RSA Studying and Reversing

{*Written by* **Evilcry**}
{evilcry@reteam.org}

[RET]
[http://www.reteam.org]
[http://evilcry.altervista.org]
[irc.azzurranet.org #crack-it]

{**Difficulty:** Intermediate}

INTRODUCTION

This article is derived from a fusion of my experience and some other articles on the web.

TOOLS

HIEW
OllyDbg
RSTool 2
Calc.exe (a decrypting tool like PowMod)

TARGET LOCATION

http://www.google.com

ABOUT THE TARGET

Lockless 3 CM (if you want to play a bit)

RSA OVERVIEW

I have been studying the RSA cryptosystem for some time now for two fundamental reasons. One because it is a great source for gaining knowledge in math&cryptology, and two, because you encounter RSA in many protection systems. Initially I did not know that this was RSA!. RSA is a public_key_cipher designed at M.I.T. on 1978 thanks to the collaboration between Ron **R**ivest and Adi **S**hamir and Les **A**ndleman (indeed RSA is derived from their names). This is an asymmetric cryptosystem which bases is "power" on a series of features that the prime numbers have. This system initiated a large field of use in computer security. Now we will take a look at an hypothetical crypting session, keep your eyes

open! This is mainly a mathematical discusion. The first step to crypt a message is:

n=p*q

N as we can see from the previous equation, is a product between **p** and **q** that are two prime numbers. With some analysis we can affirm that if P and Q are small values, then the security will be greatly reduced, so it is best to use a module N that is quite large.
Now we take a look at Ô(n).
We can do this with:

Ô(n)=(p-1)*(q-1)

This is an implementation or more precisely an extension of the Euler Theorem. Pratically *Ô(n)*, is a number RELATIVELY prime to *(p-1)*(q-1*). But at this point someone could pose the question: "What is the real meaning of the term **relatively prime*?". A number is relatively prime to another when these two numbers have no factor in common, exept obviously 1. For example: 8 and 21 aren't prime numbers, but 8 and 21 will be relatively prime to another number..which is obviously 1. the number is 1 because this is the only common factor between 8 and 21. The factors of 8 are (1,2,4,8), the factors of 21 are (1,3,7,21). Hopeful that clears up the concept of "relatively prime".

Another step and we have "finished". Very easly **E** could be retrived from:

e =Ô(n)

E represents the key (or rather the public Exponent) thanks to which our message will be crypted. So E should be a prime number. The decryption key (D) could be obtained from:

d = e^(-1) mod ((p-1)*(q-1)) ------->So we should know 'e', 'p' and 'q'

Ok, now we know in general how a session of RSA works, as you have seen during this session there are some values that we know and others that we should discover (in the case of an attack). Now we can build a table that will help us.

P and Q	Are prime numbers	Not known
N	N product of p*q	Know
Ô(n)	Ô(n)=(p-1)*(q-1)	Not know

e	Crypto key	Not know
d	Decrypt key	Known

In this table we also add X (the plaintext) and Y (the ciphertext). Now we are into the heart of the algorithm!, the famous:

C=M^e mod n

Where M is the Message to crypt and C is the Crypted message. I think that you are all able to obtain the inverse formula, but here it is:

M=C^d mod n

We are at the end of our RSA overview. As you have noticed, this is a discussion based mainly on the math, however I tried not to include the factorisation systems as the Method of the Elliptic Curve (a bit slow with the big numbers). There are also Pollard's Monte Carlo Algorithm, Continued Fraction Algorithm, Trial Division (this is the oldest system that i know). If we want to perform operations as the trial division, we can use the Miracl libraries.

PRACTICAL RSA

In this section we will begin to use RsaTool2 (of The Egoiste TMG) which will be very useful for the reversing of RSA targets. Firstly, we need to set the NumberBase of our RsaTool to 10 (indeed we work in base decimal), and next we will verify if the formula previously studied is true:

n=p*q

In the field "Modulus N" insert the number 47 and next press the button "Factor N". At this point we agree that this is a prime numeber, also 53 is a prime number. So we retrieve N=2491 from the multiplication of 847*53). Now if we insert 2491 in "Modulus N" and factor this number, we will reobtain P and Q!. Now we have P and Q, and if we analyze the formula:

d = e^(-1) mod ((p-1)*(q-1))

We can see that we should obtain from this formula the decryption key D. To obtain "D" we need only of the Public Exponent "E" and with P and Q, there is a specific button which is "Calc D". So we also know the decription key, if we want to decrypt some text, we can easily use the formula:

M=(C^d) mod n

If the numbers considered are little, we can make some attempt of encryption/decryption just using only Win calc, but the best is to write a quick program which uses the Miracl libraries.

THE FACTORIZATION (ADENDUM)

This little section, is not of fundamental importance to understand RSA, but it could help you to understand which are the major problems that can emerge in an hypothetical RSA Attack. As I hope you understanded, to be able to overcome a code protected with RSA we need to FACTORIZE the modulus N. If this number is not so big there aren't problems, but if N is really big we need to use probabilistic algorithms. The algorithms for the factorisation are divised in two groups:

Deterministic algorithms

and

Montecarlo's algorithms (which are based on a semi-casual choice of the initial data)

The deterministic algorithms are divised in three subgroups:

-Algorithm that from each successful attempt returns, a proper divisor (prime or not).

-Algorithm that from each successful attempt ALWAYS returns a prime divisor.

-Algorithm that from each succesful attempt returns divisors common to the starting number.

The Euclidean theorem, is the most used deterministic attack. The most used formula is

x=(x.a)(x.b)/(x.a.b) mod N

For example, N=777 if we divide this num by 3 we have 259 (which is >32), 259 could be divised for 7, and we obtain 37 (which is <72). The prime divisors of 259 are obviously (3,7,37). If you try to factorize this with RsaTool2 you will obtain these same values. This kind of deterministic algorithm could be good (for good you should think "speed of resolution") for little numbers, but for big numbers Montecarlo's algorithms are used, which are probabilistic.

REVERSING CORNER

Now it is time to see something in practice. Firstly we will begin with a simple crackme, Lockless's 3CM that you can download from the Lockless web site. Firstly we put a bpx GetDlgItemTextA and Getwindowtexta. After sice pops, we should be here:

```
Reference To: USER32.GetDlgItemTextA

004137FE Call dword ptr [00428910]

00413804 jmp 00413819 ;Jump down

Referenced by a (U)nconditional or (C)onditional Jump at Address:

00413819 pop ebp

0041381A ret 000C
```

After the call to getdlgitemtexta and the ret we will find ourselves here:

```
004029B0 push FFFFFFFF

004029B2 push 00419E18

004029B7 mov eax, dword ptr fs:[00000000]

004029BD push eax

004029BE mov dword ptr fs:[00000000], esp

004029C5 sub esp, 00000650

004029CB push esi

004029CC push edi

Possible StringData Ref from Data Obj ->"9901" ; This is a very
important value

004029CD push 004200DC

004029D2 lea ecx, dword ptr [esp+000000E4]

004029D9 call 00401130 ;In this call the number 9901 is copyed to
another location

Possible StringData Ref from Data Obj ->"12790891"; Also this num
is really important
```

```
004029DE push 004200D0

004029E3 lea ecx, dword ptr [esp+1C]

004029E7 mov dword ptr [esp+00000664], 00000000

004029F2 call 00401130 ;In this call is trasfered 12790891

Possible StringData Ref from Data Obj ->"8483678" ; Another very
important value

004029F7 push 004200C8

004029FC lea ecx, dword ptr [esp+00000274]

00402A03 mov byte ptr [esp+00000664], 01

00402A0B call 00401130 ;Idem come sopra

Possible StringData Ref from Data Obj ->"5666933" ;Last but not
least value

00402A10 push 004200C0

00402A15 lea ecx, dword ptr [esp+000001AC]

00402A1C mov byte ptr [esp+00000664], 02

00402A24 call 00401130 ;Is called here the same call

00402A29 mov edx, dword ptr [esp+00000668] ;Relative address of
the serial

00402A30 or esi, FFFFFFFF ;

00402A33 mov edi, edx ;| Common method used to retrive the length
of a string

00402A35 mov ecx, esi ;|

00402A37 xor eax, eax ;|

00402A39 mov byte ptr [esp+00000660], 03 ;|

00402A41 repnz ;|

00402A42 scasb ;|

00402A43 not ecx ;|

00402A45 dec ecx ;|
```

```
00402A46 cmp ecx, 0000000E ;if the serial isn't Eh (14 chars)

00402A49 jne 00402BB2 ;the jump to beggar off, else go

00402A4F xor ecx, ecx

00402A51 mov al, byte ptr [ecx+edx] ;char relative to the serial

00402A54 cmp al, 30

00402A56 jl 00402BB2 ;Jump to beggar off, if is less tha 30h (0)

00402A5C cmp al, 39

00402A5E jg 00402BB2 ;Jump to beggar off, if is greater than 39h
(9d)

00402A64 inc ecx ;Inc the counter

00402A65 cmp ecx, 0000000E

00402A68 jl 00402A51 ;If is less than Eh, go to the next cycle
```

A bit of analysis is necessary to fully understand this piece of code. At the beginning a serial is taken, and not the name. This should make you think that an algorithm will encrypt/decrypt our serial. In this case things are a bit easier because we know that this could be RSA. Next we can see the presence of four fixed numbers, and immediately we think of E and N.

I'll remind you of the formula:

C=M^e mod n

0042a51 starts a routine that checks if the serial inserted is a number, if isn't a number then jump to an error message. From this cycle we also understand that the NumberBase is 10. At this point is it best to leave the routine-analysis that follows the previous routine. (It's a bit long and boring). But I will try to summarize what happens:

The serial should be 14 chars long, but during the routine this number (14) will be divised into two equal parts each one of 7 characters. Each piece of the serial is now applied a series of operations. This operation also involves the number that we have previously seen (9901 and 12790891). Finally these two parts of the serial are united for the final check. From the operations that happen we can undestand that 12790891 represents the modulus N and that 9901 is the public exponent E. Now our work is to decrypt these two parts of the serial. With a bit of tracing through this routine, we can easly understand that the values 5666933 and 8483678 are the two correct parts of the serial, but

obviously crypted. So we can build a scheme of the current situation:

-We know the correct but crypted serial, in other words we know C

-We know the modulus N, that was used to crypt our serial.

-We know E (with E and P,Q we can found D).

We have all we need for a decrypting session. To decrypt the two parts of the serial we need to have further C, also N and D. If you remember N is N=P*Q, so all we need is to factorize N. In RsaTool2 we insert the supposed N and next check "Factor N", but our prog tell us that the number entered is prime. If this numer is prime it isn't N (which is composed by the multiplication of two primes), but can be our E. Now we retry to factorize 12790891 from which we obtain P (1667) e Q(7673). At this point we have E,P and Q and we can retrive the decryption key D, that derives from this formula:

d = e^(-1) mod ((p-1)*(q-1))

The value of D is 10961333, so finally we have all elements to be allowed to apply

M=C^d mod n

A hint for you: don't try to use win's calc!!, there are specific tool to decrypt this number.

M1=(8483678^10961333) mod 12790891

M2=(5666933^10961333) mod 12790891

From which we obtain:

M1=7167622

M2=3196885

These are the two parts of the decrypted serial. If you remember our routine: break the serial in two equal parts, so we make the inverse Correct_Serial=(M1+M2). That finally is 71676223196885.

HOW TO RECOGNIZE RSA

To recognize RSA in a protection system is not always simple because it could have many various forms, or in the worst case could be readapted. But I thinked to make a list of the common features that a routine *could have*:

-The serial is encrypted with C=(M^E) mod N, where M is our serial.

-As consequence we should see two fixed values, the modulus N and the public exponent E.

Now I wanted to insert a piece of code here that was taken from a crackme, that could clarify your ideas:

```
0040118D sub_40118D proc near

0040118D mov ebx, eax ;In eax we have our serial

0040118F mov ecx, dword_4030A8 ;in ecx is copyed a constant value

00401195 mov esi, dword_4030A4 ; in esi another costant value

0040119B mov eax, 1

004011A0 loc_4011A0:

004011A0 cdq ;ConvertDoubleToQuad, before a division

04011A1 mul ebx ; ebx*eax

004011A3 div esi ;division between esi and eax (in esi there is a costanr value)

004011A5 mov eax, edx ;Reminder in eax

004011A7 loop loc_4011A0 ; loop these operations, this cycleis repeated 1109 times, indeed in ecx there is 1109!
```

after there is a comparision between the correct serial, and the serial (crypted) that we typed

VALUES:

004030A4= 0EAD2C511h so 3939681553

004030A8=455h so 1109

This routine is very easy. In ebx we have our typed serial, which is multiplied for eax (first time, this contains 1) and next is divised for esi (esi = 3939681553). In the next step it is taken the remainder of the division (in other words they are performing a MOD operation!!). All these operation are iterated 1109 times!. With a bit of math analysis we can see what really happens:

eax=eax * ebx

repeated 1109 times, which means:

eax=(ebx ^ 1109)

next we have eax = eax / esi , but only the reminder is taken, and we can consider it as:

eax MOD esi (don't forget that esi is 3939681553)

So finally we see this , C=(Serial ^ 1109) MOD 3939681553

As you can see, this is only an easy introduction to what RSA can really do. I hope that this little article could help you to understand and recognize RSA.

REFERENCES

-"Cenni di Aritmetica Superiore per la Crittologia", of Marco Frigerio

-Other docs on the web.

GREETS

To all my reversing friends, especially Quequero, Ironspark, kratorius, LonelyWolf, ZaiRoN, AndreaGeddon , Misanthropic, Kreatief