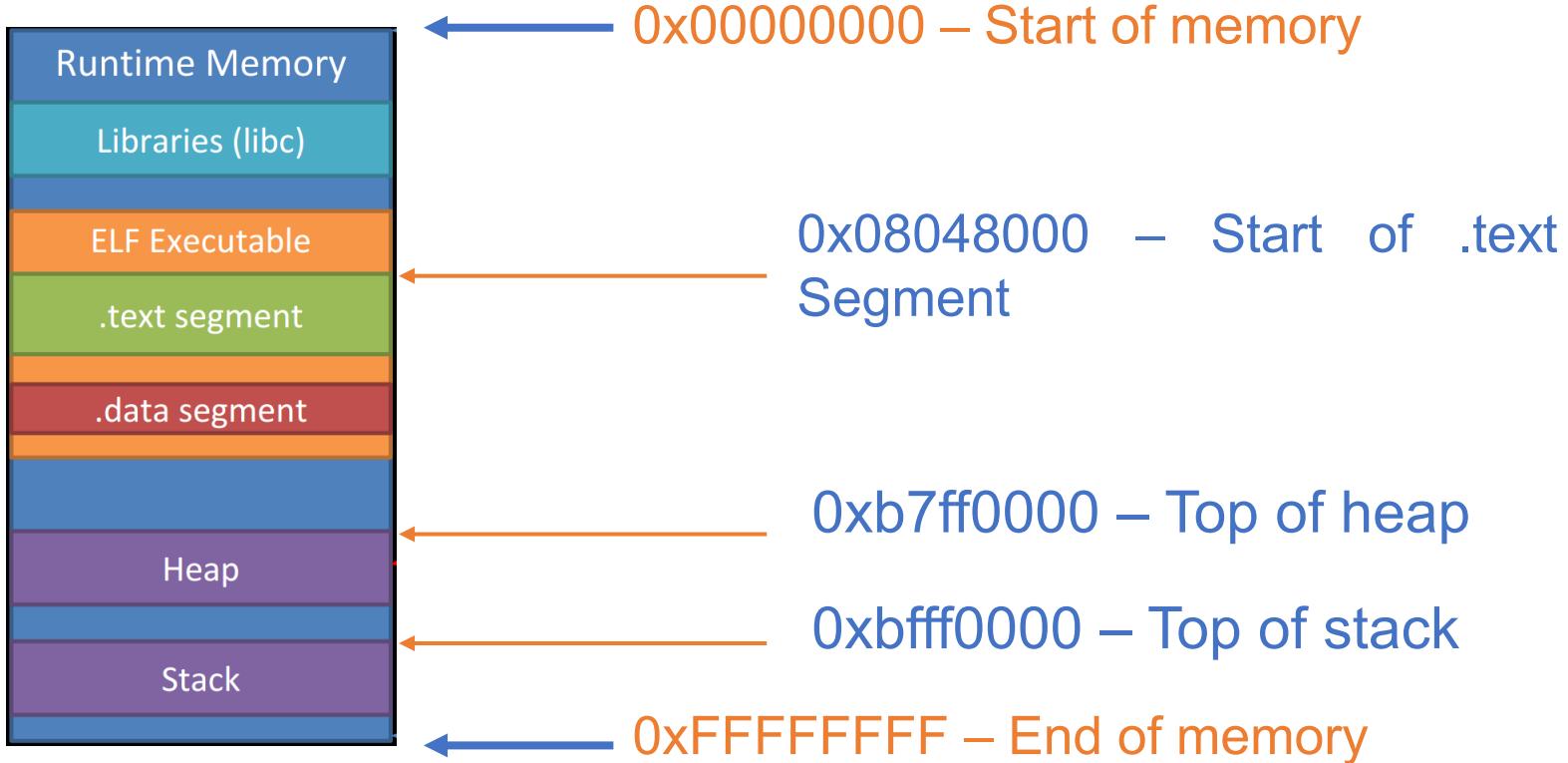


Cơ bản về bộ nhớ Heap

- Heap là vùng không gian bộ nhớ được dùng cho thao tác cấp phát động tại thời điểm chạy của chương trình:
 - Cấp phát động vùng nhớ với hàm **malloc()**
 - Giải phóng vùng nhớ trên heap với hàm **free()**



Cơ bản về bộ nhớ Heap



Bộ nhớ Heap

Lưu ý:

- Bộ nhớ để lưu trữ cho các biến con trả được cấp phát động bởi các hàm **malloc**, **calloc**, **realloc**
- Kích thước không cố định, có thể tăng giảm do đó đáp ứng được nhu cầu lưu trữ dữ liệu của chương trình
- Dữ liệu sẽ không bị hủy khi hàm thực hiện xong
→ phải tự tay hủy vùng nhớ (câu lệnh **free** đối với C, và **delete** hoặc **delete []** đối với C++), nếu không sẽ xảy ra hiện tượng rò rỉ bộ nhớ)





Bộ nhớ Heap

- Cho đoạn chương trình sau:

```
int main(){
    char * buffer = NULL;
    buffer = malloc (0x100);
    fgets(stdin, buffer, 0x100);
    printf("Hello %s!\n", buffer);
    free(buffer);
    return 0;
}
```





Bộ nhớ Heap

- Có nhiều phiên bản cài đặt/hiện thực Heap:
 - dlmalloc – trình cấp phát tổng quát
 - ptmalloc – trình cấp phát trên glibc
 - tcmalloc – trình cấp phát Google
 - jemalloc – FreeBSD và Firefox
 - nedmalloc
 - Hoard

Tham khảo:

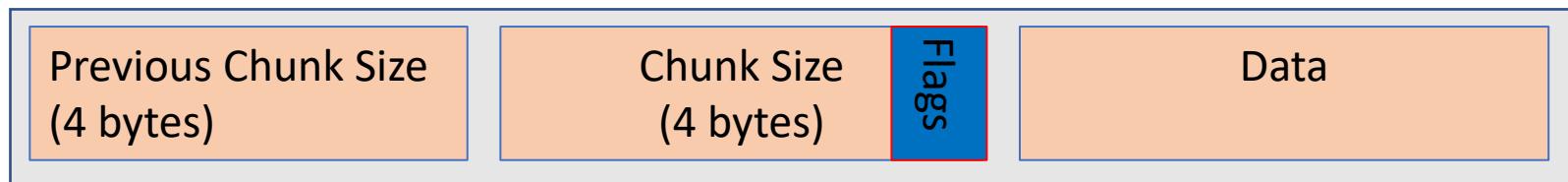
<https://sploitfun.wordpress.com/2015/02/10/understanding-glibc-malloc/>



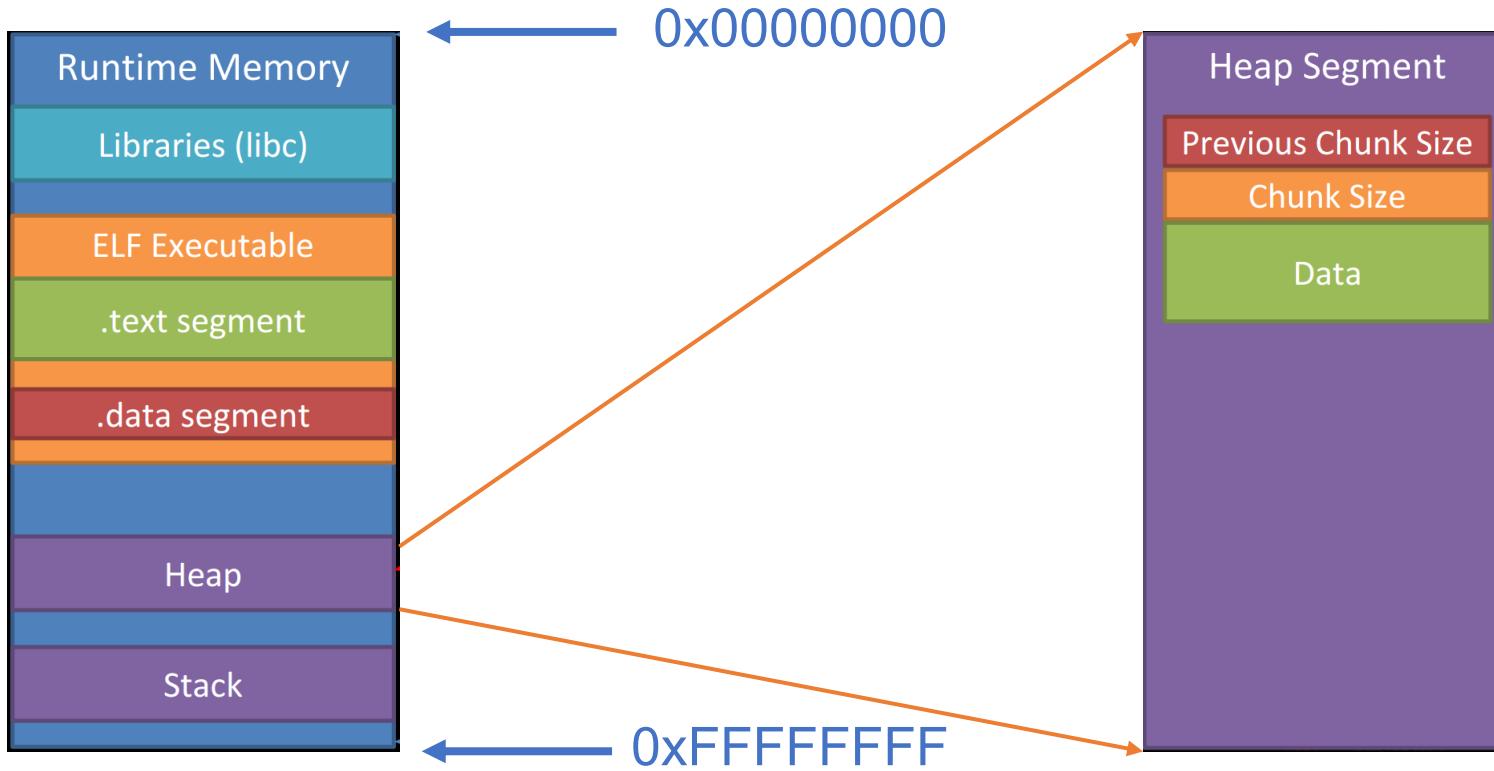
Heap Chunk: Cấp phát vùng nhớ

- Cấu trúc dữ liệu Heap Chunk (trong **ptmalloc**):

```
struct malloc_chunk {  
    // size of "previous" chunk  
    // (only valid when the previous chunk is freed, P=0)  
    size_t prev_size;  
  
    // size in bytes (aligned by double words): lower bits  
    // indicate various states of the current/previous chunk  
    // A: allocoed in a non-main arena  
    // M: mmapped  
    // P: "previous" in use (i.e., P=0 means freed)  
    size_t size;  
  
    [...]  
};
```

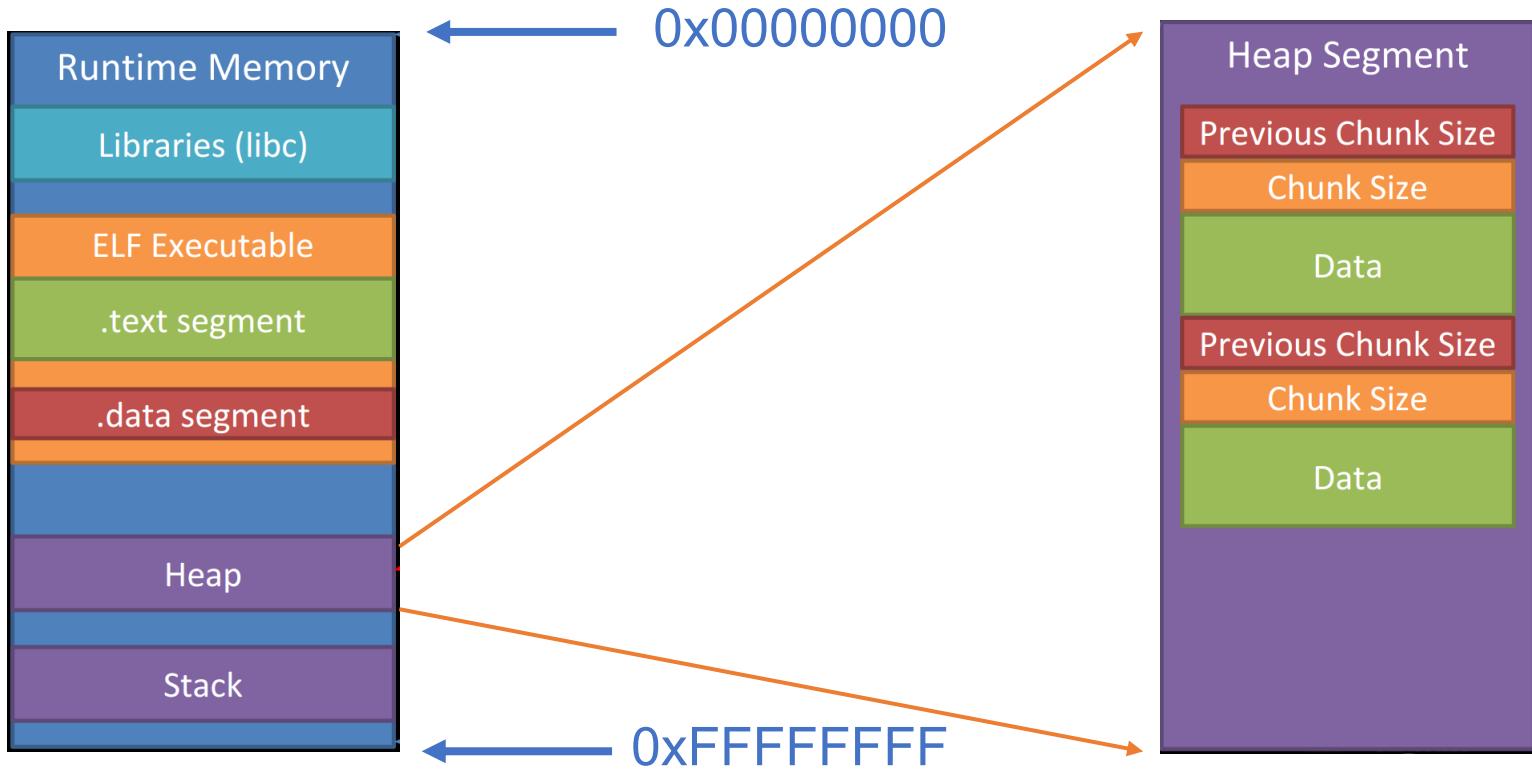


Cơ bản về bộ nhớ Heap



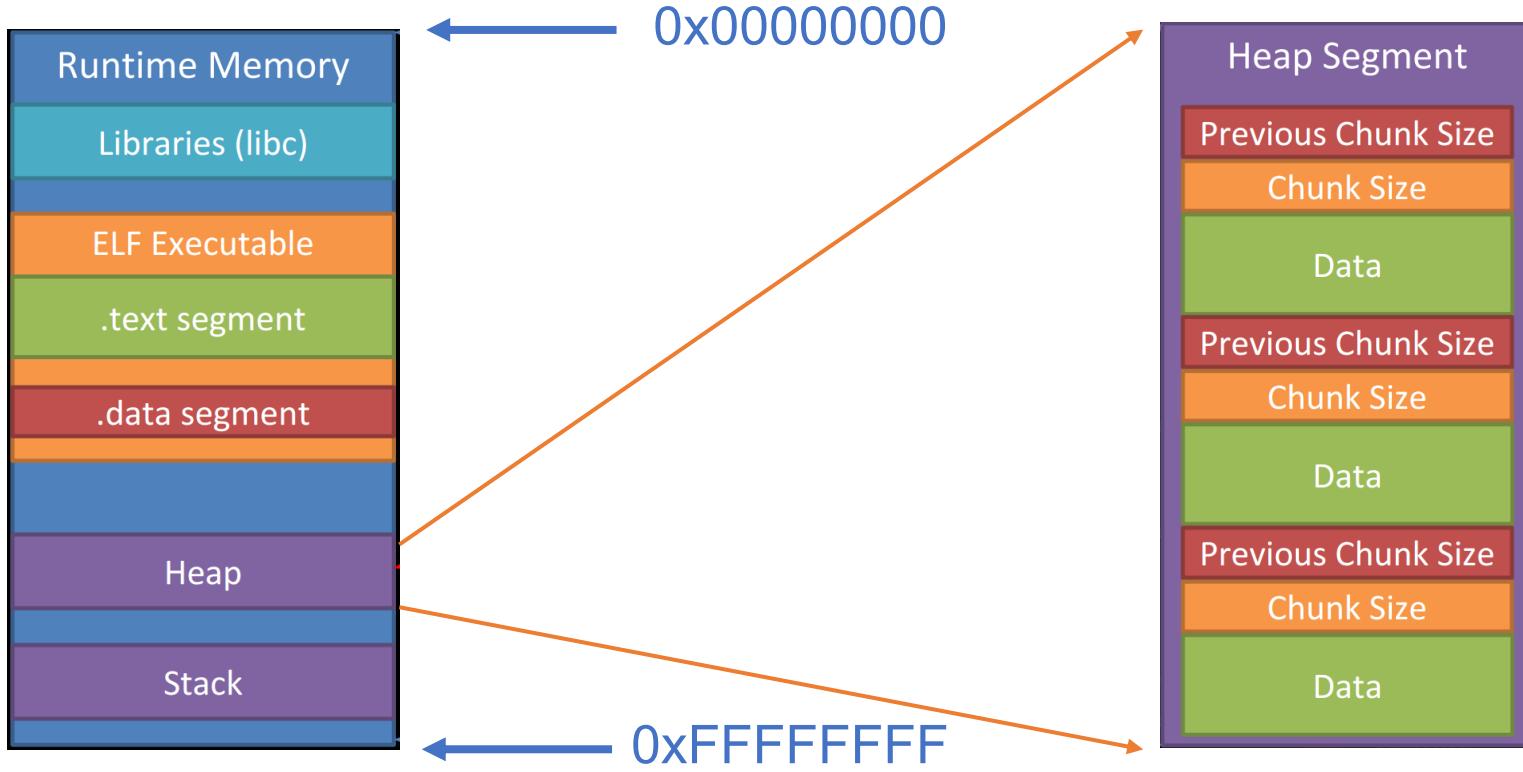
Phát triển theo vùng nhớ có địa chỉ cao

Cơ bản về bộ nhớ Heap



Phát triển theo vùng nhớ có địa chỉ cao

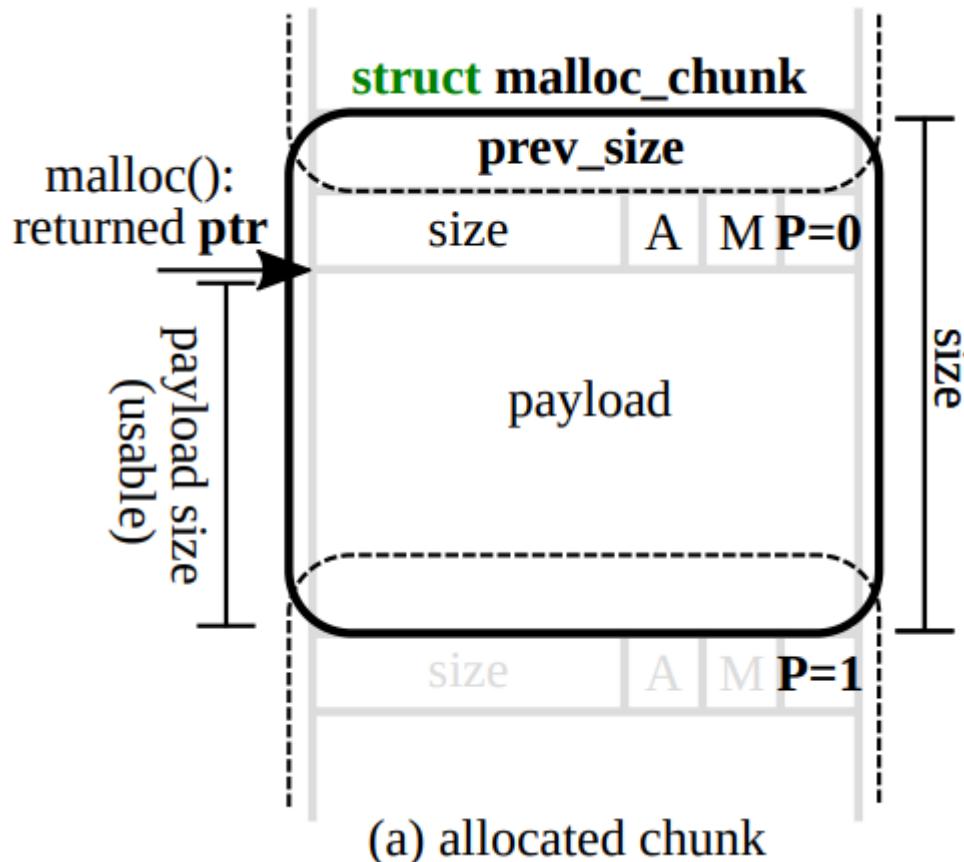
Cơ bản về bộ nhớ Heap



Phát triển theo vùng nhớ có địa chỉ cao

Heap Chunk: Cấp phát vùng nhớ

- Cấu trúc dữ liệu (trong ptmalloc):
- Mô tả chunk được cấp phát:



Heap chunk: Cấp phát vùng nhớ

- Qui trình cấp phát vùng nhớ, ứng với con trỏ **ptr** được cấp phát với những tác vụ như sau:
 - Tra cứu kích thước con trỏ
 - Kiểm tra đối tượng trước đó (previous) đã được cấp phát/giải phóng ($P=0$ or 1)
 - Lắp cho tới đối tượng tiếp theo
 - Kiểm tra đối tượng kế tiếp theo đã được cấp phát/giải phóng



Heap chunk: Giải phóng vùng nhớ

```
struct malloc_chunk {  
    [...]  
    // double links for free chunks in small/large bins  
    // (only valid when this chunk is freed)  
    struct malloc_chunk* fd;  
    struct malloc_chunk* bk;  
  
    // double links for next larger/smaller size in largebins  
    // (only valid when this chunk is freed)  
    struct malloc_chunk* fd_nextsize;  
    struct malloc_chunk* bk_nextsize;  
};
```

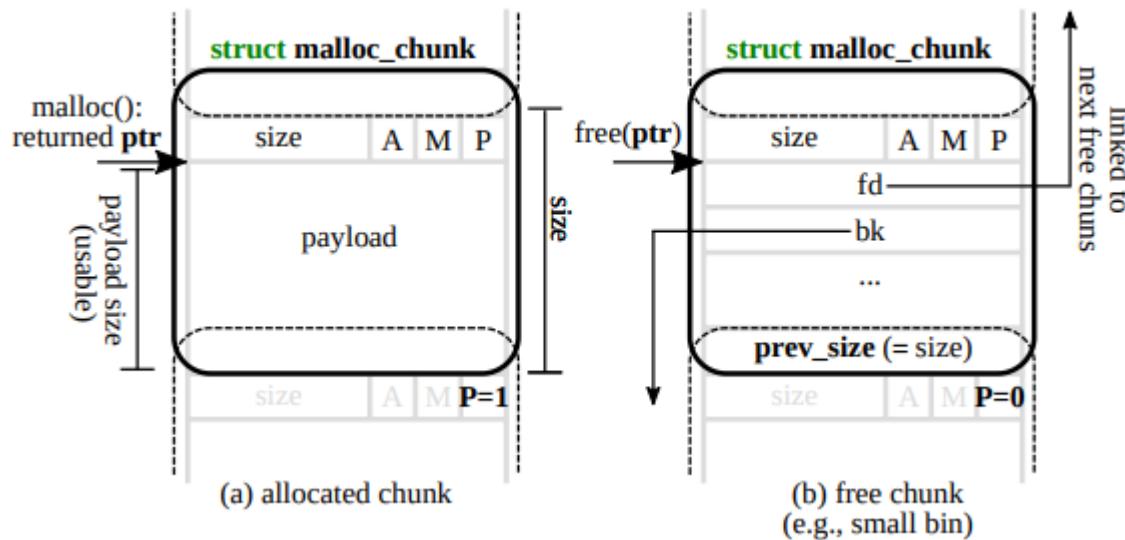
- Memory free: cho 1 con trỏ **ptr** ở trạng thái **free-ed** (giải phóng):
 - Bao gồm các tính năng của con trỏ **ptr** được cấp phát (đã đề cập ở trước đó)
 - Lắp tới các đối tượng **previous/next** trống (**free**) thông qua liên kết **fd/bk**
 - Khi gọi **free()**, kiểm tra các đối tượng **previous/next** có trống hay không và kết nối lại.



Bộ nhớ Heap: Tóm tắt

Cơ chế của Trình cấp bộ nhớ Heap cho phép:

- Tối đa hóa sử dụng bộ nhớ: sử dụng vùng nhớ đang trống
- Cấu trúc dữ liệu cho phép giảm thiểu vấn đề phân mảnh (fragmentation), (throughout linking fd/bk)
- Cấu trúc dữ liệu cho phép tối ưu hóa hiệu năng ($O(1)$ trong sử dụng free/malloc)





Phân loại lỗ hổng trên Heap

- Thông dụng/phổ biến:
 - Buffer Overflow/underflow
 - Out-of-bound read
- Đặc trưng trên Heap:
 - Use-after-free (UAF) (e.g., dangled pointers)
 - Incorrect uses (e.g., double frees)
 - Heap spraying
 - Metadata corruption



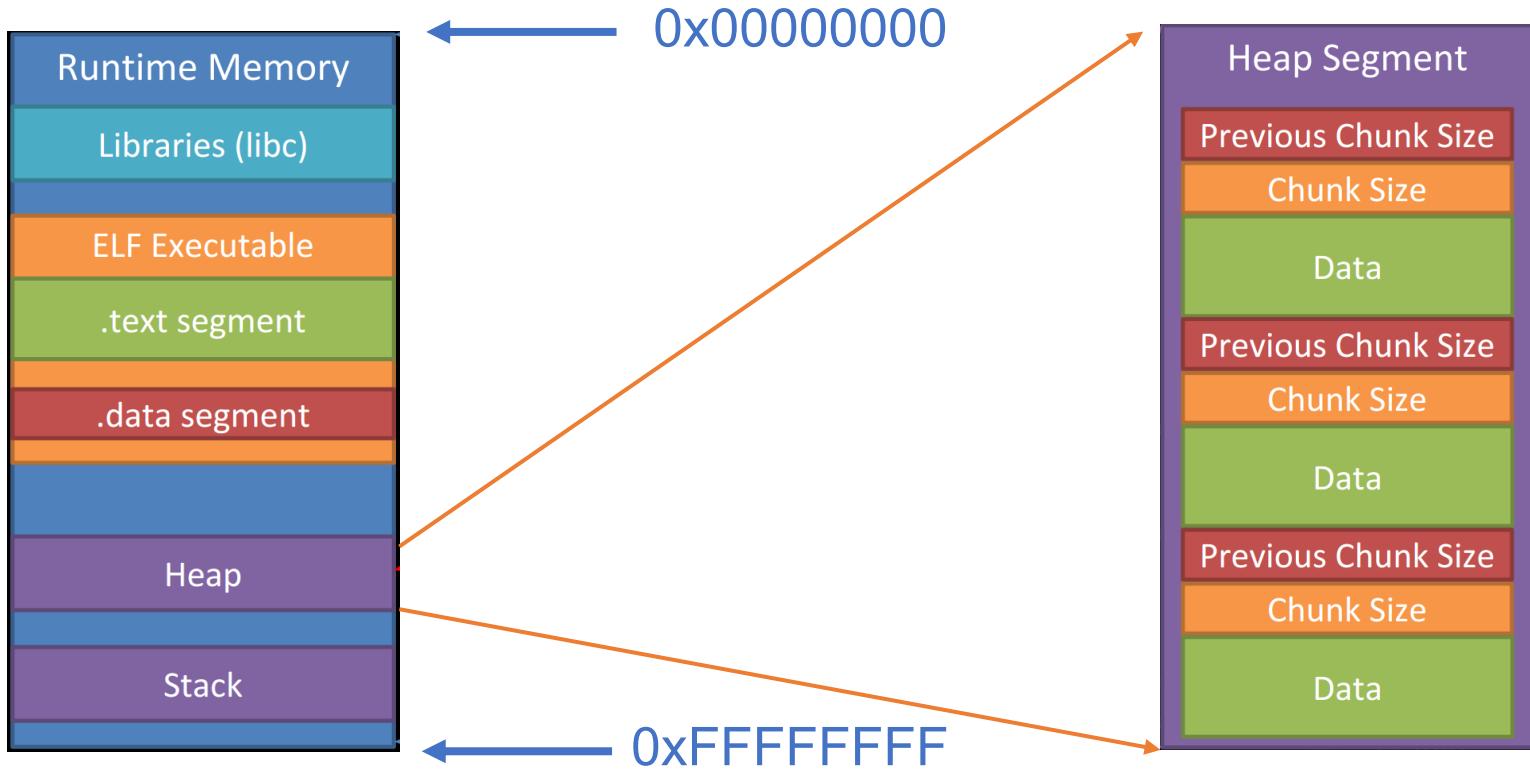


Tràn bộ nhớ Heap: Heap Overflow

- Cơ chế xảy ra lỗi trên **Heap** giống như cách thức hoạt động trên **Stack**
- Heap cookies/canaries không có vai trò.
 - Không có địa chỉ “trả về” (return) để bảo vệ.

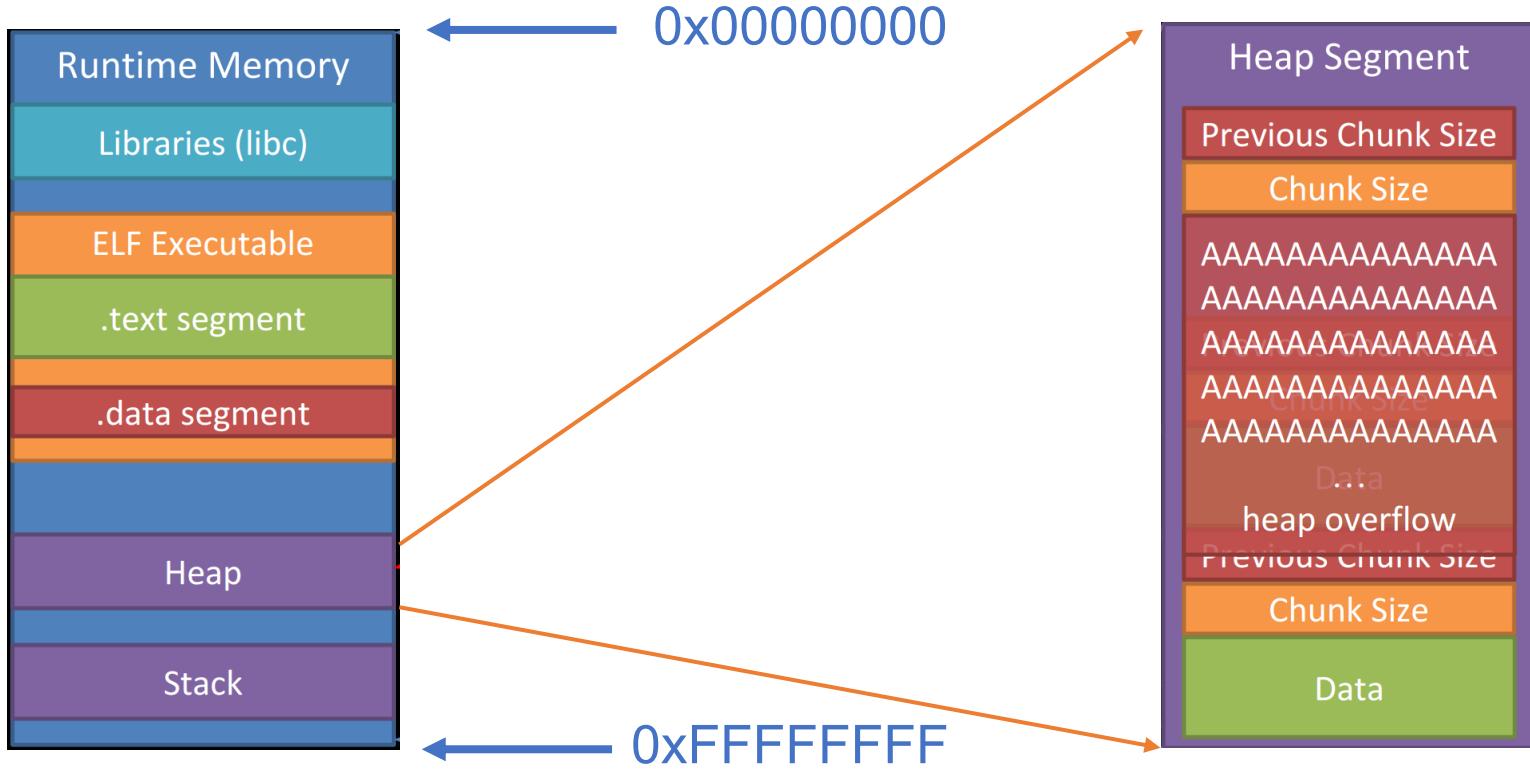


Heap Overflow



Phát triển theo vùng nhớ có địa chỉ cao

Heap Overflow



Phát triển theo vùng nhớ có địa chỉ cao

- Tràn bộ nhớ Heap



Heap Overflow

- Trong thực tế, có hàng loạt các thông tin như object/struct nằm ở trên Heap.
 - **Bất cứ thứ gì xử lý dữ liệu vừa bị hỏng (corrupted) bây giờ đều có thể trở thành bề mặt tấn công (attack surface) trong ứng dụng.**
- Người ta thường đặt các **con trỏ hàm (function pointer)** trong các cấu trúc (**struct**) thường là con trỏ được cấp phát bằng **malloc** trên heap.
 - Ghi đè lên một con trỏ hàm trên heap và buộc một codepath gọi chức năng của đối tượng đó





Heap Overflow

- Cho ví dụ sau:

```
struct toystr {  
    void (* message) (char *);  
    char buffer[20];  
}
```





Heap Overflow

```
coolguy = malloc(sizeof(struct toystr));
lameguy = malloc(sizeof(struct toystr));
coolguy->message = &print_cool;
lameguy->message = &print_meh;
printf("Input coolguy's name: ");
fgets(coolguy->buffer, 200, stdin);
coolguy->buffer[strcspn(coolguy->buffer, "\n")] = 0;
printf("Input lameguy's name: ");
fgets(lameguy->buffer, 20, stdin);
lameguy->buffer[strcspn(lameguy->buffer, "\n")] = 0;
coolguy->message(coolguy->buffer);
lameguy->message(lameguy->buffer);
```



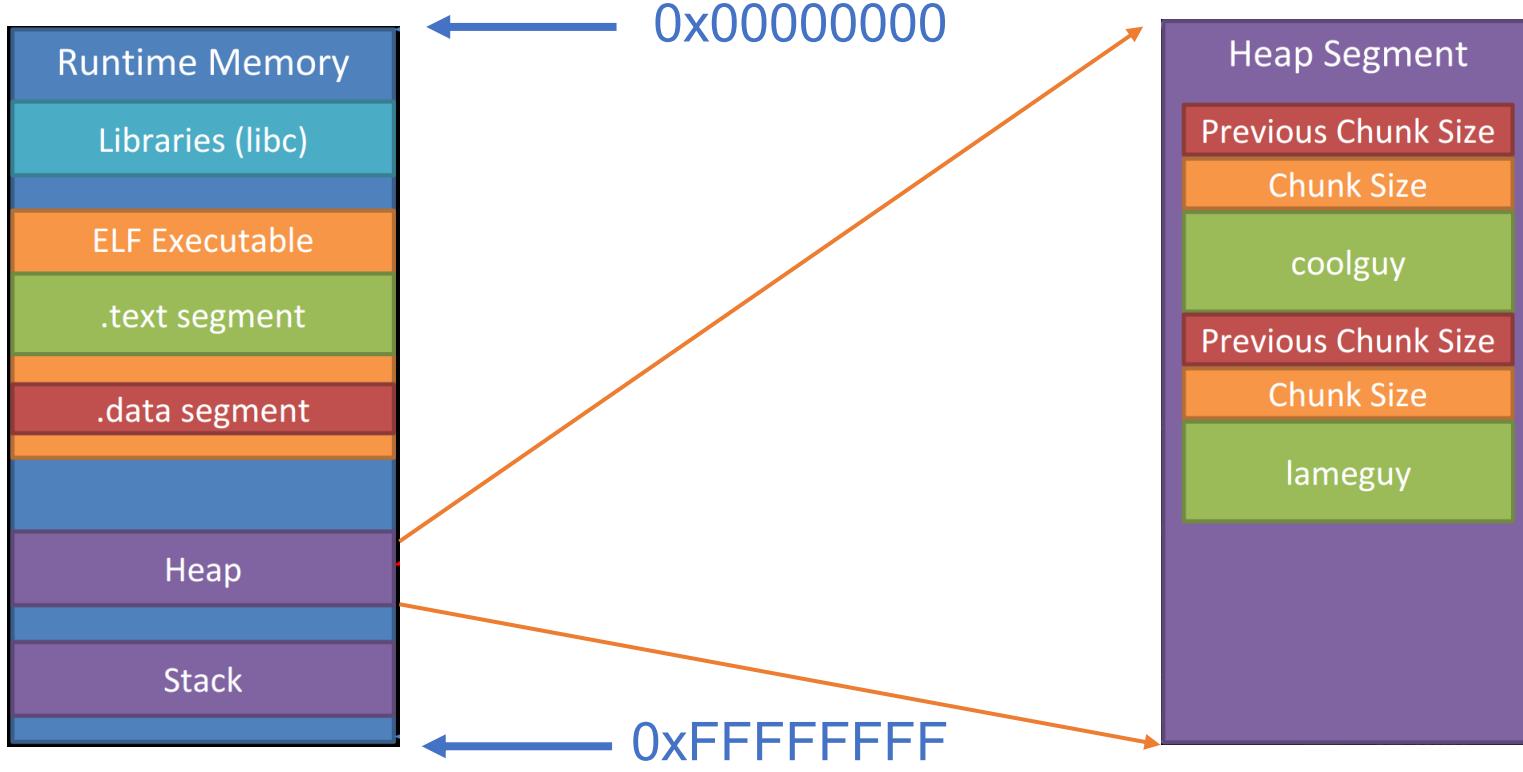


Heap Overflow

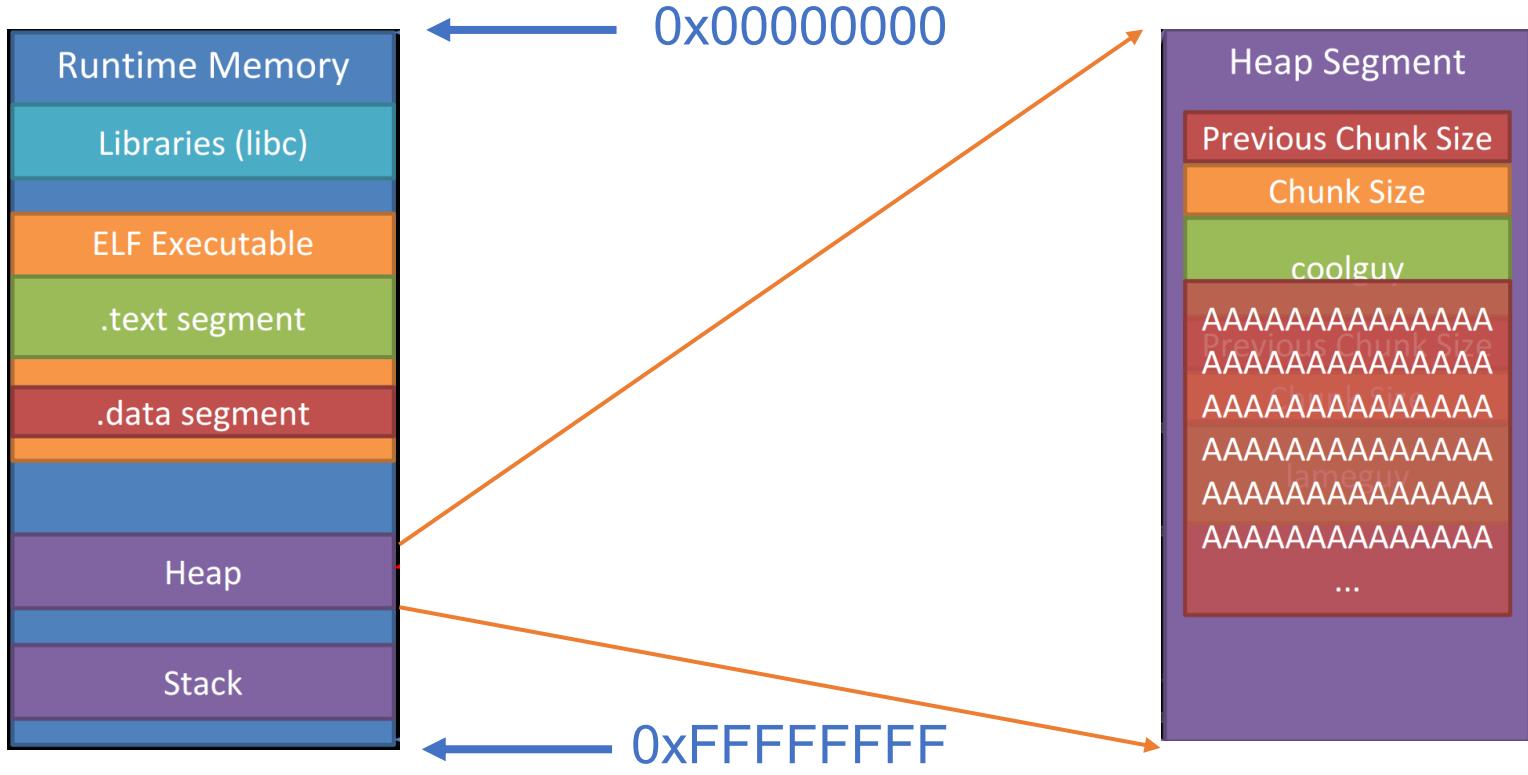
```
coolguy = malloc(sizeof(struct toystr));
lameguy = malloc(sizeof(struct toystr));
coolguy->message = &print_cool;
lameguy->message = &print_meh;
printf("Input coolguy's name: ");      Heap Overflow!!!!!!
fgets(coolguy->buffer, 200, stdin);
coolguy->buffer[strcspn(coolguy->buffer, "\n")] = 0;
printf("Input lameguy's name: ");
fgets(lameguy->buffer, 20, stdin);
lameguy->buffer[strcspn(lameguy->buffer, "\n")] = 0;
coolguy->message(coolguy->buffer);
lameguy->message(lameguy->buffer);
```



Heap Overflow



Heap Overflow



Phát triển theo vùng nhớ có địa chỉ cao



Heap Overflow

```
coolguy = malloc(sizeof(struct toystr));
lameguy = malloc(sizeof(struct toystr));
coolguy->message = &print_cool;
lameguy->message = &print_meh;
printf("Input coolguy's name: ");      Heap Overflow!!!!!!
fgets(coolguy->buffer, 200, stdin);
coolguy->buffer[strcspn(coolguy->buffer, "\n")] = 0;
printf("Input lameguy's name: ");
fgets(lameguy->buffer, 20, stdin);
lameguy->buffer[strcspn(lameguy->buffer, "\n")] = 0;
coolguy->message(coolguy->buffer);
lameguy->message(lameguy->buffer);
```

Overwritten function pointer!!!!



Use-After-Free

- **Use-After-Free (UAF):** là dạng lỗ hổng mà dữ liệu trên heap đã được giải phóng (free-ed), nhưng có một tham chiếu hay “**dangling pointer**” được sử dụng.
- Thường gặp trong Web browser, các chương trình phức tạp





Use-After-Free: Sự phỗ biến

Search Results

There are **3263** CVE entries that match your search.

Name	Description
CVE-2019-9821	A use-after-free vulnerability can occur in AssertWorkerThread due to a race condition with shared workers. This results in a potentially exploitable crash. This vulnerability affects Firefox < 67.
CVE-2019-9820	A use-after-free vulnerability can occur in the chrome event handler when it is freed while still in use. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 60.7, Firefox < 67, and Firefox ESR < 60.7.
CVE-2019-9818	A race condition is present in the crash generation server used to generate data for the crash reporter. This issue can lead to a use-after-free in the main process, resulting in a potentially exploitable crash and a sandbox escape. *Note: this vulnerability only affects Windows. Other operating systems are unaffected.*. This vulnerability affects Thunderbird < 60.7, Firefox < 67, and Firefox ESR < 60.7.
CVE-2019-9796	A use-after-free vulnerability can occur when the SMIL animation controller incorrectly registers with the refresh driver twice when only a single registration is expected. When a registration is later freed with the removal of the animation controller element, the refresh driver incorrectly leaves a dangling pointer to the driver's observer array. This vulnerability affects Thunderbird < 60.6, Firefox ESR < 60.6, and Firefox < 66.
CVE-2019-9790	A use-after-free vulnerability can occur when a raw pointer to a DOM element on a page is obtained using JavaScript and the element is then removed while still in use. This results in a potentially exploitable crash. This vulnerability affects Thunderbird < 60.6, Firefox ESR < 60.6, and Firefox < 66.
CVE-2019-9767	Stack-based buffer overflow in Free MP3 CD Ripper 2.6, when converting a file, allows user-assisted remote attackers to execute arbitrary code via a crafted .wma file.
CVE-2019-9766	Stack-based buffer overflow in Free MP3 CD Ripper 2.6, when converting a file, allows user-assisted remote attackers to execute arbitrary code via a crafted .mp3 file.
CVE-2019-9706	Vixie Cron before the 3.0pl1-133 Debian package allows local users to cause a denial of service (use-after-free and daemon crash) because of a force_rescan_user error.
CVE-2019-9489	A directory traversal vulnerability in Trend Micro Apex One, OfficeScan (versions XG and 11.0), and Worry-Free Business Security (versions 10.0, 9.5 and 9.0) could allow an attacker to modify arbitrary files on the affected product's management console.
CVE-2019-9458	In the Android kernel in the video driver there is a use after free due to a race condition. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation.
CVE-2019-9447	In the Android kernel in the FingerTipS touchscreen driver there is a possible use-after-free due to improper locking. This could lead to a local escalation of privilege with System execution privileges needed. User interaction is not needed for exploitation.
CVE-2019-9442	In the Android kernel in the mnh driver there is possible memory corruption due to a use after free. This could lead to local escalation of privilege with System privileges required. User interaction is not needed for exploitation.
CVE-2019-9431	In Bluetooth, there is a possible out of bounds read due to a use after free. This could lead to remote information disclosure with heap information written to the log with System execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-109755179
CVE-2019-9427	In Bluetooth, there is a possible information disclosure due to a use after free. This could lead to local information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-110166350
CVE-2019-9381	In netd, there is a possible out of bounds read due to a use after free. This could lead to remote information disclosure with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-122677612
CVE-2019-9350	In Keymaster, there is a possible EoP due to a use after free. This could lead to local escalation of privilege with no additional execution privileges needed. User interaction is not needed for exploitation. Product: AndroidVersions: Android-10Android ID: A-129562815



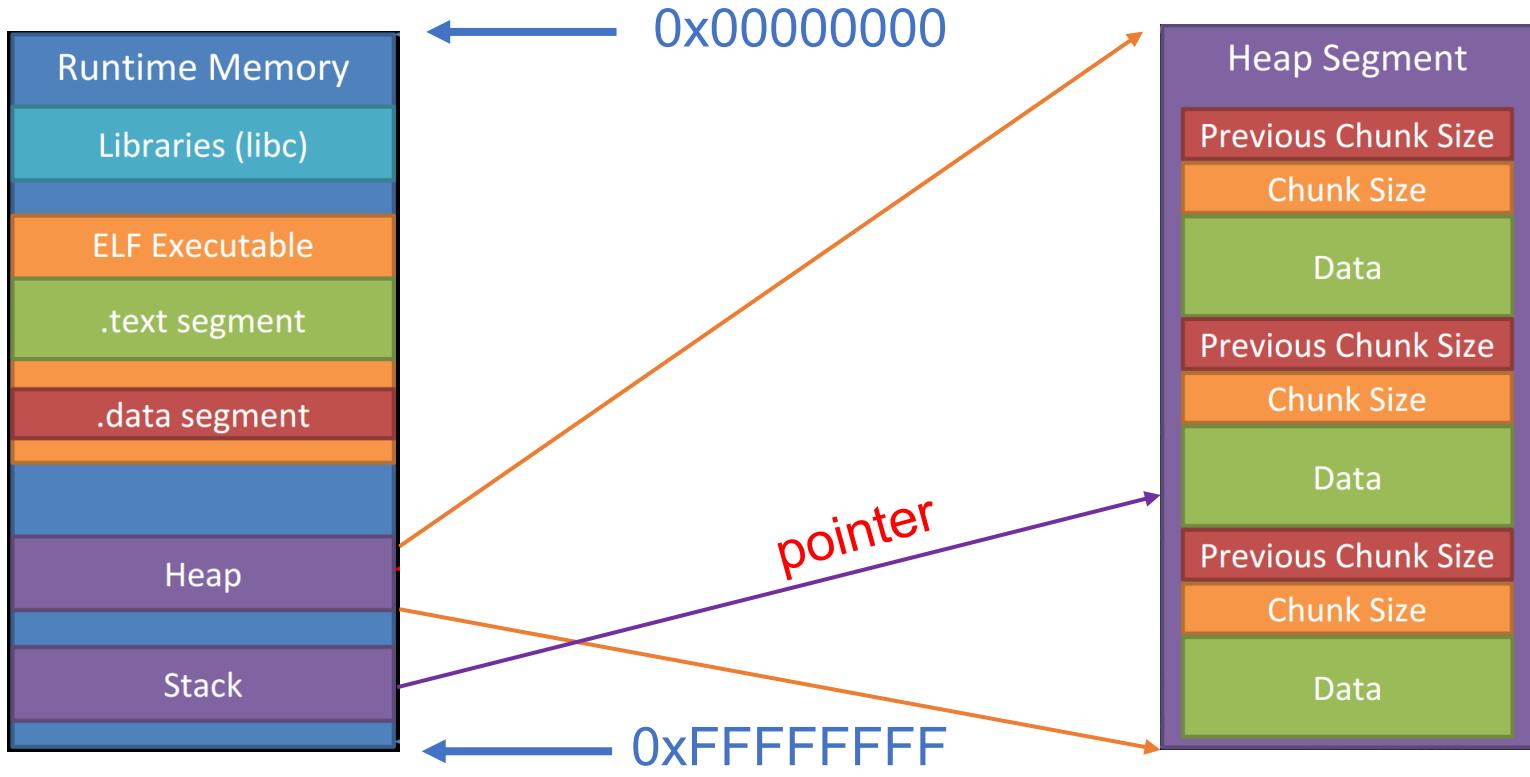
Use-After-Free: Sự phổ biến

Recent Entries (max. 50)

10/08/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	Foxit Reader/PhantomPDF up to 9.2 Javascript Engine PDF Document Use-After-Free memory corruption
10/08/2018	M	pyOpenSSL up to 17.4.x X.509 Object Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.2.0.9297 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/03/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine PDF Document Use-After-Free memory corruption
10/02/2018	M	Google Android file.c sdcardfs_open memory corruption
10/02/2018	M	Google Android 8.0/8.1 Bluetooth Service avrc_pars_tg.cc avrc_pars_browsing_cmd memory corruption
10/02/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine Use-After-Free memory corruption
10/02/2018	M	Foxit PDF Reader 9.1.0.5096 Javascript Engine Use-After-Free memory corruption

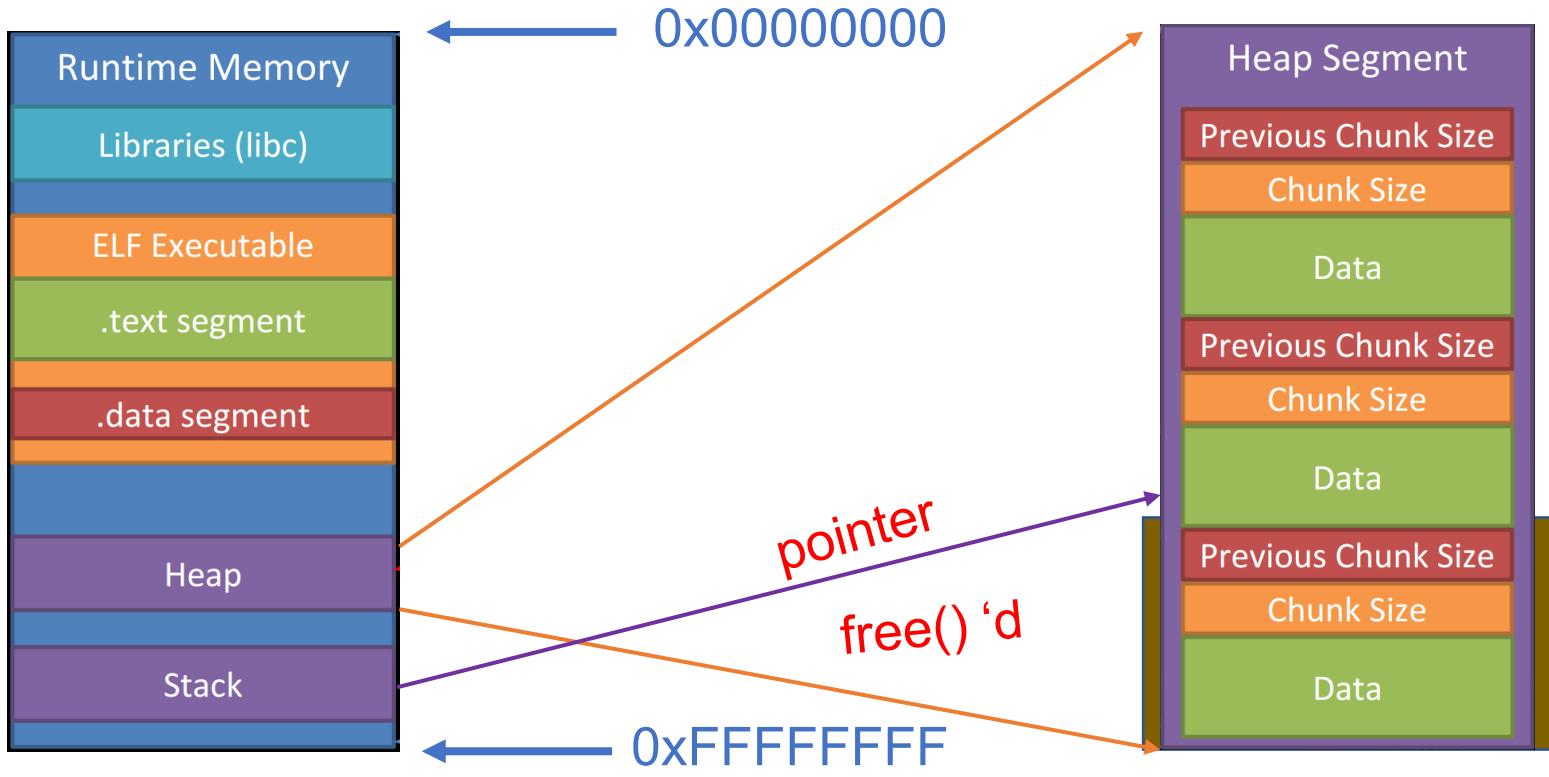


Use-After-Free



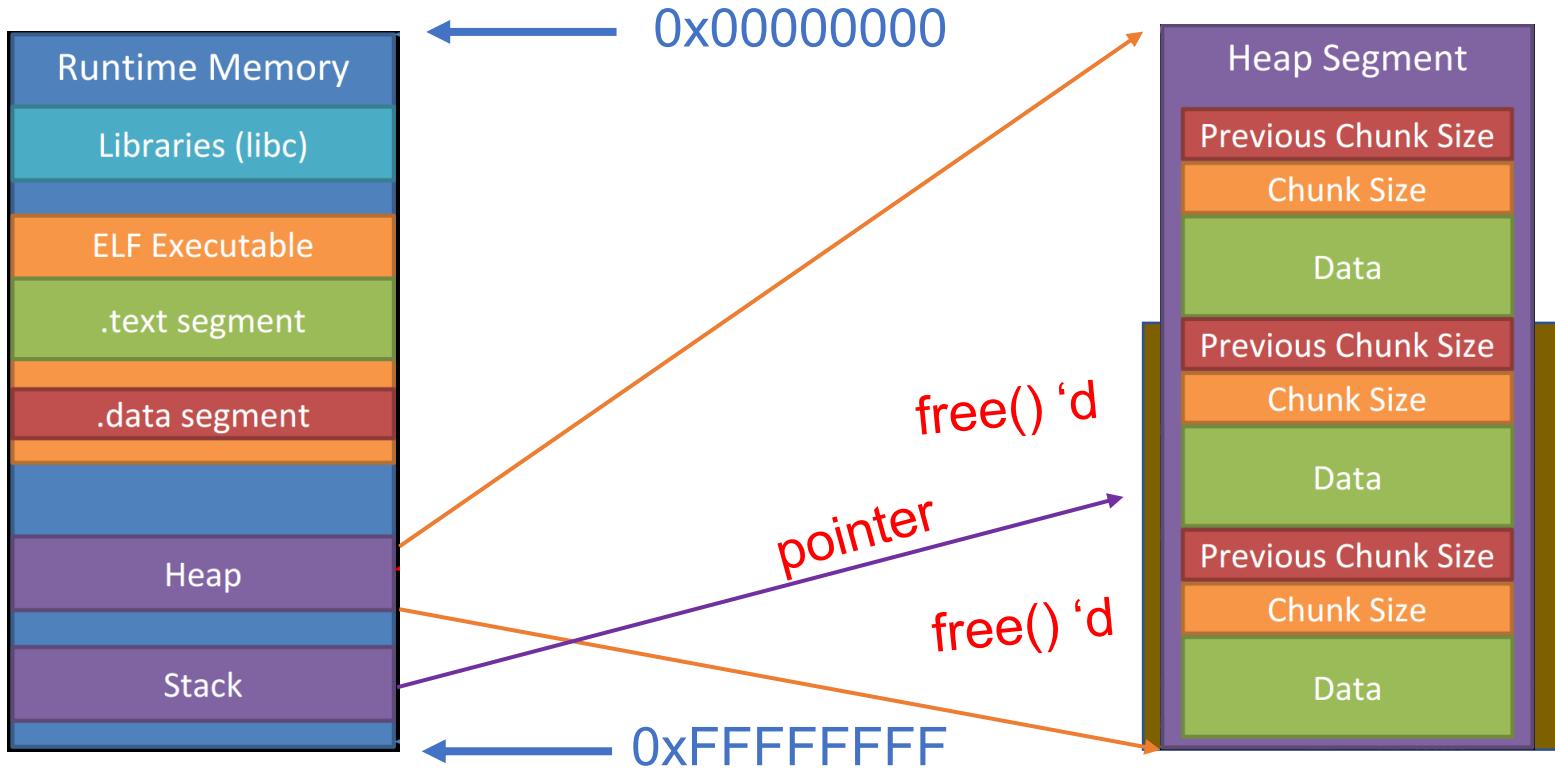
Phát triển theo vùng nhớ có địa chỉ cao

Use-After-Free



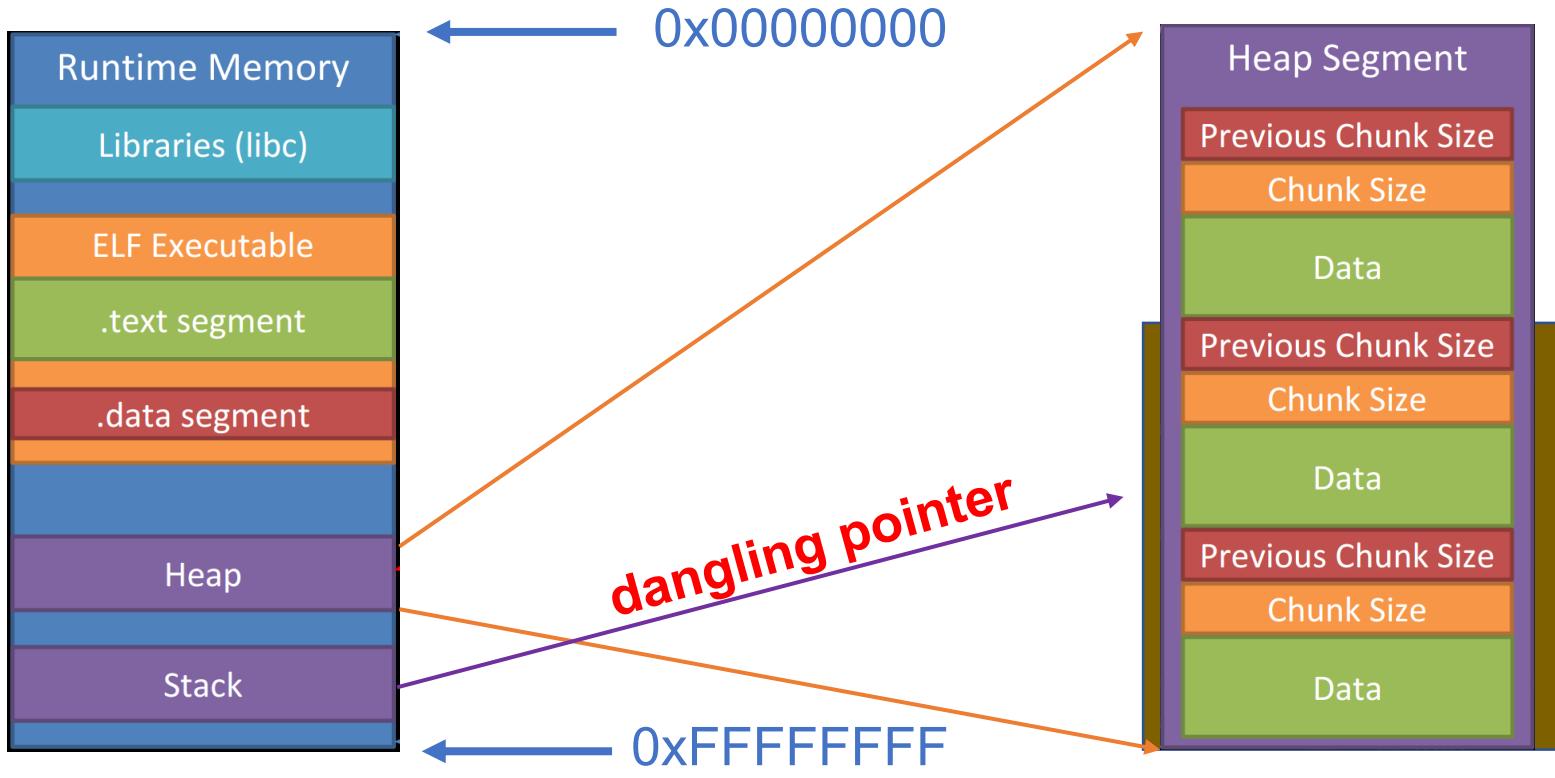
Phát triển theo vùng nhớ có địa chỉ cao

Use-After-Free



Phát triển theo vùng nhớ có địa chỉ cao

Use-After-Free



Phát triển theo vùng nhớ có địa chỉ cao



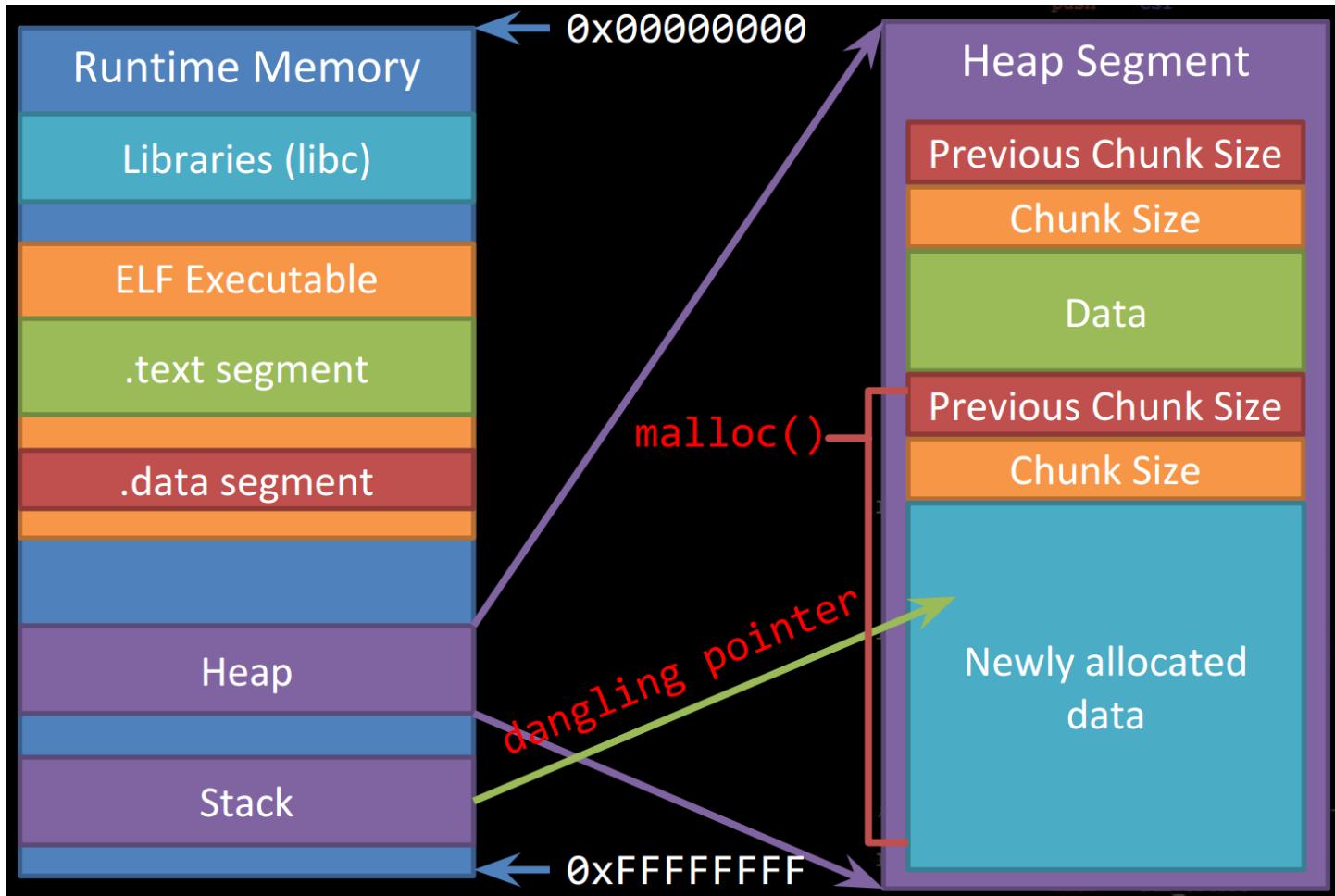
Use-After-Free: Dangling Pointer

- **Dangling Pointer:**

- Là một con trỏ còn sót lại trong mã nguồn có tham chiếu tới dữ liệu đã được giải phóng trên Heap; dễ có khả năng được sử dụng lại.
- Khi được chỉ định tới địa chỉ tại vùng nhớ đã được giải phóng, không có sự đảm bảo nào về dữ liệu tại đây
- Còn được gọi với tên khác: **stale pointer, wild pointer.**

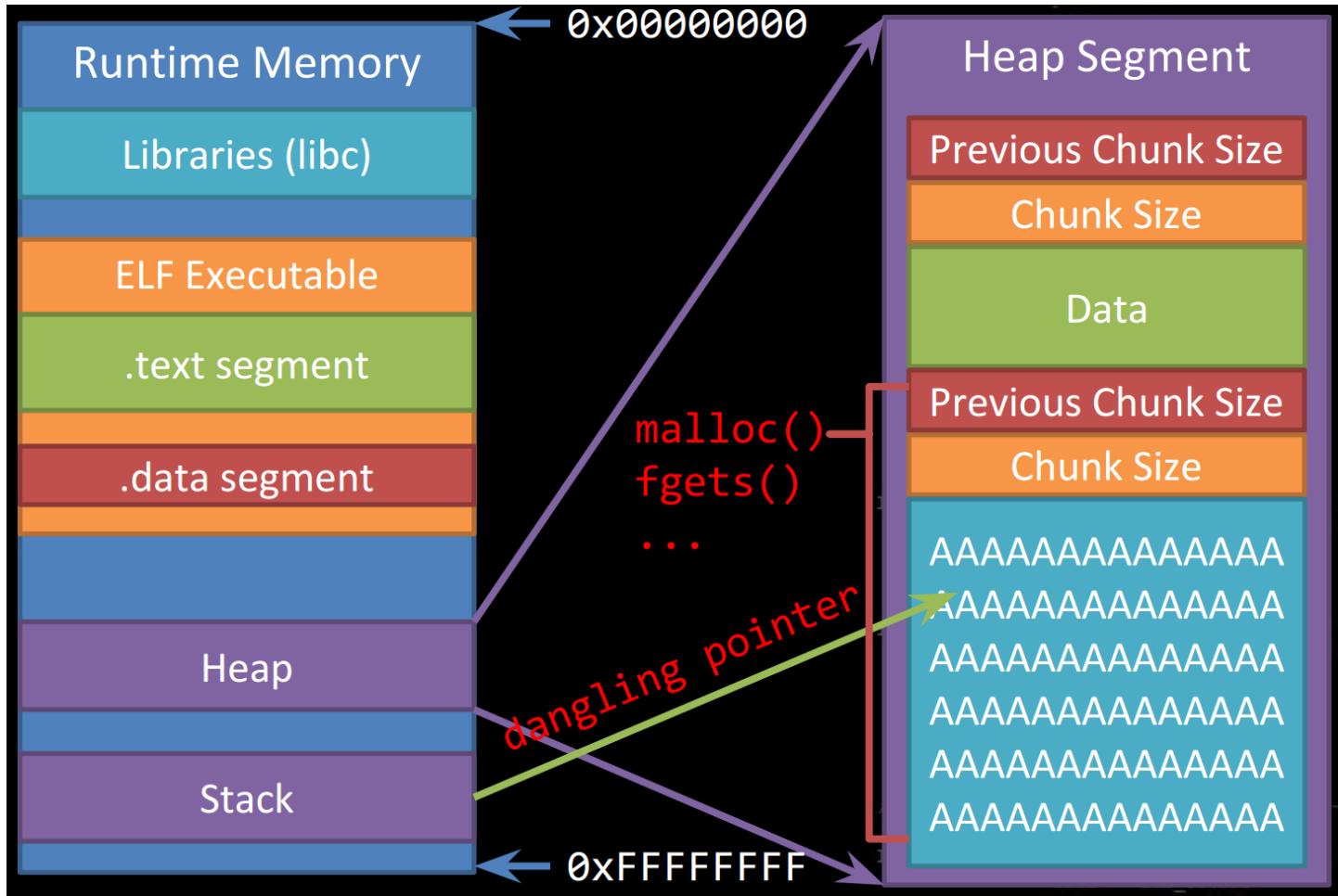


Use-After-Free



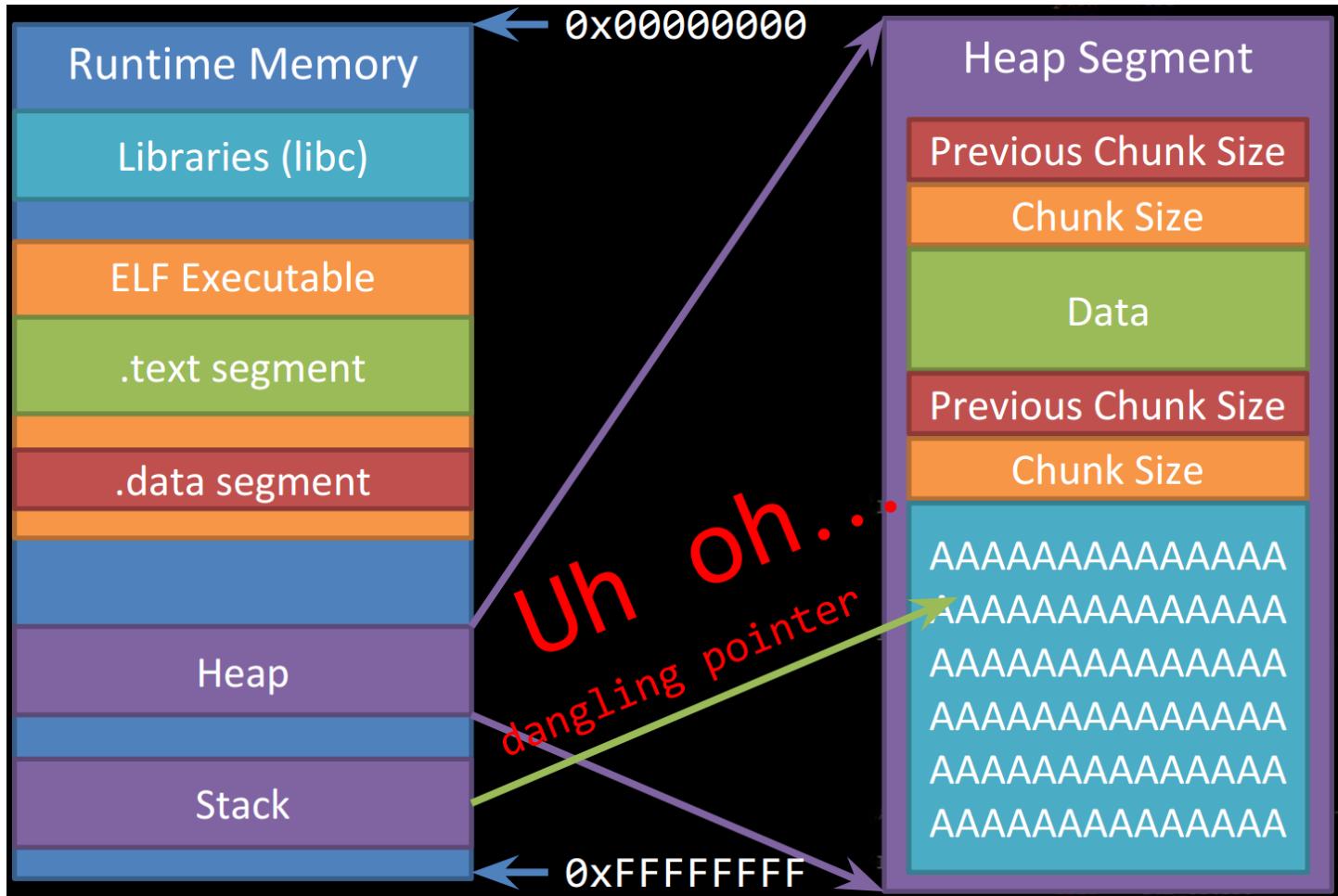
Phát triển theo vùng nhớ có địa chỉ cao

Use-After-Free



Phát triển theo vùng nhớ có địa chỉ cao

Use-After-Free



Phát triển theo vùng nhớ có địa chỉ cao

Khai thác lỗ hổng Use-After-Free

- Để khai thác **UAF**, người ta thường phải cấp phát một kiểu dữ liệu khác của đối tượng lên trên vùng nhớ đã được giải phóng.

```
struct toystr { ←  
    void (* message) (char *);  
    char buffer[20];  
}
```

1. **free()** , giả sử có 1 **dangling pointer** tồn tại

```
struct person { ←  
    int favorite_num;  
    int age;  
    char name[16];  
}
```

2. **malloc()** kiểu đối tượng mới

3. Gán **favorite_num = 0x41414141**

4. Điều khiển **dangling pointer** gọi (call) tới hàm **message()**



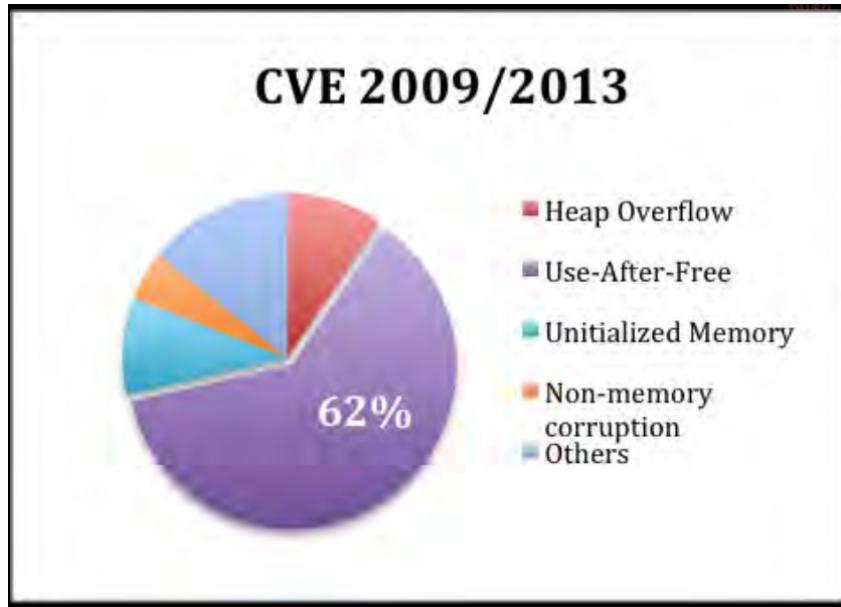
Use-After-Free

- Không cần bất kỳ hình thức “**phá hỏng bộ nhớ**” (*memory corruption*) nào để khai thác lỗ hỏng use-after-free (UAF).
- Nó chỉ đơn giản là một vấn đề hiện thực (implementation issue) trong quản lý sai con trỏ (*pointer mismanagement*).



Khai thác lỗ hổng UAF trong thực tế

- Hầu như tất cả các khai thác trình duyệt (browser) hiện đại đều dựa trên lỗ hổng **UAF**.



Tại sao UAF lại là khai thác được ưa thích?

- Không yêu cầu bất cứ hình thức memory corruption nào để sử dụng
- Có thể được dùng cho mục đích rò rỉ thông tin
- Có thể được dùng để kích hoạt memory corruption hay lấy kiểm soát EIP

Phát hiện lỗ hổng UAF

- **Việc phát hiện UAF** trong các chương trình phức tạp là **một tác vụ khó**, ngay trong môi trường công nghiệp (industry).
- Lí do:
 - UAF chỉ tồn tại trên một số trạng thái thực thi của chương trình, do đó việc quét mã nguồn tinh để tìm kiếm sẽ khó cho ra kết quả
 - Nó thường hay được tìm thấy thông qua “crash”, tuy nhiên, việc sử dụng **Symbolic Execution** và **Constrainst Solver** có thể giúp tìm ra lỗi UAF nhanh hơn.

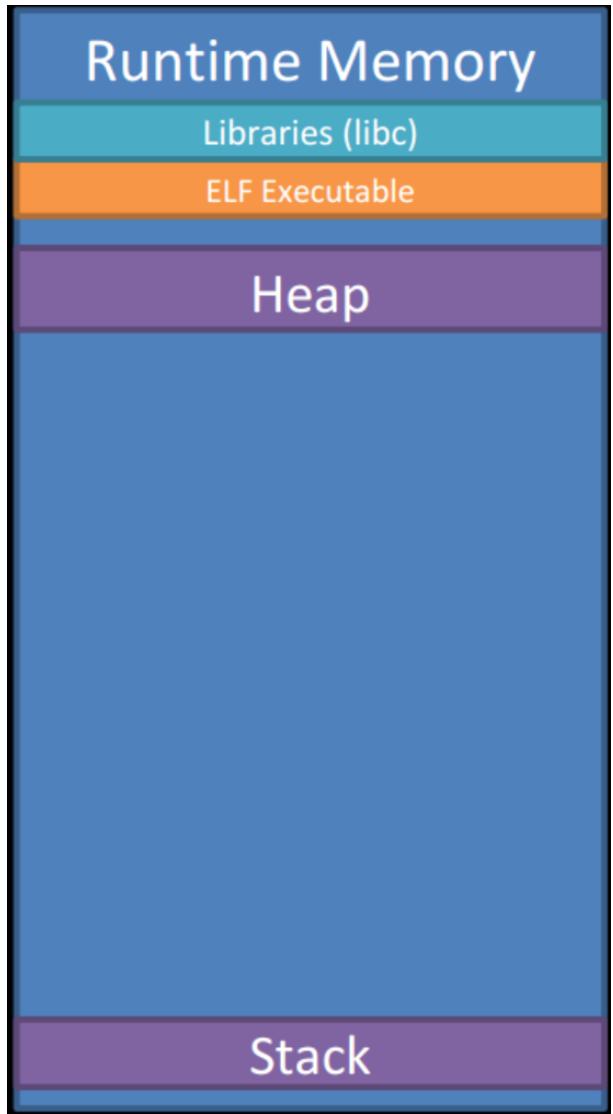


Heap Spraying

- **Heap Spraying** không phải là một cách để khai thác lỗ hổng bảo mật.
 - Đây là một kỹ thuật được sử dụng để cấp phát payload được sử dụng như một phần của một khai thác lỗ hổng.
 - Nó **không phải** là *lỗ hổng (vulnerability) hay khiếm khuyết (security flaw)*
- **Heap Spraying** là một kỹ thuật làm tăng khả năng khai thác, bằng cách lấp đầy heap với một lượng data chunk lớn tương ứng mã khai thác mà kẻ tấn công dự định đặt xuống bộ nhớ.
 - Hỗ trợ vượt qua được cơ chế bảo vệ của **ASLR**
- **Heap Spraying** tận dụng lợi thế là cho phép đặt shellcode ở một vị trí bất kỳ (có thể dự đoán được) trong bộ nhớ và sau đó “jump” về lại địa chỉ đó để thực thi đoạn shellcode.



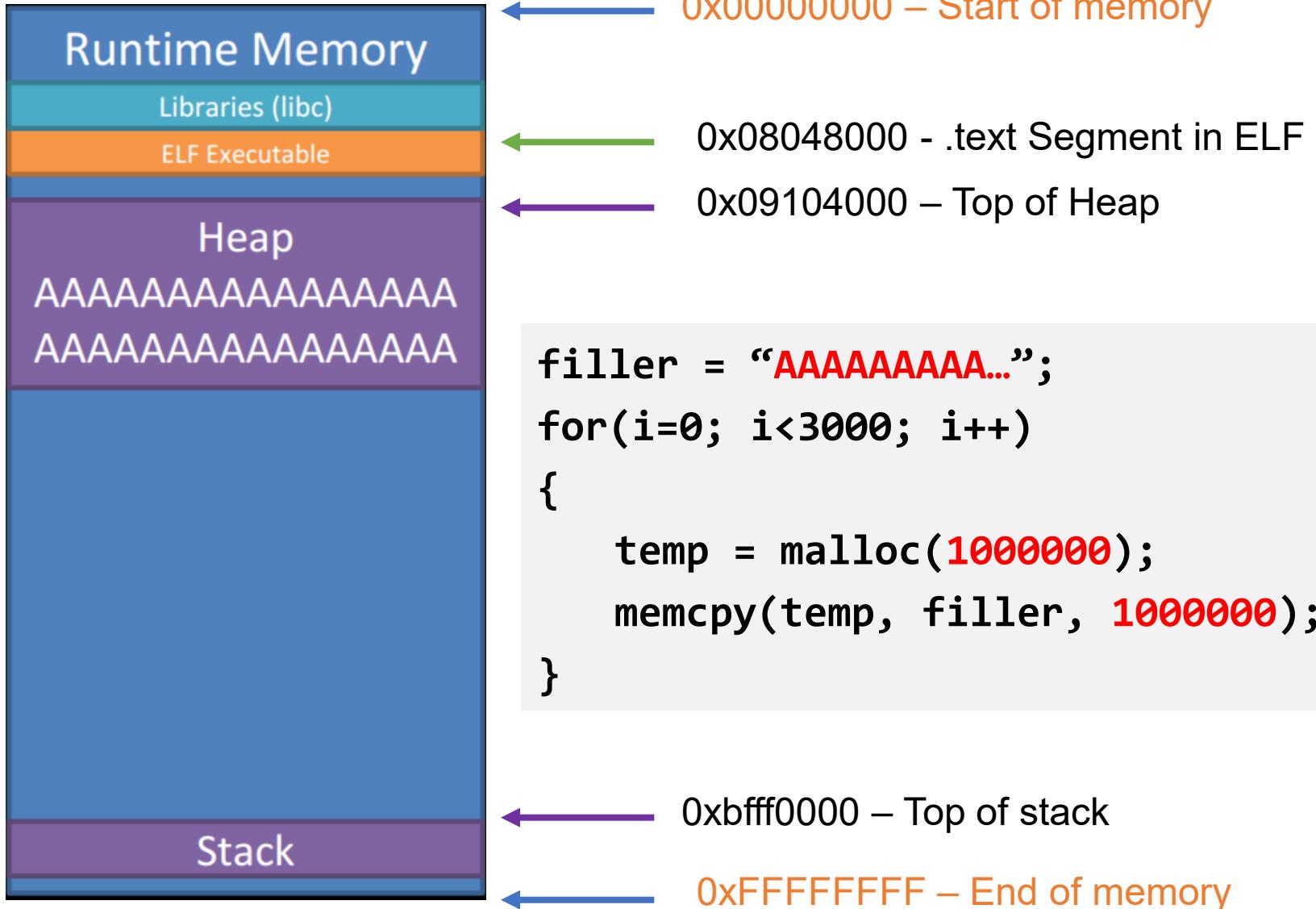
Heap Spraying



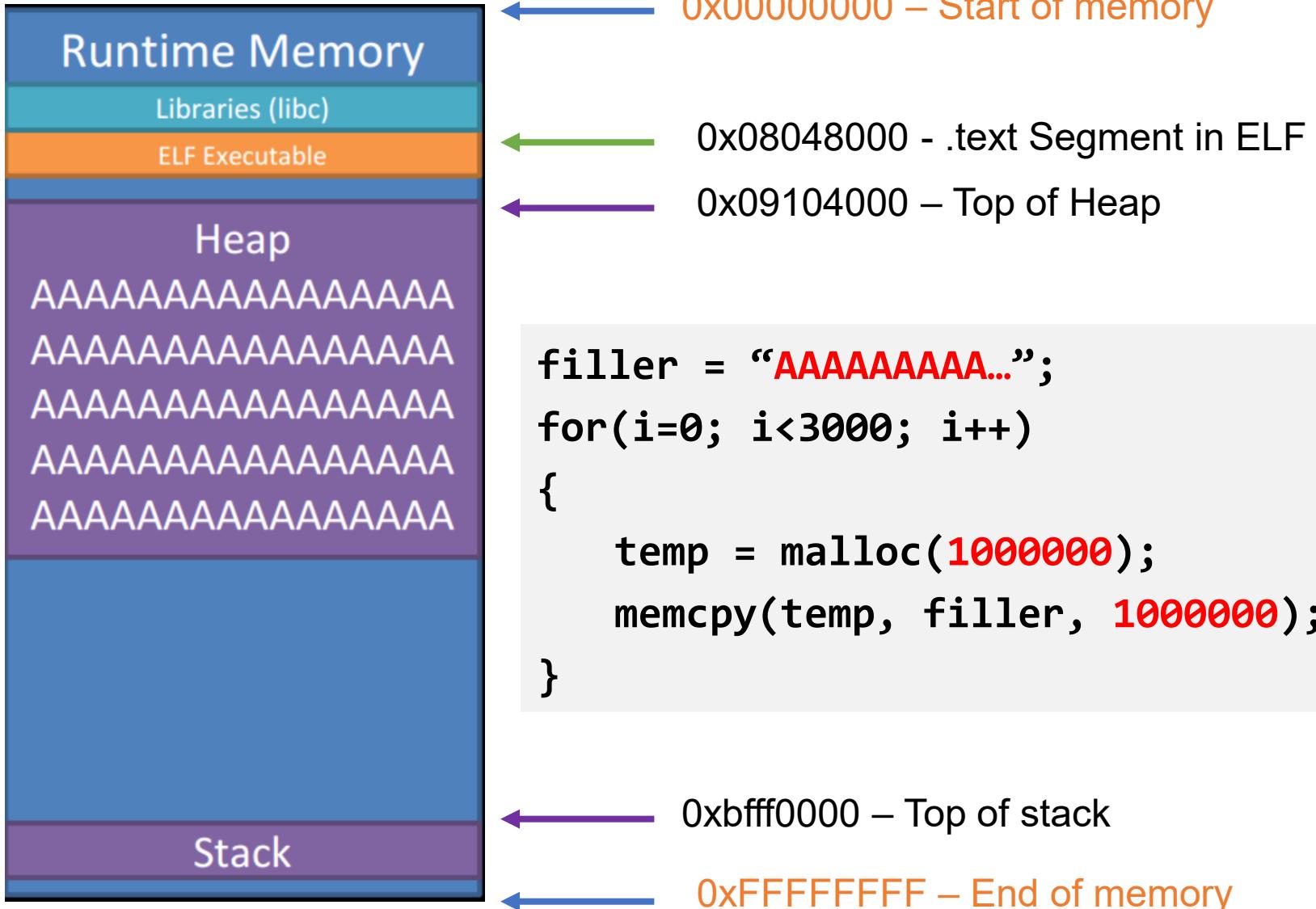
```

filler = “AAAAAAA...”;
for(i=0; i<3000; i++)
{
    temp = malloc(1000000);
    memcpy(temp, filler, 1000000);
}
  
```

Heap Spraying

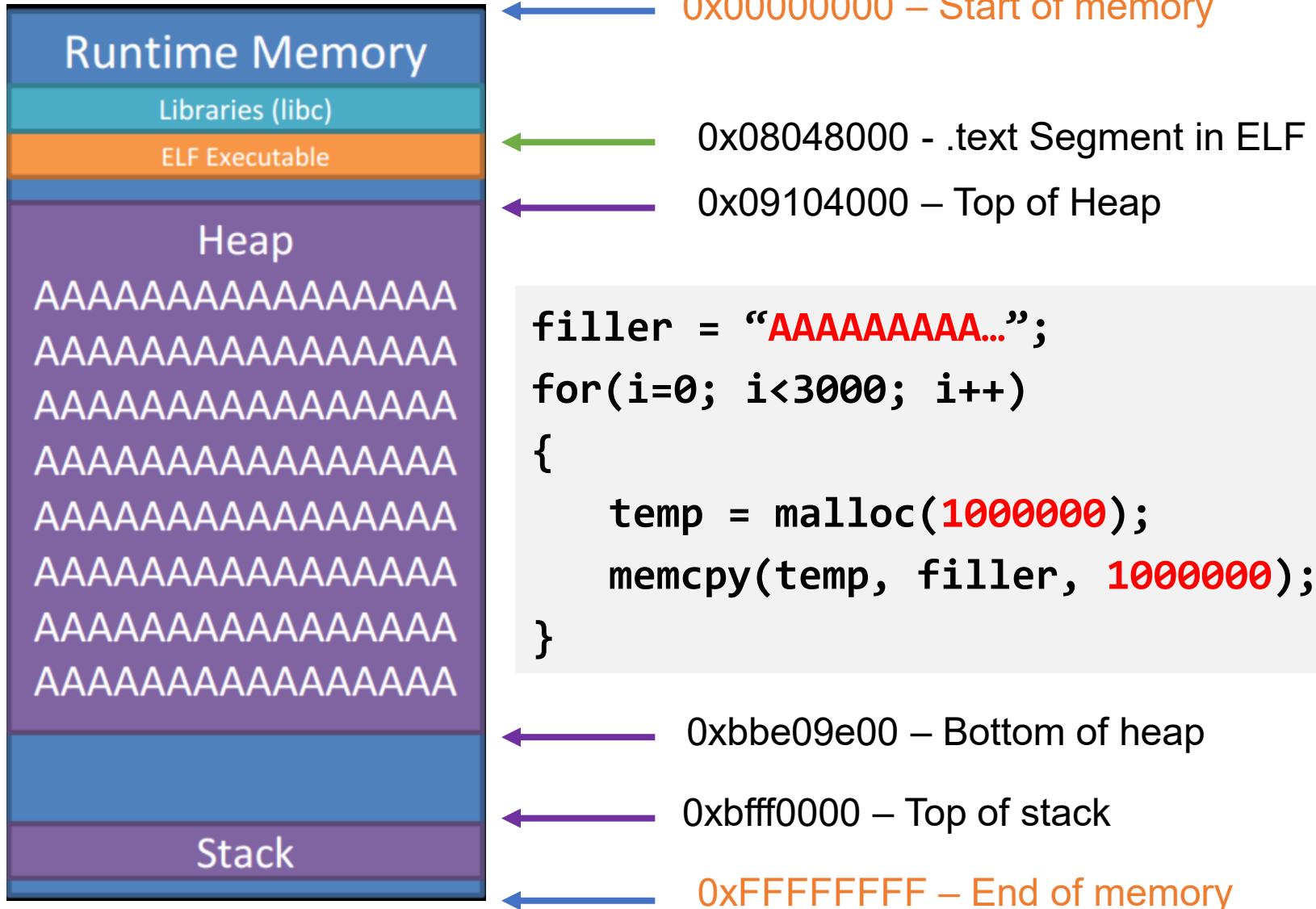


Heap Spraying





Heap Spraying



Heap Spraying

3 GB of AAAA...AA



0x00000000 – Start of memory

0x08048000 - .text Segment in ELF

0x09104000 – Top of Heap

```

filler = “AAAAAAA...”;
for(i=0; i<3000; i++)
{
    temp = malloc(1000000);
    memcpy(temp, filler, 1000000);
}

```

0xbbe09e00 – Bottom of heap

0xbfff0000 – Top of stack

0xFFFFFFF – End of memory



Heap Spraying: trong thực tế

- Thường được tìm thấy trong các khai thác trình duyệt (browser), hiếm khi có trong các thử thách CTF hay wargame – **tuy nhiên, cần chú ý.**
- Kỹ thuật heap spraying thường được dùng trong các trường hợp như đặt mã javascript trên một trang html độc hại.

```
memory = new Array();
for(i = 0; i < 0x100; i++){
    memory[i] = ROPNOP + ROP
}
```





Heap spraying trên kiến trúc 32bit

- Các hệ thống 32bit, không gian địa chỉ tối đa là 4 GB (2^{32} bytes)
- Thực hiện lấp đầy 3 GB ký tự A trên bộ nhớ Heap:
 - Có thể giúp làm **tăng 75%** cơ hội cho địa chỉ **0x23456789** trở thành **con trỏ hợp lệ**.





Heap spraying trên kiến trúc 64 bit

- Trên hệ thống 64-bit, heap spraying không thể dùng để vượt qua được cơ chế bảo vệ của ASLR.
 - Mang tính chất may mắn, cần “phun” khoảng 2^{64} bytes, tương đương ~18446744 terabyte.
- Kỹ thuật spray có chủ đích (targeted spray) vẫn có thể hữu dụng trong các ngữ cảnh chúng ta có một con trỏ ptr được ghi đè (overwrite).





Metadata Corruption

- Khai thác Metadata Corruption liên quan tới việc phá vỡ thông tin metadata của heap, như các chúng ta sử dụng các hàm nội tại của trình cấp phát bộ nhớ để ghi đè thông tin muốn thực hiện.
- Thường liên quan tới các chunk giả mạo (fake chunk), và tận dụng nó trong qui trình **coalescing** và **unlinking**.
- Khai thác Heap metadata với qui trình Heap unlink() khi giải phóng 01 chunk:
 - Lỗ hổng cũ, đã được vá trên glibc.

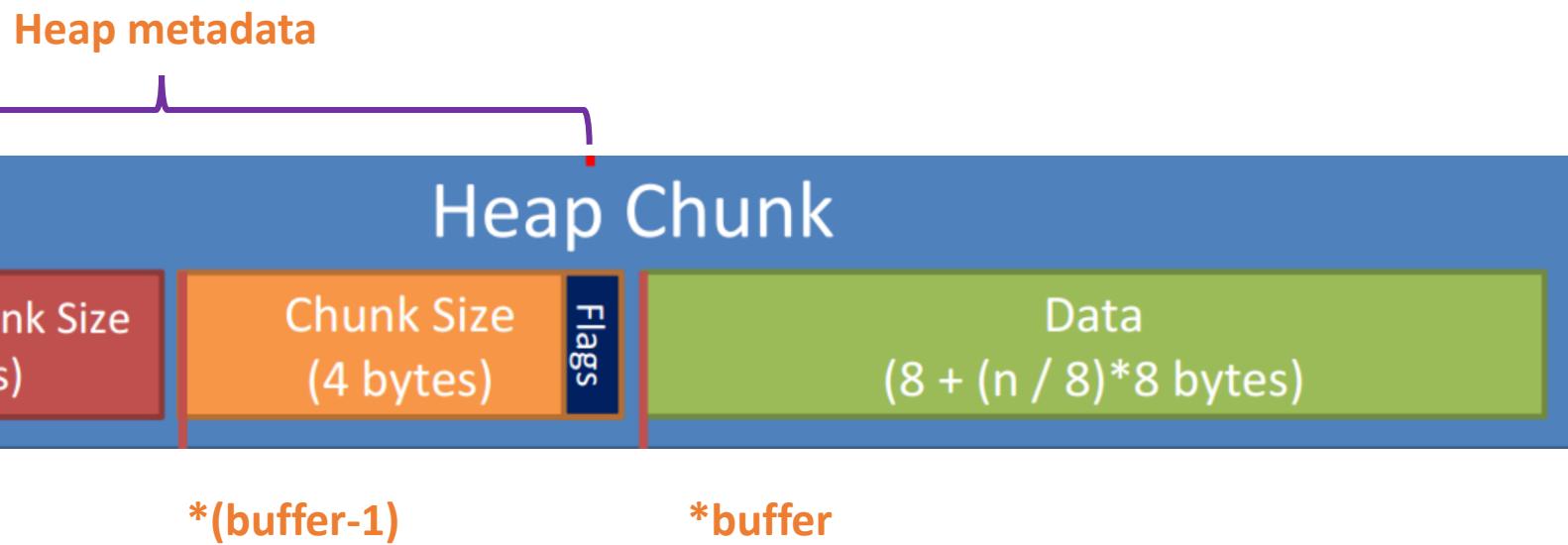
Tham khảo:

<https://sploitfun.wordpress.com/2015/02/26/heap-overflow-using-unlink/>



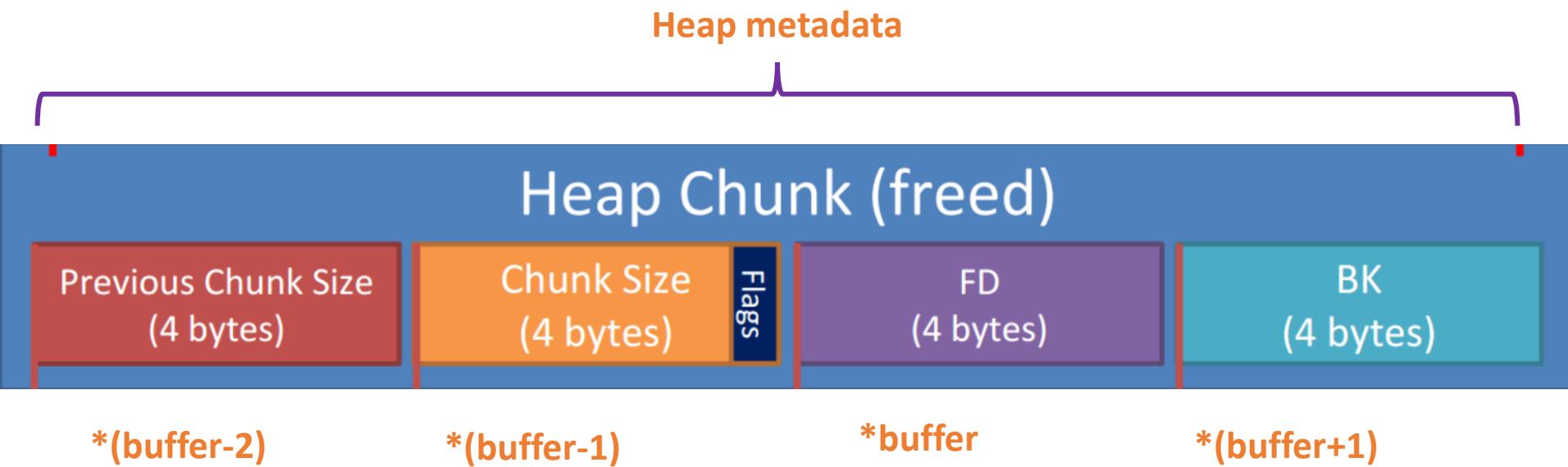
Metadata Corruption

- Heap Chunks – In Use:



Metadata Corruption

- Heap Chunks – Freed (vùng trống):





Metadata Corruption

- Các khai thác dựa trên phá vỡ metadata của heap thường rất liên quan và yêu cầu **kiến thức sâu hơn về cấu trúc nội bộ của heap.**
- Một số liên kết tham khảo:
 - <https://kitctf.de/writeups/0ctf2015/freenote/>
 - <https://sploitfun.wordpress.com/2015/03/04/heap-overflowusing-malloc-maleficarum/>
 - <http://acez.re/ctf-writeup-hitcon-ctf-2014-stkof-or-modernheap-overflow/>
 - <http://wapiflapi.github.io/2014/11/17/hacklu-oreo-withret2dl-resolve/>
 - <http://phrack.org/issues/66/10.html>
 - <http://dl.packetstormsecurity.net/papers/attack/MallocMaleficarum.txt>





Assignment 5

- Study hard, and go ahead,...



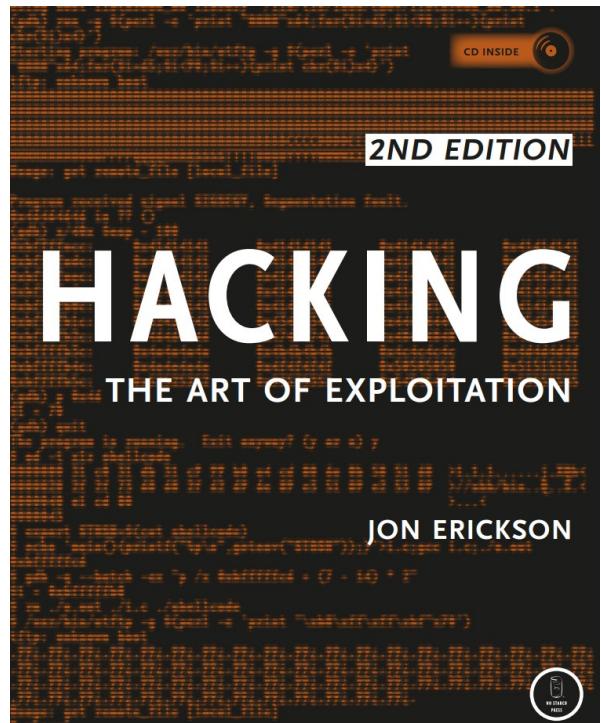
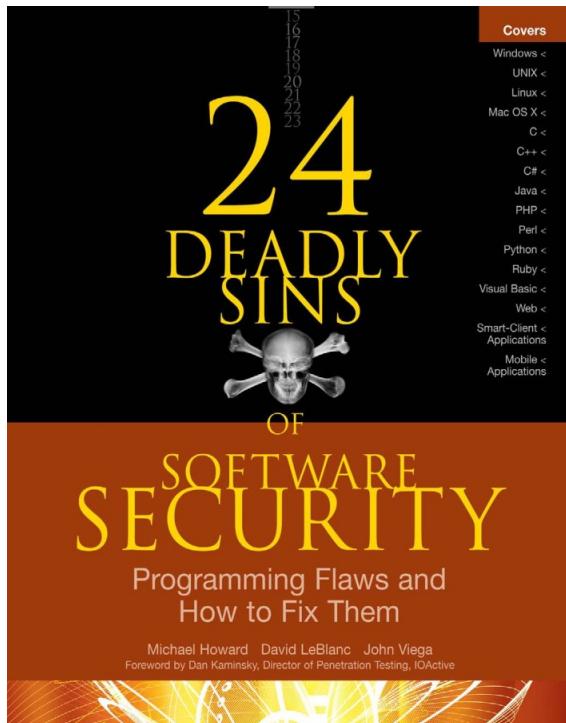
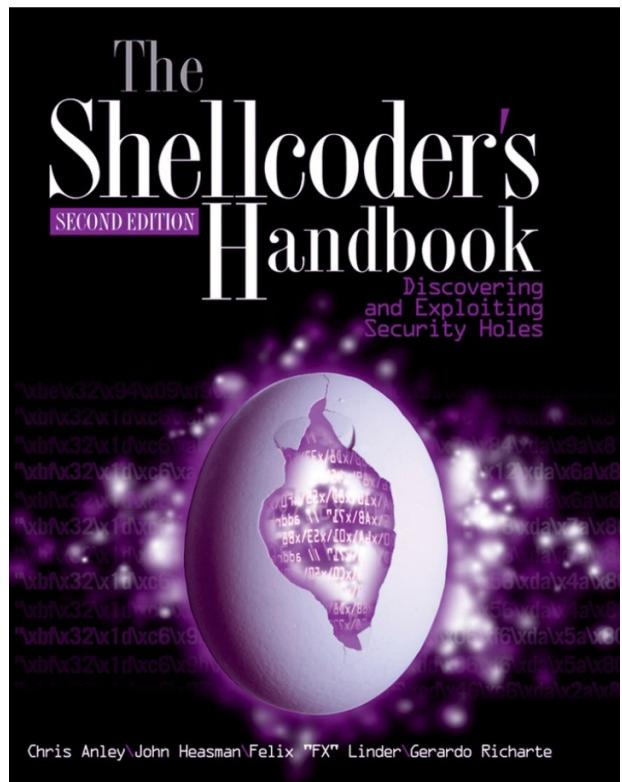


Tài liệu tham khảo: Pwn CTF

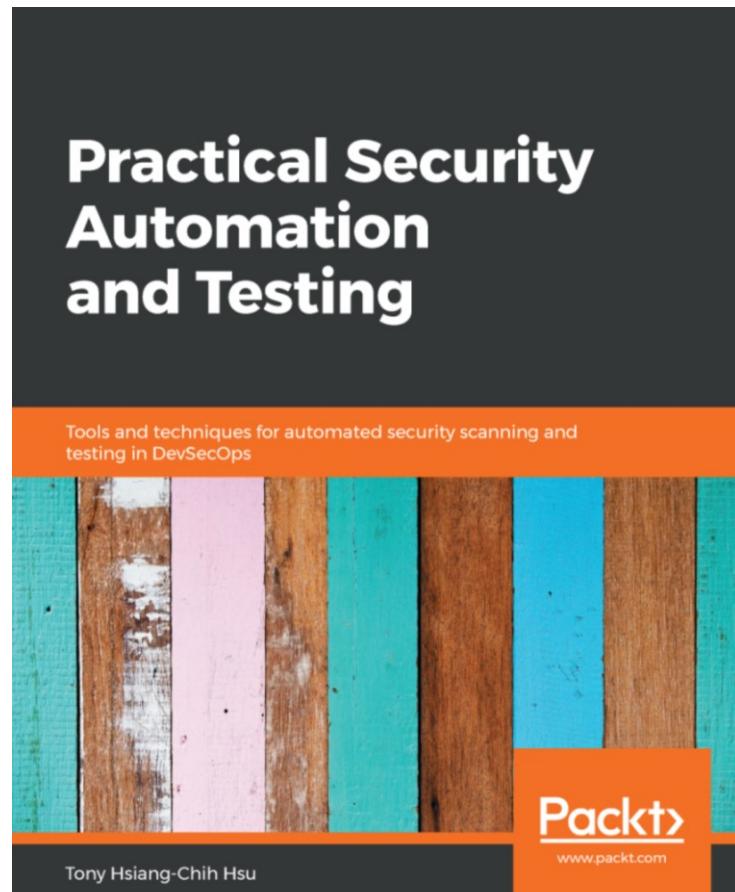
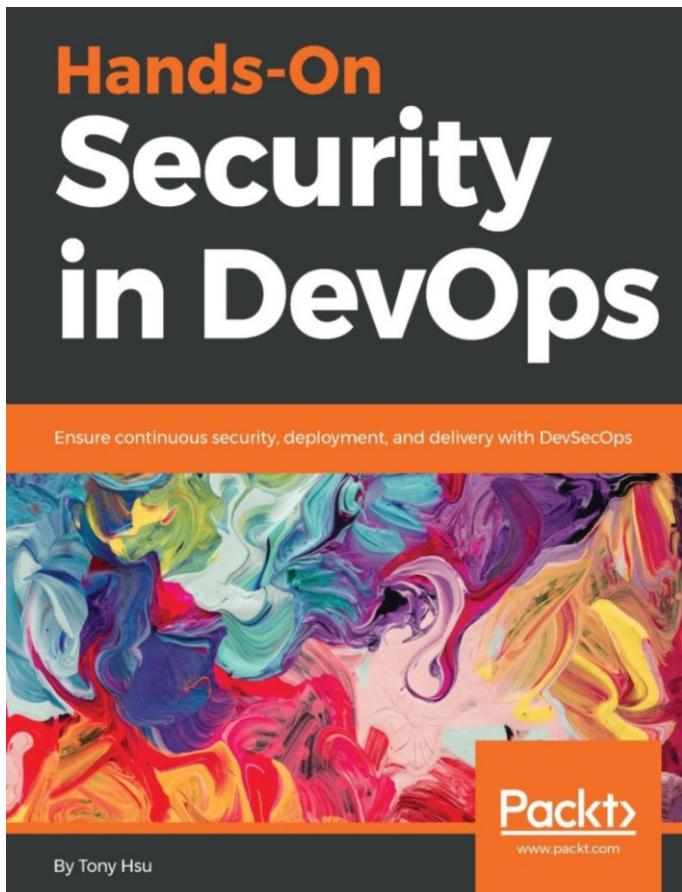
- CTF Wiki: <https://ctf-wiki.org/pwn/linux/user-mode/environment/>
- Modern Binary Exploitation - CSCI 4968: <https://github.com/RPISEC/MBE>
- NTU Computer Security Fall 2019: <https://github.com/yuawn/NTU-Computer-Security>
- Nightmare: <https://guyinatuxedo.github.io/index.html>
- CS6265: <https://tc.gts3.org/cs6265/tut/tut00-intro.html>
- **Educational Heap Exploitation:** <https://github.com/shellphish/how2heap>
- **Heap Exploitation:** <https://techlifecompilation.wordpress.com/2018/10/04/malloc1-heap-exploitation-101/>
- **GLibC Heap Exploitation:** <https://0x434b.dev/overview-of-glibc-heap-exploitation-techniques/>



Tài liệu tham khảo



Tài liệu tham khảo



Tài liệu tham khảo

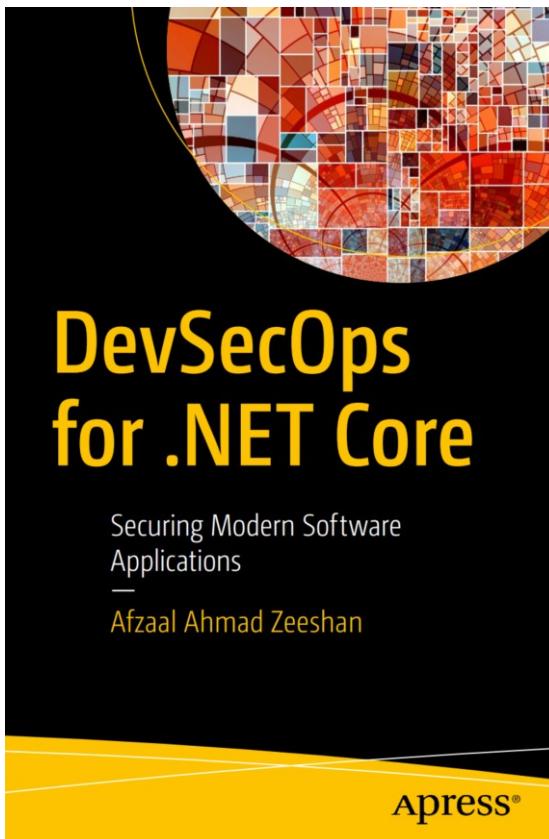
O'REILLY®



Agile Application Security

ENABLING SECURITY IN A CONTINUOUS DELIVERY PIPELINE

Laura Bell, Michael Brunton-Spall,
Rich Smith & Jim Bird



O'REILLY®

DevOpsSec

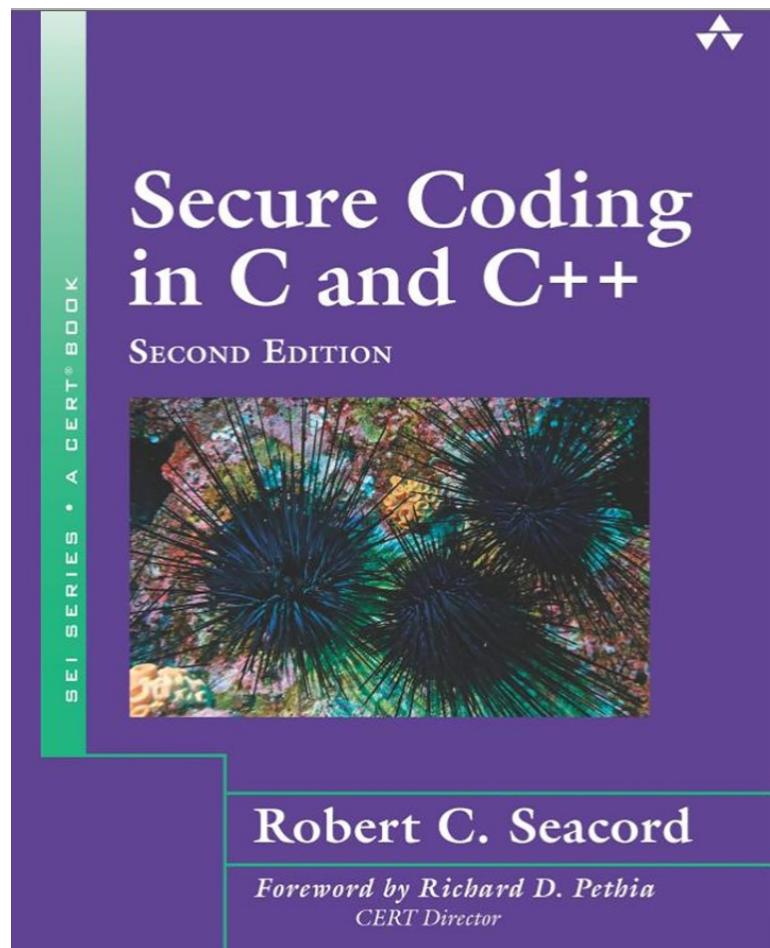
Delivering Secure Software
Through Continuous Delivery



Tài liệu tham khảo



MARK DOWD
JOHN McDONALD

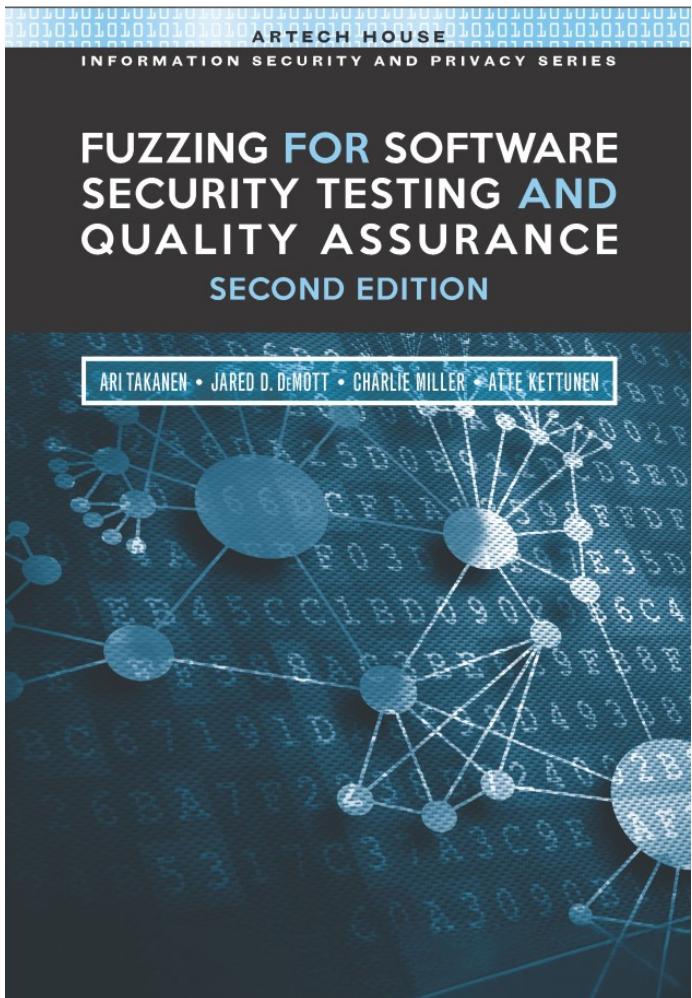


Robert C. Seacord

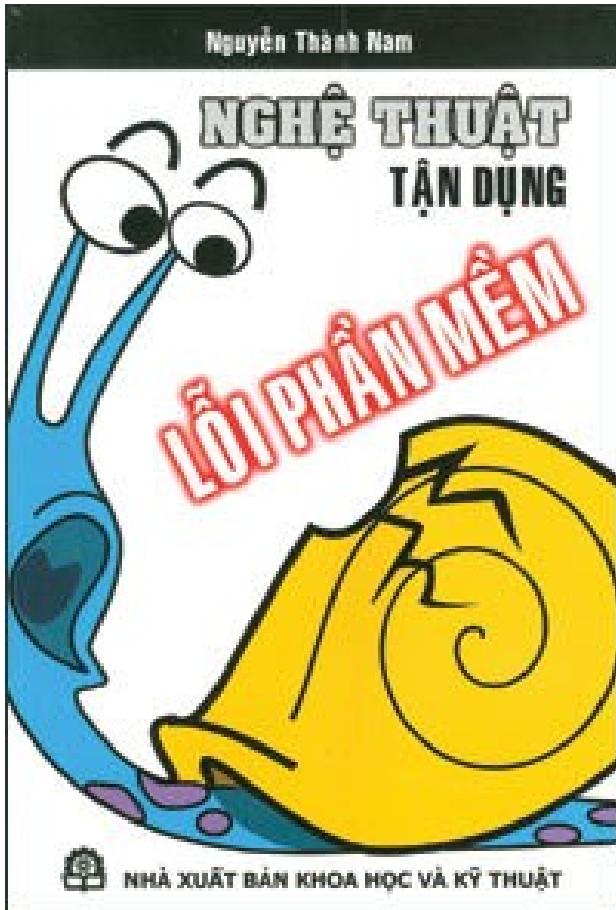
Foreword by Richard D. Pethia
CERT Director



Tài liệu tham khảo



Tài liệu tham khảo





Tài liệu tham khảo

- <https://security.berkeley.edu/secure-coding-practice-guidelines>
- https://wiki.sei.cmu.edu/confluence/display/sec_code/Top+10+Secure+Coding+Practices
- [https://owasp.org/www-pdf-archive/OWASP SCP Quick Reference Guide v2.pdf](https://owasp.org/www-pdf-archive/OWASP%20SCP%20Quick%20Reference%20Guide%20v2.pdf)
- <https://www.softwaretestinghelp.com/guidelines-for-secure-coding/>
- <http://security.cs.rpi.edu/courses/binexp-spring2015/>
- <https://www.ired.team/>



Lập trình an toàn & Khai thác lỗ hổng phần mềm



Trường ĐH CNTT TP. HCM

