Solving andrewl.us Shmoocon Crypto Pack

Numernia
crackmes.de

Jul 14, 2010

Brief overview

This crackme pack is really great, there are five crackmes enumered from 1 to 5, crackmes 1,2,3 and 5 are relativly simple but the 4th is a hard problem needed to solve. This text will decribe all crackmes in order. Note that all five crackmes have two GetDlgItemTextA in a row (name/serial), the analyze starts at that point for each crackme.

1 crypto1

This crackme is no problem if you have basic knowledge about polynomial equation matrixes. The checking procedure starts at $403D77$. Now, watch closely at the start of the loop, it performs a and operation with your serial and a table value. The most simple way to parse this is to just basicly see every operation as done over 64bit, even though that seems bit wierd at start. However the nested loop parse the and'd result and adds all cofficients. So yes, this is just a equation system over $GF(2)$. You could view it as:

$$\begin{pmatrix} x_0v_0 + x_1v_0 & \cdots & x_{39}v_0 \\ x_0v_1 + x_1v_1 & \cdots & x_{39}v_1 \\ \vdots & \ddots & \vdots \\ x_0v_{39} + x_1v_{39} & \cdots & x_{39}v_{39} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{39} \end{pmatrix}$$

Where x_i is bits from the input polynomial(serial) and v_i is bits from the table. All you need todo is to view name bits as polynomial cofficients, put those bits on the right side and solve the matrix with Gaussian Elimination.

2 crypto2

This crackme also uses polynomial maths to verify the serial, again serial is 64bit. The checking procedure starts at $403D57$. Note the two(one) constants initialized $0xA348FCCD93AEA5A7$. Again the procedure is simple and is actually just a multiplication of two polynomials modulo $0xA348FCCD93AEA5A7$. How to notice? the loop is extremely obvious with its shifting of bits and jumps. Besides, check the comparing, it does check if the result is 1, this is even more obvious to make a test to input just 1 as serial, you'll see your name is appearing in the compare instruction. Anyway, the crackme checks:

$name \cdot serial \bmod 0xA348FCCD93AEA5A7 == 1$

($0xA348FCCD93AEA5A7$ as polynomial $x^{63} + x^{61} + x^{57} + x^{56} + x^{54} + x^{51} + x^{47} + x^{46} + x^{45} + x^{44} + x^{43} + x^{42} + x^{39} + x^{38} + x^{35} + x^{34} + x^{32} + x^{31} + x^{28} + x^{25} + x^{24} + x^{23} + x^{21} + x^{19} + x^{18} + x^{17} + x^{15} + x^{13} + x^{10} + x^8 + x^7 + x^5 + x^2 + x + 1$.)

So to produce a serial you just need to convert your name bytes to a 64bit number, seen as a polynomial and perform

$name^-1 \bmod 0xA348FCCD93AEA5A7 = serial$, which could be calculated with the Extended Euclidean algorithm.

3 crypto3

This is probably the easiest crackme in the whole package and takes around 5-10 minutes to solve. Its basicly only a RSA protection with small parameters, the powmod(blackbox analyze or just follow the call line-per-line to recognize it) call is located at $00403E73$. It checks:

```
if ( powmod(serial,0xDEADBEEF,0x7FBF89F8CD4152C5) == name) { serial_good(); }
```

so we find $0xB4BDEC63 \cdot 0xB4F0C8B7 = 0x7FBF89F8CD4152C5$ and calculate $d = 0xDEADBEEF^{-1} \bmod (0xB4BDEC63 - 1)(0xB4F0C8B7 - 1) = 0x7181F6EA6026F297$.

so to generate a serial we need to do

```
powmod(name,0x7181F6EA6026F297,0x7FBF89F8CD4152C5) = serial
```

Now lets move forward to the hardest crackme.

4 crypto4

This crackme comes with source and contains some basic name/serial routines and a huge equation system. Firstly, A huge thanks to Dcoder which helped me with this crackme next what happends is that the crackme xors your name with a constant

```
0xDEADBEEFCAFFFFFF
```

(Note that this value is different in the source and the exe)

Then the low 24bits gets removed by a right shift. After this, the serial gets converted to a bit-vector(only 40 bits.. crackme rejects serials over 40 bits). Your serial is in x_i and next is the huge equation system. Hum, so what is this? Well basiclly it is just a huge equation systems over GF(2), its alot of AND and XOR operations, you are probably aware that AND is seen as multiplication and XOR as addition/subtraction in GF(2).

The result of each calculations is the bit vector y, which is later compared to the name we xor'ed in the start. So basicly we need to find a x_i vector that generates our name bits. So the approach is to use the Groebner algorithm to solve the equation system so all equations gets 0, the obvious solution for this is off course all 0, but if you just extend all equations with $+\ bit\_from\_name_i$ then apply the Groebner algorithm, it should find the solution pretty fast.

$equation_i = equation_i + name\_bit_i$

Also worth to note here is that there are two ways to exploit this crackme:
1. You could create a name that's in the xoring procedure becomes 0, the charachters wont decode properly in this text editor but you could just print them with ALT+number in the name-edit box of the crackme:

```
{0xFF, 0xFF, 0xFF, 0xCA, 0xEF, 0xBE, 0xAD, 0xDE}
```

Obviously, the serial for this name would be 0.

2. Since the name bits is shifted 24 places, you could generate a serial for the constant xored value that remains after this. Since for all names below 3 (3*8 bits) chars will be shifted out automaticly and the name will always get the same bits, solving that scheme gets the serial

```
000000975801D8EE
```

Which works for all names with three or less chars.

A Magma script to generate serial is included.

5 crypto5

So for the last one is some very basic modular arithmetics. The checking call starts at $00403ECD$ and directly you see a few push'es of constants and few call, you can simply compare these first two calls from the recent crackme that it is a powmod. Just check the stack values and see what it is computing, it is done over 64 bit integers. The third call is simply a multiplication call modulo $0xFFFFFFFB$ (4294967291) or -5 as it is shown =).
So the math goes

$serial_1 * serial_2^{2953572069^{4294967289}}$
$\equiv serial_1 * serial_2^{12685495422059050941}$
$\equiv name(\mod 4294967291)$
So simply choose a arbitary integer $serial_2 : (0 \leq serial_2 \leq 2^{32} - 1)$

Then calculate for $serial_1$:

$serial_1 = (serial_2^{12685495422059050941})^{-1} \mod 4294967291)$
This could be done very simple:

$serial_2^{2953572069^{4294967289-1}} \equiv serial_2^{12685495422059050941^{\phi(4294967291)-1}}(\mod 4294967291)$

And since $\phi(4294967291){-}1 = 4294967289$: $serial_2^{2953572069^{42949672892}} \equiv serial_2^{2953572069}(\mod 4294967291)$

so obviously all you need todo is to perform: $serial_1 = (serial_2^{2953572069} * name) \mod 4294967291$
Or even simplier, just set $serial_2 = 1$, then any name that fits into 4 bytes will obviously have the serial $name$00000001, for example:

```
num - 006D756E00000001
```

6 End

A keygenerator written in C++ is included for crackme 1,2,3 and 5, the keygen for crackme 4 is included as a MAGMA script.

6.1 Example serials

```
crypto 4
```

```
andrewl - 59ED48701D
numernia - 7FF66D5F07
test - B71CB2390
```

6.2 Thanks

Best greetings to: andrewl, cyclops, encrypto, dcoder, anicona, kilobyte, simonzack, artif, mr.haandi, hmx0101.
Special thanks to andrewl.us for this crackme pack and to Dcoder.