Layer7 CTF 고등부 Write up

서울디지텍고등학교
고등부 6위 Gyeongje

(문제파일이 보이지 않아서 기억에 의존해 쓰겠습니다..)

Warm-up
이 문제는 플래그 양식을 알려주는 문제였습니다. 그대로 복사해서 붙여넣으면 정답입니다.

MISC
이 문제는 치환암호 문제입니다.
Shal7 ohz illu dpao aol zjovvs zpujl 2001, dolu Zbuspu Pualyula Opno Zjovvs dhz klzpnuhalk hz aol mpyza PA johyhjalypzapj opno zjovvs pu Zlvbs. Zabkluaz dov hyl pualylzalk pu zljbypaf hyl ibpskpun h jsbi, huk pu aol yhyl hylhz vm ohjrpun, lhjo vaoly ohz olswlk lhjo vaoly huk ohz zabkplk zljbypaf. Aol zfzalthapj jbyypjbsbt huk aol whzzpvu vm aol jsbi tltilyz huk aol zluzl vm ylzwvuzpipspaf ohcl jvuaypibalk av aol opzavyf vm 17 flhyz. Aol Dvysk Dhy Aluupz Johtwpvuzopwz, ovzalk if aol Dvysk'z Tvza Dhualk if Klhao huk Klmluzl Tpupzayf, hyl wbispzolk pu chypvbz mplskz, pujsbkpun aol Pualyuhapvuhs Ohjrpun Jvumlylujl, vynhupglk if aol Pualyuhapvuhs Ohjrpun Jvumlylujl, huk wbispzopun pu chypvbz mplskz, pujsbkpun jvtwbalyz, Dli wvyahsz, huk ltilkklk mvythaz. Avkhf dl ohcl opkklu h mshn. MSHN{Shfly7_pz_nvvkkkkkkkkk}.
위 내용은 파일 안에 있는 데이터 내용인데 맨 아래있는 MSHN{Shfly7_pz_nvvkkkkkkkkk} 가 플래그 같아 보였습니다. 간단히 코딩해서 암호화를 풀어주었습니다.

```c
#include <stdio.h>

int main()
{
    char flag[] = "MSHN{Shfly7_pz_nvvkkkkkkkkk}";
    int a = 'F' - 'M';
    if (a < 0)
        a += 26;
    for (int i = 0; i < sizeof(flag) - 1; i++)
    {
        if (flag[i] >= 65 && flag[i] <= 90)
        {
            if (flag[i] + a > 90)
                printf("%c", flag[i] + a - 26);
            else
                printf("%c", flag[i] + a);
        }
        else if (flag[i] >= 97 && flag[i] <= 122)
        {
            if (flag[i] + a > 122)
                printf("%c", flag[i] + a - 26);
            else
                printf("%c", flag[i] + a);
        }
        else
            printf("%c", flag[i]);
    }
    return 0;
}
```


FLAG{Layer7_is_goodddddddddd}

Reversing ABC문제

ABC문제의 해당 실행파일을 다운받고 PEID 로 확인해보면 C#으로 짜여져 있는 걸 확인할수 있었습니다. 바로 c# 디컴파일 프로그램인 jetbrains dotpeek 으로 열어주었습니다.

```
public class Form1 : Form
{
  public string flag = "AKF@|D$D$D$D$z";
  private IContainer components;
  private TextBox tb_input;
  private Button btn_check;

  public Form1()
  {
    this.InitializeComponent();
  }

  private void btn_check_Click(object sender, EventArgs e)
  {
    string text = this.tb_input.Text;
    bool flag = true;
    if (this.flag.Length != text.Length)
    {
      int num1 = (int) MessageBox.Show("Failure...", "Not Good");
    }
    else
    {
      for (int index = 0; index < this.flag.Length; ++index)
      {
        if (((int) this.flag[index] ^ 7) != (int) text[index])
          flag = false;
      }
      if (flag)
      {
        int num2 = (int) MessageBox.Show("Correct!", "Good");
```

해당 문제 핵심코드 입니다

Flag 변수에 들어있는 "AKF@|DDDDz" 문자열을 xor 7한 값과 Input text값이 동일하면 Correct! 문이 나오는 문제입니다

이 문제도 역시 코딩해서 풀어주었습니다

```
1 flag = "AKF@|D$D$D$D$z"
2 for i in flag:
3     print chr(ord(i) ^ 7),
```

cs

FLAG{C#C#C#C#}

Reversing Ant문제

```
  v5 = 1078348609;
  v6 = 1483236220;
  v7 = 1483892862;
  v8 = 1885890924;
  v9 = 1936287320;
  v10 = 8009838;
  v11 = 0;
  v12 = 0;
  v13 = 0;
  v14 = 0;
  v15 = 0;
  v16 = 0;
  v17 = 0;
  v18 = 0;
  v19 = 0;
  v20 = 0;
  v2 = 0;
  for ( i = 0; i < 5 * IsDebuggerPresent() + 7; ++i )
    ++v2;
  sub_401050("Secret Key : ");
  sub_4010C0("%31s", &v13);
  for ( j = 0; j < strlen((const char *)&v13); ++j )
    *((_BYTE *)&v13 + j) ^= v2;
  v1 = strcmp((const char *)&v5, (const char *)&v13);
  if ( v1 )
    v1 = -(v1 < 0) | 1;
  if ( v1 )
    sub_401050("Nono...\n");
  else
    sub_401050("Correct!\n");
  return 0;
}
```

해당 exe문제의 핵심 함수 모습입니다.
첫번째 반복문에서 안티디버깅 함수를 통해 v2값을 정해주고 있습니다
IsDubbgerPresent() 함수는 디버깅시 1 아닐시 0의 리턴값을 반환해 주기 때문에 v2의 값은 5*0+7 = 7이 되어야 합니다.
후에 문자열을 input 받은 후 input 문자열과 v2를 xor연산 한 값과 v5~v10 문자열들을 비교하고 있습니다. 이 문제도 역시 코딩해서 풀었습니다.

```
flag=[0x41, 0x4b, 0x46, 0x40, 0x7c, 0x63, 0x68, 0x58, 0x7e, 0x68, 0x72
, 0x58,0x6c, 0x69, 0x68, 0x70, 0x58, 0x66, 0x69, 0x73, 0x6e, 0x38, 0x7a]
for i in flag:
    print chr(i ^ 7),
```

FLAG{do_you_know_anti?}

느낀점

생각보다 쉬운문제만 풀었음에도 불구하고 순위권에 오르게 되서 굉장히 놀랐습니다 (아마 선린CTF가 it고CTF랑 정보보호올림피아드와 겹쳐서 그런거 같습니다)
한가지 아쉬웠던 점은 Eazy as Pie 문제를 풀면서 다른 암호화 (shift, xor..) 방식은 역연산으로 풀어나갈 수 있었지만 Sub_400B50 함수 에 있던 SIMD(Single Instruction Multiple Data) 기법은 처음보는 프로그래밍 방식이라서 저부분은 그냥 함정인가?.. 하고 넘어가 삽질을 했었습니다
하지만 플래그가 38문자열 이라는 힌트를 받고 IDA로 동적분석을 해본 결과 SIMD 부분으로 넘어가 input 문자열을 16byte씩 2번 패치해주는 모습을 보고 굉장히 놀랐습니다.. 그당시 새벽이라 제가 비몽사몽한 상태로 똑같이 SIMD 코딩을 해보고 값을 봤는데 계속 다른 결과가 나와서 중간에 포기했던 문제입니다..
처음보는 프로그래밍 처리기법이기도 하고 되게 놀라워서 SIMD 자료를 찾아보고 공부한 후에 문제를 다시 풀어봐야겠습니다.
(아.. 그리고 삽질한 결과로 문의를 한적이 있었는데 그 당시 나온 결과는 SIMD 기법이 함정이라고 생각해서 플래그 문자열을 16글자로 잡고 평문이라고 가정도 안한채 무작정 역연산 했던 값을 답이라고 착각했었던거 같습니다.
귀찮게 해드려서 다시한번 죄송합니다..)