Advance RCE L07

2010년 10월 12일 화요일
오전 8:45

CodeEngn Advanced RCE L07

L3m0nTr33
L3m0nTr33.SUR3.org

1. Introduction

Advance RCE L07 문제는 **.NET 으로 compile** 된 Program 이다. 그러므로 **.Net Reflector** 를 사용해서 손쉽게 분석이 가능하다. 문제를 살펴본 결과 약간의 암호학 개념이 들어가는 것을 확인 하였다. Reflector 를 실행시켜보니 **Button 1 을 Click 하였을 때의 Event** 를 가르쳐 주는 함수를 볼 수 있었다.

Disassembler

```
private void button1_Click(object sender, EventArgs e)
{
    string str = "";
    string str2 = "";
    string str3 = "";
    ytrewq ytrewq = new ytrewq();
    if ((((this.textBox1.Text.Length >= 5) && (this.textBox1.Text.Length <= 0x1b)) && ((this.textBox2.Text.Length == 0x1a) && (this.textBox2.Text[8] == '-'))) && (this.textBox2.Text[0x11] == '-'))
    {
        for (int i = 0; i < 8; i++)
        {
            str = str + this.textBox2.Text[i];
        }
        uint num = Convert.ToUInt32(str, 0x10);
        for (int j = 9; j < 0x11; j++)
        {
            str2 = str2 + this.textBox2.Text[j];
        }
        uint num2 = Convert.ToUInt32(str2, 0x10);
        for (int k = 0x12; k < 0x1a; k++)
        {
            str3 = str3 + this.textBox2.Text[k];
        }
        uint fsfsdf = Convert.ToUInt32(str3, 0x10);
        uint num5 = ytrewq.qwerty(dfgsf(this.textBox1.Text));
        uint hashCode = (uint) this.textBox1.Text.GetHashCode();
        fsfsdf ^= hashCode;
        this.yreee[0] = num;
        this.yreee[1] = num2;
        this.yreee[2] = num;
        this.yreee[3] = num2;
        if ((this.vxzzz(this.yreee, this.ewrrr, 0x8ffe2225, fsfsdf) && (this.yreee[2] == hashCode)) && (this.yreee[3] == num5))
        {
            MessageBox.Show("Congratulations, mate!", "Fine!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
        }
    }
}
```

- 여기에서 우리는 **Name (textBox1.Text) 길이가 5 ~ 27 사이**인, **26 자리 B_1-B_2-B_3 형태인 Serial (textBox2.Text)** 이어야 한다는 것을 알 수 있다. 여기서 각각의 **B(lock) 은 8자리**여야 한다는 것도 **textBox2.Text[8], textBox2.Text[0x11] == '-'** 로 알 수 있다.

- num, num2, fsfsdf 는 각각 B_1-B_2-B_3 를 읽어 들여서, 16진수로 표현된 숫자의 String 을 **32Bit 부호 없는 정수로 반환된 값**이다.

Name (textBox1.Text) 의 Hash 값을 구하는 부분을 발견할 수 있다.

```
uint num5 = ytrewq.qwerty(dfgsf(this.textBox1.Text));
uint hashCode = (uint) this.textBox1.Text.GetHashCode();
```

성공을 알리는 MessageBox 부분을 발견 할 수 있다.

```
if ((this.vxzzz(this.yreee, this.ewrrr, 0x8ffe2225, fsfsdf) && (this.yreee[2] == hashCode)) && (this.yreee[3] == num5))
{
    MessageBox.Show("Congratulations, mate!", "Fine!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
}
```

- vxzzz, treee, werrr, fsfsdf 등은 추측을 하지 못하게 일부러 의미없이 만든 것이며, **vxzzz 함수가 이 Crackme 의 핵심**이라는 것을 알 수 있다,

2. Analyze

우선 **vxzzz 함수를 분석**하여 보았다.

Disassembler

```
private bool vxzzz(uint[] rwerqw, uint[] kgtsdfs, uint pgdsfa, uint fsfsdf)
{
  pgdsfa ^= fsfsdf;
  uint num = (pgdsfa % 0x39) - 1;
  uint num10 = rwerqw[0];
  uint num11 = rwerqw[1];
  uint num6 = num;
  uint num2 = pgdsfa << ((byte) (0x61 ^ (num + 0x44)));
  if (num != 0)
  {
    while (num-- > 0)
    {
      uint num5 = num6 / 0x10;
      uint num3 = num10 << ((byte) (num6 / 8));
      uint num4 = num10 >> (3 + ((byte) num5));
      uint num7 = (num6 / 4) + 3;
      uint num9 = num7;
      num7 = kgtsdfs[(int) ((IntPtr) ((num2 >> ((byte) num7)) % 4))];
      uint num8 = num2 + num7;
      num11 -= (((num3 ^ num4) + num10) ^ num8) - num;
      num2 -= pgdsfa;
      num11 -= num;
      num3 = num11 << (((byte) (num9 + 1)) ^ 8);
      num4 = num11 >> (((byte) (((num6 / 2) - num9) + 0x17)) ^ 0x19);
      if (num == num6)
      {
        num11 ^= num;
      }
      if (num == ((num6 / 2) + (num9 ^ 0x1b)))
      {
        num9 = (num3 ^ num4) + (num11 ^ num);
      }
      else
      {
        num9 = (num3 ^ num4) + num11;
      }
      num10 -= num9 ^ (num2 + kgtsdfs[(int) ((IntPtr) (num2 & 3))]);
    }
    rwerqw[0] = num10 ^ 4;
    rwerqw[1] = num11 ^ 7;
    rwerqw[2] = rwerqw[1] ^ ((byte) (((num6 + 1) / 3) - 4));
    rwerqw[3] = rwerqw[0] ^ ((byte) (((num6 - 0x15) + 1) ^ 8));
    rwerqw[0] ^= kgtsdfs[4];
    rwerqw[1] ^= kgtsdfs[5];
    return true;
  }
  return false;
}
```

- 우선 인자로 넘어온 값들을 알아 볼 수 있는 변수로 약간의 연산과 함께 **다시 이름 짓는 것**을 알 수 있다.
  4번째 인자와 3번째 인자를 XOR 연산한 것을 3번째 인자에 대입
  num = 3번째 인자 / 0x39 의 나머지 - 1
  num10 = 1번째 인자 (배열) 의 1번째 값
  num11 = 1번째 인자 (배열) 의 2번째 값
  - 이로 인해, **B_1-B_2 를 연산** 하는 함수라는 것을 알 수 있다.

  num6 = num 과 동일
  num2 = 3번째 인자를 0x61 과 (num + 0x44) XOR 한 값을 Byte 단위로 Shift 연산
  의미상 1번째 인자를 Block , 2번째 인자를 sbox, 3번째 인자를 c, 4번째 인자를 key 로 볼 수 있다.
  - 이는 보통 **대칭키 알고리즘에서 쓰이는 변수들**이다. 대칭키 알고리즘의 설명은 아래에 설명 하겠다.

- 그 후, **num 값 이 0 이 아니라는 조건 하에 while 문을 수행**한다.
  얼핏 살펴보면 **num 6 / 0x10(16)**, num 10 << ((byte)(**num6/8**));, (**num6/4**) + 3, **num6 / 2** 등이 보인다.
  - 우리는 num 값이 **적어도 16으로 나누어져야 한다**는 것은 알 수 있다(ex : **최소인 32로 값을 설정**한다면, 여러 값을 미리 계산 해 볼 수 있다.)

임의로 설정한 **변수 값들을 모두 의미 있는 값으로 두고, 과정들을 정리**하여 보았다.

```
private void vxzzz(...)
{
    uint num = 32;
    uint num10 = rwerqw[0];
    uint num11 = rwerqw[1];
    uint num2 = 1056;
    while (num-- > 0)
    {
        uint num3 = num10 << 4;
        uint num4 = num10 >> 5;
        uint num8 = num2 + kgtsdfs[0];
        num11 -= (((num3 ^ num4) + num10) ^ num8);

        num2 -= 33;
        num3 = num11 << 4;
        num4 = num11 >> 5;
        num9 = (num3 ^ num4) + num11;
        num10 -= num9 ^ (num2 + kgtsdfs[num2 & 3]);
    }
    rwerqw[2] = num11;
    rwerqw[3] = num10;

    return true;
}
private void vxzzz_reversed(...)
{
    uint num11 = <hashcode>;
    uint num10 = <num5>;
    uint num12 = <hashcode> ^ 0x8FFE2204;
    uint pgdsfa = 33;
    uint num2 = 0;
    uint num = 32;
    while (i-- > 0)
    {
        uint num3 = num11 << 4;
        uint num4 = num11 >> 5;
        uint num9 = (num3 ^ num4) + num11;
        num10 += num9 ^ (num2 + kgtsdfs[num2 & 3]);

        num2 += 33;

        num3 = num10 << 4;
        num4 = num10 >> 5;
        num8 = num2 + kgtsdfs[0];
        num11 += (((num3 ^ num4) + num10) ^ num8);
    }
}
```

3. Conclusion

위의 코드를 Compile 하여 실행하면 **해당 하는 Serial 을 획득** 할 수 있다.

답 : 11E051D1