ROP & FSB

CYDF 21 오정민

목차

1. ROP 그게 뭔데 뭐하는건데
2. FSB 넌 또 뭔데

Review (plt & got)

라이브러리에서 함수를 호출하는 루틴은 아래와 같음.

1. func@plt 실행
2. Got 참조
    a. 함수 주소가 존재? -> 그냥 점프
    b. 없음? -> 여타 다른 함수를 이용해 주소 구해주고 got에 적고 점프

이를 이용해 got에 우리가 원하는 주소를 넣는 **got overwriting**을 할 수 있다!

ex) printf(“/bin/sh”); 에서 printf got에 system 넣으면 system(“/bin/sh”)가 실행.

ROP ???

ROP = Return Oriented Programming

이전에 했던 RTL과 마찬가지로 결국 RETURN을 조작

-> 우리가 원하는 동작을 하도록 유도

왜 그런거 쓰는데?

-> 보호기법 우회 하기 위함. ROP의 경우 ASLR, NX 우회

Background for ROP (ASLR)

ASLR이 뭔데?
-> RTL을 막기 위해 여러 주소를 랜덤화 시키는 것.

자세한건 담주에 mitigation 배울 때 더 할 것.

결과적으로 heap, stack, binary section의 주소가 랜덤화 됨.
근데, 약간 통으로 옮겨서 base만 바뀌고 내부의 구조가 랜덤화 되지는 않는다.

즉, libc내부의 어떤 한 주소만 구하면 offset으로 나머지를 다 구할 수 있다!
(이 내용이 지난주에 언급한 libc base구하기 입니다.)

국룰 ROP Scenario

1. 메모리 어딘가에 “/bin/sh” 문자열 넣어주기.
2. Libc leak (임의의 함수의 got를 출력해줌.)
3. System(“/bin/sh”)로 리턴

일단 가장 단순화하면 위와 같다.

근데 leak을 하려면 한번 출력이 되어야할텐데….?

국룰 ROP Scenario

Libc leak과 system()함수의 call이 onetime에 이루어지기는 힘듦.
Why?
특정 함수의 주소를 출력해서 우리가 계산해주는 단계가 필요하기 때문.

그래서 우리는 두 단계로 나눠서 합니다.
Stage 1 : libc leak -> main으로 return
Stage 2 : system("/bin/sh");

"/bin/sh" 문자열은 언제 넣어줘도 상관 없겠죠?

ROP....

결국 이름처럼 return을 기반으로 함수 흐름을 주루룩 만들어야 함.

그 말은 사고의 틀을 좀 넓히는게 좋다~

또, rop를 이용할 때 뒤에 할 example같은 경우 그냥 bof가 있어서 바로 rop를 해줄 수 있지만, 꼭 bof만을 이용할 필요는 없음.

Ex) oob로 임의 주소 read/write될 때, 그냥 코드 로직상 임의 주소 read/write 될 때 가능할때 등등...

다양한 사고가 필요하다! -> 과제 풀면서 늘려보자구요~

Example 1: 32bit ROP

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

void init(){
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
}

int main()
{
    char buf[100];

    init();

    read(0, buf, 256);
    write(1, buf, 5);

    return 0;
}
```

```
gef➤  checksec
[+] checksec for '/mnt/c/Users/metam0ng/study/p
Canary                        : ✘
NX                            : ✓
PIE                           : ✘
Fortify                       : ✘
RelRO                         : Partial
```

보호기법 확인.

딱 봐도 read를 할 때, buffer overflow 가능.

근데 바이너리 안에 "/bin/sh"문자열도 없고 system 함수도 없기에 그냥 call은 할 수 없다.

그러면 libc의 system 쓰면 되겠네~

Aslr 켜져있는데 해당 주소 어떻게 아는데?

=> ROP Scenario 참고.

Example 2: 64bit ROP

```c
#include <stdio.h>

void init(){
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
}

int main()
{
    char buf[100];

    init();

    puts("Time to do x64 ROP");
    gets(buf);

    return 0;
}
```

```
gef➤  checksec
[+] checksec for '/mnt/c/Users/metam0ng/st
Canary                        : ✘
NX                            : ✓
PIE                           : ✘
Fortify                       : ✘
RelRO                         : Partial
```

보호기법 확인.

딱 봐도 gets를 할 때, buffer overflow 가능.

이후는 이전 문제와 동일함.

근데 64bit ROP이기 때문에 function call을 할 때 인자를 어떻게 넘겨줄 것인가! 이게 중요

해당 예제는 좀 쉽게 인자 하나만 해드렸습니당.

[Ref.] gadget

64bit 의 경우 calling convention에 의해 레지스터를 이용해 인자 처리.

-> rdi, rsi, rdx, rcx, r8, r9 … 순서

그러면 각각 인자를 pop [register] 를 이용해 레지스터에 넣어줘야 하는데, 가젯이 없는 경우도 있음.

해당 경우는 __libc_csu_init() 함수를 이용해서 처리하면 편합니다.

과제는 필요한 경우 그냥 제가 가젯을 넣어뒀으니까 나중에 여유가 될 때 return to csu를 따로 공부해보시면 좋슴당

ROP Q&A

What is Format String?

```c
int main()
{
    char *string = "Hello Cykor!\n";

    printf("%s", string);
}
```

입출력을 하는 scanf, printf, fprintf 등에서 정말 자주 사용.

보통 왼쪽의 코드 처럼 printf를 할 때 출력 형식을 지정해주며 출력을 한다.

이때 %[format] 이게 서식 지정자 이다.

(ex. %c, %s, %d ….)

What is Format String?

보통 아래의 서식 지정자 정도를 자주 사용함.

서식 지정자	출력 데이터 형태
%c	한 문자
%s	문자열
%d	부호 있는 10진수
%u	부호 없는 10진수
%f	고정 소수점으로 표현한 실수
%x	부호 없는 16진수
%p	포인터

What is Format String?

아래와 같이 다양한 지정 서식자가 존재함.

기호	설명	예시	출력
hh	출력 단위를 1바이트로	printf("%hhx", (unsigned char)0x12)	12
h	출력 단위를 2바이트로	printf("%hx", (unsigned short)0x1234)	1234
l	출력 단위를 4바이트로	printf("%lx", (unsigned long)0x12345678)	12345678
ll	출력 단위를 8바이트로	printf("%llx", (unsigned long long)0x12345678abcdef0)	12345678abcdef0
%n	지금까지 출력한 문자의 수를 저장(출력 x)	printf("%s%n", "CyKor", &num)	CyKor (+num에 5 저장)
$	몇 번째 인자를 출력할지 설정	printf("%2$d %1$d", 1, 2)	2 1
*	Field width 지정	printf("%*c", 5, 'a')	a (4칸 space + a)

What is Format String Bug?????

그래서 FSB가 뭔데
-> Format Sting을 잘못 사용해서 생기는 여러 이슈

사실 범인은 printf 함수 루틴.

Printf의 원형은 오른쪽과 같음.

SYNOPSIS top

```
#include <stdio.h>

int printf(const char *restrict format, ...);
```

가변 개수의 인자를 받을 수 있는데
내부에서 format string 과 그에 상응하는 인자의 개수를 체크하지 않음.

How to use?

결국 pwnable에 사용할 때 필요한 것 -> 임의 주소 입력, 임의 주소 출력

%(offset$)p를 이용해 leak

+

%(offset$)n을 이용해 overwrite

-> 잘 조합해서 사용하기.

자세한 건 실습 3개 보면서 합시다!

Example 1

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

void init(){
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
}

int main() {
    char fmt[0x50];
    int key = 0;
    int input = 0;

    init();

    srand(time(NULL));
    key = rand();

    printf("fmt : ");
    scanf("%80[^\n]", fmt);
    printf(fmt);

    printf("key : ");
    scanf("%d", &input);

    if (key == input)
        system("/bin/sh");
    else
        puts("wrong key :(");

    return 0;
}
```

Random 하게 생성되는 key를 input으로 넣어야 함.

Printf(fmt) 에서 fsb 가능

-> offset 계산 잘해서 key를 출력할 수 있음.

Example 2

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

int token = 0xdeadbeef;

void init(){
    setvbuf(stdin, NULL, _IONBF, 0);
    setvbuf(stdout, NULL, _IONBF, 0);
}

int main() {
    char fmt[0x50];

    init();

    printf("fmt : ");
    scanf("%80[^\n]", fmt);
    printf(fmt);

    if (token == 1234)
        system("/bin/sh");
    else
        puts("wrong token :(");

    return 0;
}
```

전역변수 token을 1234로 덮어야 함.

Printf(fmt) 에서 fsb 가능

token 주소는 고정

-> offset 구하고 계산 잘 해서 %n으로 덮어주기

Example 3

```c
// gcc oneshot.c -o oneshot -no-pie
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void gift() {
    system("/bin/sh");
}

int main() {
    char buf[0x100];

    printf("Look, if you had one shot, one opportunity\n");
    printf("To seize everything you ever wanted, in one moment\n");
    printf("Would you capture it or just let it slip? ♩♪\n\n");

    read(STDIN_FILENO, buf, sizeof(buf));
    printf(buf);
    exit(0);
}
```

Printf(buf) 에서 fsb.

Gift 실행해주면 쉘 나옴.

-> exit got를 덮어주자!

FSB Q&A

Assingment

Homework 디렉토리에 있는 문제들 푸시면 됩니다.

이건 무조건 풀기 : ez_rop, oneshot, oneshot1, onetime

Note와 helpme는 익스를 못하더라도 **최대한 분석하고**, 생각한만큼 써주시길 바랍니다.

Oneshot2는 새로 알아야할 개념이 있습니다. Note, helpme와 마찬가지로 최대한 분석하시고 여러 방면으로 고민하신 뒤 과제로 최대한 써서 제출하시면 됩니다. (hint : 보호기법)

오늘 rop, fsb만 배웠다고 해당 내용에만 치중하지 말고 두 기법을 도구로 잘 사용해주시길....

과제는 2주일 뒤인 5/1(월) 밤 12시까지 받겠습니다~
추가로 다음주는 중간고사라 쉬어 갑니다!

Assignment 요약.

ez_rop, oneshot, oneshot1, onetime -> 무조건 풀기

나머지 -> 최대한 분석하고 오늘 배운거 이외의 내용들도 있으니 충분히 찾아보고 고민한 뒤 분석내용 적어서 제출

과제 제출 기한 : 5/1 밤 12시

+질문사항 있으면 언제든 연락주세요

+ 과제는 github에 올려뒀습니다. Zip파일로 다운로드 받으셔도 되고 clone 받으셔서 사용하셔도 됩니다 :)