기술문서	15. 5. 31 작성

HUST 2015 Writeup

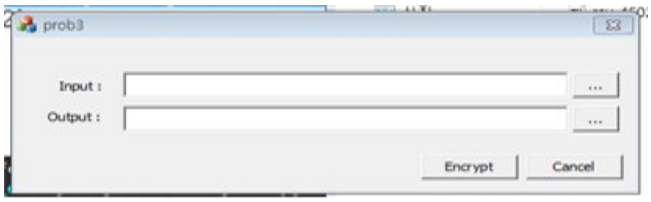
write by cert-is

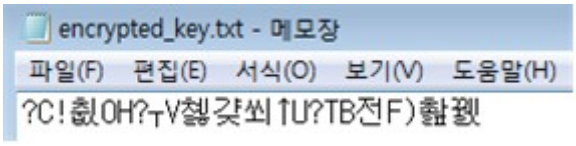
NO.	NAME	SCORE
1	Otaku	5715
2	CaptureTheSwag	5508
3	dcua	5409
4	CyKor	5407
5	MonitorHunter	5406
6	CERTIS	5302
7	newheart	3704
8	굿	3502
9	수호형여친구함	3200
10	인생망함	3003

Copyright @ 2015 CERT-IS

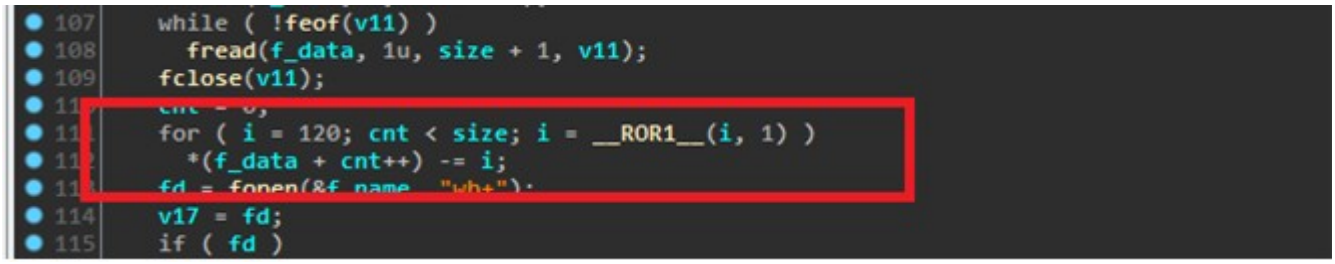
1. Reversing

1) Prob3





input파일을 넣으면 내부 연산에 의해 파일이 암호화되어 Output 파일로 저장이 된다. 따라서 리버싱으로 내부연산을 분석한 후 문제로 받은 'encrypted_key.txt'를 복호화하면 된다.



```
107   while ( !feof(v11) )
108     fread(f_data, 1u, size + 1, v11);
109   fclose(v11);
11?   cnt = 0;
11?   for ( i = 120; cnt < size; i = __ROR1__(i, 1) )
11?     *(f_data + cnt++) -= i;
11?   fd = fopen(&f_name, "wb+");
114   v17 = fd;
115   if ( fd )
```

<핵심코드- sub_4012f0>

```
data = 0xC1, 0x2A, 0x43, 0x21, 0xAE, 0x74, 0x4F, 0x48, 0xE9, 0x25, 0x16, 0x56, 0xAB, 0x9E, 0x81, 0x7
4, 0xBD, 0xFD, 0x18, 0x55, 0xDE, 0x76, 0x54, 0x42, 0xC0, 0xFC, 0x46, 0x29, 0xAC, 0x6F, 0x85, 0x75
# data is "encrypted_key.txt"
key = 0x78, 0x3c, 0x1e, 0x0f, 0x87, 0xc3, 0xe1, 0xf0
# key is result of _ROR_
a = ''
for i in range( len( data ) ):
        a += chr((data[i] +key[i%len(key)])&0x000000ff)

print "flag is", a
'''
flag is 9fa05708aa4e2abd596de95288d832fe
'''
```

 Copyright @ 2015 CERT-IS

2) rev_DB756D48

```
.text:00401050 loc_401050:                             ; CODE XREF: crypt(char *,char *,int,int)+64↓j
.text:00401050                 call    _rand
.text:00401055                 cdq
.text:00401056                 mov     ecx, 0FFh
.text:0040105B                 idiv    ecx
.text:0040105D                 inc     esi
.text:0040105E                 xor     [esi+buf-1], dl
.text:00401062                 cmp     esi, edi
.text:00401064                 jl      short loc_401050
```

코드의 핵심부분이다. 파일을 열고 거기에 따라서 1byte씩 연산하는 코드이고 그 연산은 rand()로 작동된다. 하지만 코드의 상단부에 srand(0x666)으로 고정이 되어 있으니, 문제풀이할때도 srand()를 0x666으로 코딩하면 된다. 또한 암호화를 xor연산하는데 이는, 역연산이 가능하므로 문제의 코드와 똑같이 코딩을 해주면 된다.

```
.text:00401000 buf = ebx
.text:00401000                 push    ebp
.text:00401001                 push    666h            ; seed
.text:00401006                 call    _srand
```

〈srand 설정부분〉

```
/* hust 2015 rev 풀이 */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
int main ( void )

        size_t size = 0;
        int cnt = 0;
        char buf[ 0x19000 ] = 0,;
        FILE *fd = NULL;
        srand( 0x666 );
        fd = fopen( "file.dat", "rb" );
        size = fread( buf, 1, 0x19000, fd );
        fclose( fd );
        do
                cnt++;
                buf[cnt-1] ^= (rand() % 0xff);
        while( cnt < size );
        printf("%s", buf );
        fd = fopen( "result.dat", "wb" );
        fwrite( buf, 1, size, fd );
        fclose( fd );
        return 0;
```

 Copyright @ 2015 CERT-IS

<복호화 결과>

3) 슈퍼마리오 게임

게임을 시작하니 단순한 마리오게임인줄 알았는데 뭔가 넘을수 없는 4차원의 구덩이가 있었다. 처음 접근은 클리어를 하면 키 값이 출력될것 같아서 String을 먼저 찾아보기로 했다. 문자열이 그림으로 출력되어서 간단히 찾을수 없다는걸 깨닫고 다른 접근 방법을 하기로 하였는데, 이때 팀원이 fceux라는 프로그램을 발견, 디버깅에 들어갔다. 해당 툴에서는 맵을 개조할수 있는 여러가지 기능이 있었는데, 자세한 사용법을 몰라 해매던 도중 나는 메모리 디버깅 메뉴를 발견하여 실행해 보았다. 메모리디버깅에서는 특정 오프셋에 있는 메모리가 얼마나 자주 변화하였고 현재값이 어떤 값인지 알려주는 간단한 테이블이 눈에 들어왔다. 여기서 만약 내가 점프를 하면 마리오의 현재 y축 위치, 점프 횟수, 점프 가속도 등이 어딘가에 세팅되어 있을것 같아서 점프하고 일시정지를 반복하는 방법으로 메모리가 변하는 곳을 추려내어 해당 오프셋을 찾았는데, 점프를 할때 올라가는값은 1,2,3,4 내려가는 값은 -1,-2,-3,-4 등으로 변하는걸 볼 수 있었다.

해당 롬을 에뮬레이터에 올리고 치트엔진을 attach 시켜서 점프를 하고 난 뒤 값을 검색, 내려왔을때 값을 검색.. 을 반복하여 점프에 해당하는 메모리 주소 3개를 찾아서 모두 양수로 만들고 값을 고정시켰더니 마리오가 무한점프 하며 하늘을 날기 시작했다. 이렇게 넘을수 없는 4차원의 구덩이를 넘고 그 뒤에 있는 넘을수 없는 4차원의 벽을 넘으니 벽돌로 KEY IS 라는 문자가 보이기 시작했고 해당 문자를 다 읽을때 까지 마리오는 하늘을 날았고 키 값을 인증했다.

 Copyright @ 2015 CERT-IS

4)rev 500(Vulnerability)

WhoAreYou.exe 바이너리가 주어집니다.

Ollydbg 로 열어보면 아마 패킹이 되어 있는 것으로 추측할 수 있습니다.

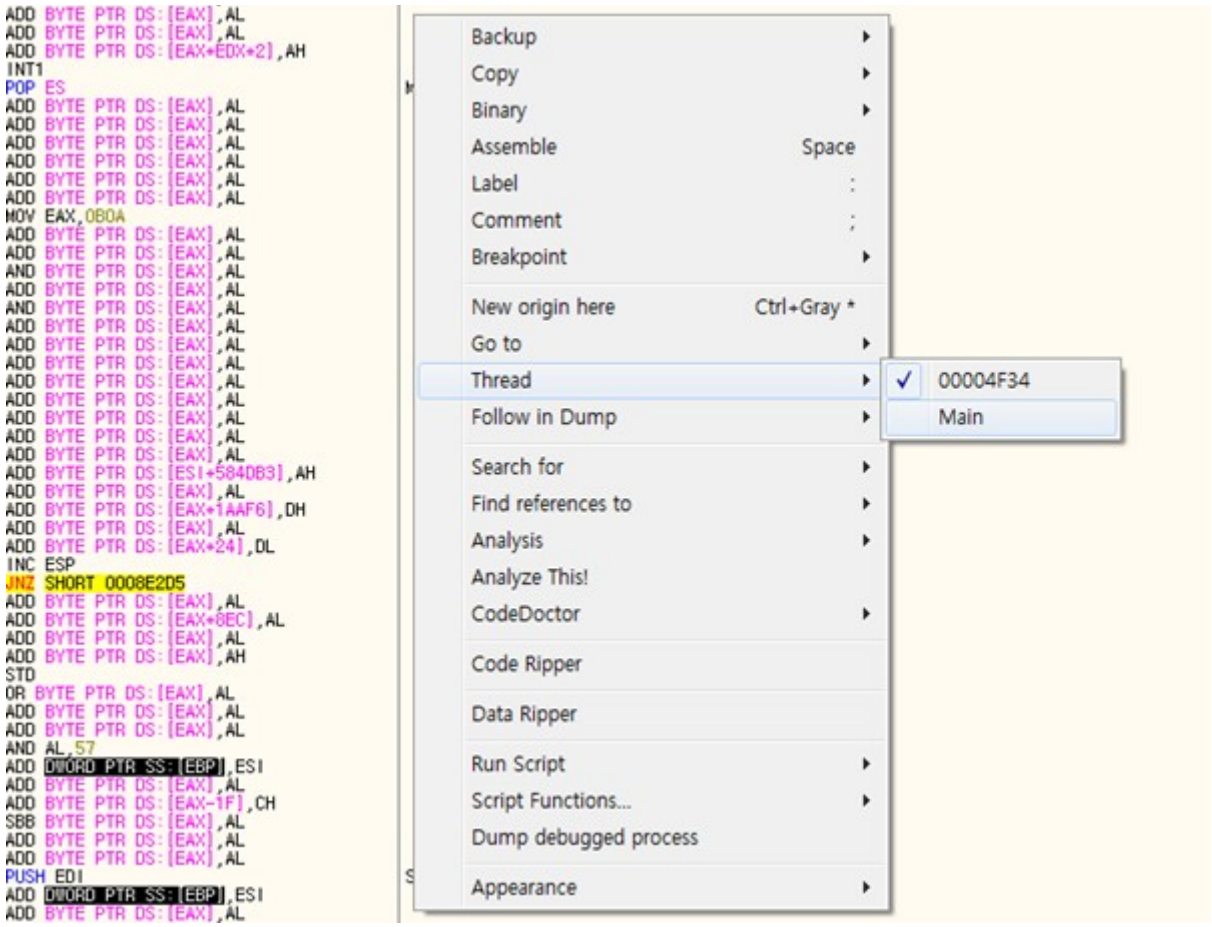
조금만 진행해보면 상당히 과정이 복잡하고 귀찮습니다.

그래서 그냥 Attach 신공을 사용합니다.

프로그램을 실행한 다음 입력을 받고 있으면 그 때 Ollydbg 로 attach 를 시도합니다.

먼저 패킹은 풀려있을 것이라 가정하고 Entry Point 를 찾아봅니다.

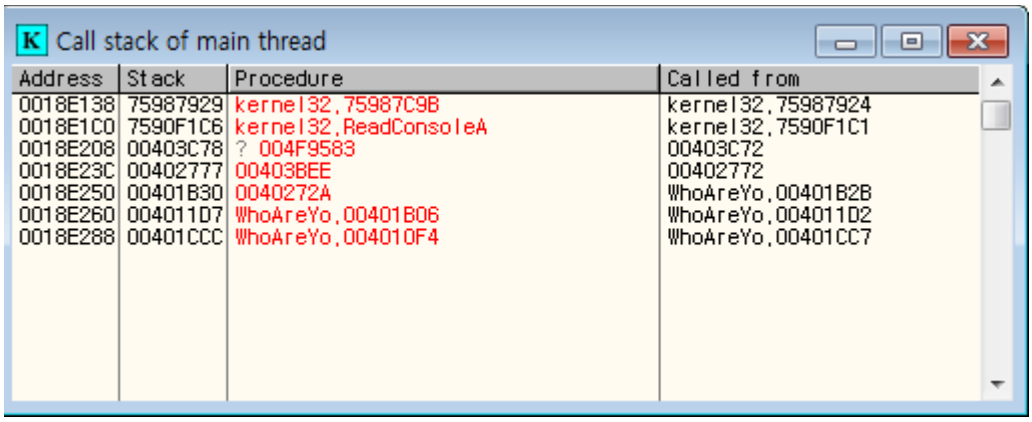
어차피 프로그램 규모가 그리 크지 않기 때문에 어렵지 않게 찾을 수 있습니다.



먼저 Thread 를 Main 으로 변경합니다.

 Copyright @ 2015 CERT-IS

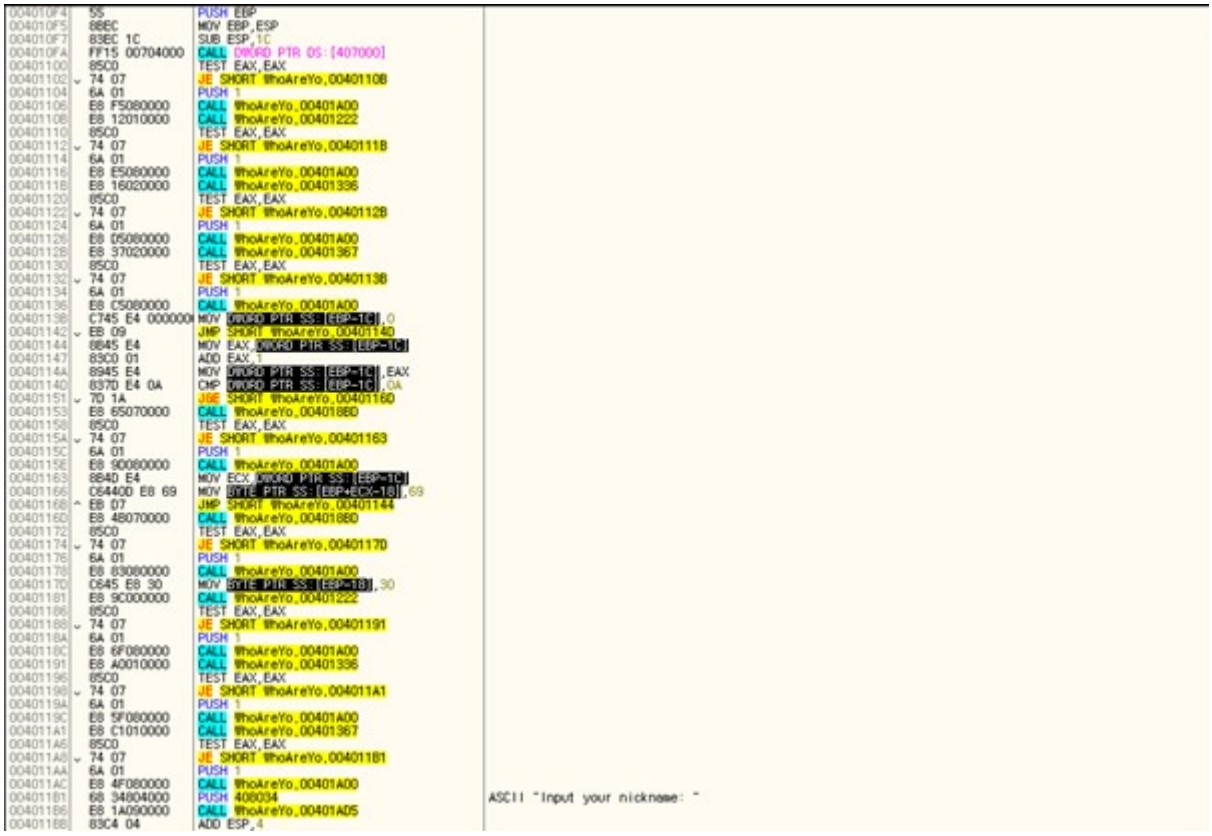
그 다음 Call Stack 을 확인합니다.



적당한 주소로 찍어 맞춰보면 됩니다.

어차피 보면 주소가 전부 401000 쪽에 있는 것으로 보아 그냥 패킹을 코드 섹션에

바로 풀었다는 것으로 생각할 수 있습니다. 그럼 일단 대충 그 쪽으로 이동합니다.



대강 위와 같습니다. 여기가 일단 실제 코드들이 있는 곳이라는 것은 쉽게 알 수 있습니다.

일단 이 쪽은 전체 크기가 0x6000 이 할당되어 있습니다. 그리 많지 않으니 눈으로 찾아봅니다.

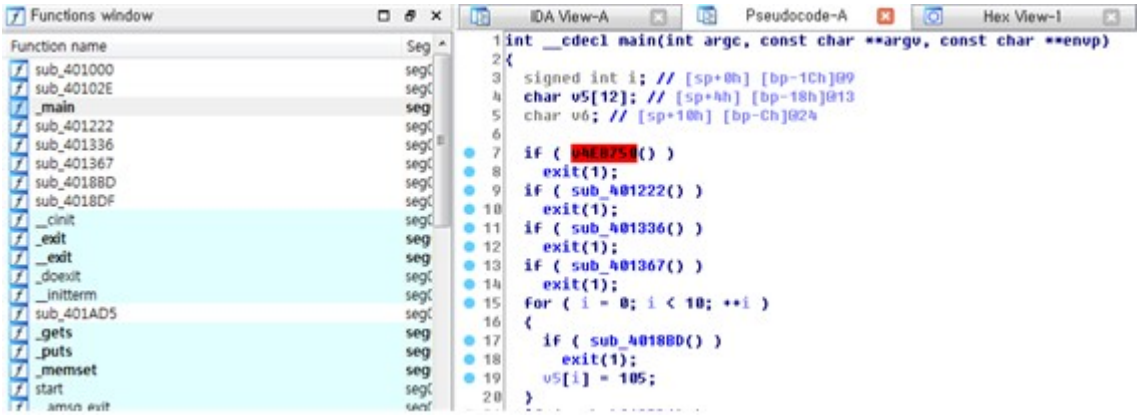
바로 아래와 같은 부분을 찾을 수 있습니다.

 Copyright @ 2015 CERT-IS

```
00401C18  55            PUSH EBP
00401C19  8BEC          MOV EBP,ESP
00401C1B  6A FF         PUSH -1
00401C1D  68 E8704000   PUSH 4070E8
00401C22  68 DC344000   PUSH 4034DC
00401C27  64:A1 00000000 MOV EAX,DWORD PTR FS:[0]
00401C2D  50            PUSH EAX
00401C2E  64:8925 000000 MOV DWORD PTR FS:[0],ESP
00401C35  83EC 10       SUB ESP,10
00401C38  53            PUSH EBX
00401C39  56            PUSH ESI
00401C3A  57            PUSH EDI
00401C3B  8965 E8       MOV DWORD PTR SS:[EBP-18],ESP
00401C3E  FF15 40704000 CALL DWORD PTR DS:[407040]
00401C44  33D2          XOR EDX,EDX
00401C46  8AD4          MOV DL,AH
00401C48  8915 20A94000 MOV DWORD PTR DS:[40A920],EDX
00401C4E  8BC8          MOV ECX,EAX
00401C50  81E1 FF000000 AND ECX,0FF
00401C56  890D 1CA94000 MOV DWORD PTR DS:[40A91C],ECX
00401C5C  C1E1 08       SHL ECX,8
00401C5F  03CA          ADD ECX,EDX
00401C61  890D 18A94000 MOV DWORD PTR DS:[40A918],ECX
00401C67  C1E8 10       SHR EAX,10
00401C6A  A3 14A94000   MOV DWORD PTR DS:[40A914],EAX
00401C6F  6A 00         PUSH 0
00401C71  E8 11170000   CALL 00403387
00401C76  59            POP ECX
00401C77  85C0          TEST EAX,EAX
00401C79  75 08         JNZ SHORT WhoAreYo.00401C83
00401C7B  6A 1C         PUSH 1C
00401C7D  E8 9A000000   CALL WhoAreYo.00401D1C
00401C82  59            POP ECX
00401C83  8365 FC 00    AND DWORD PTR SS:[EBP-4],0
00401C87  E8 D6130000   CALL 00403067
00401C8C  FF15 3C704000 CALL DWORD PTR DS:[40703C]
00401C92  A3 44BE4000   MOV DWORD PTR DS:[40BE44],EAX
00401C97  E8 99120000   CALL 00402F35
00401C9C  A3 54A94000   MOV DWORD PTR DS:[40A954],EAX
00401CA1  E8 42100000   CALL 00402CE8
00401CA6  E8 840F0000   CALL 00402C2F
00401CAB  E8 23FDFFFF   CALL WhoAreYo.00401903
00401CB0  A1 30A94000   MOV EAX,DWORD PTR DS:[40A930]
00401CB5  A3 34A94000   MOV DWORD PTR DS:[40A934],EAX
00401CBA  50            PUSH EAX
00401CBB  FF35 28A94000 PUSH DWORD PTR DS:[40A928]
00401CC1  FF35 24A94000 PUSH DWORD PTR DS:[40A924]
00401CC7  E8 28F4FFFF   CALL WhoAreYo.004010F4
00401CCC  83C4 0C       ADD ESP,0C
```

누가 봐도 “나 EntryPoint 요” 라고 하고 있습니다. 빌드한 컴파일러야 Visual Studio 일 것이니 엔트리 포인트 형태도 다 이런 형태입니다. 핵심은 FS:[0] (SEH 세팅 관련) 부분과 아래 쪽의 push 3개(argc, argv, …) 이후 main 함수 호출하는 부분만 보면 알 수 있습니다.

그러면 이제 편하게 IDA 로 분석하기 위해 이 상태로 401C18 을 EntryPoint 로 대충 덤프합니다. Rebuild IAT 는 하지 않고, ImpREC 안 써도 됩니다. 그냥 덤프한 것을 IDA 로 열어봅니다.



(단언컨대 IDA + Hex-rays 신은 가장 완벽합니다.)

 Copyright @ 2015 CERT-IS

아주 깔끔하게 분석을 해 주었습니다. 빨간색 부분이나 기타 이상한 부분은 대강 넘어갑니다.

여기서부턴 정적 분석과 동적 분석을 약간씩 같이 해가면서 하면 편합니다.

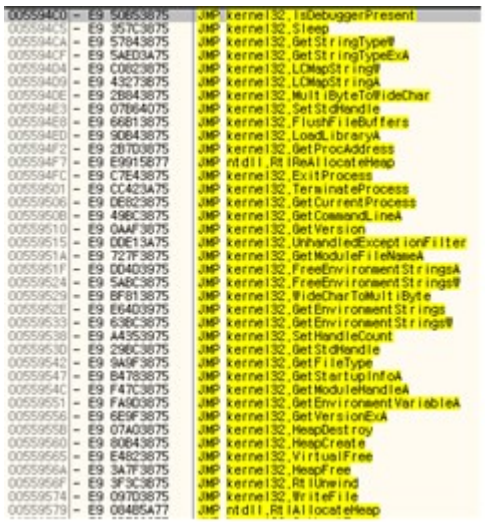
```
int sub_401222()
{
  char v1; // [sp+Ch] [bp-Ch]@1
  char v2; // [sp+Dh] [bp-Bh]@1
  char v3; // [sp+Eh] [bp-Ah]@1
  char v4; // [sp+Fh] [bp-9h]@1
  char v5; // [sp+10h] [bp-8h]@1
  char v6; // [sp+11h] [bp-7h]@1
  char v7; // [sp+12h] [bp-6h]@1

  v1 = '0';
  v2 = '1';
  v3 = '1';
  v4 = 'y';
  v5 = 'D';
  v6 = 'b';
  v7 = 'g';
  return v4EB836(&v1, 0);
}
```

코드가 전체적으로 봐도 굉장히 이상합니다. 조건식에 exit 가 많이 붙어 있습니다.

이 부분은 안티 디버깅일 가능성이 크다고 추측을 할 수 있습니다.

적당한 함수 아무거나 하나 들어가서 봅니다.



OllyDbg 라는 문자열로 보아 역시 안티 디버깅입니다.

그 외에 IDA 와 WinDbg 도 인식하고 있습니다. 그 외에 IsDebuggerPresent 와 기타 등등

많이 찾아볼 수 있습니다. 뭐 아무튼 귀찮으니 무시하고 넘어갑니다.

 Copyright @ 2015 CERT-IS

```
sub_401AD5(aInputYourNickn);
if ( sub_4018BD() )
  exit(1);
gets(&v6);
if ( v4EB750() )
  exit(1);
sub_401AD5(aHelloS, &v6);
```

빨간색은 IAT 가 제대로 복구가 되지 않아서 저렇게 보이는데, 여기선 별 다른 IAT 리다이렉션 등이 없기 때문에 OllyDump 에서 기본 지원하는 Method 로 IAT 복구만 해도 사실 다 보입니다.

여기선 그냥 동적 분석으로 해볼 겸 해서 복구를 하지 않고 진행했습니다.

Ollydbg 에서 해당 부분으로 가서 뭐가 있는지 봅니다.

편하게 테이블이 있어서 호출하는 것이 무엇인지 쉽게 알 수 있습니다.

또한 위의 코드 부분에서 아래와 같은 부분이 있습니다.

401AD5 는 문맥상 printf 나 puts 일 것임을 쉽게 알 수 있으니 안 쪽은 볼 필요가 없습니다.

그런데 보면 IDA 에서 분석을 다 제대로 못했는지, 아무 건덕지도 없습니다.

거기다 main 에서 전혀 호출하지 않는 몇 가지 함수도 있습니다.

대략 4018DF 함수에 간 뒤 xref 를 확인해 보면 왜인지 많이 나옵니다.

따라서 이 중 아무거나 들어가봅니다.

```
seg000:004012FB loc_4012FB:                             ; CODE XREF: seg000:004012F2↑j
seg000:004012FB                 push    5
seg000:004012FD                 lea     eax, [ebp-18h]
seg000:00401300                 push    eax
seg000:00401301                 lea     ecx, [ebp-10h]
seg000:00401304                 push    ecx
seg000:00401305                 call    sub_4018DF
seg000:0040130A                 add     esp, 0Ch
seg000:0040130D                 call    sub_4018BD
seg000:00401312                 test    eax, eax
seg000:00401314                 jz      short loc_40131D
seg000:00401316                 push    1
seg000:00401318                 call    _exit
```

엌ㅋ 분석이 되지 않았습니다. (아무데서도 호출을 하지 않아서)

그러면 직접 이 근방에서 분석 안된 것들을 찾아다가 Create function 을 해줍니다.

 Copyright @ 2015 CERT-IS

```
seg000:004012FB loc_4012FB:                             ; CODE XREF: seg000:004012F2↑
seg000:004012FB                 push    5
seg000:004012FD                 lea     eax, [ebp-18h]
seg000:00401300                 push    eax
seg000:00401301                 lea     ecx, [ebp-10h]
seg000:00401304                 push    ecx
seg000:00401305                 call    sub_4018DF
seg000:0040130A                 add     esp, 0Ch
seg000:0040130D                 call    sub_4018BD
seg000:00401312                 test    eax, eax
seg000:00401314                 jz      short loc_40131D
seg000:00401316                 push    1
seg000:00401318                 call    _exit
```

401263 과 4013A4 두 함수를 새로 찾을 수 있습니다. 코드를 대충 봅니다.

```
seg000:004012FB loc_4012FB:                             ; CODE XREF: seg000:004012F2↑
seg000:004012FB                 push    5
seg000:004012FD                 lea     eax, [ebp-18h]
seg000:00401300                 push    eax
seg000:00401301                 lea     ecx, [ebp-10h]
seg000:00401304                 push    ecx
seg000:00401305                 call    sub_4018DF
seg000:0040130A                 add     esp, 0Ch
seg000:0040130D                 call    sub_4018BD
seg000:00401312                 test    eax, eax
seg000:00401314                 jz      short loc_40131D
seg000:00401316                 push    1
seg000:00401318                 call    _exit
```

```
seg000:004012FB loc_4012FB:                             ; CODE XREF: seg000:004012F2↑
seg000:004012FB                 push    5
seg000:004012FD                 lea     eax, [ebp-18h]
seg000:00401300                 push    eax
seg000:00401301                 lea     ecx, [ebp-10h]
seg000:00401304                 push    ecx
seg000:00401305                 call    sub_4018DF
seg000:0040130A                 add     esp, 0Ch
seg000:0040130D                 call    sub_4018BD
seg000:00401312                 test    eax, eax
seg000:00401314                 jz      short loc_40131D
seg000:00401316                 push    1
seg000:00401318                 call    _exit
```


Copyright @ 2015 CERT-IS

```
  v13 = '1';
  v14 = '2';
  v15 = '3';
  v16 = 'n';
  v17 = 'm';
  v18 = 'i';
  v19 = 'o';
  if ( v4EB750('dcba', 'hgfe') )
    exit(1);
  v6 = 'h';
  v7 = 'e';
  v8 = 'l';
  v9 = 'l';
  v10 = 'o';
  v11 = ' ';
  v12 = 'w';
  v1 = 'o';
  v2 = 'r';
  v3 = 'l';
  v4 = 'd';
  v5 = '!';
  if ( v4EB750() )
    exit(1);
  sub_4018DF((int)&v6, (int)&v1, 5);
  if ( sub_4018BD() )
    exit(1);
  if ( sub_4018BD() )
    exit(1);
  return sub_40102E();
}
```

아무것도 없습니다. 귀찮으니 대강 넘어갑니다.

4018DF 같은 건 함수 인자만 봐도 무슨 함수인지 추측할 수 있습니다.

그럼 4013A4 를 가봅니다.

 Copyright @ 2015 CERT-IS

```
      exit(1);
    if ( sub_4018BD() )
      exit(1);
    sub_4018DF((int)&v1, (int)&v10, 3);
    if ( sub_4018BD() )
      exit(1);
    sub_4018DF((int)&v1, (int)&v7, 3);
    puts(&v1);
    if ( sub_4018BD() )
      exit(1);
    v4EB859(v22);
    exit(1);
  }
  if ( v4EB750() )
    exit(1);
  if ( v4EB750() )
    exit(1);
  if ( sub_4018BD() )
    exit(1);
  v4EB845(v22, &v51, 12, 0);
  if ( sub_4018BD() )
    exit(1);
  v4EB84A(v22, &v23, 30, 0);
  if ( sub_4018BD() )
    exit(1);
  if ( sub_4018BD() )
    exit(1);
  if ( sub_4018BD() )
    exit(1);
  puts(&v23);
  v4EB755(5000);
  if ( sub_4018BD() )
    exit(1);
  v4EB859(v22);
  return v4EB84F();
}
```

길어서 좀 번거롭습니다. 제일 뒤에 puts 가 있는 것으로 보아 운 좋으면 그냥 바로 답을 출력해주는 함수일 가능성이 있습니다. 이 함수는 호출하는 곳이 전혀 없으므로 Ollydbg 에서 attach 후에 강제로 이 쪽으로 넘겨버려서 실행을 시키면 됩니다.

그러나 중간에 많은 안티디버깅들이 있습니다. 다행히 유형은 매우 단순하고 개수로는 몇 개 되지도 않습니다. 그러니 그냥 그 부분만 골라서 코드 패치를 해줍니다. 함수 호출하는 부분인 경우 그냥 앞 부분을 xor eax,eax 후 ret 만 하도록 패치를 해줍니다.

IsDebuggerPresent 를 직접 호출하거나 하는 부분은 플러그인이 있으면 자동으로 되고 없다면 그냥 마찬가지로 IsDebuggerPresent 실제 코드부로 가서 똑같이 xor eax,eax 후 ret 으로 패치해주면 간단합니다. 중간에 소켓 연결 같은 것이 있어 지금은 접속이 안됩니다. 어쨌든 이렇게 키를 바로 얻습니다.

 Copyright @ 2015 CERT-IS

5)rev1000(showmethemoney)

exe 파일 하나와 HelpMe.encrypted 파일이 주어집니다.

이것도 아까 리버싱 문제처럼 암호화 하는 프로그램과 이미 암호화된 파일을 줬다는 것을

대략 추측할 수 있습니다. 그러나 아까 문제와는 너무나 다릅니다.

일단 왜인지 실행 자체가 되지 않습니다. 이유가 꽤 많은 이유가 있는데,

디버깅하면서 하나씩 우회하면서 실행해보면 됩니다.

```
WPARAM __stdcall WinMain(HINSTANCE hInstance, int a2, int a3, int a4)
{
  WPARAM result; // eax@2
  WNDCLASSEXW v5; // [sp+8h] [bp-50h]@1
  struct tagMSG Msg; // [sp+38h] [bp-20h]@4

  v5.cbSize = 48;
  v5.style = 3;
  v5.lpfnWndProc = WndProc;
  v5.cbClsExtra = 0;
  v5.cbWndExtra = 0;
  v5.hInstance = hInstance;
  v5.hIcon = LoadIconW(hInstance, 0x7F00);
  v5.hCursor = LoadCursorW(0, 0x7F00);
  v5.hbrBackground = 6;
  v5.lpszMenuName = 0;
  v5.lpszClassName = L"win32app";
  v5.hIconSm = LoadIconW(hInstance, 0x7F00);
  if ( RegisterClassExW(&v5)
    && (dword_42F3F0 = hInstance,
        CreateWindowExW(
          0,
          L"win32app",
          L"Sh0w m3 7he m0n3y",
          0xCF0000u,
          2147483648,
          2147483648,
          500,
          100,
          0,
          0,
          hInstance,
          0)) )
```

먼저 IDA 로 일단 열어봅니다.

(WinMain 이나 WndProc 등의 이름은 제가 IDA 에서 붙여놨었던 것들입니다.)

 Copyright @ 2015 CERT-IS

```
v107 = 0;
memset(&v108, 0, 0x31u);
if ( FindWindowW(L"WinDbgFrameClass", 0) )
  ExitProcess(0);
if ( Msg > 0x111 )
{
  if ( Msg != 0x112 )
  {
    if ( Msg == 0x113 )
    {
      if ( wParam == 1 )
      {
        ++dword_42F3F4;
        sub_4067D0();
        if ( dword_42F3F4 == 20150522 )
        {
          KillTimer(hWnd, 1u);
          ExitProcess(1u);
        }
      }
      else if ( wParam == 2 )
      {
```

일단 Window 를 생성하고, 프로시저 함수가 있는 것을 볼 수 있습니다.

여긴 볼 게 없으니 바로 WndProc 으로 가봅니다.

(위의 ExitProcess 같은 것도 모두 이름을 붙여 놓은 것들입니다.

메시지 switch 문 일것인데, 코드와 메시지를 연관시켜 가면서 분석을 약간 해보면

아래쪽의 수 많은 코드는 WM_CREATE 일 때 실행되는 코드이고 그 이외에는

전부 큰 의미가 없는 코드라는 것을 추측할 수 있습니다.

아래 쪽에는 아래와 같은 코드가 있습니다.

 Copyright @ 2015 CERT-IS

```
SetTimer(hWnd, 1, 5000, 1);
SetTimer(hWnd, 2, 6000, v6);
SendMessageW(hWnd, 0x113u, 1u, 0);
SendMessageW(hWnd, 0x113u, 2u, 0);
BlockInput();
sub_403BB0();
v10 = CreateThread(sub_4013A0, v7, v8, v9);
Handles = v10;
v11 = v10;
v13 = v12;
sub_4061F0(v10);
unused2 = CreateThread(sub_408300, v14, v13, v11);
sub_4077F0("def91def05def91def50def91defd1defd1def10def91def52def91def51", &MultiByteStr);// nagpie
v15 = sub_407C30(&MultiByteStr);
v16 = (&Type - v15);
do
{
  v17 = *v15;
  ++v15;
  *&v16[v15 - 2] = v17;
}
while ( v17 );
v18 = sub_407B40(&Type, 0x68u, &Type);
v19 = (&v87 - v18);
do
{
  v20 = *v18;
  ++v18;
  *&v19[v18 - 2] = v20;
}
while ( v20 );
v21 = sub_407CA0(&v87);
```

(위 코드 역시 SetTimer, BlockInput, CreateThread 등은 이름을 붙인 것입니다.)

저기에서 원래는 API 가 대부분 이름이 없는 것들이 많습니다. 이것들은 인자로 대충 추측하거나 또는 동적 분석으로 알아내면 됩니다. 내부적으로 일종의 해시 테이블을 갖고 있는지 암호화된 것인지 따로 분석은 안했으나 특정 문자열에 대해 특정한 문자열을 리턴하는 함수가 있습니다. 동적으로 무엇이 리턴되는지 확인해서 분석하면 됩니다.

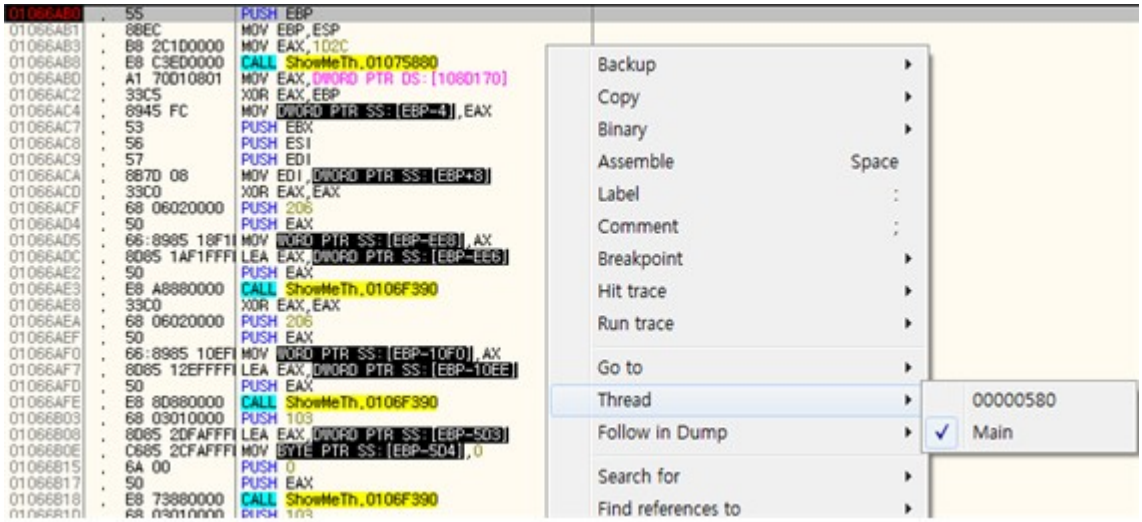
여기서부턴 동적 분석을 해봅니다.

먼저 WndProc 에 bp 를 걸고 WM_CREATE 가 올 때까지 진행을 합니다.

—

현재 메시지가 어떤 메시지인지 보려면 스택의 2번째 인자를 보면 됩니다.

 Copyright @ 2015 CERT-IS



현재 메시지가 어떤 메시지인지 보려면 스택의 2번째 인자를 보면 됩니다.

```
002FF614  761C62FA  RETURN to USER32.761C62FA
002FF618  00D512E0
002FF61C  00000024
```

위처럼 0x24 가 메시지 입니다. WM_CREATE 는 1 이므로 그 때까지 F9 를 누릅니다.

```
002FF5E4  761C62FA  RETURN to USER32.761C62FA
002FF5E8  00D512E0
002FF5EC  00000001
```

이제 IDA 에서 코드를 보면 대충 이렇습니다.

먼저 WM_CREATE 가 아닌 경우를 조금 보겠습니다.

 Copyright @ 2015 CERT-IS

```
if ( Msg > 0x111 )
{
  if ( Msg != 0x112 )
  {
    if ( Msg == 0x113 )
    {
      if ( wParam == 1 )
      {
        ++dword_42F3F4;
        sub_4067D0();
        if ( dword_42F3F4 == 20150522 )
        {
          KillTimer(hWnd, 1u);
          ExitProcess(1u);
        }
      }
      else if ( wParam == 2 )
      {
        sub_4069A0();
        OutputDebugStringW(L"%s%s%s%s%s%s%s%s%s%
        return 0;
      }
      return 0;
    }
    v4 = Msg == 0x200;
    goto LABEL_47;
  }
  goto LABEL_55;
}
if ( Msg == 0x111 )
{
  if ( !wParam )
    ExitProcess(1u);
```

아래에도 좀 더 있지만, 아무튼 ExitProcess 가 많이 보입니다.

뭐든 간에 일단 이런 건 실행하는데에 전혀 도움이 안될거라는 건 확실합니다.

저 중에선 WM_TIMER 도 있으므로, 뭐든 SetTimer 같은 건 실행 안 해보면 됩니다.

(만약 우회해서 문제 되는 게 있으면 그냥 다시 실행 하면 됩니다.)

```
SetTimer(hWnd, 1, 5000, 1);
SetTimer(hWnd, 2, 6000, v6);
SendMessageW(hWnd, 0x113u, 1u, 0);
SendMessageW(hWnd, 0x113u, 2u, 0);
BlockInput();
sub_403BB0();
v10 = CreateThread(sub_4013A0, v7, v8, v9);
Handles = v10;
v11 = v10;
v13 = v12;
sub_4061F0(v10);
unused2 = CreateThread(sub_408300, v14, v13, v11);
```

일단 위와 같이 시작합니다. SetTimer 가 벌써 2개나 보입니다.

 Copyright @ 2015 CERT-IS

뭔지 분석할 필요 없이, 걸리적 거린다 싶으면 그냥 올리디버거에서 실행 하지 말고
넘겨버리면 됩니다. 그리고 SendMessage 는 WM_TIMER 를 바로 호출하도록 하고 있습니다.
이것도 필요 없으니 넘겨버립니다. 그리고 BlockInput 은 실행해보면 별 문제 없이 넘어가니
일단 가봅니다. 그런데 뒤의 403BB0() 함수는 실행하면 안에서 나올 생각을 하지 않습니다.
대강 내용을 보면 별 의미가 없어 보이는 더미코드가 꽤 많고 외부에 영향을 줄 것 같은
부분도 없습니다. 특별히 리턴하는 함수도 아니므로 그냥 무시하고 넘겨버립니다.

뒤에 쓰레드 생성을 두 번 합니다. 이것도 분석하면 별 의미를 모르겠으니 그냥 넘깁니다.
여기서 뒤 쪽에 CreateThread 의 리턴값(쓰레드 핸들)을 쓰는 함수가 여기저기 나옵니다.
그 부분들은 보면 아무 의미 없는 함수들이므로 마찬가지로 넘겨야 합니다.
안 넘기면 핸들이 없으므로 당연히 제대로 동작하지 않습니다.

```
sub_4077F0("def91def05def91def50def91defd1defd1def10def91def52def91def51", &MultiByteStr);// magpie
v15 = sub_407C30(&MultiByteStr);
v16 = (&Type - v15);
do
{
  v17 = *v15;
  ++v15;
  *&v16[v15 - 2] = v17;
}
while ( v17 );
v18 = sub_407B40(&Type, 0x68u, &Type);
v19 = (&v87 - v18);
do
{
  v20 = *v18;
  ++v18;
  *&v19[v18 - 2] = v20;
}
while ( v20 );
v21 = sub_407CA0(&v87);
memset(&MultiByteStr, 0, 0x200u);
memset(&Type, 0, 0x400u);
sub_4061F0(unused2);
GetTempPathW(0x104u, &Buffer);
v22 = sub_407CA0(&Buffer);
sub_4077F0("def51defd0def91def51def91def85defd1def51def91defc4def11def12def91def85defd1def11", &v103);// SeoulHot
sub_4077F0("def11def51def11def11def11def11", &v105);// EDD
sub_4077F0("def91def44defd1def10def91defd1", &v107);// jpg
_sprintf(&FileName, "%s%s-%s-%s.%s", v22, &v103, &v105, "262", &v107);
```

아래 쪽엔 이러한 코드가 있습니다.
여기서 보이는 긴 문자열들은 전부 항상 같은 특정 문자열에 대응이 되는 것들입니다.
보면 같은 함수(4077F0) 에 항상 넘겨줍니다. 저건 동적 분석으로 어떤 결과가 나오는지
보고 기록해두면 편합니다. 안에서 복호화를 하는지 해시 테이블인지 그건 알 필요 없습니다.
저건 나중에 다른 함수의 내부에서도 저런 식으로 쓰므로 그때마다 동적으로 알아내면 됩니다.

저 407B40 에서는 리소스에서 뭔가 하나 끄집어낸 다음 %Temp% 디렉토리에 magpie 라는
파일명으로 그 내용을 씁니다. 그리고 아래쪽에서 %Temp% 에 SeoulHot-EDD-262.jpg 라는

 Copyright @ 2015 CERT-IS

이름으로 또 파일을 씁니다. 그리고 아래쪽에서 호출하는 407A60 에서 아까 쓴 magpie 파일을

연 다음 xor 같은 느낌으로 복호화하여 jpg 이미지를 뽑아냅니다.

```
sub_4077F0("def51defd0def91def51def91def85defd1def51def91defc4def11def12def91def85defd1def11", &v103);// SeoulHot
sub_4077F0("def11def51def11def11def11def11", &v105);// EDD
sub_4077F0("def91def4%defd1def10def91defd1", &v107);// jpg
_sprintf(&FileName, "%s%s-%s-%s.%s", v22, &v103, &v105, "262", &v107);
sub_407A60(v21, &FileName, 127);
none(&v87);
sub_4087A0(&FileName);
DeleteFileA(&FileName);
sub_4077F0("defd1def4%def91def52defd1def10", &MultiByteStr);// zip
_sprintf(&v97, "%s.%s", &FileName, &MultiByteStr);
memset(&MultiByteStr, 0, 0x200u);
memset(&Type, 0, 0x400u);
v23 = CreateFileA(&v97, 0xC0000000, 0, 0, 3u, 0x80u, 0);
v25 = SetFilePointer(v23, 0, v24, 1);
memset(v76, 0, 0x400u);
```

(아까 거기임)

저 이미지의 407A60 에서 그렇게 jpg 파일을 쓰고 Temp 디렉토리를 확인해보면

낚시 이미지가 나옵니다. 뭐 일단 넘기고, 그 다음 아래쪽에선 4087A0 이 있습니다.

```
v35 = 0;
v1 = this;
memset(&v36, 0, 0x1FFu);
sub_4077F0(
  "def51def11def91def12def91def52defd1defd0def11def52defd1defd0def11def05def
  &v35);
v33 = 0;
memset(&v34, 0, 0x103u);
_sprintf(&v33, "%s.%s", v1, "zip");
v2 = sub_408710(v1);
v21 = sub_4044A0;
v22 = sub_404550;
v23 = sub_4045D0;
v24 = sub_404650;
v25 = sub_404690;
v26 = sub_404700;
v27 = sub_404730;
v28 = 0;
v4 = sub_404ED0(v3, &v21);
v20 = v4;
if ( v4
  && (v29 = 0,
      _mm_storeu_si128(&v30, 0i64),
      _mm_storeu_si128(&v31, 0i64),
      sub_408640(v1, &FatTime),
      !sub_405010(v1, v4, &v29, v9, v10, v11, v12, v13, v14, v15, v16, v17,
  && (v6 = fopen(v1, "rb")) != 0 )
```

4087A0 의 내부는 이렇게 되어 있습니다. 주석을 안 달아놨지만 저 긴 문자열의 경우

“ThisIsMyOwnSecretWayToUseTheData” 입니다. 사실 아래쪽의 zip 과 결합해서 대충 감으로

봐도 이건 zip 비밀번호라는 걸 알 수 있습니다. 저 아래쪽의 인자가 더럽게 많이 넘어가는

 Copyright @ 2015 CERT-IS

것 처럼 보이는 405010 에 이 문자열이 넘어가고, zip 헤더 구성할 때 실제로 이 인자의

존재에 따라 암호화 여부 플래그를 설정하는 것으로 확실해집니다.

```
while ( ReadFile(v23, v76, 0x3FFu, &NumberOfBytesRead, 0) )
{
  if ( !NumberOfBytesRead )
    break;
  v26 = 0;
  do
  {
    value7 = v76[v26];
    v27 = value7;
    v28 = __ROR1__(value7 ^ 0x21, 1);
    v28 = __ROR1__(v28, 1);
    v76[v26++] = ~v28;
  }
  while ( v26 < 1024 );
  SetFilePointer(v23, v25, v27, 0);
  WriteFile(v23, v76, NumberOfBytesRead, &NumberOfBytesWritten, 0);
  v25 = SetFilePointer(v23, 0, v29, 1);
  memset(v76, 0, 0x400u);
}
CloseHandle(v23);
```

뭐 일단 중요한 건 아까의 jpg 이미지를 여기서 zip 압축을 시키게 됩니다.

물론 비밀번호는 저 문자열일 것입니다.

다음 아래 쪽에서 다시 이 zip 을 열어서, xor, ror, not 으로 간단히 암호화 비슷하게 합니다.

뭐 이건 바로 역연산이 가능한 부분이니 거의 무시해도 됩니다.

 Copyright @ 2015 CERT-IS

```
sub_4022C0(&Type, &v109);
memset(&MultiByteStr, 0, 0x200u);
memset(&Type, 0, 0x400u);
sub_4077F0(
  "def90def85def91def12defd1def51defd1defd0defd1def11def90def85def91def44defd1def51defd
  &MultiByteStr);                               // /hust/just/lust/trust/first.php?f=1024
v34 = MultiByteToWideChar(0, 0, &MultiByteStr, -1, 0, 0);
v35 = sub_40EDC6(2 * v34 | -(v34 >> 31 != 0));
MultiByteToWideChar(0, 0, &MultiByteStr, strlen(&MultiByteStr) + 1, v35, v34);
v36 = (&Type - v35);
do
{
  v37 = *v35;
  ++v35;
  *&v36[v35 - 2] = v37;
}
while ( v37 );
sub_4022C0(&Type, &v113);
memset(&MultiByteStr, 0, 0x200u);
memset(&Type, 0, 0x400u);
sub_406660();
if ( v38 )
  ExitProcess(1);
if ( sub_4065C0() )
  ExitProcess(1);
BlockInput(1);
v39 = sub_40E691(&v109);
v71 = 0x424448B;                                // mov eax, dword ptr [esp+4]
v40 = v39;
v72 = 0xC340u;                                  // inc eax , ret
v41 = 7;
do
{
```

그리고 아래쪽엔 위와 같은 코드가 있습니다. 인터넷 연결을 해서 뭔가 이것저것 하는데,

지금은 이게 뭔 의미였는지 기억이 잘 안나지만 그냥 무시하고 넘겼는지 실행시켰는지

기억이 안 납니다. 아무튼 어차피 무슨 랜덤한 게 오는 것도 아니고 고정이었던 걸로 기억하니

별 의미없는 부분이라고 보면 됩니다.

그렇게 의미없는 코드를 넘어서 앞으로 가면..

 Copyright @ 2015 CERT-IS

```
do
{
  v48 = VirtualAlloc(v66, v68);
  v49 = v48;
  LibFileName = 0;
  *v48 = 0x424448B;
  *(v48 + 2) = 0xC340u;
  memset(&v80, 0, 0x1FFu);
  ProcName = 0;
  memset(&v84, 0, 0x1FFu);
  sub_4077F0(
    "def11def84def91def51defd1def90def91def45def91def51def91defc4defd0defd0defd0def90def
    &LibFileName);
  sub_4077F0(
    "def51def91def91def52defd1def90defd1def11defd1def51def91def50def91defc4def51def10defc
    &ProcName);
  v50 = LoadLibraryA(&LibFileName);
  v51 = GetProcAddress(v50, &ProcName);
  if ( v51 )
    (v51)(v49, 4096, 16, &v71);
  value3 = (v49)(value3);
  v81 = 0;
  memset(&v82, 0, 0x1FFu);
  v77 = 0;
  memset(&v78, 0, 0x1FFu);
  sub_4077F0(
    "def11def84def91def51defd1def90def91def45def91def51def91defc4defd0defd0defd0def90def
    &v81);
  sub_4077F0(
    "def51def91def91def52defd1def90defd1def11defd1def51def91def50def91defc4def11def91defc
    &v77);
  v52 = LoadLibraryA(&v81);
  v53 = GetProcAddress(v52, &v77);
```

뭘 불러다가 이것저것 하려고 하는데 그냥 귀찮아서 동적으로 돌려보고 의미없다고 생각되서 무시했습니다. 사실 분석할 때는 그냥 잠깐 돌려보고 문제없으면 넘기고 문제있으면 우회하는 식으로 슥슥 했기 때문에 별 생각이 없습니다.

```
sub_4077F0(
  "def91def51def91def45def91defd0defd1def90defd1def52defd1def10defd1def11def91def51def91def11",// encrypted
  &MultiByteStr);
_sprintf(&v91, "%s.%s", &FileName, &MultiByteStr);
memset(&MultiByteStr, 0, 0x200u);
memset(&Type, 0, 0x400u);
sub_405D60(&v97, &v91, value7, value3);
DeleteFileA(&v97);
sub_4077F0("def91def05def91def50def91defd1defd1def10def91def52def91def51", &MultiByteStr);// magpie
v54 = MultiByteToWideChar(0, 0, &MultiByteStr, -1, 0, 0);
v55 = sub_40EDC6(2 * v54 | -(v54 >> 31 != 0));
MultiByteToWideChar(0, 0, &MultiByteStr, strlen(&MultiByteStr) + 1, v55, v54);
v56 = (&Type - v55);
do
{
  v57 = *v55;
  ++v55;
  *&v56[v55 - 2] = v57;
}
while ( v57 );
```

다음 아래쪽엔 이런 부분이 있습니다. 사실상 가장 중요한 부분이라고 봐도 됩니다.

저 405D60 함수가 실질적으로 핵심이 되는 함수입니다. 한 마디로 아까 살짝 암호화한 zip 파일에다가 또 암호화를 하는 함수입니다. 그 외에 아래쪽에는 또 리소스에서 하나 끄집어내서 html 파일을 만드는데, 이건 그냥 랜섬웨어에서 나오는 입금을 위한 안내 파일입니다. 무시하고 실질적으로는 여기서 작업이 다 끝났다고 보면 됩니다.

 Copyright @ 2015 CERT-IS

그럼 핵심인 405D60 을 보면..

```
do
{
  v11 = fread(&v22, 1u, 0x100u, v9);
  v12 = 0;
  v18 = v11;
  v19 = 0;
  if ( v11 > 0 )
  {
    do
    {
      v13 = *(&v22 + v12);
      v14 = 0x4F;
      v15 = 1;
      do
      {
        if ( v14 % 2 == 1 )
          v15 = v13 * v15 % value0x85;
        v14 >>= 1;
        v13 = v13 * v13 % value0x85;
      }
      while ( v14 > 0 );
      v10 = v17;
      v21 = v15;
      fwrite(&v21, 4, 1, v17);
      v11 = v18;
      v12 = v19 + 1;
      v19 = v12;
    }
    while ( v12 < v18 );
    v9 = v20;
  }
}
while ( v11 >= 256 );
```

여기서 value0x85 라든가 이런 건 아까 제가 의미 없다고 한 그 부분에서 만드는 값을 통해서 나온 것들인데, 고정이기 때문에 그냥 저렇게 써놔도 됩니다. 뭐 일단 암호화한 zip 파일을 읽어서, 여기에 저런 연산을 통해 또 다시 암호화를 하고, 뒤에 encrypted 를 붙여서 파일을 쓰게 됩니다.

지금 당장 저걸 분석하진 않고, 일단은 실제로 파일을 만들면 어떤 형태로 나오는지를 봐야 할 것이기 때문에 동적으로 쭉쭉 넘겨서 저기까지 가서 파일을 쓰도록 유도합니다.
그리고 %Temp% 디렉토리에 가서 한 번 가져와서 봅니다.

 Copyright @ 2015 CERT-IS

SeoulHot-EDD-262.jpg.encrypted

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	1E	00	00	00	42	00	00	00	23	00	00	00	31	00	00	00	....B...#...1...
00000010	2D	00	00	00	32	00	00	00	72	00	00	00	32	00	00	00	-...2...r...2...
00000020	22	00	00	00	32	00	00	00	17	00	00	00	63	00	00	00	"...2.......c...
00000030	29	00	00	00	26	00	00	00	0F	00	00	00	7A	00	00	00	)...&.......z...
00000040	64	00	00	00	47	00	00	00	1C	00	00	00	00	00	00	00	d...G...........
00000050	3E	00	00	00	32	00	00	00	5D	00	00	00	40	00	00	00	>...2...]...@...
00000060	3E	00	00	00	32	00	00	00	39	00	00	00	32	00	00	00	>...2...9...2...
00000070	32	00	00	00	32	00	00	00	67	00	00	00	39	00	00	00	2...2...g...9...
00000080	1B	00	00	00	15	00	00	00	76	00	00	00	09	00	00	00	........v.......
00000090	61	00	00	00	0A	00	00	00	6A	00	00	00	6B	00	00	00	a.......j...k...
000000A0	1B	00	00	00	0F	00	00	00	5A	00	00	00	5F	00	00	00	........Z..._...
000000B0	1E	00	00	00	1B	00	00	00	78	00	00	00	5B	00	00	00	........x...[...
000000C0	0A	00	00	00	42	00	00	00	27	00	00	00	43	00	00	00	....B...'...C...
000000D0	0A	00	00	00	25	00	00	00	23	00	00	00	5A	00	00	00	....%...#...Z...
000000E0	75	00	00	00	75	00	00	00	23	00	00	00	34	00	00	00	u...u...#...4...
000000F0	3A	00	00	00	34	00	00	00	74	00	00	00	2D	00	00	00	:...4...t...-...
00000100	26	00	00	00	59	00	00	00	3D	00	00	00	0E	00	00	00	&...Y...=.......
00000110	23	00	00	00	11	00	00	00	50	00	00	00	83	00	00	00	#.......P...ƒ...
00000120	09	00	00	00	44	00	00	00	10	00	00	00	49	00	00	00	....D.......I...
00000130	79	00	00	00	3F	00	00	00	0D	00	00	00	02	00	00	00	y...?...........
00000140	27	00	00	00	09	00	00	00	61	00	00	00	65	00	00	00	'.......a...e...
00000150	76	00	00	00	77	00	00	00	2B	00	00	00	0B	00	00	00	v...w...+.......

대략 위와 같은 형태입니다. 코드를 보면 알지만 0x85 로 mod 연산을 하고 4바이트씩 쓰므로

당연히 저렇게 나올 수 밖에 없을 것입니다. 그렇다면 이제 처음 주어진 암호화된 파일을 봅니다.

HelpMe.encrypted

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	02	0E	00	00	DE	03	00	00	84	24	00	00	85	33	00	00
00000010	68	44	00	00	C8	13	00	00	8D	2F	00	00	C8	13	00	00
00000020	9E	2B	00	00	C8	13	00	00	AB	49	00	00	BC	28	00	00
00000030	A0	43	00	00	45	33	00	00	43	0C	00	00	02	0E	00	00
00000040	74	45	00	00	AD	2B	00	00	18	58	00	00	8D	2F	00	00
00000050	C8	13	00	00	C8	13	00	00	8D	00	00	00	93	4C	00	00
00000060	C8	13	00	00	C8	13	00	00	EC	2C	00	00	C8	13	00	00
00000070	C8	13	00	00	C8	13	00	00	79	3B	00	00	1C	04	00	00
00000080	42	63	00	00	5A	66	00	00	3B	4A	00	00	05	1F	00	00
00000090	F7	06	00	00	5A	66	00	00	C6	1B	00	00	EC	2C	00	00
000000A0	40	16	00	00	69	33	00	00	59	2C	00	00	6B	32	00	00
000000B0	45	33	00	00	42	20	00	00	05	1F	00	00	F7	06	00	00
000000C0	69	33	00	00	AC	4A	00	00	C6	1B	00	00	09	4D	00	00
000000D0	42	63	00	00	C8	38	00	00	8F	3A	00	00	C1	0B	00	00

!? 0x85 는 커녕 전부 2바이트 값으로 저장이 되어 있습니다.

이 암호화 파일은 주어진 바이너리에선 나올 수 없는 파일입니다.

그러니 아마도 다른 바이너리에서 나왔을 것으로 생각할 수 있습니다.

그렇다면 이것을 풀려면 아마도 다를 저 mod 값을 먼저 구해야 합니다.

 Copyright @ 2015 CERT-IS

다행히도, 항상 zip 파일을 암호화하기 때문에 초기 zip 파일의 헤더가 항상 같으니

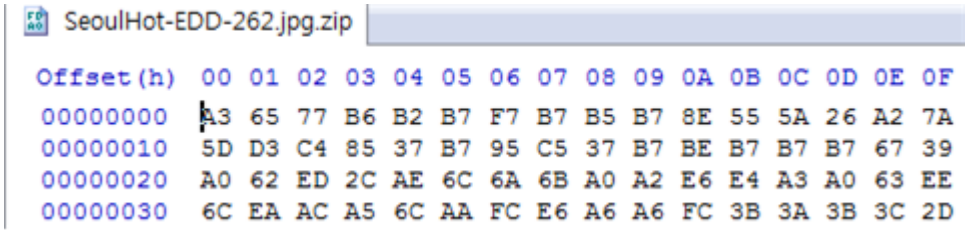
(굳이 플래그 부분까지 갈 필요도 없이 항상 같은 2~4바이트 시그니쳐만 같아도 충분합니다.)

이걸 이용해서 초기값을 추측해볼 수 있습니다.

또한 그냥 zip 파일이 아니라 여기서 xor/ror/not 이 되어 있는 것에 대해서 암호화를 하니

실제로는 이것으로 추측해야 합니다. 물론 어차피 역연산이 가능하고 항상 같은 결과가 나오니

상관없습니다.



암호화한 zip 파일은 이렇게 되어 있습니다 .즉 여기서 A3 65 77 B6 정도는 어떤 zip 파일을

압축해도 같을 것입니다. 1바이트씩 읽어서 4바이트를 출력하기 때문에 충분합니다.

방법은 간단합니다. 위의 코드에 있는 연산을 그대로 적용해서, HelpMe.encrypted 의

시작부분인 0x0E02 , 0x03DE 등이 나오는 mod 값을 찾으면 됩니다.

하나로는 중복된 결과가 나올 수 있으나 2~3개면 그 확률이 극히 적습니다.

그렇게 해서 코드를 돌리면, 0x6E27 로 mod 를 했다는 사실을 알 수 있습니다.

그럼 이제 남은 것은 간단합니다. 다행히도 0x00 ~ 0xFF 까지 저 연산을 적용하면

중복된 경우가 안 나오는 것으로 보입니다. 즉 A 라는 바이트에 대한 결과는 항상

B 가 나오고, A2 라는 바이트에 대해 B 가 나올 수는 없다는 얘기입니다.

(물론 확실한 것은 아니고, 추측입니다. 결과가 제대로 나왔으니 맞겠지만)

 Copyright @ 2015 CERT-IS

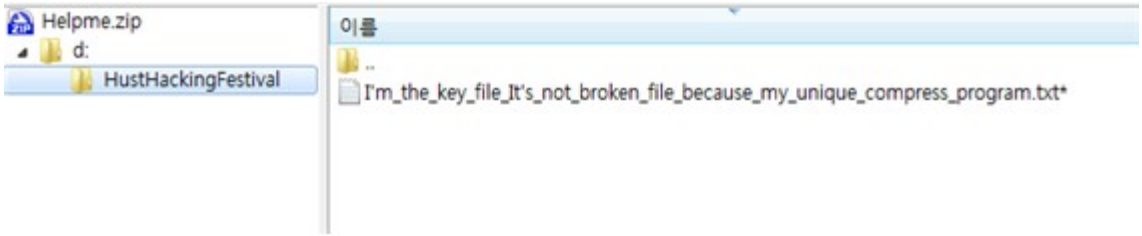
```
fp = fopen("HelpMe.encrypted","rb");
fseek(fp,0,SEEK_END);
ii = ftell(fp);
fseek(fp,0,SEEK_SET);
ii /= 4;
for( i = 0; i < ii; i++ )
    fread(&data[i],4,1,fp);
fclose(fp);

fp = fopen("a.txt","wb");
for( i = 0; i < ii; i++ ){
    for( c = 0; c < 0xff; c++ ){
        n = 0x4f;
        m = 1;
        unsigned int t = c;
        do{
            if( n % 2 == 1 )
            m = t * m % 0x6E27;
            n >>= 1;
            t = t * t % 0x6E27;
        }while( n > 0 );
        if( m == data[i] )
            fwrite(&c,1,1,fp);
    }
}
fclose(fp);
```

그러면 00 부터 FF 까지 하나씩 루프를 돌려서 HelpMe.encrypted 에 있는 결과가 나오는지 확인해서 1바이트씩 복원을 하면 됩니다. 코드는 아래와 같습니다.

특별할 것도 없고, 그냥 그대로 똑같이 작성했습니다.

이를 돌리면 짠 하고 정상적인 zip 파일이 나옵니다.

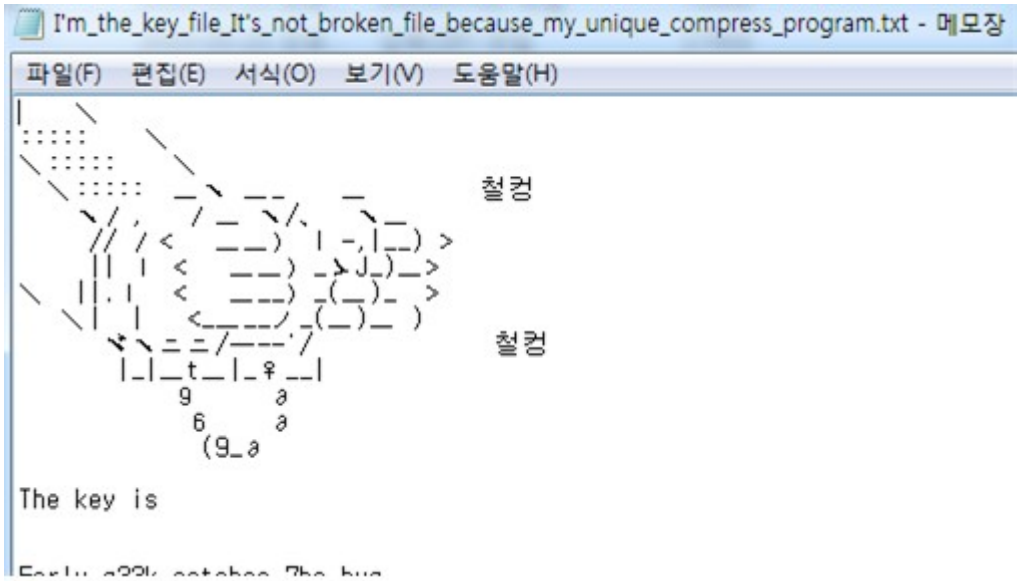


다 푼 줄 알았는데, 여기서 아까 위에서 찾은 비밀번호로 추측되는 문자열인 “ThisIsMyOwnSecretWayToUseTheData” 을 계속 집어넣어서 풀려고 하였으나 왜인지 절대로 풀리지 않았습니다. 아까 SeoulHot 에 대해서도 사실 사전에 테스트를 했으나 풀리지 않아서 그냥 이건 안 되는 거겠지 싶어서 넘어갔으나 여기서도 안 되어서 좌절했습니다.

그러나 뭔가 잘못 생각했겠지 하고 코드를 더 분석해도 전혀 나오는 것이 없습니다.

 Copyright @ 2015 CERT-IS

분명히 패스워드는 저게 맞아야 하고, 감으로 대충 봐도 저거 아니면 패스워드가 될 건 없습니다.



그렇게 오랜 시간 고민에 고민을 하고 분석하고 좌절하던 중

그냥 별 생각 없이 무의식적으로 반X집 이 아닌 7-zip 을 이용해서 풀어보았습니다.

(아아아아아아악)

…. 7-zip 으로 바로 풀려서 문제를 풀게 되었습니다.

이 원인은 귀찮아서 따로 생각하지 않았습니다. 반X집의 버그인지 모르겠습니다.

 Copyright @ 2015 CERT-IS

2. Web

1) Web 200

시작하면 개인이 운영하는 블로그가 하나 있다. 블로그 답게 자신이 쓴 글이 즐비하게 있고 login 을 할 수 있는 버튼이 있으며 운영자와 contact 할수 있는 공간도 마련되어 있었다. 지금은 서버가 닫혀있어 첨부할수 없지만, 로그인을 한 후 화면을 클릭하면 분홍색 네모가 하나씩 떨어지는걸 볼 수 있고 이렇게 클릭을 한번 할때마다 score가 오른다.

처음 이 문제를 접했을때 score를 조작하여 flag를 얻거나, admin계정의 score에 flag가 박여있을것으로 생각하고 수년간 단련한 오른손의 힘으로 재빨리 클릭을 했는데 이것이 brute force로 판명되어 IP ban을 당했다. 요청을 하여 ban을 풀고 기다리니 get_score 라는 힌트가 공개되었고, 조금 어리둥절 하던 마당에 login이 있으니 sql이 될수 있지 않냐 라고 하는 팀원의 말에 따라 우선 set_score 부분에서의 blind sql이 가능하다는걸 발견했다.

그렇게 id를 확인하니 root라는걸 알아내고 webshell을 올리는 문제라고 판단했다. 하지만 우리가 파일을 쓸 수 있는 디렉토리를 찾는데 문제가 발생하여 단순 SQL injection이 아니겠냐는 팀원의 말을 듣고 DB를 뒤져보기로 했다. 하지만 불안정한 서버의 상황때문에 blind sql로 테이블을 가져와서 확인한다는건 명백하게 무리가 있었다. 때문에 우리는 서버에서 제공하는 js파일을 받아 임의로 변조하여 score 대신 받아오는 json을 그대로 출력하게 만들어서 출력되는 문자를 점수 뿐 아닌 정보 그 자체를 출력하게 만들고 get_score를 통한 sql injection을 시도하였다. 다행히도 쿼리는 정상작동 하였고, keytable 이라는 테이블 밑의 value 라는 컬럼을 발견했다.

이를 통해 키 값을 얻어냈는데, 인증이 되지 않아서 팀원들과 고민한 끝에 가져온 값 중에 key_is 에 해당하는 문자열을 떼고 인증하니 정상적으로 인증이 되었다.

 Copyright @ 2015 CERT-IS

2) web 400

풀 수 있을 것이라고 기대하지 않았으나 운이 좋게 풀린 문제입니다.

대략 공을 움직여서 가운데 막혀 있는 열쇠에 가면 키를 주는 식의 문제입니다.

```
socket = io.connect('http://223.194.105.181:12009');

socket.on('conn', function(data){
    console.log("connected");
});

socket.on('login', function(data){
    var json=JSON.parse(data);

    isIngame = true;
    ingameInit(json);
});
socket.on('wall', function(data){
    var json=JSON.parse(data);
    wall = json;
});

socket.on('around_object_data', function(data){
    var json=JSON.parse(data);
    setAroundObjectData(json);
});
socket.on('around_object_data2', function(data){
    var json=JSON.parse(data);
    setAroundObjectData2(json);
});
socket.on('allFeed', function(data){
    var json=JSON.parse(data);
    makeFeed(json);
});
socket.on('keysRocation', function(data){
    var json=JSON.parse(data);
    makeKeys(json);
});


socket.on('moving', function(data){
    var json=JSON.parse(data);
});


socket.on('LoLHereIsKey', function(data){
        console.log(data);
});
```

웹소켓을 사용하고 있으며, 거의 모든 연산을 서버에서 처리한다는 특징이 있습니다.
즉 뭔가 좌표 조작 등으로 열쇠로 가는 것은 거의 불가능합니다.

메인 소스를 봐도 특별히 허점이 있어 보이지 않습니다.
그러나 딱 한 부분, 아래와 같은 부분이 있습니다.

 Copyright @ 2015 CERT-IS

```
/*      for(var key in feedObj){
            if(feedObj[key].beforePoint != feedObj[key].point){
                socket.emit("overlabFeed",feedObj[key]);
            }
        }
        */
```

여기에 주석이 된 부분이 있는데, overlabFeed 라는 앞에는 나오지 않던 커맨드를
서버로 전달하고 있습니다. 일단 이건 그냥 의심 정도로 넘어갔습니다.

```
window.makeFeed = function(json){
    for(var key in json){
        if(json[key] == null){
            if(typeof feedObj[key] !== 'undefined'){
                feedObj[key].deleteObj();
            }
            continue;
        }else if(!(key in feedObj)){
            var ObjectInfo = {
                socketId : key,
                point : json[key].point,
                beforePoint : json[key].point,
                range : json[key].range,
                score : json[key].score,
                color : json[key].color,
                color2 : json[key].color2
            };
            new Feed(ObjectInfo);
            for(var key2 in feedObj){
                var dx = Math.sqrt(Math.pow(feedObj[key2].point[0] - feedObj[key].point[0]) + Math.pow(feedObj[key2].point[1] - feedObj[key].point[1]));
                if(dx < feedObj[key2].range + feedObj[key].range){
                    socket.emit("overlabFeed",feedObj[key]);
                    break;
                }
            }
        }else if(key in feedObj){
            feedObj[key].point = json[key].point;
            feedObj[key].beforePoint = json[key].point;
            feedObj[key].range = json[key].range;
        }


    }
    for(var key in feedObj){
        if(!(key in json)){
            feedObj[key].deleteObj();
        }
    }
}
```

아래 쪽에 makeFeed (아마 음식(공이 먹는..)을 만드는 부분) 핸들러가 있습니다.
이 코드를 대충 보면, 음식(…) 을 만드는데 두 음식이 충돌했을 경우, 즉 거의 비슷한 자리에
만들어졌을 경우 overlabFeed 라는 커맨드를 서버로 보내고 feedObj[key] 를 넘깁니다.
저 feedObj[key] 는 그냥 충돌한 두 음식 중 하나입니다. 정확히는 새로 만든 음식입니다.

왜 overlap 이 아닌 overlab 으로 썼는지는 모르겠으나 아무튼 건덕지는 여기밖에 없어 보입니다.
여기서 생각할 수 있는 것은, 아마도 충돌이 일어났다면 하나의 좌표를 옆으로 약간 옮겨서
충돌을 없애든지 해서 처리할 가능성이 높다는 것입니다.

희한하게도 이 메인 코드에선 가운데 있는 열쇠도 음식의 하나로 치고 있습니다.
비록 Object 구조는 약간 다르지만 없는 것들은 그냥 억지로 넣어서 충분히 만들 수 있습니다.

즉 코드를 변조하는데, key 의 object 는 아래와 같습니다.

```
window.Keys = function(ObjectInfo) {
    this.socketId = ObjectInfo.socketId;
    this.point = ObjectInfo.point;
    this.range = ObjectInfo.range;
    this.img = new PIXI.Sprite(PIXI.Texture.fromIma
    this.img.anchor.x = 0.5;
    this.img.anchor.y = 0.5;
    this.img.scale.x = this.img.scale.y = 0.15;
    game_screen.addChild(this.img);
    feedObj[this.socketId] = this;
    KEYS = this;
}
Keys.prototype.constructor = Keys;
```

 Copyright @ 2015 CERT-IS

아까의 이미지에서 ObjectInfo 를 보면 음식의 Object 구조가 있는데
이거랑 약간 다릅니다. 하지만 핵심은 socketId 와 point , range 일 뿐 다른 건 그다지
중요해 보이진 않습니다. 그러니 그냥 beforePoint 같은 건 point 랑 같게 해주고
score, color 모두 적당히 음식 하나하고 똑같이 맞춰줍니다.

그렇게 해서 저 같은 경우는 저 충돌 체크하는 for 문을 없애버리고, 무조건
Key 에 대해 overlabFeed 를 계속 보내도록 하였습니다.

그렇게 하고 들어가보니, 갑자기 가운데 key 가 사라졌습니다.
좀 화면을 축소해서 보니 key 가 위아래 위위아래 계속 빠르게 움직이고 있습니다.
이거 같은 경우 실제 서버에서 key 의 좌표가 분명 바뀌고 있는 것이라고 확신해서
근처에 가서 키의 가운데에 어떻게든 데였습니다.

그렇게 하고 console log 를 살펴보니 정상적으로 키가 출력되어서 인증하였습니다.

3. Pwn

1) pwn 700

ssh 데몬이 열려있는 시스템이 하나 주어집니다. 안 쪽을 아무리 찾아봐도 뭔가 건덕지가 보이지 않습니다. 메모리

커럽션 류의 취약점이라 생각하기도 어렵습니다.

ifconfig 라는 힌트에서 알 수 있는 것은 이 컴퓨터가 아닌 다른 네트웍에 연결해서 무언가를 하라는 의미로 생각할 수 있습니다. 그리고 만약 그 네트웍이 외부에서 접근 가능한 것이면 의미가 없을 테니 아마도 해당 서버에서만 접속 가능한 내부 네트웍일 것입니다.
eth1 에 10.0.0.10 이라는 IP 를 할당해놓은 것을 볼 수 있습니다. 이 가상 네트웍에서 연결되어 있는 컴퓨터를 찾기 위해 arp 테이블을 확인해 봅니다. ARP 테이블에는 10.0.0.20 이라는 IP 만 유일하게 정상적인 맥 어드레스가 있습니다. 이 쪽에 열려있는 서비스 포트를 찾아봅니다.

여러 가지 방법이 있겠지만, 서버에 nmap 을 올려서 포트 스캐닝을 하는 방법 또는 잘 알려진 named 포트들을 nc 를 올려서 직접 연결 테스트를 해보는 방법 등이 있습니다. 이 경우, 어차피 이게 문제의 의도가 맞다면 보통 내부 네트웍의 ssh 나 기타 원격 접속프로토콜일 가능성이 크다고 추측할 수 있습니다. 물론 아니면 다른 것도 시도해보면 됩니다. 그래서 ssh, telnet, 그 외 파일 공유 프로토콜이나 vnc 등 다양하게 시도를 해 봅니다. 그러면 5900 번 포트에서 RFB 관련 메시지가 오는 것을 확인할 수 있습니다. VNC 서버가 열려있는 것으로 생각할 수 있고 당연히 여기에 접속을 해야 합니다. 그러나 이는 그래픽 모드이기 때문에 콘솔에서 그대로 접속하는 것은 어렵습니다. 따라서 프록시처럼 외부로 포트를 빼낸 다음 로컬에서 직접 VNC Viewer 로 봐야 합니다.

시스템에서 아래의 명령어를 이용합니다. nc 를 올려놨으니 방법은 쉽습니다.

```
mkfifo abc
nc –l 7777 < abc | nc 10.0.0.20 5900 > abc
```

fifo 를 통해 IPC 를 하도록 하여 10.0.0.20:5900 연결을 7777 포트로 바인딩합니다. 다음 VNC Viewer (저는 대충 RealVNC 를 썼습니다) 로 Hust 서버 IP:7777 로 접속합니다. 비밀번호에 대한 정보가 없으므로 게싱으로 1234 을 추측하여 접속합니다. 그러면 바로 Windows 화면이 뜨고 Key 이미지가 떠있는 것을 확인할 수 있습니다.

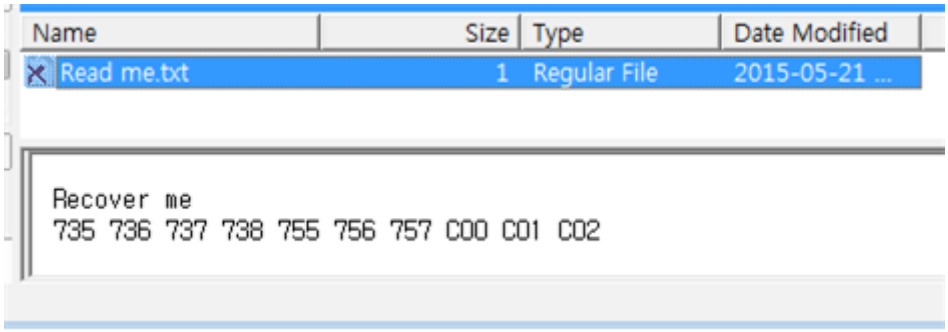
 Copyright @ 2015 CERT-IS

4. ETC

1) Forensic

```
blackcon@bk:~$ file for_a7058c13
for_a7058c13: x86 boot sector
```

<문제파일>



FTK imager 툴을 이용하여 문제의 파일을 넣어보면 "Read me.txt"파일을 볼 수 있다. 파일 내용에적힌 숫자의 의미는 n번째 클러스터를 나타낸다. 또한 문제의 파일시스템은 FAT32인데 해당 파일시스템의 헤더와 비교를 하면 클러스트의 위치와 사이즈를 알 수 있다. (위치 : 첫 디렉토리 - 0x1000, 클러스터 크기 : 0x800)

따라서 0x800 사이즈만큼의 데이터를 분할하여 Coving한 후 Recover me에 적힌 n번째 데이터를 합쳐주면 된다.

```
import time
fd = open( "for_a7058c13", "rb" )
data = fd.read()
fd.close()
data = data[data.find( "TEST       " )-0x1000:]   # first directory in FAT32
routine = len( data ) / 0x800
for i in range( 0xc04):
        data2 = data[ 0x800*i : 0x800*i+0x800]
        fname = "%0003x.dat" % i
        print "[+] create %s" % fname
        fd = open( fname, "wb" )
        fd.write( data2 )
        fd.close()
        time.sleep(0.1)

print "[+] done"

a = "735", "736", "737", "738", "755", "756", "757", "c00", "c01", "c02"
data = ''
```

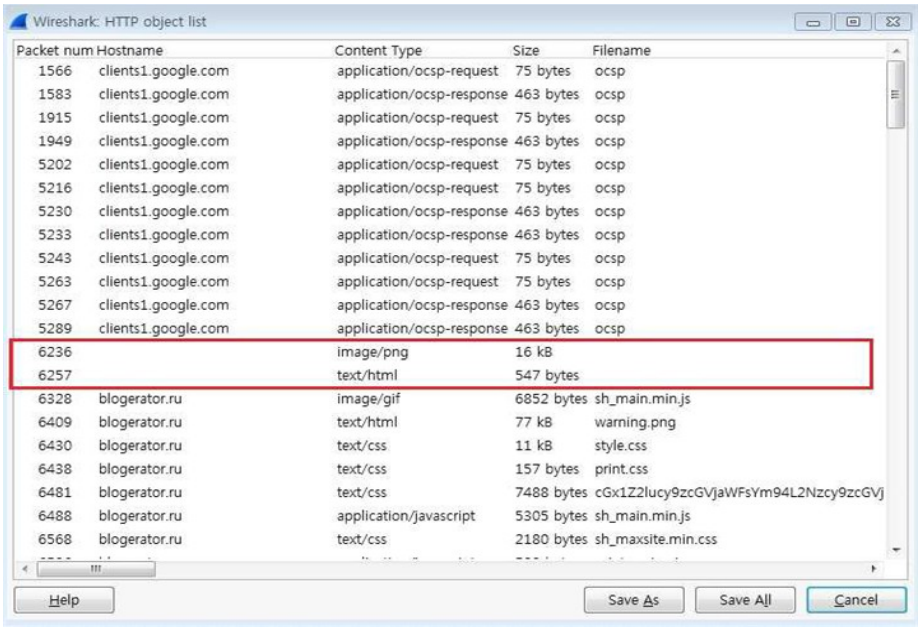
 Copyright @ 2015 CERT-IS

```
for i in a:
        fname = i + ".dat"
        fd = open( fname, "rb" )
        data += fd.read()
        fd.close()

fd = open( "flag.png", "wb" )
fd.write( data )
fd.close()
```

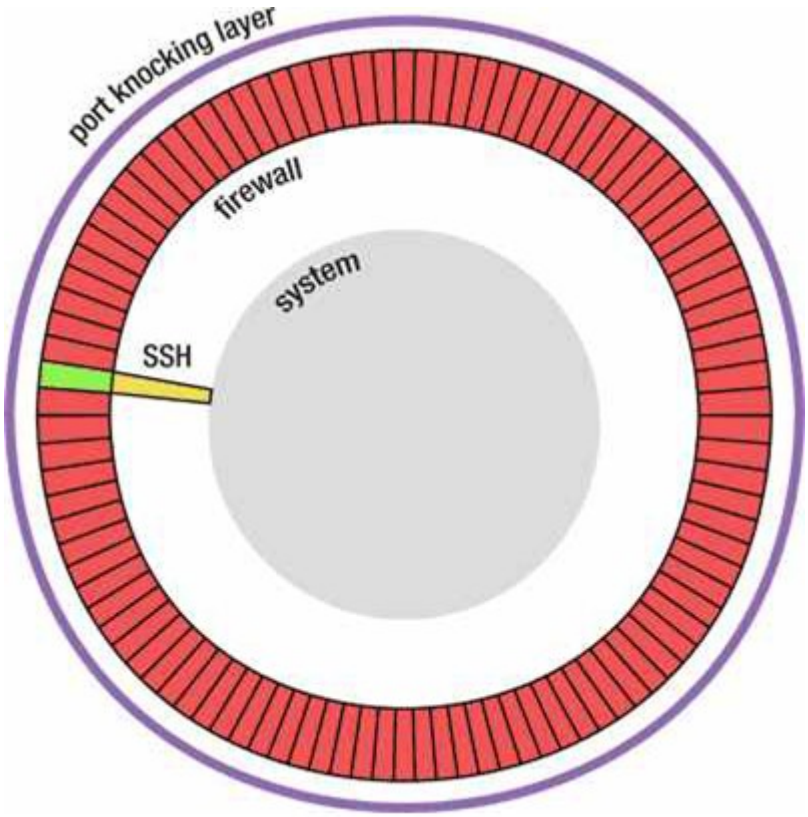
<flag.png>

2) Network

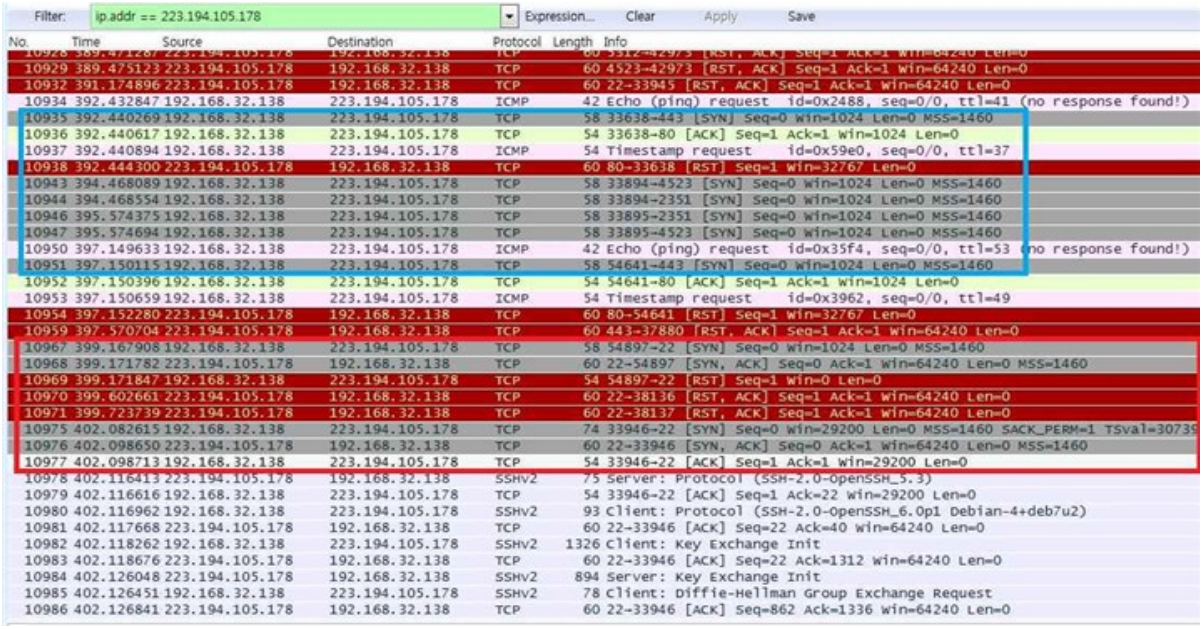


 Copyright @ 2015 CERT-IS

이미지의 빨간색 박스가 이번 문제의 키워드이다. 파일 열어보면 knock에 대해 설명이 되어있는 홈페이지 한 개와 아래의 그림이 주어진다.



이 두 개를 통해 리눅스의 knockd를 이용하여 풀어야 된다는 힌트를 얻었다. 패킷들을 보면 첫 부분빼고는 일반적으로 인터넷서핑을 하면서 여러 사진파일을 받는 그냥 평범한 패킷들만 있다. 앞 부분에 ssh에 접속을 하는데 그 주소로 필터링을 해서 보면 처음에 랜덤한 포트 여러곳으로 몇천개의 SYN패킷을 보낸다. 이 SYN패킷이 노크라는 거라 생각하고, 그 중에서 ssh접속하기 직전에 몇 번 포트에 SYN패킷을 보냈는지 확인을 했다.

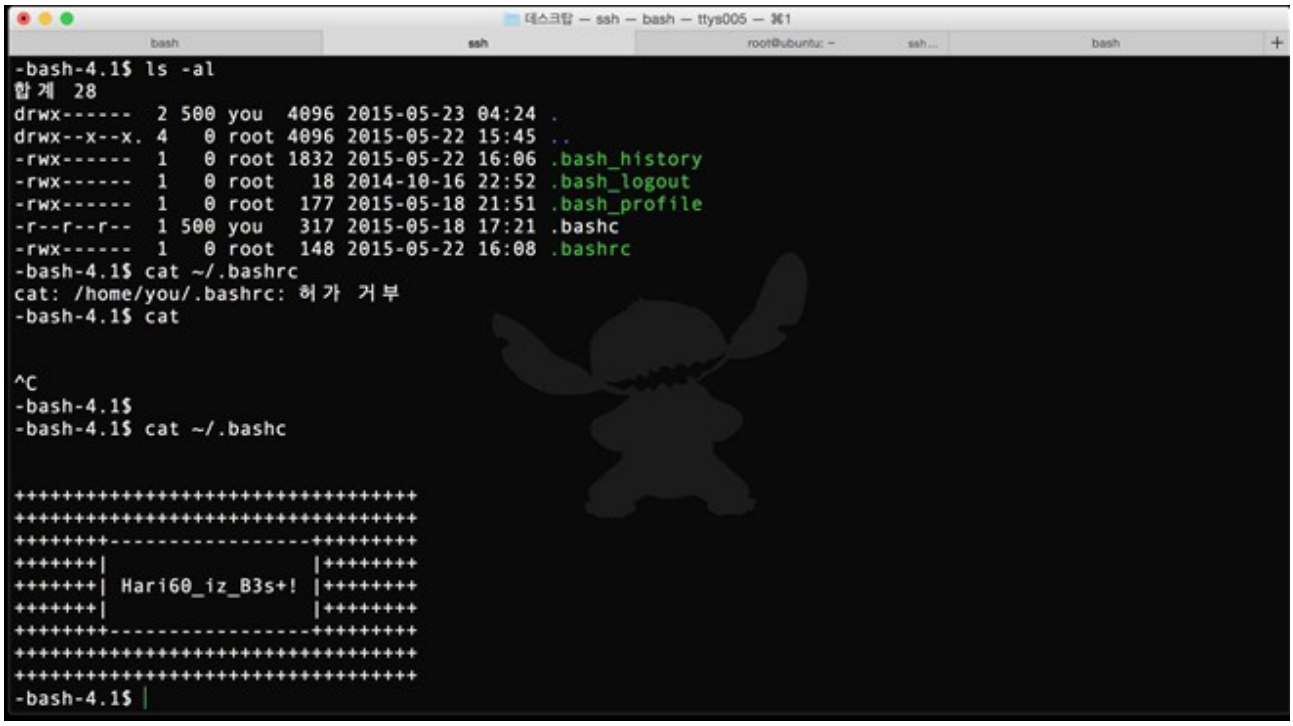

Filter: ip.addr == 223.194.105.178 Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
10929	389.475123	223.194.105.178	192.168.32.138	TCP	60	4523→42973 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
10932	391.174896	223.194.105.178	192.168.32.138	TCP	60	22→33945 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
10934	392.432847	192.168.32.138	223.194.105.178	ICMP	42	Echo (ping) request id=0x2488, seq=0/0, ttl=41 (no response found!)
10935	392.440269	192.168.32.138	223.194.105.178	TCP	58	33638→443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10936	392.440617	192.168.32.138	223.194.105.178	TCP	54	33638→80 [ACK] Seq=1 Ack=1 Win=1024 Len=0
10937	392.440894	192.168.32.138	223.194.105.178	ICMP	54	Timestamp request id=0x59e0, seq=0/0, ttl=37
10938	392.444300	223.194.105.178	192.168.32.138	TCP	60	80→33638 [RST] Seq=1 Win=32767 Len=0
10943	394.468089	192.168.32.138	223.194.105.178	TCP	58	33894→4523 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10944	394.468554	192.168.32.138	223.194.105.178	TCP	58	33894→2351 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10946	395.574375	192.168.32.138	223.194.105.178	TCP	58	33895→2351 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10947	395.574694	192.168.32.138	223.194.105.178	TCP	58	33895→4523 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10950	397.149633	192.168.32.138	223.194.105.178	ICMP	42	Echo (ping) request id=0x35f4, seq=0/0, ttl=53 (no response found!)
10951	397.150115	192.168.32.138	223.194.105.178	TCP	58	54641→443 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10952	397.150396	192.168.32.138	223.194.105.178	TCP	54	54641→80 [ACK] Seq=1 Ack=1 Win=1024 Len=0
10953	397.150659	192.168.32.138	223.194.105.178	ICMP	54	Timestamp request id=0x3962, seq=0/0, ttl=49
10954	397.152280	223.194.105.178	192.168.32.138	TCP	60	80→54641 [RST] Seq=1 Win=32767 Len=0
10959	397.570704	223.194.105.178	192.168.32.138	TCP	60	443→37880 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
10967	399.167908	192.168.32.138	223.194.105.178	TCP	58	54897→22 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
10968	399.171782	223.194.105.178	192.168.32.138	TCP	60	22→54897 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
10969	399.171847	192.168.32.138	223.194.105.178	TCP	54	54897→22 [RST] Seq=1 Win=0 Len=0
10970	399.602661	223.194.105.178	192.168.32.138	TCP	60	22→38136 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
10971	399.723739	223.194.105.178	192.168.32.138	TCP	60	22→38137 [RST, ACK] Seq=1 Ack=1 Win=64240 Len=0
10975	402.082615	192.168.32.138	223.194.105.178	TCP	74	33946→22 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=30739
10976	402.098650	223.194.105.178	192.168.32.138	TCP	60	22→33946 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
10977	402.098713	192.168.32.138	223.194.105.178	TCP	54	33946→22 [ACK] Seq=1 Ack=1 Win=29200 Len=0
10978	402.116413	223.194.105.178	192.168.32.138	SSHv2	75	Server: Protocol (SSH-2.0-OpenSSH_5.3)
10979	402.116616	192.168.32.138	223.194.105.178	TCP	54	33946→22 [ACK] Seq=1 Ack=22 Win=29200 Len=0
10980	402.116962	192.168.32.138	223.194.105.178	SSHv2	93	Client: Protocol (SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2)
10981	402.117668	223.194.105.178	192.168.32.138	TCP	60	22→33946 [ACK] Seq=22 Ack=40 Win=64240 Len=0
10982	402.118262	192.168.32.138	223.194.105.178	SSHv2	1326	Client: Key Exchange Init
10983	402.118676	223.194.105.178	192.168.32.138	TCP	60	22→33946 [ACK] Seq=22 Ack=1312 Win=64240 Len=0
10984	402.126048	223.194.105.178	192.168.32.138	SSHv2	894	Server: Key Exchange Init
10985	402.126451	192.168.32.138	223.194.105.178	SSHv2	78	Client: Diffie-Hellman Group Exchange Request
10986	402.126841	223.194.105.178	192.168.32.138	TCP	60	22→33946 [ACK] Seq=862 Ack=1336 Win=64240 Len=0

빨간색 네모가 ssh와 핸드셰이킹을 하는 부분이고, 이후에는 접속이 되는 것을 볼 수 있다. 그래서 직전에 보낸 SYN패킷을 보면 knock포트를 확인할 수 있을거라 생각했다. 파란색 네모부분을 보면 송신포트가 같은 부분끼리 묶으

 Copyright @ 2015 CERT-IS

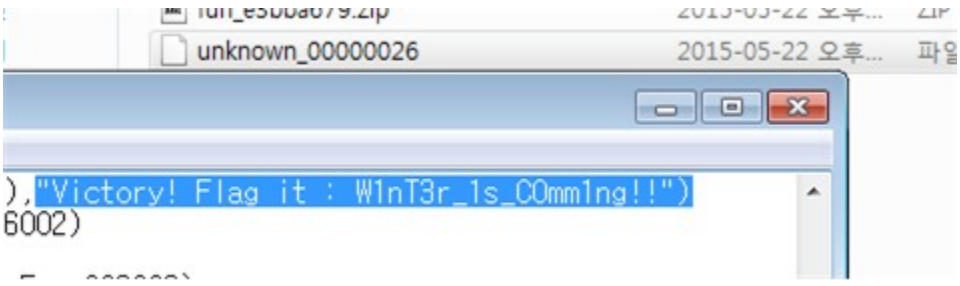
면 3개의 경우의 수가 생긴다. 443, 4523-2351, 2351-4523 이 3순서 중에 한 개를 knock하면 포트가 열릴거 같은 느낌같은느낌이 들었다. 4523,2351을 knock했을 때 22번 포트가 열려서 들어 갈수 있었고 홈디렉토리에서 키 값을 찾을 수 있었다.



5. Fun

1) w3x

문제파일로는 와우의 맵이 주어진다. 이 파일과 인터넷에 돌아다니는 원본 파일을 맵제작 프로그램을 이용하여 내부를 비교했다. 비교한 결과 **“unknown_00000026”**에서 차이가 있는 것을 발견했으 이 며파일을 메모장으로 열어서 확인한 결과 flag를 확인할 수 있었다.



<flag>

 Copyright @ 2015 CERT-IS