Keygenning andrewl.us’s UPSKiRT crackme

Numernia
Crackmes.de

1. Analyze

General info:
Protection: modular arithmetic

We know from crackme description that crackme is not packed in any way. Also directly we get impressed by cool gfx, cool and original really ☺. Open it in ollydbg scroll down a bit and we see the main loop, the interesting part of it is here.

```
00403E75  |. 8D45 EC        ||LEA     EAX, DWORD PTR SS:[EBP-14]
00403E78  |. 57             ||PUSH    EDI
00403E79  |. 50             ||PUSH    EAX
00403E7A  |. 885C3D EC      ||MOV     BYTE PTR SS:[EBP+EDI-14], BL
00403E7E  |. E8 F1F4FFFF    ||CALL    UPSKiRT.00403374
00403E83  |. 83FF 4A        ||CMP     EDI, 4A
00403E86  |. 59             ||POP     ECX
00403E87  |. 59             ||POP     ECX
00403E88  |. 75 0E          ||JNZ     SHORT UPSKiRT.00403E98
00403E8A  |. 8D45 EC        ||LEA     EAX, DWORD PTR SS:[EBP-14]
00403E8D  |. 50             ||PUSH    EAX
00403E8E  |. E8 D2F5FFFF    ||CALL    UPSKiRT.00403465
00403E93  |. 85C0           ||TEST    EAX, EAX
00403E95  |. 59             ||POP     ECX
00403E96  |. 75 21          ||JNZ     SHORT UPSKiRT.00403EB9
```

This code runs all the time and jumps in the call at 00403E8E when serial length is 0x4A.

In this call it first seperates the serial parts, the parts are s1-s2-s3, where all is 24 chars long. Next

```
004034A0  |> 8A8A F0604000  /MOV     CL, BYTE PTR DS:[EDX+4060F0]
004034A6  |. 33C0           |XOR     EAX, EAX
004034A8  |> 384C05 28      |/CMP     BYTE PTR SS:[EBP+EAX+28], CL
004034AC  |. 74 06          ||JE      SHORT UPSKiRT.004034B4
004034AE  |. 40             ||INC     EAX
004034AF  |. 83F8 18        ||CMP     EAX, 18
004034B2  |.^7C F4          |\JL      SHORT UPSKiRT.004034A8
004034B4  |> 83F8 18        |CMP     EAX, 18
004034B7  |. 0F84 13010000  |JE      UPSKiRT.004035D0
004034BD  |. 42             |INC     EDX
004034BE  |. 83FA 0B        |CMP     EDX, 0B
004034C1  |.^7C DD          \JL      SHORT UPSKiRT.004034A0
```

This checks s1 for the charset “#cRaCkInG4NeWbIeS”, and makes sure there are not duplicates of chars in this charset, for example only one ‘S’ is allowed.

```
004034C5  |> 8A4C05 34      /MOV     CL, BYTE PTR SS:[EBP+EAX+34]
004034C9  |. C0E1 04        |SHL     CL, 4
004034CC  |. 304C05 28      |XOR     BYTE PTR SS:[EBP+EAX+28], CL
004034D0  |. 40             |INC     EAX
```

```
004034D1  |. 83F8 0C         |CMP     EAX, 0C
004034D4  |.^7C EF           \JL      SHORT UPSKiRT.004034C5
```

This loop performs some operations on s1, and builds a 12byte array, this is later used as a big number, call it m.

2. Reversing

This algoritm goes as

$$for(i \to 0; i < 12; i++)$$
$$s_i \leftarrow s_i \oplus (s_{i+12} << 4)$$

s_i then represents a bignumber, I call it m

After that is follows lots of bigint installations. Then it initializes two bignumbers, both primes.

p_1 = 19FBD41D69AA3D86009A968D
p_2 = 1B6F141F98EEB619BC036051

Then the vertification goes

```
00403587  |. E8 B40D0000    CALL    multiply.00404340
0040358C  |. 83C4 40        ADD     ESP, 40
0040358F  |. 8D85 90FEFFFF  LEA     EAX, DWORD PTR SS:[EBP-170]
00403595  |. 50             PUSH    EAX
00403596  |. 8D85 18FFFFFF  LEA     EAX, DWORD PTR SS:[EBP-E8]
0040359C  |. 50             PUSH    EAX
0040359D  |. 50             PUSH    EAX
0040359E  |. E8 9D0D0000    CALL    multiply.00404340
004035A3  |. 8D85 18FFFFFF  LEA     EAX, DWORD PTR SS:[EBP-E8]
004035A9  |. 50             PUSH    EAX
004035AA  |. 8D45 A0        LEA     EAX, DWORD PTR SS:[EBP-60]
004035AD  |. 50             PUSH    EAX
004035AE  |. 50             PUSH    EAX
004035AF  |. E8 3C0B0000    CALL    sub.004040F0
004035B4  |. 56             PUSH    ESI
004035B5  |. 8D85 80FDFFFF  LEA     EAX, DWORD PTR SS:[EBP-280]
004035BB  |. 50             PUSH    EAX
004035BC  |. 8D45 A0        LEA     EAX, DWORD PTR SS:[EBP-60]
004035BF  |. 50             PUSH    EAX
004035C0  |. E8 CB090000    CALL    compare.00403F90
```

It verifies if $(s_3 * p_2) - (s_2 * p_1) = m$ then good serial number
So the task is to generate s_3 and s_2.

$$\left(s_3 * p_2\right) - \left(s_2 * p_1\right) = m$$

$$\frac{m + \left(p_1 * s_2\right)}{p_2} = s_3$$

$$m + \left(p_1 * s_2\right) = p_2 * n$$

Idea is to generate s_2 such that

$$\left(m + \left(p_1 * s_2\right)\right) \equiv 0 (\text{mod } \ p_2)$$

$$\left(p_1 * s_2\right) \equiv \left(p_2 - m\right) (\text{mod } \ p_2)$$

$$\left(p_1 * s_2\right) = \left(p_2 - m\right) + p_2 * h$$

$$x = -h$$

$$y = s_2$$

Then we got the diophantic equation

$$p_2 * x + p_1 * y = 1$$

Then we find by using the extended euclidean algorithm
x=5A875A72F4478758DD6F5FB
y=-5F94CF4E06C11B198137892

(1B6F141F98EEB619BC036051*5A875A72F4478758DD6F5FB) +
(19FBD41D69AA3D86009A968D*-5F94CF4E06C11B198137892) = 1

(1B6F141F98EEB619BC036051*5A875A72F4478758DD6F5FB) -
(19FBD41D69AA3D86009A968D*5F94CF4E06C11B198137892) = 1

Then we multiply whole eq with

$$\left(p_2 - m\right)$$

Since 5F94CF4E06C11B198137892 will always result the same, we can use this in our keygen., then we can compute y by

Key=5F94CF4E06C11B198137892

$$y = -\mathrm{k}ey * \left(\mathrm{p}_2 - m\right) - p_2 * k$$

$$k \in \mathrm{Z}$$

$$s_2 = y$$

I choosed k such that

$$k = \left\lfloor -\left(-key * \left(p_2 - m\right)\right) / \ p_2 \right\rfloor + 1$$

Then we will get a proper value to work with

After finding that, we can compute s_3 easy

$$s_3 = \frac{(m + (p_1 * s_2))}{p_2}$$

3. Done

After I was done I also saw that the crackme seemed to wanted the veritification to result such as, $(s_3 * p_2) - (s_2 * p_1) = -m$, so we just change $(p_2 - m)$ to $(p_2 + m)$ and its ok.

Some serials:
4WScGekeIRIaCnbN#DLJJAWV-0E8815F5E5AED3A0106F0072-
0DC36FCB907A6988FA8F33F9

4#aNcGIeRbeWCnSkIPMYAUIT-15B94785FE5E894831C31DB1-
14934D5A973D5B34E7916186

Greetings to all my friends and to andrewl.us, a nice multimedia indeed.
And sorry for English mistakes and check source for more information, there are also other ways solving this.

Keygen is implemented using miracl, also I hardcoded two chars of the seed, so $m < p_2$ always.

Thanks to everyone who supporting crackmes.de!

Numernia