

DEFCON CTF 2009

Binary Leetness 100-500 Solutions

박 찬암 (hkpc)

chanam.park@hkpc.kr

<http://hkpc.kr/>

2009. 7. 4

Contents

1. DEFCON Capture The Flag의 소개와 변화
2. Binary Leetness 100
3. Binary Leetness 200
4. Binary Leetness 300
5. Binary Leetness 400
6. Binary Leetness 500
7. +@, funny pwn100
8. Thanks

Capture The Flag의 소개와 변화

- 매년 라스베가스에서 개최되는 DEFCON의 한 행사로, 상대방의 시스템을 공격하고 자신의 시스템을 방어하는 해킹대회
- CTF라고도 불린다
- 4년마다 운영 단체가 바뀌며 작년까지는 kenshoto, 올해부터는 Diutinus Defense Technologies Corp(ddtek or DDTech)
- 6월경에 예선을 거치며, 상위 10개팀(작년 우승팀 포함)이 본선에 진출
본선은 8월 초 전후를 기점으로 매년 라스베가스에서 개최된다

Capture The Flag의 소개와 변화

- 예선 문제는 빙고판에 공개
- 각 분야별로 100점부터 500점까지 5개의 문제가 제공됨

- 초록색 빙고칸은 현재 풀 문제
- 보라색 빙고칸은 우리가 현재 선택한 문제
- 파란색 빙고칸은 아직 풀지 못한 문제
- 황토색 빙고칸은 어떠한 팀도 풀지 못한 문제
- 회색은 아직 공개되지 않은 문제

- 공개된 문제를 한 팀 이상이 풀어야 그 다음 문제를 추가적으로 공개
- 만약, 아무도 문제를 풀지 못하면 다음 문제가 공개되지 않음

Capture The Flag의 소개와 변화

WOWHACKER Score: 1200

Logout

| Pursuits Trivial | Crypto Badness | Packet Madness | Binary L33tness | Pwntent Pwnables | Forensics |
|---------------------|-------------------|-------------------|--------------------|---------------------|-----------|
| 100 | 100 | 100 | 100 | 100 | 100 |
| 200 | 200 | 200 | 200 | 200 | 200 |
| 300 | 300 | 300 | 300 | 300 | 300 |
| 400 | 400 | 400 | 400 | 400 | 400 |
| 500 | 500 | 500 | 500 | 500 | 500 |

Name the algorithm implemented in this [file](#). Note that several smaller algorithms may be utilized in implementing the overall solution. The answer is the overall algorithm.

CORRECT!

Leaders

1. VedaGodz (1600)
2. WOWHACKER (1200)
3. sk3wlm4st3r (1100)
4. The B Squad (1100)
5. Shellphish (1100)
6. Sexy Pwndas (1100)
7. Taekwon-V (1100)
8. Routards (1000)
9. sutegoma (1000)
10. CLIP (1000)
11. G0t r00t (1000)
12. clgt (800)
13. Razer (800)
14. pwnrthrst (800)
15. Song of Freedom (800)

Capture The Flag의 소개와 변화

< 예선 문제 형식 >

- Binary, Pwn, Forensics, Trivia, Web
- Binary, Pwn, Forensics, Trivia, Realworld
- Binary, Pwn, Forensics, Trivia, Crypto, Packet

* 전체적인 시스템이나 문제 형식 등은 크게 바뀌지 않고 유지됨

Capture The Flag의 소개와 변화

Binary

주어진 바이너리를 분석하여 문제를 푸는 분야

Pwn

주어진 바이너리 분석 뒤, 리모트 공격을 수행하여 문제를 푸는 분야

Forensics

포렌식 분야

Trivia

문화적 혹은 넌센스적인 문제를 푸는 분야. 하지만, 올해부터는 조금 바뀌었음

Capture The Flag의 소개와 변화

Web

웹 상의 환경을 공격하여 문제를 푸는 분야. 2007년을 마지막으로 사라짐.

Realworld

상대적으로 좀 더 실제환경과 유사한 컨셉으로 구성되어진 분야.
2008년을 마지막으로 사라짐.

Crypto

암호학적 지식을 문제와 결부시켜 풀이를 해나가는 분야. 올해부터 생겨남.

Packet

제공되는 패킷 등을 바탕으로 분석을 통해 문제를 푸는 분야. 올해부터 생겨남.

Binary Leetness 100

What is the password?

문제를 위한 바이너리가 주어졌으며, 분석을 통하여 패스워드를 찾아야 함

➤ Windows

- IDA, Ollydbg

➤ Unix-like System

- GDB, Objdump

Binary Leetness 100

- file 명령을 통한 기본 조사

```
[hkpc@servera122 CTF]$ file bin100
```

```
bin100: ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux),  
statically linked, stripped
```

- ELF 파일포맷의 Linux 바이너리이며
- 정적 컴파일(static compile) 하였고
- Stripped(디버깅 정보가 삭제) 되어있다.

Binary Leetness 100

- strings 명령을 통한 추가 정보 수집

.....

/X3H

GCC: (Debian 4.3

u_l0_stdin_used\$

checklong_real

h_im

.....

\$Info: This file is packed with the UPX executable packer <http://upx.sf.net> \$

\$Id: UPX 3.01 Copyright (C) 1996-2007 the UPX Team. All Rights Reserved. \$

PROT_EXEC|PROT_WRITE failed.

/proc/se

lf/exe

Binary Leetness 100

```
hkpc@cs408-Server:~$ ./upx -d bin100
```

Ultimate Packer for eXecutables

Copyright (C) 1996 - 2008

UPX 3.03
2008

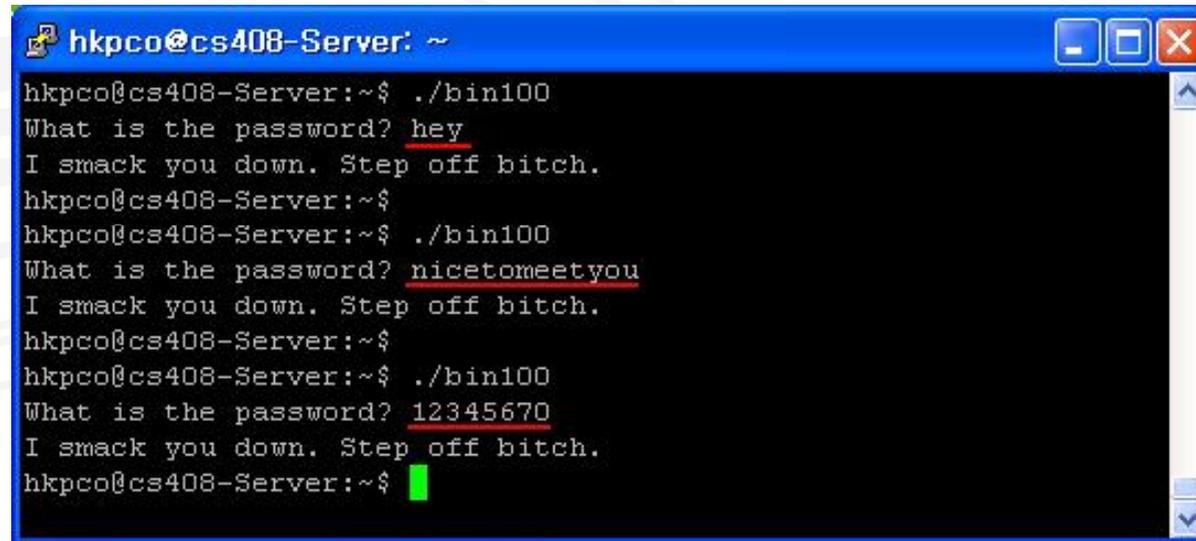
Markus Oberhumer, Laszlo Molnar & John Reiser Apr 27th

| File size | Ratio | Format | Name |
|-----------|-------|--------|------|
|-----------|-------|--------|------|

```
upx: bin100: Exception: checksum error
```

```
Unpacked 1 file: 0 ok, 1 error.
```

Binary Leetness 100



```
hkpc@cs408-Server: ~  
hkpc@cs408-Server:~$ ./bin100  
What is the password? hey  
I smack you down. Step off bitch.  
hkpc@cs408-Server:~$  
hkpc@cs408-Server:~$ ./bin100  
What is the password? nicetomeetyou  
I smack you down. Step off bitch.  
hkpc@cs408-Server:~$  
hkpc@cs408-Server:~$ ./bin100  
What is the password? 12345670  
I smack you down. Step off bitch.  
hkpc@cs408-Server:~$ █
```

- 모든 패킹, 코드 암호화 등은 실행 과정에서 정상적인 형태를 취해야 함
- 그러므로, bin100 바이너리 실행 시 우선적으로 패킹이 풀릴 것이며
- 입력을 받는 과정에서 이미 메모리에 해당 패스워드가 존재할 것이라 예상

Binary Leetness 100

A terminal window with a blue title bar. The title bar text is "hkpc@cs408-Server: ~". The terminal content shows the command "./bin100" being executed, followed by the prompt "What is the password?" and a green cursor. The window has standard Linux window controls (minimize, maximize, close) in the top right corner and a vertical scrollbar on the right side.

```
hkpc@cs408-Server: ~  
hkpc@cs408-Server:~$ ./bin100  
What is the password? █
```

Binary Leetness 100

```
hkpc@cs408-Server: ~  
hkpc@cs408-Server:~$ ps -aux|grep bin100  
Warning: bad ps syntax, perhaps a bogus '-'? See http://procps.sf.net/faq.html  
hkpc 15665 0.0 0.0 2108 488 pts/2 S+ 19:17 0:00 ./bin100  
hkpc 15744 0.0 0.1 3364 808 pts/3 S+ 19:18 0:00 grep bin100  
hkpc@cs408-Server:~$ gdb -q bin100 15665  
  
warning: no loadable sections found in added symbol-file /home/hkpc/bin100  
(no debugging symbols found)  
Attaching to program: /home/hkpc/bin100, process 15665  
(no debugging symbols found)  
0xb7fcc430 in __kernel_vsyscall ()  
(gdb) x/s 0x804c73b  
0x804c73b: ""  
(gdb)  
0x804c73c: "\001"  
(gdb)  
0x804c73e: "\002"  
(gdb)  
0x804c740: "What is the password? "  
(gdb)  
0x804c757: "visilooksgoodinhotpants"  
(gdb)  
0x804c76f: ", %d byte packets"  
(gdb)  
0x804c781: "CWR ECN "  
(gdb)  
0x804c78a: "ACK "  
(gdb)  
0x804c78f: "SYN "  
(gdb)  
0x804c794: "Destination not reached\n"  
(gdb)  
0x804c7ad: "Bad destination address: %s\n"  
(gdb) █
```

Binary Leetness 200

Name the algorithm implemented in this file. Note that several smaller algorithms may be utilized in implementing the overall solution.

The answer is the overall algorithm.

- 주어진 바이너리를 분석하여 알고리즘의 이름을 맞추는 문제

< 3가지 방법 >

- Brute force
- Signature
- Analysis

Binary Leetness 200

▪ Bruteforce

- 문제 출제자의 원래 의도가 바이너리 파일을 정식으로 분석하여 해당 알고리즘을 도출하는 것이라면, 굉장히 생소한 알고리즘을 문제로 내지는 않았을 것이기 때문에 대표적인 알고리즘 리스트를 이용하여 무차별 대입

▪ Signature

- 각 알고리즘마다 루프 횟수, 할당 크기, 주 사용 자료구조 등 구분 지을만한 특정한 시그니처가 있을 것이기 때문에 이를 이용하여 구글 등의 검색으로 도출

▪ Analysis

- 바이너리를 정식으로 분석하여 해당 알고리즘을 찾아냄

Binary Leetness 200

Huffman coding

In computer science and information theory, **Huffman coding is an entropy encoding algorithm used for lossless data compression.** The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. It was developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

From Wikipedia, the free encyclopedia

Binary Leetness 200

< Huffman Code >

- 무손실 데이터 압축 알고리즘
- 데이터에 특성에 따라 20 ~ 90% 까지 절약 가능
- 문자의 빈도 수에 따라 그 크기가 달라지는 *가변 길이 코드(variable-length code)*를 사용하여 기존 *고정 길이 코드(fixed-length code)*를 절약

Binary Leetness 200

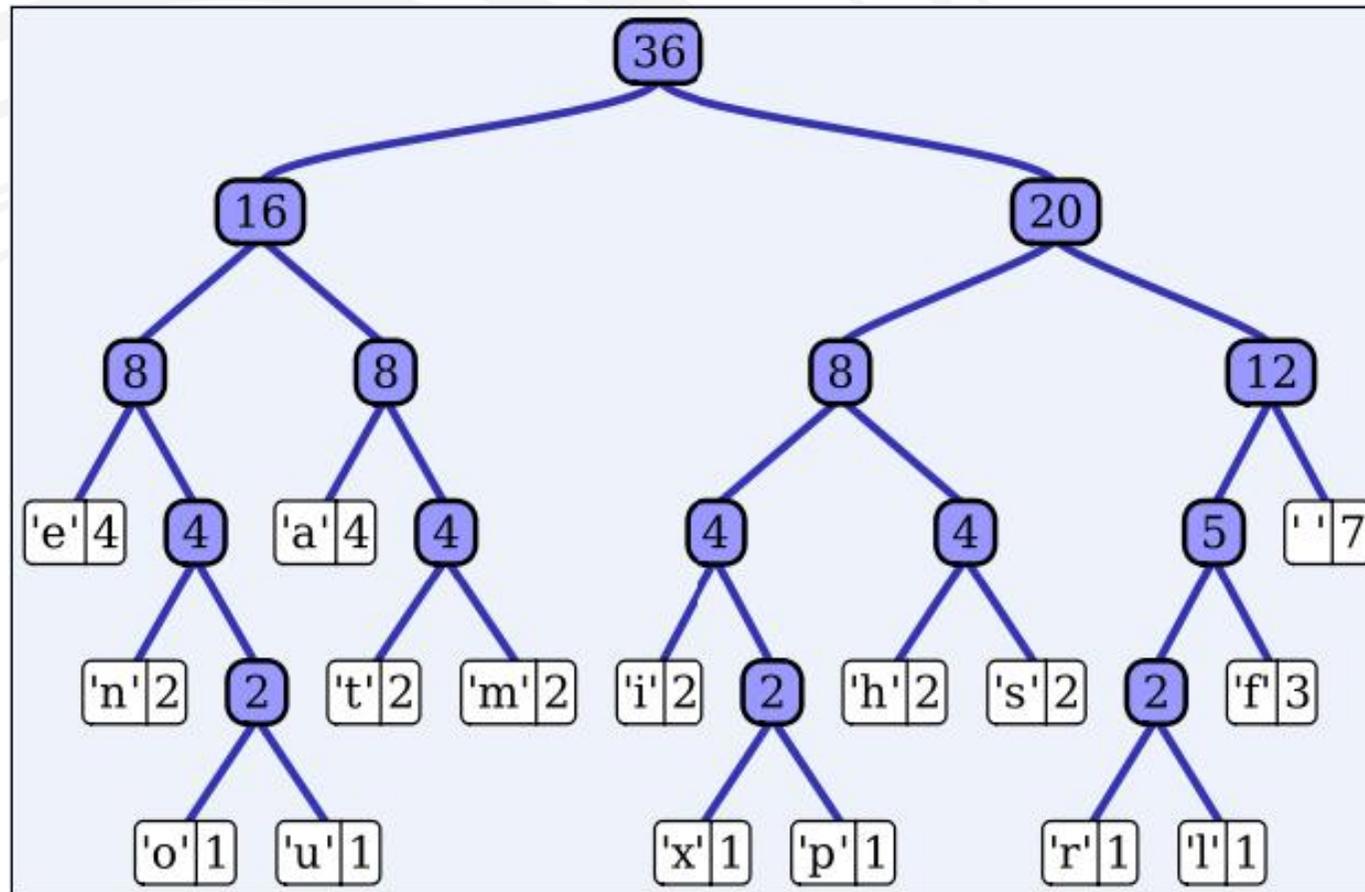
- 가변 길이 코드(variable-length code) 테이블

| | a | b | c | d | e | f |
|----------|------|------|------|------|------|------|
| 빈도수 | 4500 | 1300 | 1200 | 1600 | 900 | 500 |
| 고정 길이 코드 | 000 | 001 | 010 | 011 | 100 | 101 |
| 가변길이 코드 | 0 | 101 | 100 | 111 | 1101 | 1100 |

- 고정 길이 코드: $(4500 + 1300 + 1200 + 1600 + 900 + 500) * 3 = 300,000$
- 가변 길이 코드: $(4500*1 + 1300*3 + 1200*3 + 1600*3 + 900*4 + 500*4) = 224,000$
(약 25% 절약)

Binary Leetness 200

< Huffman Tree >



Binary Leetness 200

< 간단한 과정 >

- 1. 대상 파일의 모든 문자들의 빈도수를 구한다.
- 2. 빈도 수에 따라 코드의 길이를 재지정한다.
 - 이 과정에서 **tree**가 사용됨
 - 트리는 **linked-list**를 통하여 구현

Binary Leetness 200

< 간단한 과정 >

- 1. 대상 파일의 모든 문자들의 빈도수를 구한다.
- 2. 빈도 수에 따라 코드의 길이를 재지정한다.
 - 이 과정에서 tree가 사용됨
 - 트리는 linked-list를 통하여 구현

Binary Leetness 200

```
push    ebp
mov     ebp, esp
push    edi
push    esi
sub     esp, 460h
lea    eax, [ebp+intbuf_400h]
mov     [ebp+var_C], 0
mov     [ebp+p_intbuf], eax
mov     dword ptr [esp+8], 400h
mov     dword ptr [esp+4], 0
mov     [esp], eax
call   _memset
mov     eax, [ebp+arg_4]
test   eax, eax
jz     short loc_1CB6 ; 초기화 루틴
```

```
N ㄴ
xor     edx, edx
xor     eax, eax
nop    dword ptr [eax+00000000h]
nop    dword ptr [eax+eax+00000000h]
```

```
N ㄴ
loc_1CA0:
mov     ecx, [ebp+arg_0]
inc     edx
movsx   eax, byte ptr [ecx+eax]
inc     [ebp+eax*4+intbuf_400h]
mov     eax, edx
cmp     [ebp+arg_4], edx
jnz    short loc_1CA0
```

Binary Leetness 200

```
text:00001C8D          xor    edx, edx
```

```
text:00001C8F          xor    eax, eax
```

; 레지스터 초기화

```
text:00001CA0 loc_1CA0:
```

```
text:00001CA0          mov    ecx, [ebp+arg_0]
```

; 대상 base address 저장

```
text:00001CA3          inc    edx
```

; 루프를 위한 edx 레지스터 증가

Binary Leetness 200

```
text:00001CA4          movsx  eax, byte ptr [ecx+eax]
```

; base address에서 루프 카운터 eax 만큼 더한 offset을 eax에 저장

```
text:00001CA8          inc   [ebp+eax*4+intbuf_400h]
```

; 대상 문자에 해당하는 변수공간의 카운터 1 증가

*; 카운터를 저장하는 변수가 4byte이기 때문에 *4*

```
/*
```

intbuf_400h는 4byte 변수 100h(256)개가 저장되어 있는데,

이는 ASCII 코드의 문자 집합이 총 $2^8(256)$ 개이기 때문

```
*/
```

Binary Leetness 200

```
text:00001CAF          mov     eax, edx
```

; 카운터 값을 eax에 저장

```
text:00001CB1          cmp     [ebp+arg_4], edx
```

```
text:00001CB4          jnz    short loc_1CA0
```

; 함수 호출 당시 지정된 길이와 비교하여 같지 않다면 루프 재수행

! 결국, intbuf_400h 공간에 전체 문자의 카운터 값이 저장됨

Binary Leetness 200

< 간단한 과정 >

- 1. 대상 파일의 모든 문자들의 빈도수를 구한다.
- 2. 빈도 수에 따라 코드의 길이를 재지정한다.
 - 이 과정에서 **tree**가 사용됨
 - 트리는 **linked-list**를 통하여 구현

Binary Leetness 200

- Huffman Coding에 사용되는 구조체

```
typedef struct _huf
{
    long count;    // 빈도수
    int data;      // 문자
    struct _huf *left, *right
} huf;
```

-> 총 16byte

Binary Leetness 200

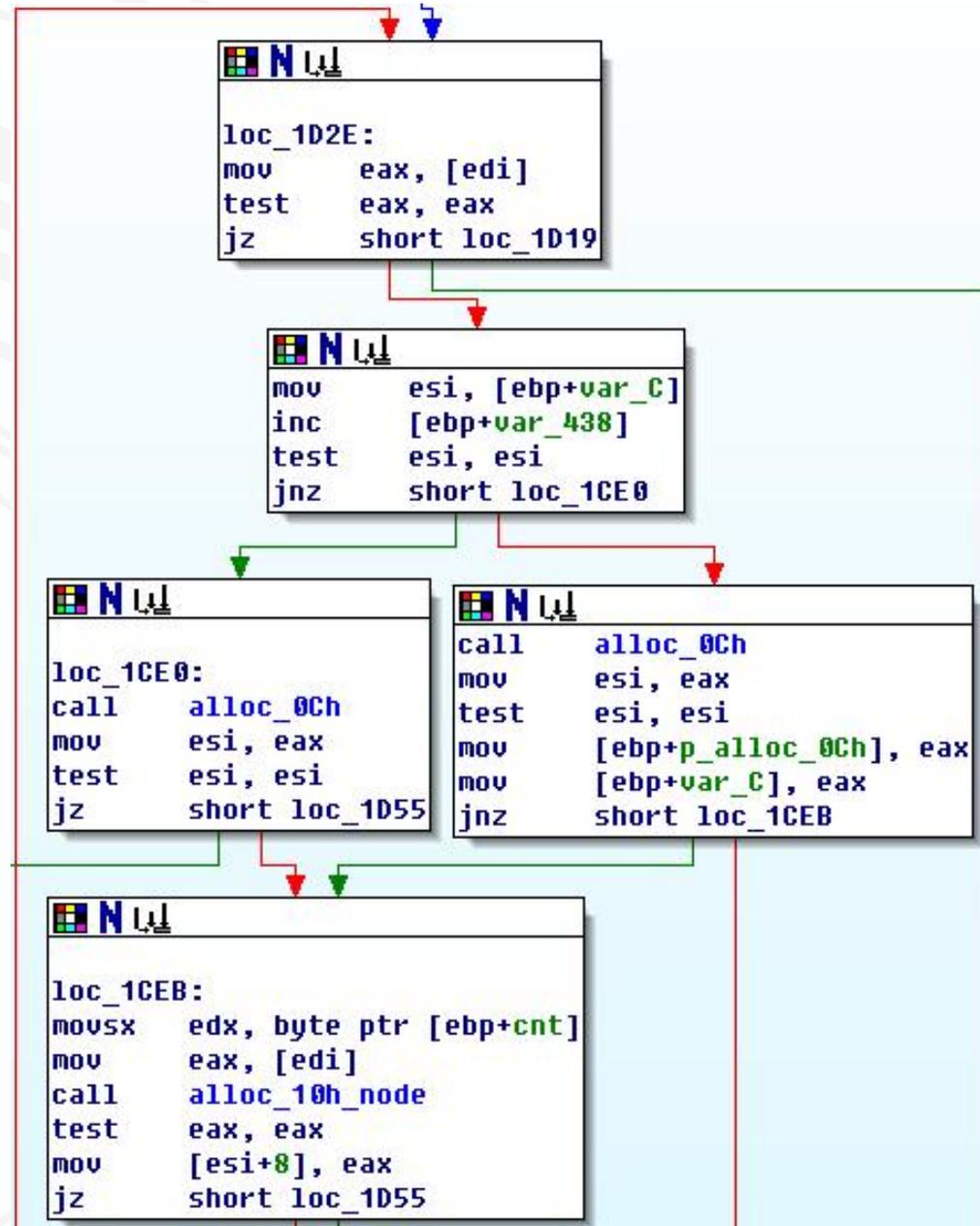
```
void construct_trie(void)
{
...
/* 초기 단계 */
for ( i = nhead = 0; i < 256; i++)
{
...
    if((h = (huf*)malloc(sizeof(huf))) == NULL)           // 16byte 할당
...
    h->count = freq[i];                                     // 빈도수 저장
    h->data = i;                                           // 문자 저장
...
}
}
...
...

```

Binary Leetness 200

다음 범위가
Huffman Tree를
위한 초기화 과정

[00001D2E ~ 00001D2C]



Binary Leetness 200

```
text:00001CB6          mov     edi, [ebp+p_intbuf]
```

; 각 문자의 빈도수가 존재하는 intbuf 배열의 주소를 edi에 저장

```
text:00001CBC          mov     [ebp+cnt], 0
```

; 카운터 값 초기화

Binary Leetness 200

```
text:00001CEB          movsx  edx, byte ptr [ebp+cnt]
```

; 카운터 값을 edx 레지스터에 저장

```
text:00001CF2          mov    eax, [edi]
```

; 현재 가리키고 있는 intbuf의 특정 오프셋 주소를 eax에 저장

```
text:00001CF4          call  alloc_10h_node
```

; alloc_10h_node 함수 호출

Binary Leetness 200

▪ (Alloc_10h_node 함수 내부)

```
text:00001A06          mov     dword ptr [esp], 10h
```

...

```
text:00001A17          call   _malloc
```

; Huffman tree를 위한 구조체 크기인 10h 만큼 공간 할당

```
text:00001A1C          test   eax, eax
```

```
text:00001A1E          jz     short loc_1A3C
```

; 실패시 점프

Binary Leetness 200

▪ (Alloc_10h_node 함수 내부)

```
text:00001A20          mov     edx, edi
```

; intbuf의 특정 오프셋을 가리키는 주소를 edx로 이동

```
text:00001A22          mov     dword ptr [eax+0Ch], 0
```

```
text:00001A29          mov     dword ptr [eax], 0
```

```
text:00001A2F          mov     dword ptr [eax+4], 0
```

; 구조체 초기화

Binary Leetness 200

▪ (Alloc_10h_node 함수 내부)

```
text:00001A36      mov    [eax+8], esi
```

*; esi 레지스터(intbuf의 특정 오프셋을 가리키고 있으며, 빈도수 값)의 값을
; 구조체의 8h번째 오프셋에 저장*

```
text:00001A39      mov    [eax+0Ch], dl
```

; dl 레지스터(문자 카운터 값)의 값을 구조체의 0Ch번째 오프셋에 저장

Binary Leetness 200

```
text:00001D19          inc  [ebp+cnt]
```

; 카운터 증가

```
text:00001D1F          add  edi, 4
```

; 각 문자의 빈도수를 저장하는 배열 카운터 증가

```
text:00001D22          cmp  [ebp+cnt], 100h
```

```
text:00001D2C          jz   short loc_1D61
```

; 100h(256)개가 아니라면 루프의 처음으로 점프

Binary Leetness 200

- 모든 문자에 대한 빈도수를 구함
- 트리를 사용하며 해당 구조체는 16byte
- 트리의 초기 과정에서 구조체에 문자와 해당 문자의 빈도수를 저장하며 이는 Huffman Tree의 초기 과정과 동일
- 이러한 과정을 총 256번(아스키의 빈도수와 동일) 수행

Binary Leetness 200

It's you!, Huffman baby~

Binary Leetness 300

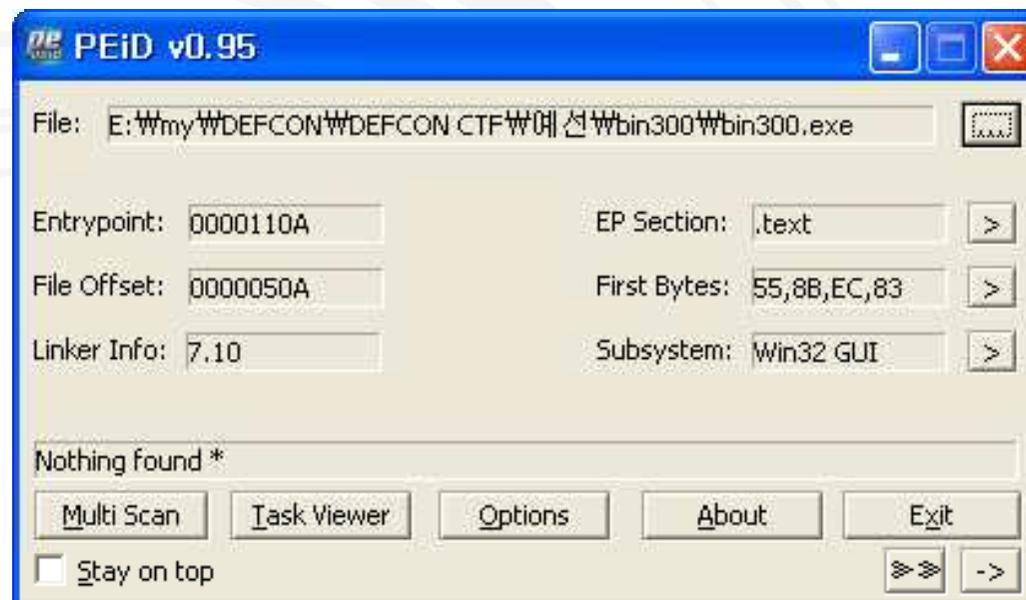
The answer is the 16 byte decryption key used in the file linked here. Enter in the form: \xaa\xbb\xcc...

바이너리에서 사용되는 16byte Decryption Key를 찾는 문제

- Windows 실행파일

```
[hkpc@server122 ~]$ file bin300
bin300: MS-DOS executable PE for MS Windows (GUI) Intel 80386 32-bit
```

Binary Leetness 300



Binary Leetness 300

- [CPU - main thread, module bin300]

File View Debug Plugins Options Window Help

Paused

| | | | |
|----------|---------------|--|-----------------|
| 0040110A | 55 | push ebp | |
| 0040110B | 8BEC | mov ebp, esp | |
| 0040110C | 83EC 1C | sub esp, 1C | |
| 00401110 | BE F9280000 | mov esi, 28F9 | |
| 00401115 | FF15 5C104000 | call near dword ptr ds:[<&KERNEL32.GetCommandLineA>] | GetCommandLineA |
| 0040111B | 8BF0 | mov esi, eax | |
| 0040111D | 2BDE | sub ebx, esi | |
| 0040111F | 81C0 B5540000 | add eax, 54B5 | |
| 00401125 | 8D3D C9464100 | lea edi, dword ptr ds:[4146C9] | |
| 0040112B | 81F6 17530000 | xor esi, 5317 | |
| 00401131 | BE AC104000 | mov esi, bin300.004010AC | |
| 00401136 | FFD6 | call near esi | |
| 00401138 | FF15 4C104000 | call near dword ptr ds:[<&KERNEL32.ExitThread>] | ExitThread |
| 0040113E | C3 | ret | |
| 0040113F | CC | int3 | |
| 00401140 | CC | int3 | |
| 00401141 | CC | int3 | |
| 00401142 | A6 | db A6 | |
| 00401143 | 12 | db 12 | |
| 00401144 | 32 | db 32 | CHAR '2' |
| 00401145 | 89 | db 89 | |
| 00401146 | A9 | db A9 | |
| 00401147 | 47 | db 47 | CHAR 'G' |
| 00401148 | 06 | db 06 | |
| 00401149 | B2 | db B2 | |
| 0040114A | 29 | db 29 | CHAR ')' |
| 0040114B | 1B | db 1B | |
| 0040114C | D6 | db D6 | |
| 0040114D | 4C | db 4C | CHAR 'L' |
| 0040114E | B4 | db B4 | |
| 0040114F | 4E | db 4E | CHAR 'N' |
| 00401150 | 0F | db 0F | |
| 00401151 | 8C | db 8C | |
| 00401152 | DD | db DD | |
| 00401153 | 10 | db 10 | |

Registers (FPU)

EAX 00000000
 ECX 0006FFB0
 EDX 7C93E514 ntdll,KiFastSystemCallRet
 EBX 7FFD6000
 ESP 0006FFC4
 EBP 0006FFFD
 ESI FFFFFFFF
 EDI 7C940228 ntdll,7C940228
 EIP 0040110A bin300.<ModuleEntryPoint>
 C 0 ES 0023 32bit 0(FFFFFFFF)
 P 1 CS 001B 32bit 0(FFFFFFFF)
 A 0 SS 0023 32bit 0(FFFFFFFF)
 Z 1 DS 0023 32bit 0(FFFFFFFF)
 S 0 FS 003B 32bit 7FFDF000(FFF)
 T 0 GS 0000 NULL
 D 0
 D 0
 LastErr ERROR_FILE_NOT_FOUND (00000002)
 EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
 ST0 empty -UNORM BCED 01050104 00000000
 ST1 empty 0,0
 ST2 empty 0,0
 ST3 empty 0,0
 ST4 empty 0,0
 ST5 empty 0,0
 ST6 empty 0,0
 ST7 empty 0,0
 FST 0020 Cond 0 0 0 0 Err 0 0 1 0 0 0 0 0 (GT)
 FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1

ebp=0006FFFD
 Jump from 004010DA

bin300.<ModuleEntryPoint>

| Address | Hex dump | UNICODE | 0006FFC4 |
|----------|---|---------|--------------------------------------|
| 004A9000 | 2D B8 5F C7 96 56 B3 24 E2 3F 12 69 73 7B EC 3E | 원일영(李) | 7C7E7077 RETURN to kernel32.7C7E7077 |
| 004A9010 | 48 81 B9 66 60 D5 7D C2 99 18 6A 77 79 88 06 05 | 원영일(李) | 7C940228 ntdll,7C940228 |
| 004A9020 | 80 84 BE 40 04 50 E5 72 B5 77 65 12 39 75 88 DB | 원영일(李) | FFFFFFFF |
| 004A9030 | 84 38 FB 3F EB 18 13 57 0B 0F 3D 00 F6 56 9D A4 | 원영일(李) | 7FFD6000 |
| 004A9040 | 5B C7 81 E4 81 ED 0F 8A 96 8D 8C 8B 15 D1 8C D8 | 원영일(李) | 805532FA |
| 004A9050 | DE 81 29 0C D2 89 37 00 36 D9 03 76 AA 81 80 7F | 원영일(李) | 0006FFC8 |
| 004A9060 | F9 C5 78 FC F2 A5 15 30 E6 B5 6B 34 6A E3 E3 26 | 원영일(李) | 86917A58 |
| 004A9070 | 74 2C 5F 2A 6A 90 48 CC A9 C9 7F 5A 53 9F C8 79 | 원영일(李) | FFFFFFFF End of SEH chain |
| 004A9080 | 51 D0 1D 62 25 28 C7 18 6A 7E 7E 0C 4F A5 E1 1B | 원영일(李) | 7C809A08 SE handler |
| 004A9090 | 4B 99 DD 17 13 30 B6 AF C7 30 2B 56 43 00 3F 1E | 원영일(李) | 7C7E7080 kernel32,7C7E7080 |
| 004A90A0 | C1 42 FE C4 45 2A E7 F8 4A 01 9F A1 2D 9E A2 A4 | 원영일(李) | 00000000 |
| 004A90B0 | 73 31 6C 6C 14 C6 18 5E 0A 8D 08 6C 08 5B D8 C3 | 원영일(李) | 00000000 |
| 004A90C0 | 73 31 6C 6C 14 C6 18 5E 0A 8D 08 6C 08 5B D8 C3 | 원영일(李) | 00000000 |

Command:

Analysing bin300: 1 heuristical procedure, 2 calls to known functions

Binary Leetness 300

< binary 300의 개략적 실행 흐름 >

- ① 930000, 990000 메모리 할당
- ② 930000, 990000에 데이터를 써넣음
- ③ 930000 영역 내부 코드(9303BD)가 990000 영역 메모리 복호화
- ④ 400000 메모리 할당
- ⑤ 930000루틴이 990400의 데이터를 (407000~407000*(E00*4))에 써넣음(0040A590는 UPX 구문)
- ⑥ CreateRemoteThread로 해당 UPX 루틴 실행
- ⑦ helloworld.txt 파일에 "Hello world!"라는 문자열을 기록

사용된 기술 - 코드 난독화, 코드 암호화, 패킹

Binary Leetness 300

< binary 300의 개략적 실행 흐름 >

- ① 930000, 990000 메모리 할당
- ② 930000, 990000에 데이터를 써넣음
- ③ 930000 영역 내부 코드(9303BD)가 990000 영역 메모리 **복호화**
- ④ 400000 메모리 할당
- ⑤ 930000루틴이 990400의 데이터를 (407000~407000*(E00*4))에 써넣음(0040A590는 UPX 구문)
- ⑥ CreateRemoteThread로 해당 UPX 루틴 실행
- ⑦ helloworld.txt 파일에 "Hello world!"라는 문자열을 기록

사용된 기술 - 코드 난독화, 코드 암호화, 패킹

Binary Leetness 300

< binary 300의 개략적 실행 흐름 >

- ① A, B 메모리 할당
- ② A, B에 각각 데이터를 써넣음
- ③ A 영역이 암호화 된 B 영역 데이터를 **복호화**
- ④ C 메모리 할당
- ⑤ A 루틴이 복호화 된 B의 데이터를 C에 써넣음
- ⑥ CreateRemoteThread로 C영역 루틴 실행
- ⑦ helloworld.txt 파일에 "Hello world!"라는 문자열을 기록

사용된 기술 - 코드 난독화, 코드 암호화, 패킹

Binary Leetness 300

< 분석 시 장애물 >

- 코드 난독화가 이루어져 있기 때문에 분석 툴이 어셈블리 코드를 제대로 해석하지 못함
- 프로그램의 엔트리 포인트를 엉뚱한 곳으로 지정
- 패킹

Binary Leetness 300

< 분석 방법론 >

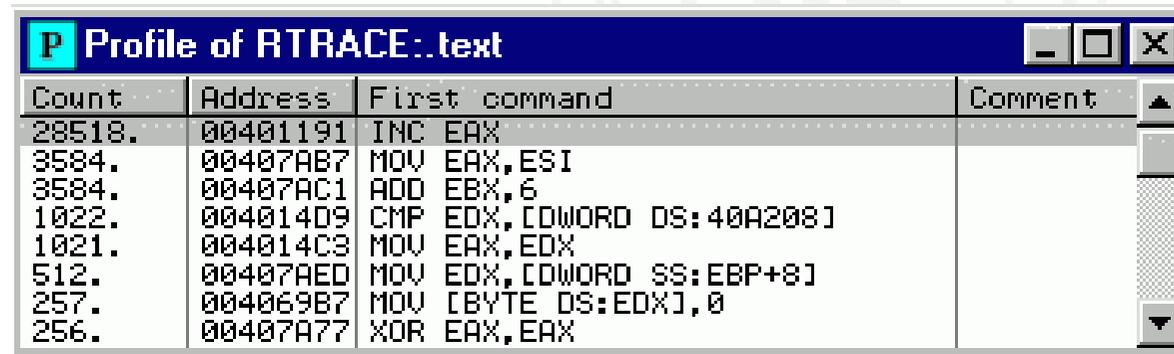
- 코드 난독화, 암호화, 패킹 등 어떠한 방법이든 실행 시에는 모두 그 원래의 형태로 진행되어야 함
- 여기서는 그러한 특성을 이용하여 동적 분석으로 쉽게 해결할 수 있음

Binary Leetness 300

두 가지 방법(더 있겠지만 여기서 언급하는것에 해당)

▪ Run trace

- 프로그램을 직접 실행하며 해당 명령의 실행 카운터 등을 구해주는 Ollydbg의 기능
- 참가 팀중 한 명이 해당 방법으로 풀이한 보고서를 작성
- 복호화 과정 시 동일한 명령의 반복 루틴이 많을것이라 예상하고 Run trace를 이용하여 빈도수가 많은 명령 근처에서 복호화 루틴을 쉽게 찾음



| Count | Address | First command | Comment |
|--------|----------|---------------------------|---------|
| 28518. | 00401191 | INC EAX | |
| 3584. | 00407AB7 | MOV EAX,ESI | |
| 3584. | 00407AC1 | ADD EBX,6 | |
| 1022. | 004014D9 | CMP EDX,[DWORD DS:40A208] | |
| 1021. | 004014C3 | MOV EAX,EDX | |
| 512. | 00407AED | MOV EDX,[DWORD SS:EBP+8] | |
| 257. | 004069B7 | MOV [BYTE DS:EDX],0 | |
| 256. | 00407A77 | XOR EAX,EAX | |

Binary Leetness 300

▪ Breakpoint의 활용

- 바이너리 내부의 API가 아닌 DLL 파일 자체의 API에 breakpoint를 걸어두고 실행 시 breakpoint가 걸리는 지점 근처의 루틴을 분석
- 300번 문제의 경우는 패킹을 푸는 과정에서 암호화 된 루틴을 복호화 시키는 등의 진행을 거치고 있음
- 바이너리가 실행압축 되어있다고 가정하고 그에 자주 사용되는 API인 **VirtualAllocEx()**에 breakpoint를 걸어둔 뒤 분석

Binary Leetness 300

- [CPU - main thread]

File View Debug Plugins Options Window Help

Paused

| Address | Hex dump | UNI CODE |
|----------|---|---------------------|
| 0093040F | E8 A9 FF FF FF 88 75 FC 88 46 3C 03 45 FC 89 03 | CALL EB A9 FF FF |
| 0093041F | 88 03 88 50 50 88 03 88 40 34 88 40 08 E8 60 FE | CALL EB 03 88 50 |
| 0093042F | FF FF 88 55 F8 89 02 88 45 F8 83 38 00 74 6A 8B | CALL EB FF FF |
| 0093043F | 03 88 48 54 88 55 F8 88 12 88 C6 E8 1E FC FF FF | CALL EB 03 88 48 54 |
| 0093044F | 88 03 E8 93 FE FF FF 88 F0 88 03 0F B7 40 06 48 | CALL EB 88 03 E8 93 |
| 0093045F | 85 C0 72 45 40 89 45 F0 33 DB 88 55 F8 88 12 80 | CALL EB 85 C0 72 45 |
| 0093046F | 3C 96 03 54 FE 0C 88 4C FE 10 88 44 FE 14 03 45 | CALL EB 3C 96 03 54 |
| 0093047F | FC E8 E8 FB FF FF 80 45 F4 50 6A 40 88 44 FE 08 | CALL EB FC E8 E8 FB |
| 0093048F | 50 86 45 F8 88 00 03 44 FE 0C 50 6A FF 88 45 08 | CALL EB 50 86 45 F8 |
| 0093049F | FF 50 14 43 FF 40 F0 75 C1 5F 5E 58 88 E5 50 C2 | CALL EB FF 50 14 43 |
| 009304AF | 0C 00 56 53 89 C6 64 8B 05 30 00 00 88 40 0C | CALL EB 0C 00 56 53 |
| 009304BF | 82 00 0C 88 00 88 70 18 7E 50 80 50 18 80 48 1C | CALL EB 82 00 0C 88 |

```

009303F1 C3      letr
009303F2 8040 00  lea eax, dword ptr ds:[eax]
009303F5 55      push ebp
009303F6 8BEC   mov ebp, esp
009303F8 83C4 F0 add esp, -10
009303FB 53      push ebx
009303FC 56      push esi
009303FD 57      push edi
009303FE 8BD9   mov ebx, ecx
00930400 8955 F8 mov dword ptr ss:[ebp-8], edx
00930403 8945 FC mov dword ptr ss:[ebp-4], eax
00930406 8B4D 0C mov ecx, dword ptr ss:[ebp+C]
00930409 8B55 10 mov edx, dword ptr ss:[ebp+10]
0093040C 8B45 FC mov eax, dword ptr ss:[ebp-4]
0093040F E8 A9FFFFFF call 0093038D
00930414 8B75 FC mov esi, dword ptr ss:[ebp-4]
00930417 8B46 3C mov eax, dword ptr ds:[esi+3C]
0093041A 0345 FC add eax, dword ptr ss:[ebp-4]
0093041D 8903   mov dword ptr ds:[ebx], eax
0093041F 8B03   mov eax, dword ptr ds:[ebx]
00930421 8B50 50 mov edx, dword ptr ds:[eax+50]
00930424 8B03   mov eax, dword ptr ds:[ebx]
00930426 8B4D 34 mov eax, dword ptr ds:[eax+34]
00930429 8B4D 08 mov ecx, dword ptr ss:[ebp+8]
0093042C E8 60FEFFFF call 00930291
00930431 8B55 F8 mov edx, dword ptr ss:[ebp-8]
00930434 8902   mov dword ptr ds:[edx], eax
00930436 8B45 F8 mov eax, dword ptr ss:[ebp-8]
00930439 8338 00 cmp dword ptr ds:[eax], 0
0093043C 74 6A  je short 009304A8
0093043E 8B03   mov eax, dword ptr ds:[ebx]
00930440 8B48 54 mov ecx, dword ptr ds:[eax+54]
00930443 8B55 F8 mov edx, dword ptr ss:[ebp-8]
00930446 8B12   mov edx, dword ptr ds:[edx]
00930448 8BC6   mov eax, esi
    
```

Registers (FPU)

```

EAX 00940000
ECX 0091000C
EDX 00062400
EBX 0006F674
ESP 0006F62C
EBP 0006F648
ESI 00910000
EDI 00062400
EIP 0093040F
C 1 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_INVALID_ADDRESS (000001E7)
EFL 00000203 (NO,B,NE,BE,NS,PO,GE,G)
ST0 empty -UNORM BB80 01050104 00000000
ST1 empty 0,0
ST2 empty 0,0
ST3 empty 0,0
ST4 empty 0,0
ST5 empty 0,0
ST6 empty 0,0
ST7 empty 0,0
    
```

| Address | Hex dump | UNI CODE |
|----------|----------|----------------------------------|
| 0006F62C | 00062400 | |
| 0006F630 | 00910000 | |
| 0006F634 | 00920000 | |
| 0006F638 | 7C7D9B60 | kernel32,7C7D9B60 |
| 0006F63C | FFFFFFFF | |
| 0006F640 | 0006F670 | |
| 0006F644 | 00940000 | |
| 0006F648 | 0006F678 | |
| 0006F64C | 00930529 | RETURN to 00930529 from 009303F5 |
| 0006F650 | 00920000 | |
| 0006F654 | 00910000 | |
| 0006F658 | 00062400 | |
| 0006F65C | 00400000 | |

Command: bp VirtualAllocEx

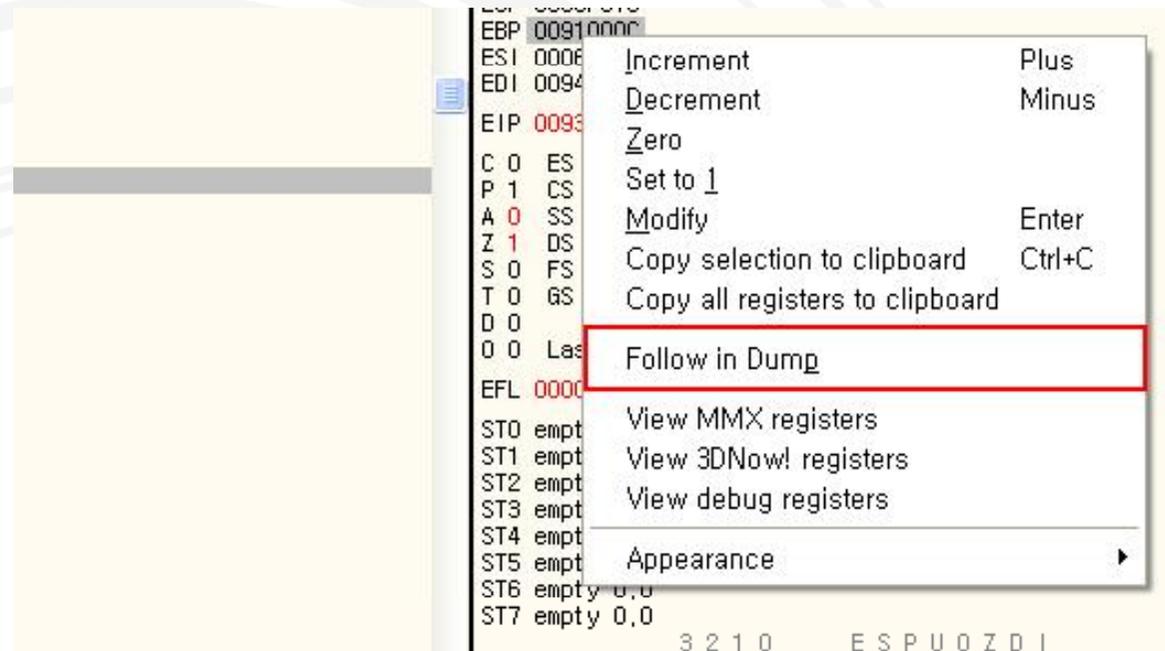
Hardware breakpoint 1 at 0093040F

Binary Leetness 300

| | | |
|----------|-----------|-------------------------------|
| 009303C2 | 74 2A | je short 009303EE |
| 009303C4 | 8BF8 | mov edi, eax |
| 009303C6 | 8BE9 | mov ebp, ecx |
| 009303C8 | 33C0 | xor eax, eax |
| 009303CA | 8BF2 | mov esi, edx |
| 009303CC | 4E | dec esi |
| 009303CD | 85F6 | test esi, esi |
| 009303CF | 72 10 | jb short 009303EE |
| 009303D1 | 46 | inc esi |
| 009303D2 | 33D2 | xor edx, edx |
| 009303D4 | 8A4C05 00 | mov cl, byte ptr ss:[ebp+eax] |
| 009303D8 | 300C17 | xor byte ptr ds:[edi+edx], cl |
| 009303DB | 8BCA | mov ecx, edx |
| 009303DD | 32C8 | xor cl, al |
| 009303DF | 300C17 | xor byte ptr ds:[edi+edx], cl |
| 009303E2 | 40 | inc eax |
| 009303E3 | 83F8 10 | cmp eax, 10 |
| 009303E6 | 75 02 | jnz short 009303EA |
| 009303E8 | 33C0 | xor eax, eax |
| 009303EA | 42 | inc edx |
| 009303EB | 4E | dec esi |
| 009303EC | 75 E6 | jnz short 009303D4 |
| 009303EE | 5D | pop ebp |
| 009303EF | 5F | pop edi |
| 009303F0 | 5E | pop esi |
| 009303F1 | C3 | retn |
| 009303F2 | 8D40 00 | lea eax, dword ptr ds:[eax] |
| 009303F5 | 55 | push ebp |
| 009303F6 | 8BEC | mov ebp, esp |
| 009303F8 | 83C4 F0 | add esp, -10 |
| 009303FB | 53 | push ebx |
| 009303FC | 56 | push esi |
| 009303FD | 57 | push edi |
| 009303FE | 8BD9 | mov ebx, ecx |
| 00930400 | 8955 F8 | mov dword ptr ss:[ebp-8], edx |

| Registers (FPU) | |
|-----------------|----------|
| EAX | 00000000 |
| ECX | 0091000C |
| EDX | 00000000 |
| EBX | 0006F674 |
| ESP | 0006F61C |
| EBP | 0091000C |
| ESI | 00062400 |
| EDI | 00940000 |
| EIP | 009303D4 |

Binary Leetness 300



Binary Leetness 300

| | | |
|----------|---------|-------------------------------|
| 009303F0 | 3C | pop esi |
| 009303F1 | C3 | ret |
| 009303F2 | 8D40 00 | lea eax, dword ptr ds:[eax] |
| 009303F5 | 55 | push ebp |
| 009303F6 | 8BEC | mov ebp, esp |
| 009303F8 | 83C4 F0 | add esp, -10 |
| 009303FB | 53 | push ebx |
| 009303FC | 56 | push esi |
| 009303FD | 57 | push edi |
| 009303FE | 8BD9 | mov ebx, ecx |
| 00930400 | 8955 F8 | mov dword ptr ss:[ebp-8], edx |

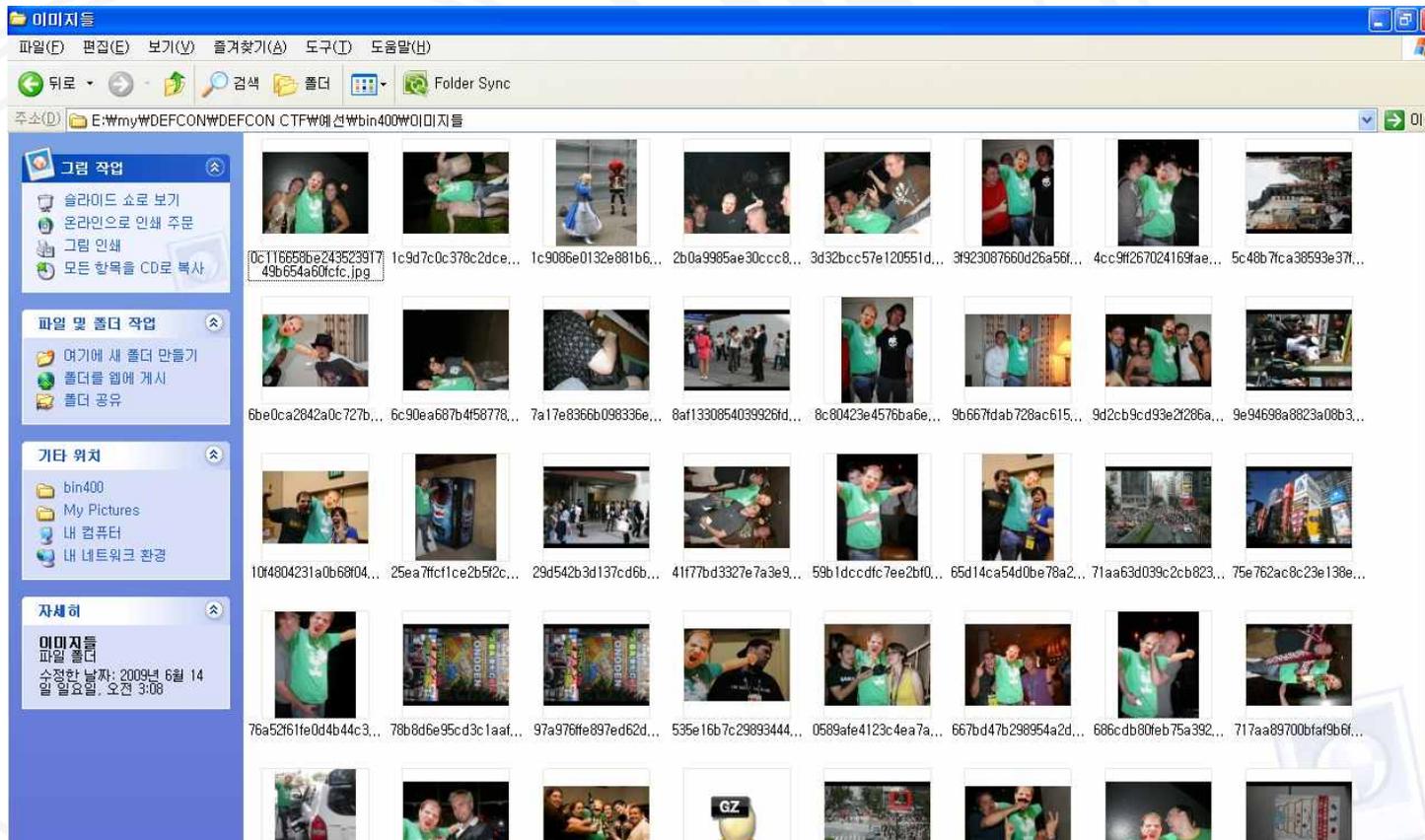
ss:[0091000C]=87
 cl=0C (Form Feed)

| Address | Hex dump | UNICODE |
|----------|---|---------|
| 0091000C | 87 03 4E 03 34 FE 56 87 1E 4C 86 67 1B 05 23 15 | ... |
| 0091001C | 00 00 92 00 12 9B 7D 7C 00 00 00 00 00 00 00 00 | ... |
| 0091002C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091003C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091004C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091005C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091006C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091007C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091008C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 0091009C | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |
| 009100AC | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ... |

Binary Leetness 400

The answer to your second questions is 'by hash'.

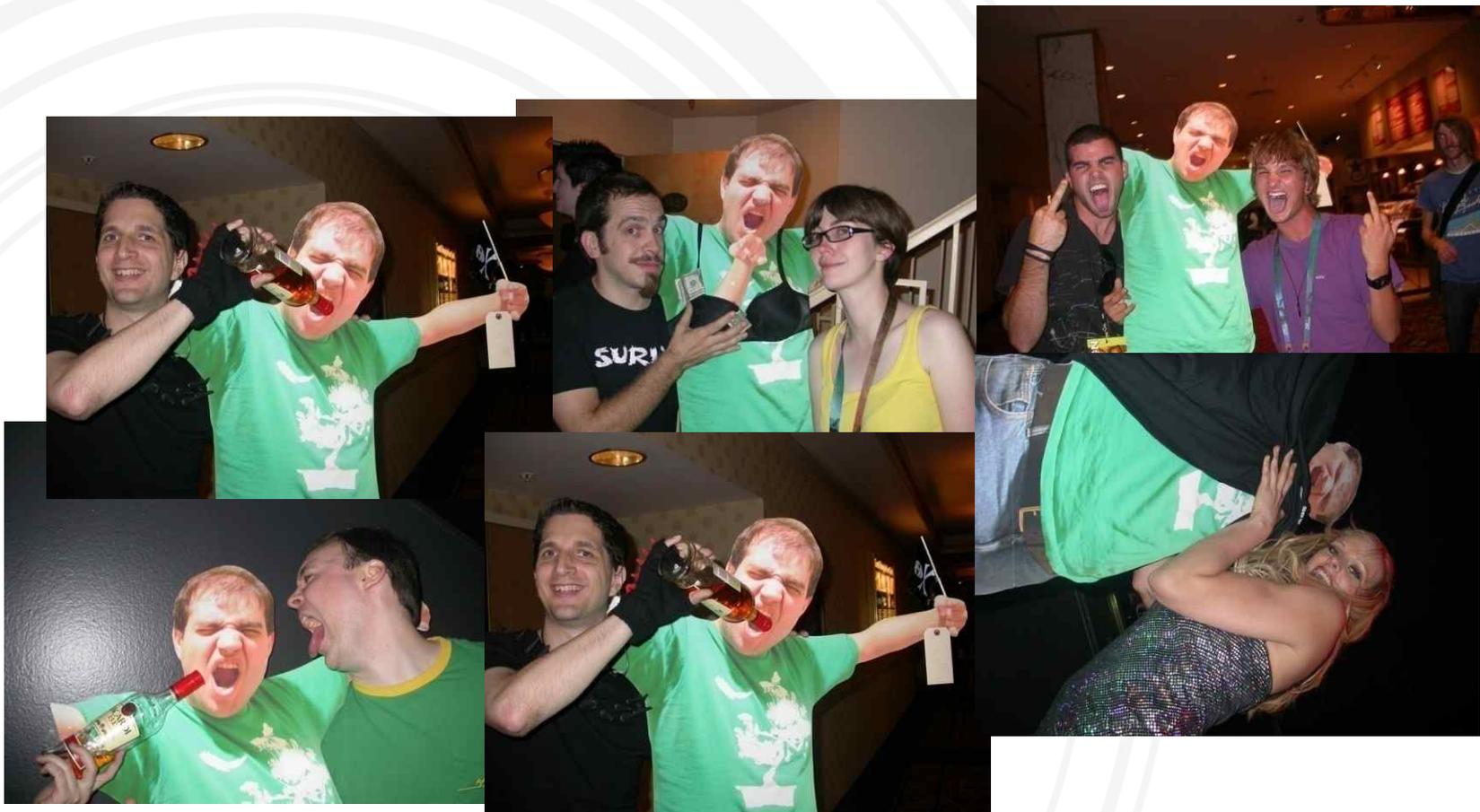
- 압축 파일이 주어지고, 해당 압축을 풀어보면 이미지 파일 수십개가 나옴



Binary Leetness 400

Dan Kaminsky?

Damn it Kaminsky!!



Binary Leetness 400

- 해당 이미지 파일 포맷은 EXIF, JFIF 둘 중 하나

```
hkpc@server122:~/bin400
1c9086e0132e881b64f944d2c50e66ed.jpg: JPEG image data, JFIF standard 1.01
1c9d7c0c378c2dce8ebdd529f85e4d99.jpg: JPEG image data, JFIF standard 1.01
25ea7ffcf1ce2b5f2cb806e412f69525.jpg: JPEG image data, JFIF standard 1.01
29d542b3d137cd6b4f13028fb3fc0538.jpg: JPEG image data, JFIF standard 1.01
2b0a9985ae30ccc8de66bae3cf22cfce.jpg: JPEG image data, JFIF standard 1.01
3d32bcc57e120551de6f8c3959129de3.jpg: JPEG image data, JFIF standard 1.01
3f923087660d26a56f3a7fc6a076bcbf.jpg: JPEG image data, JFIF standard 1.01
41f77bd3327e7a3e966c81cb70479018.jpg: JPEG image data, JFIF standard 1.01
4375bdb29f0bceedbcbdaa6c9f74749a5.jpg: JPEG image data, EXIF standard
48161ea04af162f34c50080add53d884.jpg: JPEG image data, JFIF standard 1.01
4cc9ff267024169fae7d0934e166b6cc.jpg: JPEG image data, JFIF standard 1.01
535e16b7c2989344410b589d8db1c0d8.jpg: JPEG image data, JFIF standard 1.01
5425dbb89a0834f26aa3023d67d51be1.jpg: JPEG image data, JFIF standard 1.01
57843b4b0280a03014703288c65e7831.jpg: JPEG image data, JFIF standard 1.01
59b1dcccdfc7ee2bf0bf5c0ff94d81e06.jpg: JPEG image data, JFIF standard 1.01
5c48b7fca38593e37f9e6a4ab514f784.jpg: JPEG image data, JFIF standard 1.01
65d14ca54d0be78a20319334dbc1cf81.jpg: JPEG image data, JFIF standard 1.01
667bd47b298954a2d75387a6e67837d5.jpg: JPEG image data, JFIF standard 1.01
686cdb80feb75a39230e8d99dcf82632.jpg: JPEG image data, JFIF standard 1.01
6be0ca2842a0c727b46a84ab04f136c2.jpg: JPEG image data, JFIF standard 1.01
6c90ea687b4f58778a6a34e94b84b37b.jpg: JPEG image data, JFIF standard 1.01
717aa89700bfaf9b6f62ea5bbfb28d76.jpg: JPEG image data, JFIF standard 1.01
71aa63d039c2cb823a173918c3997b56.jpg: JPEG image data, JFIF standard 1.01
7387b571323c5cbd713f3700bb50ff00.jpg: JPEG image data, JFIF standard 1.01
75e762ac8c23e138e019645be7612996.jpg: JPEG image data, JFIF standard 1.01
76a52f61fe0d4b44c3e57bd9112d25fb.jpg: JPEG image data, JFIF standard 1.01
770cc6a1698e46065fe9379b22244edb.jpg: JPEG image data, JFIF standard 1.01
7818f7e5c40ce02ee6127a1f999eebe8.jpg: JPEG image data, EXIF standard
7a17e8366b098336ed787215d3f7188b.jpg: JPEG image data, JFIF standard 1.01
```

Binary Leetness 400

- Exif(Exchangeable image file format)
 - Exchangeable image file format (Exif) is a specification for the image file format used by digital cameras.
- JFIF(JPEG File Interchange Format)
 - The JPEG File Interchange Format (JFIF) is an image file format standard.
- 이미지 파일 내에 특정 데이터가 숨겨져 있을것으로 예상
- 하지만, 헤더 구조를 잘 알지않는 이상 이를 구별해 내기가 힘들
- 관련 파일 정보 등을 분석해 주는 툴을 이용하면 편리
 - Ex) PhotoME

Binary Leetness 400

PhotoME - [E:\WmyWDEFCONWDEFCON CTFW에 선Wbin400W이미지들W75e762ac8c23e138e019645be7612996.jpg]

File Menu

- Open file...
- Save file
- Save as...
- Import/Export
- PhotoBrowser...
- Settings
- Exit

Histogram

Thumbnail

| Image | Thumbnail Info | Camera | Manufacturer notes | ICC Profile |
|---|---|--------|-----------------------|-----------------|
| ISO speed rating | 400/27° | 8827 | ISOSpeedRatings | SHORT |
| Exif version | Version 1.0 | 9000 | ExifVersion | UNDEFINED(4) |
| Date and time of original data generation | 2007-09-16 15:37:50 | 9003 | DateTimeOriginal | ASCII(20) |
| Date and time of digital data generation | 2007-09-16 15:37:50 | 9004 | DateTimeDigitized | ASCII(20) |
| Meaning of each component | YCbCr | 9101 | ComponentsConfig... | UNDEFINED(4) |
| Shutter speed | 11.55 Tv (1/3000") | 9201 | ShutterSpeedValue | SRATIONAL |
| Aperture | 4.97 Av (F5.6) | 9202 | ApertureValue | RATIONAL |
| Exposure bias | ±0 EV | 9204 | ExposureBiasValue | SRATIONAL |
| Metering mode | Pattern | 9207 | MeteringMode | SHORT |
| Flash | Flash did not fire, compulsory flash mode | 9209 | Flash | SHORT |
| Lens focal length | 17 mm | 920A | FocalLength | RATIONAL |
| Manufacturer notes | 0x00000283 | 927C | MakerNote | UNDEFINED(4868) |
| ??? | <Binary Data> | 7A69 | | UNDEFINED(4104) |
| Supported Flashpix version | Version 1.0 | A000 | FlashpixVersion | UNDEFINED(4) |
| Color space | sRGB | A001 | ColorSpace | SHORT |
| Image width | 3888 px | A002 | PixelXDimension | SHORT |
| Image height | 2592 px | A003 | PixelYDimension | SHORT |
| Focal plane X resolution | 4433.3 ppi | A20E | FocalPlaneXResolu... | RATIONAL |
| Focal plane Y resolution | 4453.61 ppi | A20F | FocalPlaneYResolu... | RATIONAL |
| Focal plane resolution unit | inch | A210 | FocalPlaneResoluti... | SHORT |
| Custom image processing | Normal process | A401 | CustomRendered | SHORT |
| Exposure mode | Auto exposure | A402 | ExposureMode | SHORT |
| White balance | Auto | A403 | WhiteBalance | SHORT |
| Scene capture type | Standard | A406 | SceneCaptureType | SHORT |
| Contrast | Hard | A408 | Contrast | SHORT |

Find:

Binary Leetness 400

- Python의 Python Imaging Library를 이용하여 해당 데이터 추출
- 이미지 파일 안에 숨겨진 데이터는 Tag-ID 7A69에 존재하며
- PIL의 Exif 관련 함수로 추출 가능
- 참고로, Exif 표준은 JFIF의 거의 모든 기능을 지원하므로 JIFF를 고려해 줄 필요는 없음

Binary Leetness 400

- 추출된 데이터들은 압축(ZIP) 파일의 조각들이며
- 이를 특정 규칙으로 merge 해야함

- 이는 jpg 파일 전체 내용의 SHA-1 값 정렬 순으로 합치면 됨
- 다양한 hash를 기준으로 정렬해 보면 그 시작 부분이 ZIP 파일의 헤더로 나오는것을 보고 알 수 있음

Binary Leetness 400

이미지 파일에서 데이터를
추출하여 SHA-1 순으로
합치는 스크립트

```
extract_data.py - 메모장
파일(E) 편집(E) 서식(O) 보기(V) 도움말(H)
import Image
import ExifTags
import glob
import hashlib

files_jpg = glob.glob('./*.jpg')
for fname in files_jpg:
    im = Image.open(fname)
    exinfo = im._getexif()
    val = exinfo[31337]
    val = val[8:]
    fname = fname.replace( 'jpg', 'slice' )
    fp = open(fname, 'wb')
    fp.write(val)
    fp.close()

files_zip = glob.glob('./*.jpg')
dicts = {}
for fname in files_zip:
    fp = open( fname, 'rb' )
    data = fp.read()
    dicts[fname] = hashlib.sha1(data).hexdigest()
    fp.close()

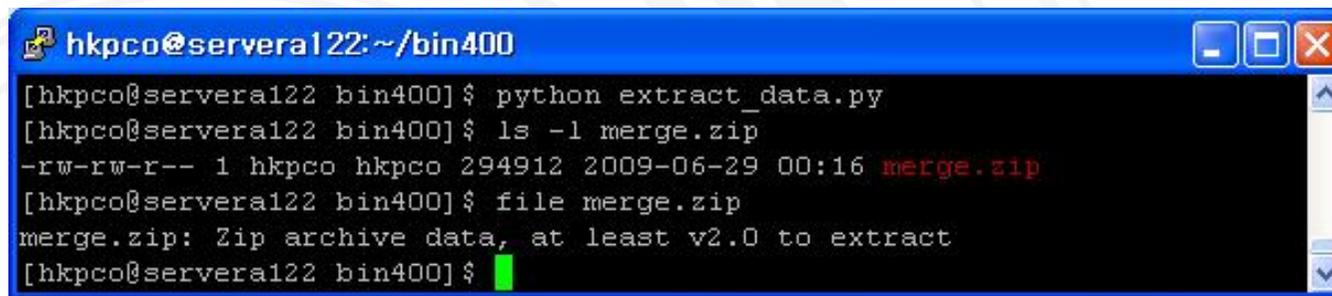
diclist = []
for i in dicts.keys():
    diclist.append( (i, dicts[i]) )

diclist.sort( key = lambda(x, y):y )

merge_zip = open( 'merge.zip', 'ab' )
for file in diclist:
    ff = file[0].replace( 'jpg', 'slice' )
    f_tmp = open( ff, 'rb' )
    slice_data = f_tmp.read()
    merge_zip.write(slice_data)
    f_tmp.close()

merge_zip.close()
```

Binary Leetness 400

A terminal window with a blue title bar and standard window controls. The title bar text is 'hkpc@servera122:~/bin400'. The terminal content shows a sequence of commands and their outputs: 'python extract_data.py' is executed; 'ls -l merge.zip' shows file permissions and metadata; 'file merge.zip' identifies the file as a zip archive.

```
hkpc@servera122:~/bin400
[hkpc@servera122 bin400]$ python extract_data.py
[hkpc@servera122 bin400]$ ls -l merge.zip
-rw-rw-r-- 1 hkpc hkpc 294912 2009-06-29 00:16 merge.zip
[hkpc@servera122 bin400]$ file merge.zip
merge.zip: Zip archive data, at least v2.0 to extract
[hkpc@servera122 bin400]$
```

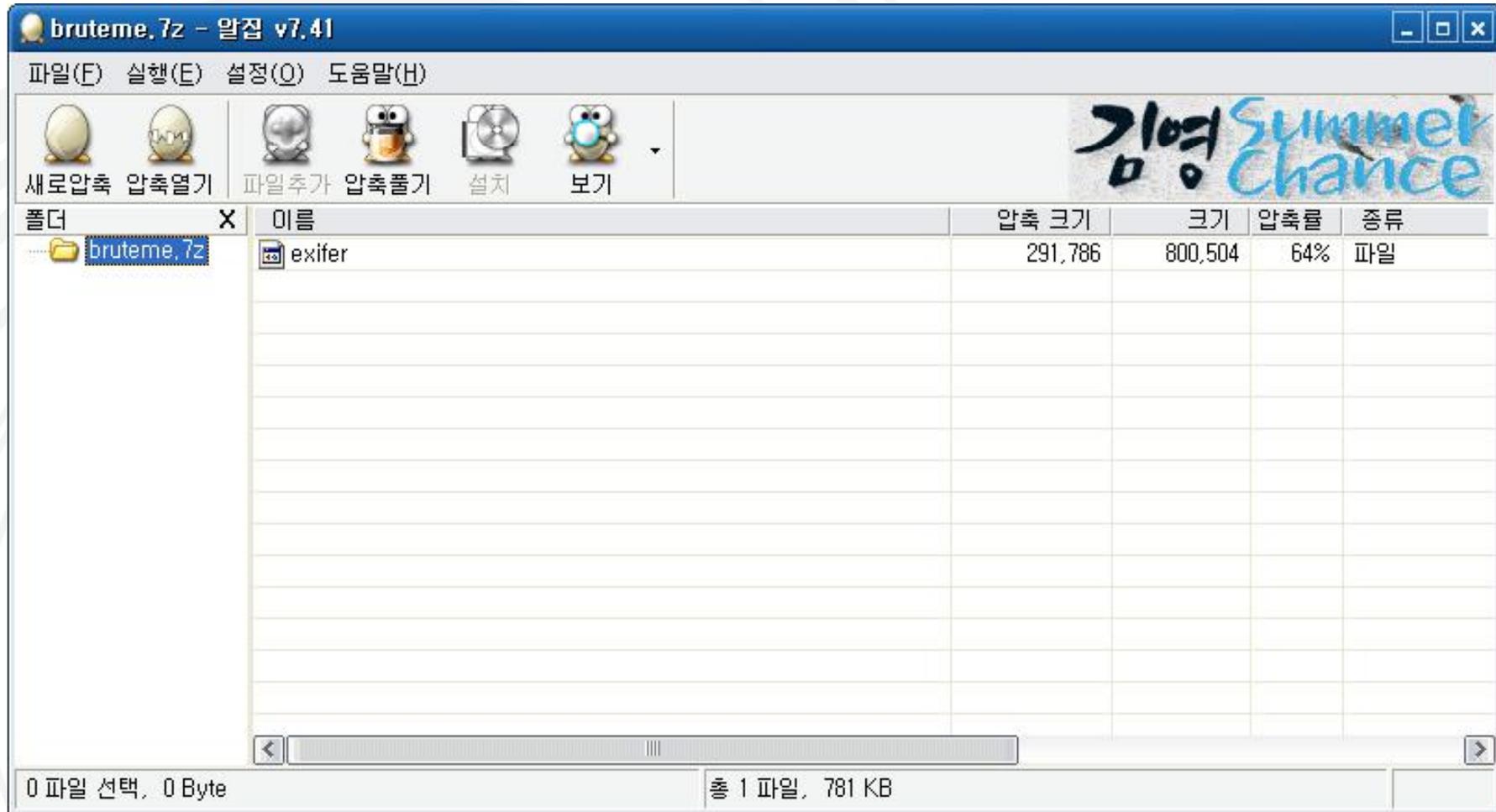
Binary Leetness 400



Binary Leetness 400

- 압축 파일에 비밀번호가 걸려있음
- 이는 압축 툴(알집 등) 자체의 암호 찾기 기능이나 **Advanced Archive Password Recovery** 프로그램 등을 사용하여 풀 수 있음
- 압축 파일의 비밀번호는 **weak**

Binary Leetness 400



Binary Leetness 400

- 압축을 풀면 바이너리가 나오며, 간략한 정보는 다음과 같음

```
[hkpc@servera122 bin400]$ file exifer
```

```
exifer: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), statically  
linked, for GNU/Linux 2.6.9, stripped
```

Binary Leetness 400

```
call    sys_accept      ; accept
mov     [ebp+var_C], eax
mov     eax, [ebp+var_C]
mov     [esp+4], eax
mov     dword ptr [esp], offset aPastAcceptClie ; "past accept: client is %d\n"
call    sub_8063D90
cmp     [ebp+var_C], 0FFFFFFFh
jz      short loc_80488F8
```

```
call    sys_fork
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 0FFFFFFFh
jz      short loc_80488F8
```

```
cmp     [ebp+var_8], 0
jnz     short loc_80488ED
```

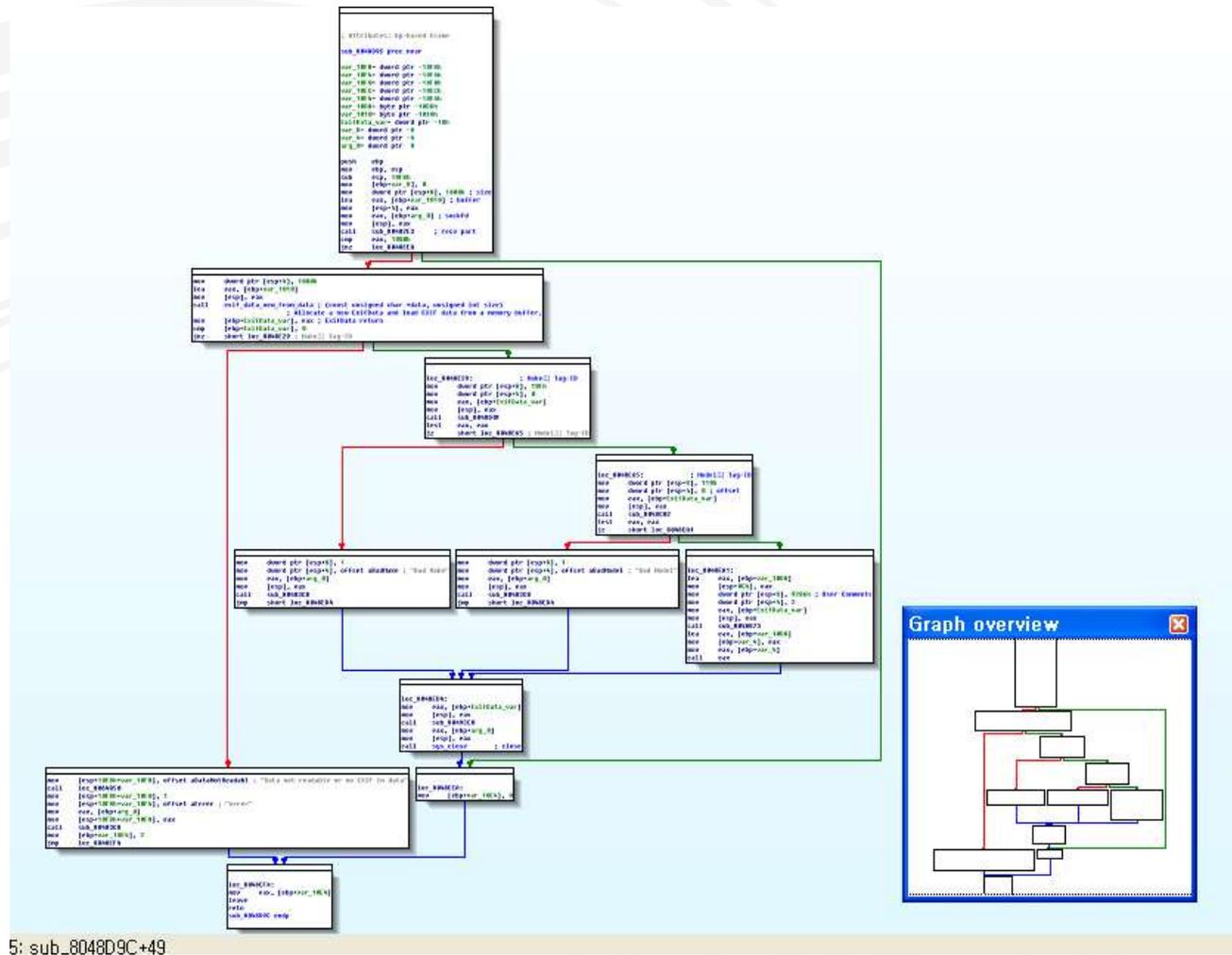
```
mov     eax, [ebp+arg_0]
mov     [esp], eax
call    sys_close
mov     eax, [ebp+var_C]
mov     [esp], eax
mov     eax, [ebp+arg_4]
call    eax
mov     [ebp+var_4], eax
mov     eax, [ebp+var_C]
```

```
loc_80488ED:
mov     eax, [ebp+var_C]
mov     [esp], eax
call    sys_close
```

Binary Leetness 400

- 서버 프로그램의 루틴은 보통 `accept()` 함수로 연결을 수락한 다음, 스레드 혹은 자식 프로세스를 생성하여 주요 루틴을 수행하도록 작성이 됨.
- 대회 문제는 특히 안정성을 위하여 스레드 보다는 자식 프로세스 생성으로 처리를 많이 하는편임
- 그래서 `accept()` 함수 또는 `fork()` 함수 등의 호출 구문 이후의 루틴을 살펴보면 서버에서 처리하는 핵심 루틴을 쉽게 찾을 수 있음
- 여기서는 두 번째 인자의 주소값을 호출하는것을 볼 수 있음

Binary Leetness 400



Binary Leetness 400

- 파일 이름 등 조금만 살펴보면 해당 바이너리는 **Exif** 관련 라이브러리를 사용한다는 것을 알 수 있음
- 문제 바이너리는 **statically**에 디버깅 정보까지 삭제되어 있기 때문에 기본적으로 함수명을 알 수 없으며, 대신 인자값의 개수나 함수 내의 고유 스트링 등을 **libexif(Exif 라이브러리)**에 존재하는 함수 원형이나 소스코드 등의 내용과 비교하여 쉽게 추측할 수 있음(그 이외에도 방법은 많음)

Binary Leetness 400

```
var_4= dword ptr -4
arg_0= dword ptr 8

push    ebp
mov     ebp, esp
sub     esp, 10F8h
mov     [ebp+var_8], 0
mov     dword ptr [esp+8], 1000h ; size
lea     eax, [ebp+var_1010] ; buffer
mov     [esp+4], eax
mov     eax, [ebp+arg_0] ; sockfd
mov     [esp], eax
call    sub_80482E3 ; recv part
cmp     eax, 1000h
jnz     loc_8048EEA
```

```
mov     dword ptr [esp+4], 1000h
lea     eax, [ebp+var_1010]
mov     [esp], eax
call    exif_data_new_from_data ; (const unsigned char *data, unsigned int size)
                                     ; Allocate a new ExifData and load EXIF data from a memory buffer.
mov     [ebp+ExifData_var], eax ; ExifData return
cmp     [ebp+ExifData_var], 0
jnz     short loc_8048E29 ; Make Tag-ID
```

Binary Leetness 400

```
loc_8048E29:                ; Make의 Tag-ID
mov     dword ptr [esp+8], 10Fh
mov     dword ptr [esp+4], 0
mov     eax, [ebp+ExifData_var]
mov     [esp], eax
call    sub_8048D0F
test   eax, eax
jz     short loc_8048E65 ; Model의 Tag-ID
```

```
loc_8048E65:                ; Model의 Tag-ID
mov     dword ptr [esp+8], 110h
mov     dword ptr [esp+4], 0 ; offset
mov     eax, [ebp+ExifData_var]
mov     [esp], eax
call    sub_8048C82
test   eax, eax
jz     short loc_8048EA1
```

```
loc_8048E29:                ; Make의 Tag-ID
mov     dword ptr [esp+8], 1
mov     dword ptr [esp+4], offset aBadMake ; "Bad Make"
mov     eax, [ebp+arg_0]
mov     [esp], eax
call    sub_80483CB
jmp     short loc_8048ED4
```

```
loc_8048E65:                ; Model의 Tag-ID
mov     dword ptr [esp+8], 1
mov     dword ptr [esp+4], offset aBadModel ; "Bad Model"
mov     eax, [ebp+arg_0]
mov     [esp], eax
call    sub_80483CB
jmp     short loc_8048ED4
```

```
loc_8048EA1:
lea     eax, [ebp+
mov     [esp+0Ch],
mov     dword ptr
mov     eax, [ebp+
```

- 함수 호출 이후의 에러 메시지나 인자로 주어지는 값만 보아도 어디를 비교하는지 쉽게 알 수 있음

Binary Leetness 400

Make(Tag-ID = 010Fh)
필드의 값이 **ddtek**인지
비교하는 루틴

```
mov     [esp], edx
call    find_tag_id    ; 비교함수
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jz      short loc_8048D8A
```

```
mov     dword ptr [esp+8], 400h ; 버퍼 사이즈
lea     eax, [ebp+buffer]
mov     [esp+4], eax    ; 버퍼
mov     eax, [ebp+var_4]
mov     [esp], eax
call    exif_entry_get_value ; libexif 라이브러리 함수
lea     eax, [ebp+buffer]
mov     [esp], eax
call    null_add
mov     dword ptr [esp+4], offset aDdtek ; "ddtek"
lea     eax, [ebp+buffer]
mov     [esp], eax
call    comp_some_strings ; ddtek 문자와 비교
test    eax, eax
jnz     short loc_8048D8A
```

```
mov     [ebp+var_414], 0
jmp     short loc_8048D94
```

```
loc_8048D8A:
mov     [ebp+var_414], 0FFFFFFFh
```

```
mov     [ebp+var_414], 0
```

Binary Leetness 400

Model(Tag-ID = 0110h)
필드의 값이 P200 인지
비교하는 루틴

```
mov     [esp+4], eax
mov     [esp], edx
call    find_tag_id
mov     [ebp+var_4], eax ; tag_id offset
cmp     [ebp+var_4], 0
jz      short loc_8048CFD
```

```
N ㄴ
mov     dword ptr [esp+8], 400h
lea     eax, [ebp+var_404]
mov     [esp+4], eax
mov     eax, [ebp+var_4]
mov     [esp], eax
call    exif_entry_get_value ; libexif 라이브러리 함수
lea     eax, [ebp+var_404]
mov     [esp], eax
call    null_add
mov     dword ptr [esp+4], offset aP200 ; "P200"
lea     eax, [ebp+var_404]
mov     [esp], eax
call    comp_some_strings ; arg0에서 null을 만나기 전까지 문자가 같으면 성공
test    eax, eax
jnz     short loc_8048CFD
```

```
N ㄴ
mov     [ebp+var_414], 0
jmp     short loc_8048D07
```

```
N ㄴ
loc_8048CFD:
mov     [ebp+var_414], 0FFFFFFFFh
```

Binary Leetness 400

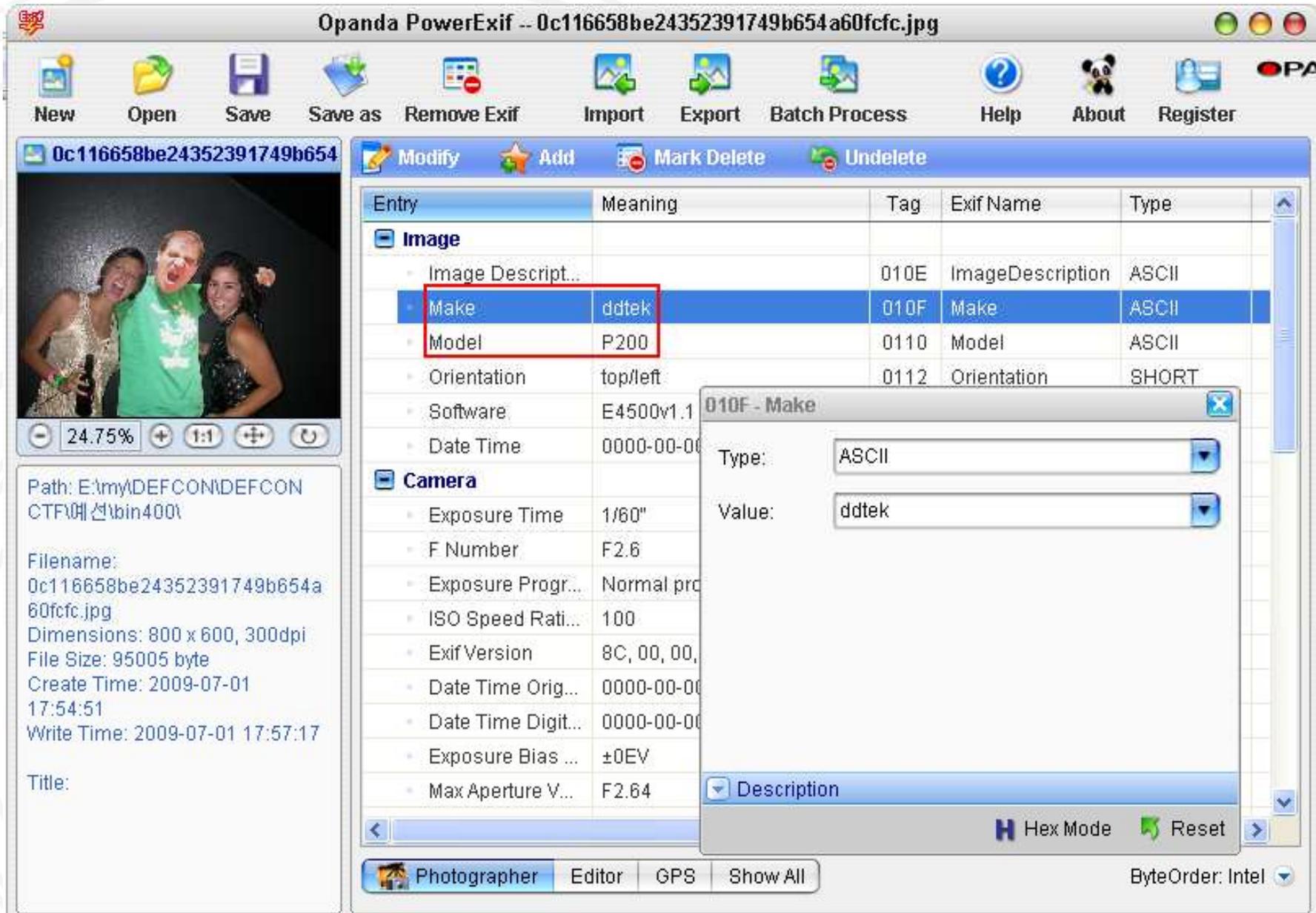
- User Comments의 값을 가져와서 실행하는 루틴

```
loc_8048EA1:  
lea    eax, [ebp+var_10D8]  
mov    [esp+0Ch], eax  
mov    dword ptr [esp+8], 9286h ; User Comments  
mov    dword ptr [esp+4], 2  
mov    eax, [ebp+ExifData_var]  
mov    [esp], eax  
call   sub_8048B73  
lea    eax, [ebp+var_10D8]  
mov    [ebp+var_4], eax  
mov    eax, [ebp+var_4]  
call   eax
```

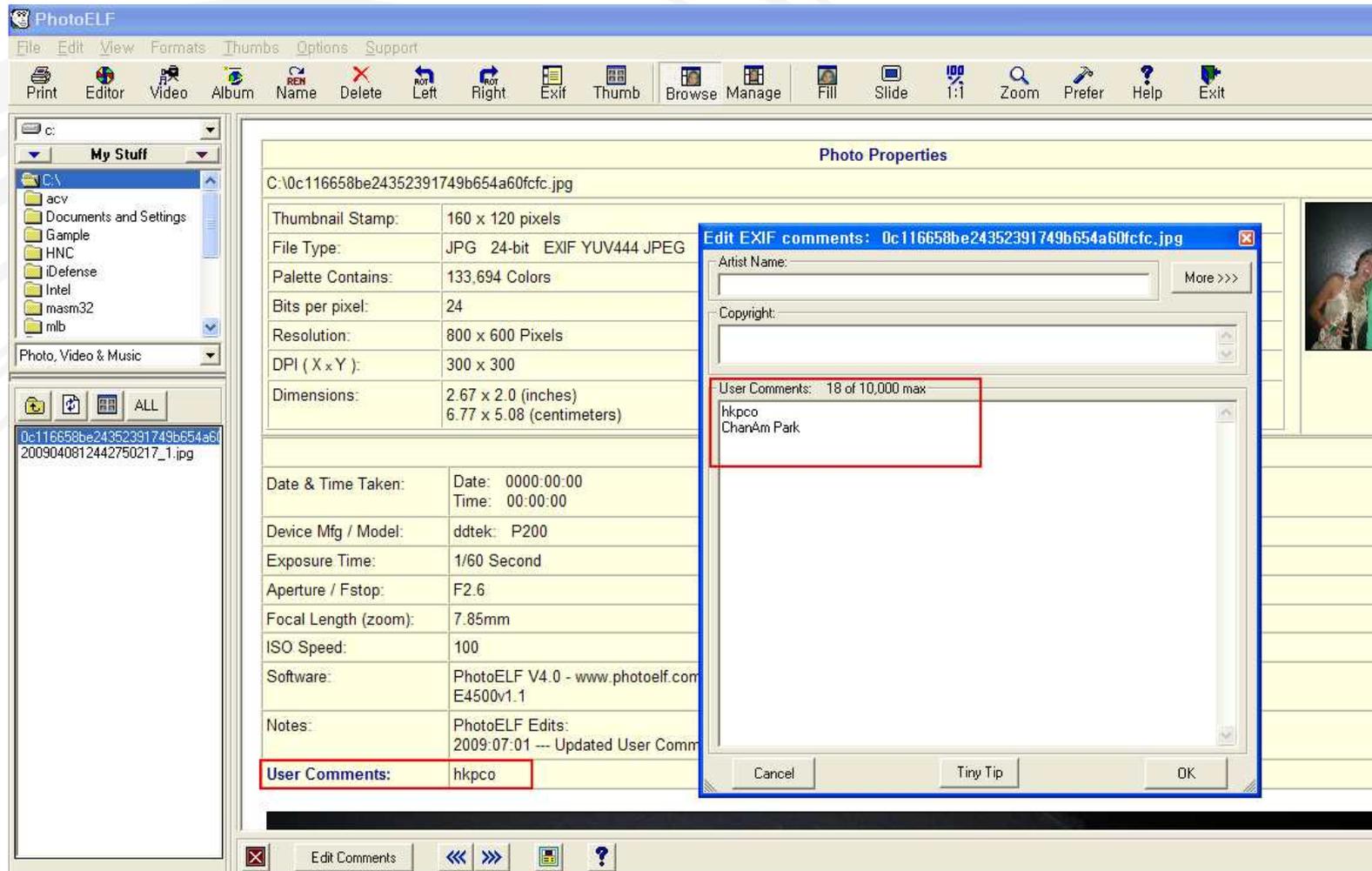
Binary Leetness 400

- Exif의 Make, Model 필드와 User Comments를 각각 *ddtek*, *P200*, *실행을 원하는 코드로* 조작해야 하며, 그러한 방법은 많고 특히, 각종 툴이 잘 만들어져 있기 때문에 쉽게 바꿀 수 있음
- Make, Model은 **Opanda PowerExif**
- User Comments는 **PhotoELF**

Binary Leetness 400



Binary Leetness 400



Binary Leetness 400

최종적으로 다음과 같이 이미지 파일을 조작하여 문제를 풀 수 있음

- Make(010Fh): ddtek
- Model(0110h): P200
- User Comments: 실행을 원하는 기계어 코드

Binary Leetness 500

Help, we lost our assembler and manuals.

Assemble and run the given code on the given interpreter
and you will get a prize.

< 문제에서 제공되는 두 가지 >

- a. 정체모를 어셈블리 코드
- b. 정체모를 바이너리

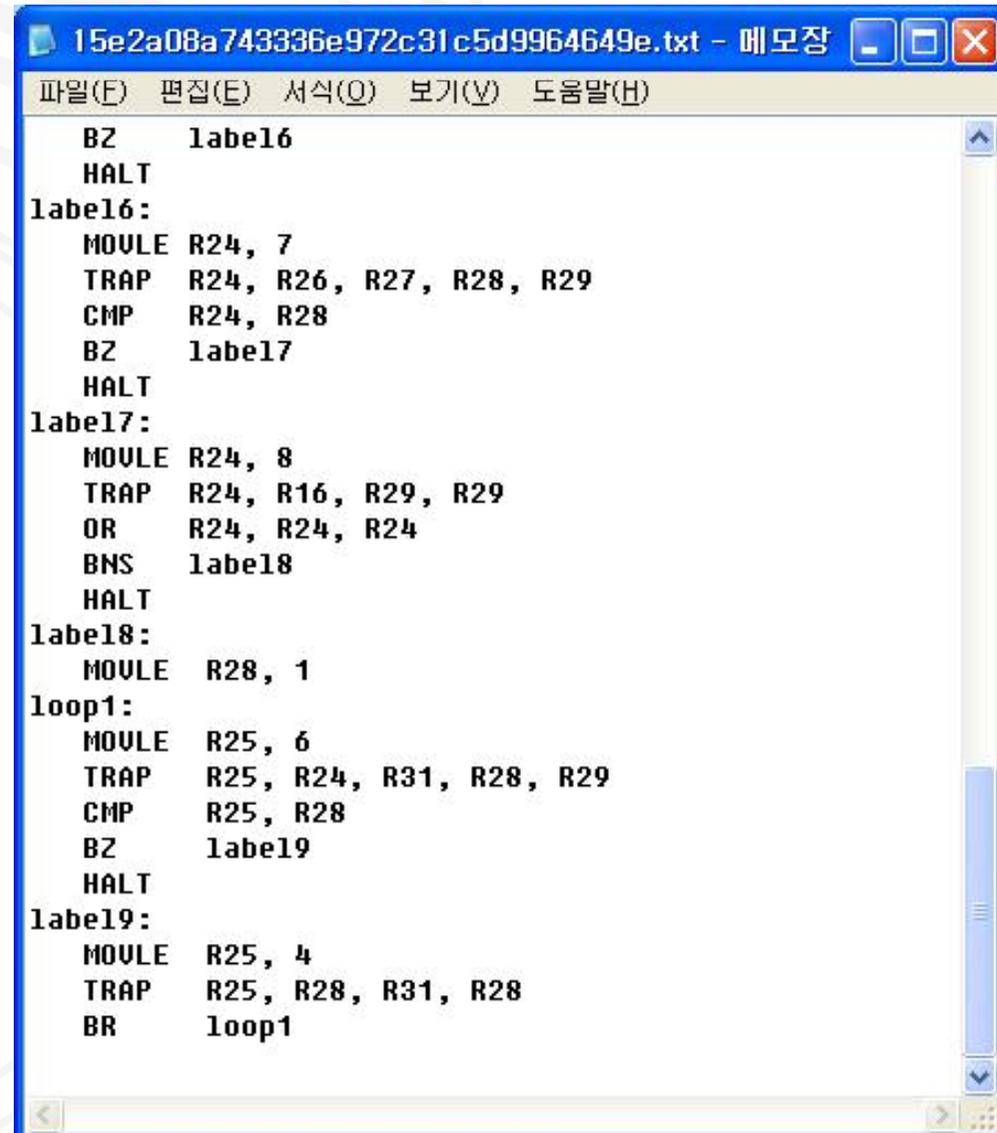
< 이들의 정체 >

- a. 임의로 작성된 가상 어셈블리 코드
- b. 가상 어셈블리 코드를 해석하는 인터프리터

Binary Leetness 500

■ 문제에서 제공된 asm 코드

```
start:
    EOR R29,R29,R29
    MOVLE R28, 1
    ...
TRAP R26, R27, R28, R29
    OR R26, R26, R26
    BNS label1
    HALT
label1:
    MOVLE R16, 5
    ...
MOVLE R28, 1
loop1:
    MOVLE R25, 6
    TRAP R25, R24, R31, R28, R29
    ...
label9:
    MOVLE R25, 4
    ...
```



The screenshot shows a text editor window titled "15e2a08a743336e972c31c5d9964649e.txt - 메모장". The window contains the following assembly code:

```
BZ label16
HALT
label16:
    MOVLE R24, 7
    TRAP R24, R26, R27, R28, R29
    CMP R24, R28
    BZ label17
    HALT
label17:
    MOVLE R24, 8
    TRAP R24, R16, R29, R29
    OR R24, R24, R24
    BNS label18
    HALT
label18:
    MOVLE R28, 1
loop1:
    MOVLE R25, 6
    TRAP R25, R24, R31, R28, R29
    CMP R25, R28
    BZ label19
    HALT
label19:
    MOVLE R25, 4
    TRAP R25, R28, R31, R28
    BR loop1
```

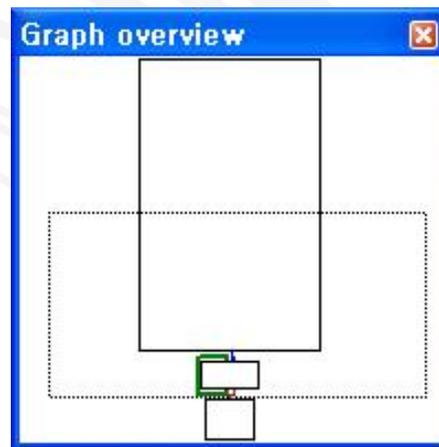
Binary Leetness 500

- 문제에서 제공한 바이너리

```
[hkpc@hkpc bin500]$ file bin500
```

```
bin500: ELF 32-bit LSB executable, Intel 80386, version 1 (FreeBSD),  
statically linked, stripped
```

Binary Leetness 500



Binary Leetness 500

The screenshot shows the IDA Pro interface with the following assembly code in the main window:

```
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 144h
mov     [esp+150h+var_14C], 0
mov     [esp+150h+var_150], offset aBin_dat ; "bin.dat"
call   sub_8049914
mov     [ebp-8], eax
lea     eax, [ebp-128h]
mov     [esp+150h+var_148], 120h
mov     [esp+150h+var_14C], eax
mov     eax, [ebp-8]
mov     [esp+150h+var_150], eax
call   sub_804990A
mov     eax, [ebp-8]
mov     [esp+14Ch+var_14C], eax
call   sub_804990F
call   sub_80498A0
lea     eax, [ebp-128h]
mov     [esp+14Ch+var_148], 120h
mov     [esp+14Ch+var_14C], eax
call   sub_8049850
```

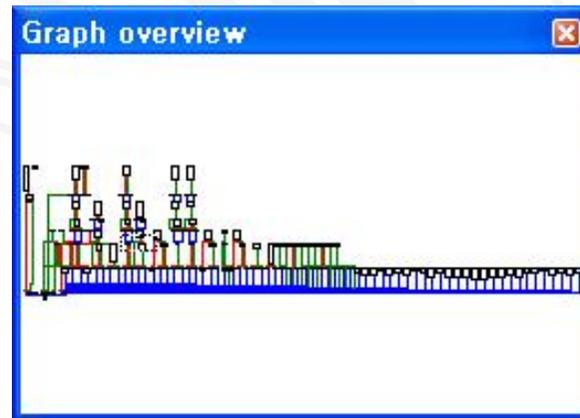
The control flow graph below the assembly shows a loop structure:

```
loc_80480EE:
call   sub_8048110
test   eax, eax
jnz    short loc_80480EE

add    esp, 144h
pop    ecx
pop    ebp
lea   esp, [ecx-4]
retn
```

The status bar at the bottom indicates the current address is 000480AB: sub_8048080+2B. A 'Graph overview' window is also visible on the right side of the interface.

Binary Leetness 500



Binary Leetness 500

IDA - E:\my\WDEFCON\WDEFCON CTF\win500\binary500 - [IDA View-A]

File Edit Jump Search View Debugger Options Windows Help

IDA View-A Hex View-A Exports Imports Names Functions Structures Enums

```
loc_8048A62:
mov     eax, ds:dword_807ABC4
and     eax, 0FBFFFFFFh
mov     ds:dword_807ABC4, eax
048A71

loc_80495F4:
mov     eax, [ebp+var_90]
shr     eax, 16h
and     eax, 1Fh
mov     edx, ds:dword_804AB40[eax*4]
mov     eax, [ebp+var_90]
shr     eax, 11h
and     eax, 1Fh
mov     eax, ds:dword_804AB40[eax*4]
not     eax
and     edx, eax
mov     eax, [ebp+var_90]
shr     eax, 18h
mov     eax, ds:dword_804AB40[eax*4]
not     eax
and     eax, edx
test    eax, eax
js      short loc_8049678

loc_8048181:
mov     eax, [ebp+var_90]
mov     ecx, eax
shr     ecx, 18h
mov     eax, [ebp+var_90]
shr     eax, 16h
and     eax, 1Fh
mov     edx, ds:dword_804AB40[eax*4]
mov     eax, [ebp+var_90]
shr     eax, 11h
and     eax, 1Fh
mov     eax, ds:dword_804AB40[eax*4]
or      eax, edx
mov     ds:dword_804AB40[ecx*4], eax
mov     eax, ds:dword_807ABC4
and     eax, 0FFFFFFEh
mov     ds:dword_807ABC4, eax
mov     eax, ds:dword_807ABC4
and     eax, 0FBFFFFFFh
mov     ds:dword_807ABC4, eax
mov     eax, [ebp+var_90]
shr     eax, 18h
mov     eax, ds:dword_804AB40[eax*4]
test    eax, eax
jnz     short loc_804820C

loc_8048A71:
mov     eax, [ebp+var_58]
mov     eax, [ebp+var_54]
mov     ds:dword_8048A80, eax
short loc_8048A8D

loc_804820C:
mov     eax,
and     eax,
mov     ds:dword_8048A80, eax
shr     eax,
mov     eax,
test    eax,
```

100.00% (3546.2164) (740.271) 00000110 08048110: sub_8048110

The initial autoanalysis has been finished. AU: idle Down: Disk: 116GE Click and drag to delete from selection: Db: Click on edge to jump to its source: Wheel to scroll horizontally

Binary Leetness 500

IDA - E:\Wmy\WDEFCON\WDEFCON CTF\해선\bin500\bin500.idb (bin500) - [IDA View-A]

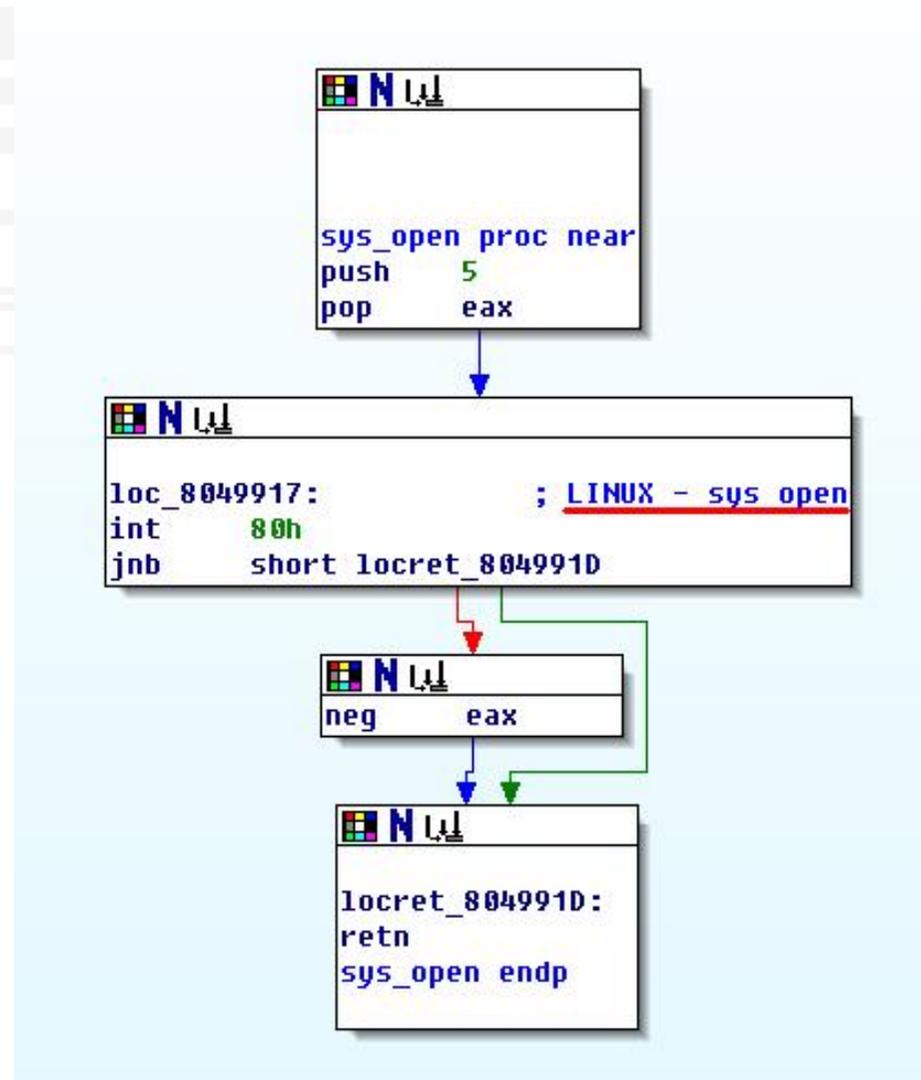
File Edit Jump Search View Debugger Options Windows Help

IDA View-A Exports Imports Problems Names Functions Libraries Types

7,28% (-21,-1997) (674,166) 00000179 08048179: interpreter+69

AU: idle Down Disk: 116GE Click and drag to delete from selection: DbIClick on edge to jump to its source; Wheel to scroll horizontally

Binary Leetness 500



Binary Leetness 500

```
call    initial_clear    ; 초기화 함수
lea     eax, [ebp-128h]
mov     dword ptr [esp+4], 120h
mov     [esp], eax       ; bin.dat에서 읽은 데이터
call    copy_to_alloc_buf ; func( eax, 0x120 )
```

```
loc_80480EE:
call    interpreter
test    eax, eax
jnz     short loc_80480EE ; 0이 아닌 수를 리턴하면
                          ; 정상적으로 interpreting
                          ; 되고있다는 뜻
```

```
add     esp, 144h
pop     ecx
pop     ebp
lea     esp, [ecx-4]
retn
sub_8048080 endp
```

Binary Leetness 500

- Interpreter 함수의 초기 루틴

```
loc_8048139:  
mov     edx, ds:_EIP  
mov     eax, ds:bin_dat_data[edx*4] ; bin.dat의 첫번째 데이터  
mov     [ebp+dat_4byte_value], eax  
lea     eax, [edx+1] ; EIP 증가  
mov     ds:_EIP, eax ; EIP 증가  
mov     eax, [ebp+dat_4byte_value]  
and     eax, 7Fh ; 4byte 중 하위 7bit를 나타냄  
mov     [ebp+var_8C], eax  
cmp     [ebp+var_8C], 2Bh ; 테이블 크기 초과 체크  
ja      loc_804982A
```

```
mov     eax, [ebp+var_8C]  
shl     eax, 2 ; 4byte table  
mov     eax, ds:VM_OPCODES[eax]  
jmp     eax
```

Binary Leetness 500

```
.text:08048139          mov     edx, ds:_EIP
```

```
.text:0804813F          mov     eax, ds:bin_dat_data[edx*4]
```

*; bin.dat의 edx*4 번째 데이터*

```
.text:08048146          mov     [ebp+dat_4byte_value], eax
```

; bin.dat에서 _EIP 오프셋에 해당하는 값 4byte를 저장

```
.text:0804814C          lea    eax, [edx+1]
```

```
.text:0804814F          mov     ds:_EIP, eax
```

; EIP 증가

Binary Leetness 500

```
.text:08048154      mov     eax, [ebp+dat_4byte_value]
```

```
.text:0804815A      and     eax, 7로
```

; 4byte 중 하위 7bit를 나타냄

```
.text:0804815D      mov     [ebp+var_8C], eax
```

```
.text:08048163      cmp     [ebp+var_8C], 2Bh
```

; 테이블 크기 초과 체크

```
.text:0804816A      ja     loc_804982A
```

; 초과한다면, 종료

Binary Leetness 500

```
.text:08048170          mov     eax, [ebp+var_8C]
```

; 하위 7바이트

```
.text:08048176          shl     eax, 2
```

; table의 각 크기가 4byte이기 때문에, shift 연산으로 4를 곱함

```
.text:08048179          mov     eax, ds:VM_OPCODES[eax]
```

```
.text:0804817F          jmp     eax
```

; 해당 오프셋의 가상 코드 테이블로 점프

Binary Leetness 500

■ 가상 명령 테이블의 일부

```
.rodata:080499C8 VM_OPCODES      dd offset __OR                ; DATA XREF: interj
.rodata:080499CC                dd offset loc_8048255
.rodata:080499D0                dd offset loc_8048264
.rodata:080499D4                dd offset loc_80483BE
.rodata:080499D8                dd offset loc_80483EB
.rodata:080499DC                dd offset loc_8048427
.rodata:080499E0                dd offset loc_8048481
.rodata:080499E4                dd offset loc_80484B6
.rodata:080499E8                dd offset loc_8048515
.rodata:080499EC                dd offset __BR
.rodata:080499F0                dd offset loc_8048580
.rodata:080499F4                dd offset __MOVLE             ; 단순 mov
.rodata:080499F8                dd offset loc_80485EB
.rodata:080499FC                dd offset loc_804861E
.rodata:08049A00                dd offset loc_8048775
.rodata:08049A04                dd offset loc_80487B5
.rodata:08049A08                dd offset __EOR
.rodata:08049A0C                dd offset loc_8048946
.rodata:08049A10                dd offset loc_8048982
.rodata:08049A14                dd offset __INC
.rodata:08049A18                dd offset loc_8048AC0
.rodata:08049A1C                dd offset loc_8048AF3
.rodata:08049A20                dd offset __MOVU
.rodata:08049A24                dd offset __TRAP
.rodata:08049A28                dd offset loc_8048FA7
```

Binary Leetness 500

- Interpreter 함수가 성공하면 1을, 그렇지 않으면 0을 반환

```
lea    eax, [ebp-128h]
mov    dword ptr [esp+4], 120h
mov    [esp], eax    ; bin.dat에서 읽은 데이터
call   copy_to_alloc_buf ; func( eax, 0x120 )
```

```
N ㄴ
loc_80480EE:
call   interpreter
test   eax, eax
jnz    short loc_80480EE ; 0이 아닌 수를 리턴하면
                        ; 정상적으로 interpreting
                        ; 되고있다는 뜻
```

```
N ㄴ
add    esp, 144h
pop    ecx
pop    ebp
lea    esp, [ecx-4]
retn
sub_8048080 endp
```

Binary Leetness 500

- VM_OPCODES 테이블에 있는 수십개의 루틴들을 분석 후,
- 주어진 어셈블리 명령에 해당하는 루틴을 찾아야 함

- 다음 단계는 가상 코드 몇 가지를 분석해본 후...

Binary Leetness 500

- 가상 테이블에 있는 increase 명령
(가장 간단한 루틴)

```
__INC:  
mov     eax, [ebp+dat_4byte_value]  
mov     edx, eax  
shr     edx, 1Bh      ; 왼쪽으로 27bit 만큼 shift  
mov     eax, ds:registers[edx*4]  
add     eax, 1        ; 1 증가  
mov     ds:registers[edx*4], eax  
jmp     loc_8049836
```

Binary Leetness 500

.text:08048A9F **__INC:**

.text:08048A9F mov eax, [ebp+dat_4byte_value]

; bin.dat에서 특정 오프셋만큼 떨어진 위치의 4byte 데이터

.text:08048AA5 mov edx, eax

.text:08048AA7 **shr** edx, 1Bh

; 오른쪽으로 27bit 만큼 shift

; 이후, 특정 레지스터를 지정하는 오프셋이 됨

Binary Leetness 500

```
.text:08048AAA          mov    eax, ds:registers[edx*4]
```

; edx 오프셋의 레지스터 값을 eax에 복사

```
.text:08048AB1          add    eax, 1          ; 1 증가
```

```
.text:08048AB4          mov    ds:registers[edx*4], eax
```

; 증가한 값을 레지스터에 이동

```
.text:08048ABB          jmp    loc_8049836
```

; 1을 리턴하는 루틴으로 점프(종료)

Binary Leetness 500

- 특정 함수를 호출해주는 TRAP 명령

```
.text:08048B29 __TRAP:
```

```
.text:08048B29      mov     eax, [ebp+dat_4byte_value]
```

```
.text:08048B2F      shr     eax, 1Bh
```

; bin.dat의 특정 오프셋 4byte 중 1Bh만큼 shift한 값을 eax에 저장

```
.text:08048B32      mov     eax, ds:registers[eax*4]
```

```
.text:08048B39      mov     [ebp+var_9C], eax
```

; 해당 오프셋의 레지스터 값을 eax에 저장 후 var_9C에 저장

Binary Leetness 500

```
.text:08048B3F          cmp    [ebp+var_9C], 0Bh
```

```
.text:08048B46          ja     loc_8048F98
```

; 레지스터 값이 trap table 사이즈인 0Bh(11)보다 크면 종료

```
.text:08048B4C          mov    edx, [ebp+var_9C]
```

```
.text:08048B52          mov    eax, ds:TRAP_TABLE[edx*4]
```

; 우리가 지정한 오프셋을 TRAP_TABLE에 적용하여 eax에 저장

```
.text:08048B59          jmp    eax
```

; 해당 trap 루틴으로 점프

Binary Leetness 500

- Trap 루틴이 저장된 테이블

```
.rodata:08049A78 TRAP_TABLE      dd offset wrap_exit
.rodata:08049A7C                dd offset wrap_open
.rodata:08049A80                dd offset wrap_close
.rodata:08049A84                dd offset trap_read
.rodata:08049A88                dd offset wrap_write
.rodata:08049A8C                dd offset wrap_socket
.rodata:08049A90                dd offset wrap_recv
.rodata:08049A94                dd offset wrap_send
.rodata:08049A98                dd offset wrap_accept
.rodata:08049A9C                dd offset wrap_bind
.rodata:08049AA0                dd offset wrap_listen
.rodata:08049AA4                dd offset wrap_connect
.rodata:08049AA4 _rodata      ends
```

Binary Leetness 500

.text:08048BD5 **wrap_close:**

.text:08048BD5 mov eax, [ebp+dat_4byte_value]

.text:08048BDB mov ebx, eax

.text:08048BDD shr ebx, 1Bh

; 4byte 중 특정 비트를 ebx 레지스터에 저장

.text:08048BE0 mov eax, [ebp+dat_4byte_value]

.text:08048BE6 shr eax, 16h

.text:08048BE9 and eax, 1로

; 4byte 중 특정 비트를 eax 레지스터에 저장

Binary Leetness 500

```
.text:08048BEC      mov     eax, ds:registers[eax*4]
```

```
.text:08048BF3      mov     [esp], eax
```

```
.text:08048BF6      call   sys_close
```

; 특정 비트의 값을 close 함수의 인자로 전달

```
.text:08048BFB      mov     ds:registers[ebx*4], eax
```

; 리턴 값을 특정 레지스터에 저장

```
.text:08048C02      jmp     loc_8049836
```

; 종료

Binary Leetness 500

- <> = 1Bh
- [] = 16h
- {} = 11h
- () = 0Ch, in the traps
- // = 7h, !! = for traps, ** = for jump
- ~~ = 7Fh, jump to table

<32 31 30 29 28> [27 26 25 24 23] {22 21 20 19 18}
(17 16 /15 14 13) !12 *11 10 9 8*!/ ~7 6 5 4 3 2 1~

Binary Leetness 500

```
asm.txt - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

label2:
    PUSH R29          ; 0
    PUSH R29          ; 0
    PUSH R29          ; 0
    MOVL R28, 2        ; R28 = 2
    MOUU R28, 0x5634   ; [R28+2] = 0x5634
    PUSH R28          ; R31 is a esp maybe.
                        ; (sin_addr = R29) -> htonl(INADDR_ANY) == 0
                        ; (sin_family = AF_INET)
                        ; (sin_port = 0x5634) -> ntohs = 13398

    MOULE R25, 9       ; R25 = 9
    MOULE R24, 16      ; R24 = 16
    TRAP R25, R16, R31, R24
// cli_sock(R25) = bind( sockfd_2, st_addr , 16 );

    OR R25, R25, R25
    BNS label13
    HALT

label13:
    MOULE R25, 10
    TRAP R25, R16, R24
// R25 = listen( sockfd_2, 16 );

    OR R25, R25, R25
    BNS label14
    HALT

label14:
    ADDIB R31, R31, 4

    MOUL R28, 0xC58C   ; R28 = 0xC58C
    MOUU R28, 0x586F   ; [R28+2] = 0x586F
    PUSH R28          ; for esp
                        ;
                        ; 111.88.140.197
                        ; 140.197.111.88 <- this one

    MOUL R28, 2        ; R28 = 2
    MOUU R28, 0x3456   ; [R28+2] = 0x3456
    PUSH R28          ; for esp
                        ; (sin_family = AF_INET)
                        ; (sin_port = 0x3456) -> ntohs = 22068
    MOULE R25, 11      ; R25 = 11
    MOULE R24, 16      ; R24 = 16
    TRAP R25, R26, R31, R24
```

Binary Leetness 500

```
sockfd_1(R26) = socket( 2, 1, 0 );
sockfd_2(R16) = socket( 2, 1, 0 );

/*
    (sin_addr = R29) -> htonl(INADDR_ANY) == 0
    (sin_family = AF_INET)
    (sin_port = 0x5634) -> ntohs = 13398
*/
cli_sock(R25) = bind( sockfd_2, st_addr , 16 );

R25 = listen( sockfd_2, 16 );

/*
    (sin_family = AF_INET)
    (sin_port = 0x3456) -> ntohs = 22068
    target address - 140.197.111.88
*/
connect( sockfd_1, st_sock, 16 );
send( sockfd_1, 284, 4, 0 );
// 보낼 바이트 수를 넘겨줌

send( sockfd_1, bin.dat의시작, 284, 0 );
cli_sock(R24) = accept( sockfd_2, st_client, st_client );

loop:
    recv( cli_sock, buf, 1, 0 );
    write( 1, buf, 1 );
```

Binary Leetness 500

< 최종 과정 >

- 문제 바이너리를 분석하여 가상 코드를 기반으로 한 `bin.dat` 작성
- 바이너리에서 `bin.dat`을 해석

`bin.dat`의 수행 과정

- 로컬 서버 오픈
- 문제 서버로 접속하여 `bin.dat` 파일의 크기와 내용 전송
- 대상 서버에서 체크를 거친 후 일치하면 답을 로컬 서버로 전송

+@, funny pwn100

- 문제로 윈도우 바이너리와 서버가 주어짐
- 서버는 해당 바이너리를 실행시킨 것
- 취약성은 간단한 Buffer Overflow
- 하지만 일반적인 Windows Buffer Overflow로는 공격이 안됨
- 이유는...

+@, funny pwn100

- Nmap 등의 툴을 통하여 fingerprinting을 해보면 문제 서버의 OS가 FreeBSD라고 나옴
- 이는 즉, FreeBSD 기반의 서버에서 Wine으로 윈도우 바이너리를 실행시킨 것으로 볼 수 있음
- 결론적으로 공격을 위해서는 FreeBSD 셸 코드를 써야함
- **Wine?** *Wine is a free software application that aims to allow Unix-like computer operating systems on the x86 or x86-64 architecture to execute programs written for Microsoft Windows.*

Thank you!!