Null@Root – 2차 문제 Key 및 풀이

Author: Cai (박세준: bpak@andrew.cmu.edu)

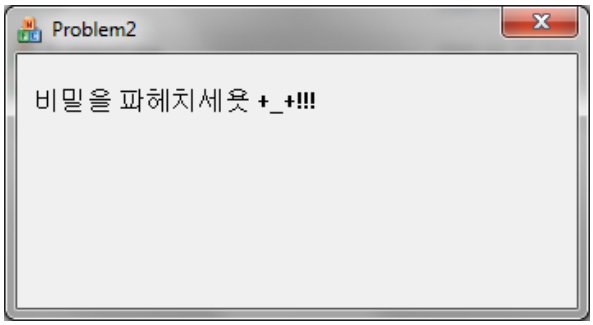
Date: 2009-04-21

---

1. Table of Content

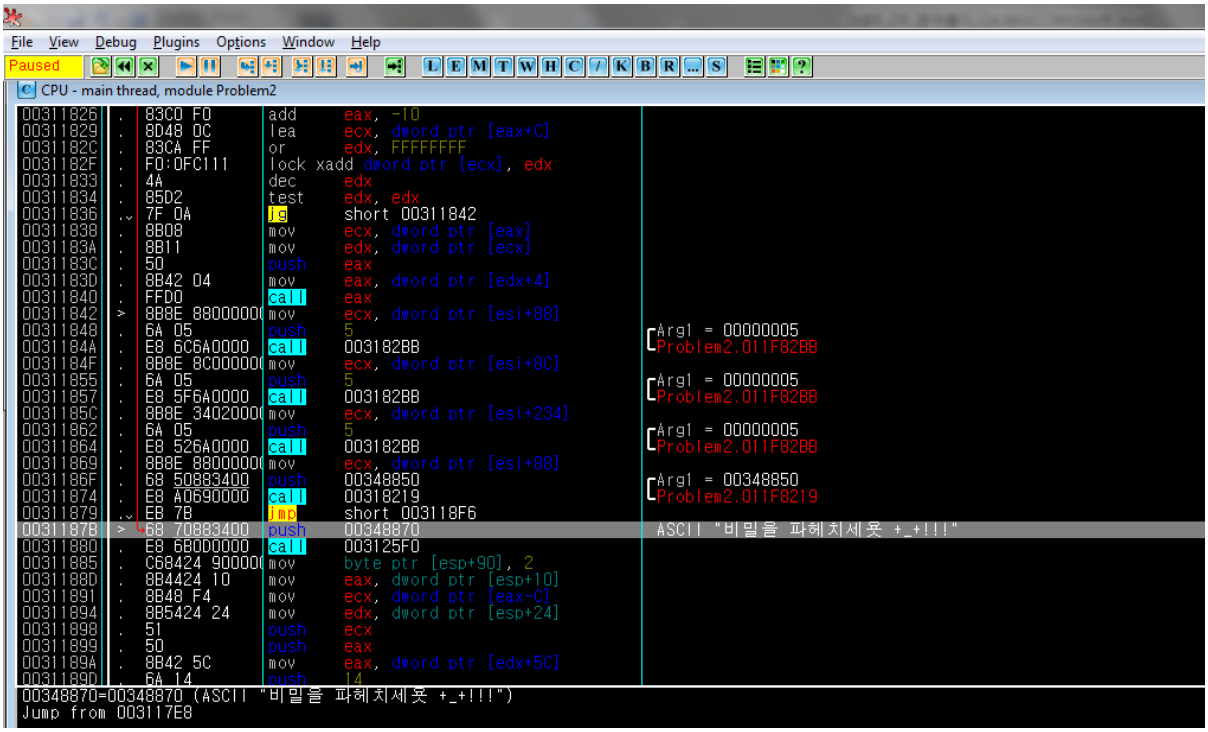
2. Reverse Engineering Target Program

- 주어진 프로그램, Problem2.exe를 실행시켜 보면 다음과 같은 프로그램이 실행됩니다.



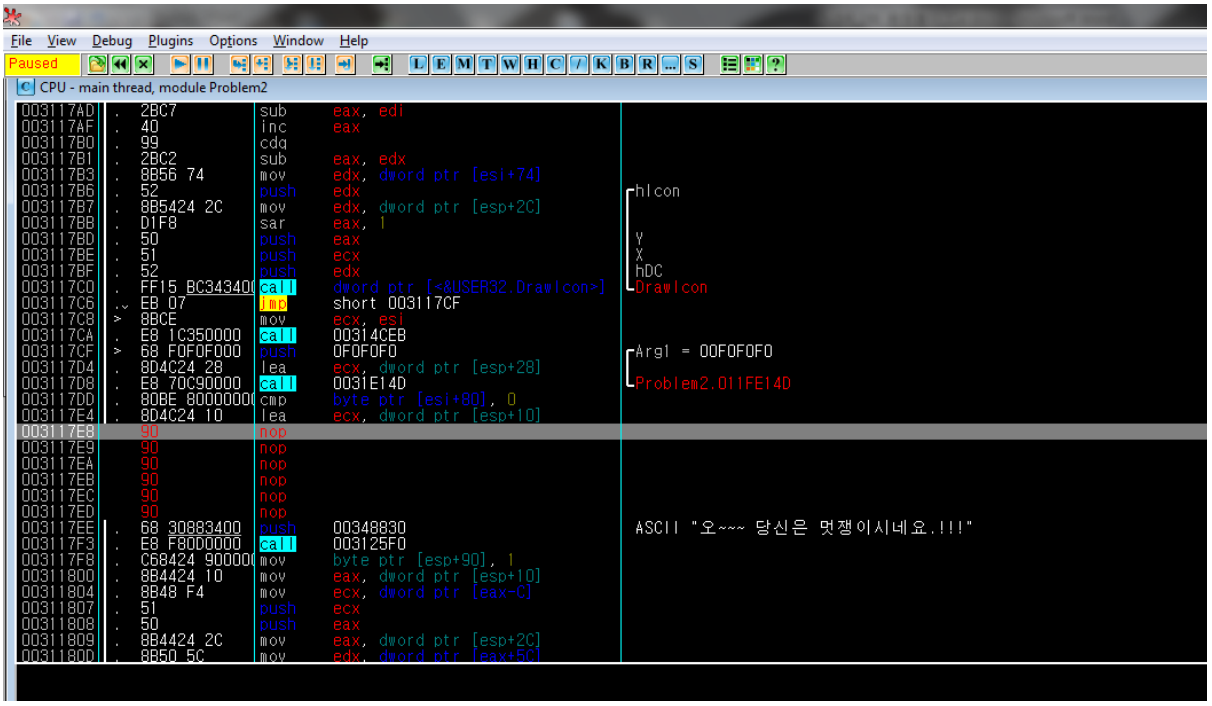
아무 일도 일어나지 않죠.

OllyDbg로 열어서 확인을 해보면,

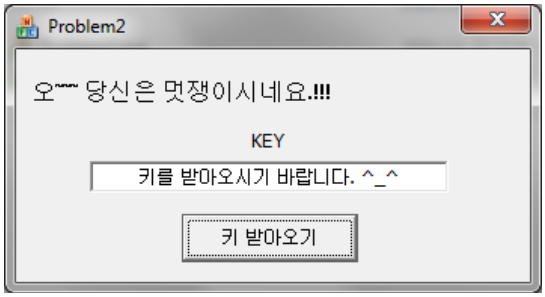


003117E8에서 점프를 하여서 내려오는 것을 확인 할 수 있습니다.

점프를 하지 않아야 우리가 원하는 곳으로 갈 수 있으니, nop처리 해줍니다.



이렇게 패치 된 프로그램을 실행시켜 보면,



이런 귀엽게(?) 변화된 창을 보실 수 있습니다.

키 받아오기 버튼을 누르면,

0193f42ce5a0c62c29a005c61073c74d

위와 같은 총 32자로 이루어진 키를 받을 수 있습니다.

두 글자씩 나누어서 보면

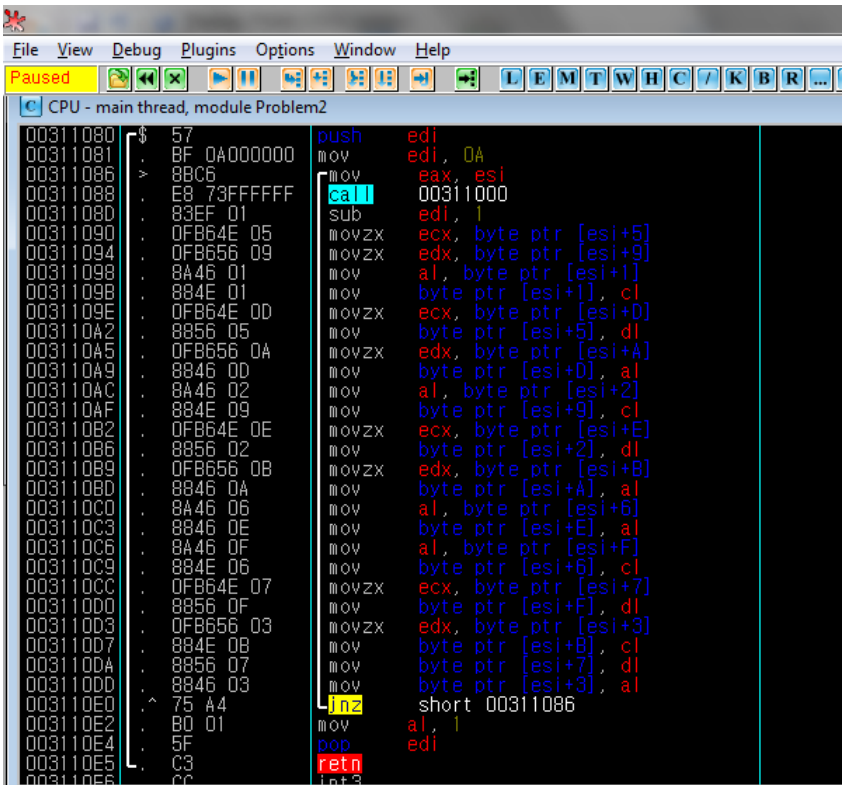
0x01,0x93,0xf4,0x2c,0xe5,0xa0,0xc6,0x2c,0x29,0xa0,0x05,0xc6,0x10,0x73,0xc7,0x4d

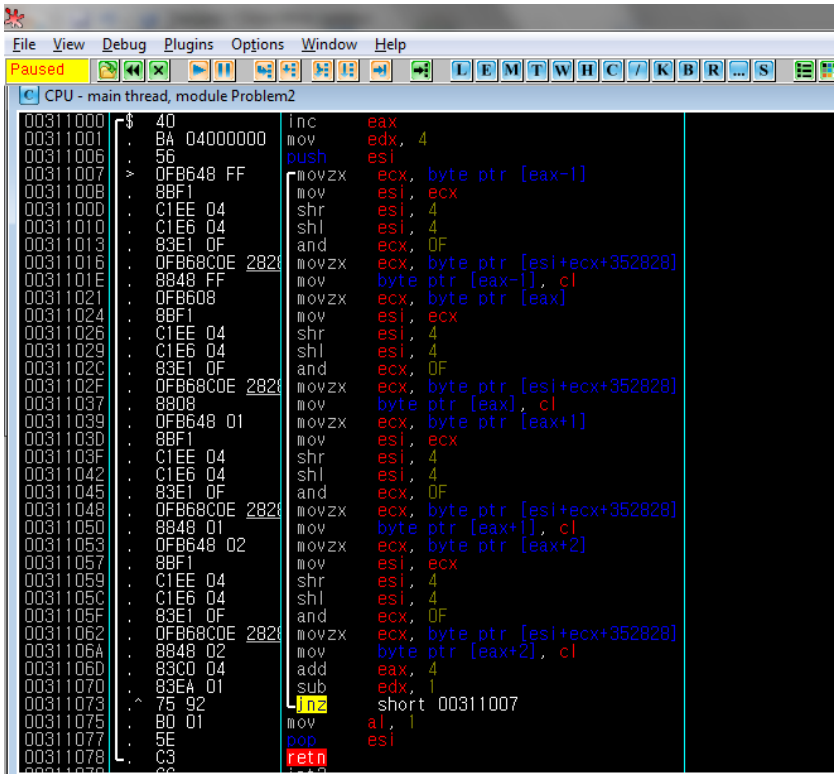
위와 같이 총 16byte의 암호화된 문자열을 확인 하실 수 있습니다.

이 암호화된 문자열을 평문으로 복호화하는 것이 이번 문제의 핵심입니다 :)

3. Encrypt Algorithm Analysis

- 역시, OllyDbg로 NRForever,^_^;;; 라는 문자열을 찾아서 키 받아오기 버튼을 눌렀을 때 어떤식으로 암호화가 되서 넘어가는지 확인을 해보면, 다음과 같은 암호화 알고리즘을 거친 후 보내지는 것을 알 수 있습니다.





- 이제 어디서 암호화 작업을 하는지 알았으니, 어떤 식으로 하는지 알아봐야 합니다.
- 일단 문자열은 00311000를 call하면서 넘어가서, 특정한 룰에 따라서 치환됩니다:
  - 총 16byte를 본다.
  - 현재 byte를 right shift를 4번 한 뒤, left shift를 4번 한다. 이를 esi에 저장한다.
  - 현재 byte를 0x0F와 and operation (&) 을 한 뒤, 이를 ecx에 저장한다.
  - [esi+ecx+table base address]에 있는 byte를 읽어온다. 이를 ecx에 저장한다.
  - ecx에 있는 byte을 현재 byte에 덮어쓴다.
  - 16byte를 모두 치환할 때까지, 위를 반복한다.
- 위의 알고리즘에서 한 가지 눈 여겨 볼 것이 있다면, esi+ecx 부분입니다. Right shift 4번, left shift 4번을 하는 과정에서 esi에는 upper 4 bits가 저장이 되어있고, 0x0F와 and 작업을 하는 과정에서 ecx에는 lower 4 bits가 저장되어있어서, esi+ecx는 현재 byte를 그대로 복원해서 줍니다.
- 위에서 언급된 알고리즘을 10번 반복합니다. 하지만 그냥 반복하는 것이 아니라, 매번 할 때마다 byte의 순서를 다음과 같은 규칙으로 섞습니다 (0-based).
  - X→Y (X번째의 byte을 Y번째로 옮김)
  - 0→0 8→8
  - 1→5 9→D
  - 2→A A→2
  - 3→F B→7
  - 4→4 C→C
  - 5→9 D→1
  - 6→E E→6
  - 7→3 F→B

4. Decrypt Algorithm Analysis

- 어떻게 암호화가 이루어지는지 알았으니, 거꾸로 적용하면 복호화가 됩니다.
- 즉, byte 순서이동을 모두 거꾸로 해주면서, 10번을 다음과 같은 알고리즘으로 실행합니다:
  - 총 16byte를 본다.
  - 현재 byte와 같은 값을 table에서 찾는다. 찾았을 때의 index를 기억해둔다.
    - ◆ 여기서 index는 암호화 루틴에 의하여, esi+ecx가 된다.
  - esi+ecx의 값은 위에서도 말했듯이, 현재 byte가 암호화 되기 전의 값.
  - 현재 byte를 새로 구한 복호화된 byte값으로 덮어쓴다.
  - 16byte를 모두 치환할 때까지, 위를 반복한다.
- 위의 모든 작업을 하고 나면, 우리가 갖게 되는 결과값은 원문이 되겠죠.

5. Decrypter Code

- 다음과 같은 코드를 Java로 작성했습니다.
- 쓰여진 알고리즘은 4번에서 설명한 것과 동일합니다.

<N2R_2nd.java>

```
/**
 *
 * Null@Root 2nd Challenge Decrypter (2009/04/21)
 *
 * @author Cai
 *
 */
public class N2R_2nd {

	// Look-up Table & input data
	static int[] table =
{0x63,0x7C,0x77,0x7B,0xF2,0x6B,0x6F,0xC5,0x30,0x01,0x67,0x2B,0xFE,0xD
7,0xAB,0x76,0xCA,0x82,0xC9,0x7D,0xFA,0x59,0x47,0xF0,0xAD,0xD4,0xA2,0x
AF,0x9C,0xA4,0x72,0xC0,0xB7,0xFD,0x93,0x26,0x36,0x3F,0xF7,0xCC,0x34,0
xA5,0xE5,0xF1,0x71,0xD8,0x31,0x15,0x04,0xC7,0x23,0xC3,0x18,0x96,0x05,
0x9A,0x07,0x12,0x80,0xE2,0xEB,0x27,0xB2,0x75,0x09,0x83,0x2C,0x1A,0x1B
,0x6E,0x5A,0xA0,0x52,0x3B,0xD6,0xB3,0x29,0xE3,0x2F,0x84,0x53,0xD1,0x0
0,0xED,0x20,0xFC,0xB1,0x5B,0x6A,0xCB,0xBE,0x39,0x4A,0x4C,0x58,0xCF,0x
D0,0xEF,0xAA,0xFB,0x43,0x4D,0x33,0x85,0x45,0xF9,0x02,0x7F,0x50,0x3C,0
x9F,0xA8,0x51,0xA3,0x40,0x8F,0x92,0x9D,0x38,0xF5,0xBC,0xB6,0xDA,0x21,
0x10,0xFF,0xF3,0xD2,0xCD,0x0C,0x13,0xEC,0x5F,0x97,0x44,0x17,0xC4,0xA7
,0x7E,0x3D,0x64,0x5D,0x19,0x73,0x60,0x81,0x4F,0xDC,0x22,0x2A,0x90,0x8
8,0x46,0xEE,0xB8,0x14,0xDE,0x5E,0x0B,0xDB,0xE0,0x32,0x3A,0x0A,0x49,0x
06,0x24,0x5C,0xC2,0xD3,0xAC,0x62,0x91,0x95,0xE4,0x79,0xE7,0xC8,0x37,0
x6D,0x8D,0xD5,0x4E,0xA9,0x6C,0x56,0xF4,0xEA,0x65,0x7A,0xAE,0x08,0xBA,
0x78,0x25,0x2E,0x1C,0xA6,0xB4,0xC6,0xE8,0xDD,0x74,0x1F,0x4B,0xBD,0x8B
,0x8A,0x70,0x3E,0xB5,0x66,0x48,0x03,0xF6,0x0E,0x61,0x35,0x57,0xB9,0x8
6,0xC1,0x1D,0x9E,0xE1,0xF8,0x98,0x11,0x69,0xD9,0x8E,0x94,0x9B,0x1E,0x
87,0xE9,0xCE,0x55,0x28,0xDF,0x8C,0xA1,0x89,0x0D,0xBF,0xE6,0x42,0x68,0
x41,0x99,0x2D,0x0F,0xB0,0x54,0xBB,0x16};
	static int[] input =
{0x01,0x93,0xf4,0x2c,0xe5,0xa0,0xc6,0x2c,0x29,0xa0,0x05,0xc6,0x10,0x7
3,0xc7,0x4d};
	static int[] inputcopy;

	public static void main(String[] args) {

		System.out.print("Encrypted:\t");
		for(int i=0; i<input.length; i++)
			System.out.printf("%#x ",input[i]);
		System.out.println();
```

```
		// Decrypt Process
		for(int k=0; k<10; k++)
		{
			inputcopy = input.clone();
			put(1, 5);
			put(2, 10);
			put(3, 15);
			put(5, 9);
			put(6, 14);
			put(7, 3);
			put(9, 13);
			put(10, 2);
			put(11, 7);
			put(13, 1);
			put(14, 6);
			put(15, 11);

			for(int i=0; i<input.length; i++)
			{
				int enc = input[i];
				int dec=0;
				for(int j=0; j<table.length; j++){
					if(table[j] == enc){
						dec = j;
						break;
					}
				}
				input[i] = dec;
			}
		}

		System.out.print("Decrypted:\t");
		for(int i=0; i<input.length; i++)
			System.out.printf("%#x ", input[i]);

		System.out.print("\nIn String:\t");
		for(int i=0; i<input.length; i++)
			System.out.printf("%c", input[i]);
	}

	private static void put(int a, int b)
	{
		input[b] = inputcopy[a];
	}

}
```

<Output>

```
Encrypted:    0x1 0x93 0xf4 0x2c 0xe5 0xa0 0xc6 0x2c 0x29 0xa0 0x5
0xc6 0x10 0x73 0xc7 0x4d
Decrypted:    0x4b 0x65 0x79 0x20 0x69 0x73 0x20 0x57 0x33 0x6c 0x63
0x30 0x6d 0x65 0x54 0x30
In String:    Key is W3lc0meT0
```

즉, 이번 문제에 숨겨진 key는 **W3lc0meT0** 였네요!