How break RSA-1024 ?
Asprotect 1.0/1.1/1.11c
- BETA v3 : DON'T SHARE IT -

by Amenesia//TKM!

Mais le vice n'a point pour mère la science,
Et la vertu n'est pas fille de l'ignorance.
Théodore Agrippa d'Aubigné

1 Introduction

In order to learn basis i decided to study some well-known protections and their weakness. The first is Asprotect and the weakness discovered by Recca few years ago. We know that Recca has been able to break Asprotect registration scheme using a weakness in the random generator so let's go...

> ASProtect - the system of software protection of applications, designed for quick implementation of application protection functions, especially targeted for software developers. ASProtect is designed for such specific tasks as working with registration keys and creation of evaluation and trial application versions.

2 Asprotect 1.0/1.1 [*built-in key*]

When a project is created several parameters are generated and stored in a file called project_name.aspr. The more important are A, D, E and N which are numbers in base 64 (written from right to left).

A = INDtrZliM4t...0czFJpN42UQ==
D = U2atlST1lQ...kHcbIGwJU8=
E = EQAAAAAA...AAAAAAAA=
N = X7lD2zsvq3QW...ha6mOrdvULEAM8=

Then when a registration key is computed:

1 - H1 = RipeMD-160(A)
2 - H2 = MD5(Registration Information—H1)
3 - Key = RSA(D,N, [H2—Registration Information—H1])

Key will look like:

PYgt/87koSvbYPluc+/crrilfWI+ssZSU7UhgCLmK3D1C+x+
EX9n7ukwM5sKmI+nsH66V7L28BFTziNz5DOPLRHAqnI11wN5
Nd/dm0Esw20mm66V7L28BFTziNz55DOP4kzt+bie/rW4grgG
+e8/hsIuotMqUXguWKBnOXsoQ89Kg92T0MkB4FCZYuZQo=

So this scheme seems secure...

2.1 How D, N and E are created ?

The routine is in a DLL called RSAVR.dll:

1- Generate two prime numbers P and Q (512-bits)
1.1- Create a random number 512-bits
1.2- Find the nearest prime number (very slow)
2- Compute $N = P * Q$
3- Compute $D = E^{-1} phi(N)$ (with $E = 17$)

So if we can find P and Q we will be able to factorize N.

2.2 How P and Q are created ?

```
unsigned long _rand()
{
    Seed *= 214013;
    Seed += 2531011;
    return( ((Seed >> 16) & 0x7FFF) );
}

unsigned long RandInt()
{
  for(i=0;i<4;i++)  {   rval = ((rval << 8) +_ rand()); }
  return(rval);
}

Seed = ( time() + ThreadId()) xor TickCount();
for(ri=0;ri<16;ri++) { BigNumber1[ri] = RandInt();  }
BigNumber1[ri] = BigNumber1[ri] xor C0000000h;
P= nextPrime(BigNumber1);

for(ri=0;ri<16;ri++) { BigNumber2[ri] = RandInt();  }
BigNumber2[ri] = BigNumber2[ri] xor C0000000h;
Q= nextPrime(BigNumber2);
```

So if we can guess *time() + ThreadId()) xor TickCount()* value, we will be able to find P and Q...

2.3 Attack

I don't know how Recca did to find the right seed but it seems that the only way is to brute-force it (there are 2^{32} possibilities):

1 - Choose a seed
2 - Compute $BigNumber1$ and $BigNumber2$
3 - Find $P = nextPrime(BigNumber1)$ and $Q = nextPrime(BigNumber2)$
4 - If $N! = P * Q$ then try with an other seed...

But primality testing is time consuming so it's impossible to check quickly the 2^{32} differents seeds.

2.3.1 Reduce space size

The first improvement is based on the fact that ThreadId and TickCount are, the most of the time, quite smalls. So the hightest bits of the seed are fixed by time(). Consequently, we can check only seeds whose value is near from the releasing date of the protected software.

2.3.2 Improve the algorithme

Primality testing is a very slow step so we have to find an algorithm to check if a seed is the right one without computing the real value of P and Q. The trick is that nextPrime() function modify only the lowest dword of BigNumber1 and BigNumber2.

$P = BigNumber1 + \Delta_1$ with $\Delta_1 \leq 2^{64}$
$Q = BigNumber2 + \Delta_2$ with $\Delta_2 \leq 2^{64}$
$N = P * Q = (BigNumber1 + \Delta_1) * (BigNumber2 + \Delta_2)$
$N = BigNumber1 * BigNumber2 + \Delta_3$ with $\Delta_3 \leq 2^{(512+64+2)}$
So: $N_{high} = BigNumber1_{high} * BigNumber2_{high}$

Algorithm:

1- Choose a seed
2- Compute $BigNumber1_{high}$ and $BigNumber2_{high}$
3- If $N_{high}! = BigNumber1_{high} * BigNumber2_{high}$ then choose an other seed
4- Compute $BigNumber1$ and $BigNumber2$ (to avoid collisions)
5- Find $P = NextPrime(BigNumber1)$
and $Q = NextPrime(BigNumber2)$
6- If $N! = P * Q$ then choose an other seed

2.4 Example

Asprotect 1.1 is protected with Asprotect 1.1. Hard to find a nicer exemple to check if our algorithme is efficient or not...

N and E values are stored in the protected software:

> N = EB1D4EADA4815F6277519791BFFA8B4C0B872D1C436515AB
> D9572B22BF6A03FECB4E5CC49AF1EE35C31344617A1210663056
> 90529B9CE7F13ED2D37CD7034A3EDD096853EC61243BCCAC5A58
> 800B0330A4DD85E9AA237F2F2AE60CA049B1D2777B2E0C5FF51E
> 058382A86C3EC12F7AB41642022772FF2A2D3DBA704725702199
>
> E = 17

Asprotect was released in october 2000 so we can choose 39000000h as minimal seed value. We find 398BBB72h (collision) then 399BACC4h in 1 minute (1 hour to check the 2^{32} possibilities):

> D= l8F1EGKSQWCw9Et5klCpkm9/TIQFw0xOxibd+bQNndzGYoIX
> 4PmHXcdZtN3VWRQfuYS/cLeEf0i+kG3Cd7kaqKCkBO3xiAFgZMf
> vW8D+bov+AfjDICITq5/Lhex7PykLGtUNnH8LSsmIDSWqldwX3Q
> 9o8U4HcJSjSJIfS4bumc=

3 Asprotect 1.11c

Quickly A.S. publish a new version of Asprotect in order to patch this little problem. But it seems that until now nobody break it... :)

The random number generator is:

```
unsigned long _rand()
{
    Seed = ( time() + ThreadId()) xor TickCount();
    Seed *= 214013;
    Seed += 2531011;
    return( ((Seed >> 16) & 0x7FFF) );
}

unsigned long RandInt()
{
  for(i=0;i<4;i++)
   {  rval = ((rval << 8) +_ rand()); }
```

```
  return(rval);
}

for(ri=0;ri<16;ri++) { BigNumber1[ri] = RandInt();  }
BigNumber1[ri] = BigNumber1[ri] xor C0000000h;
P= nextPrime(BigNumber1);

for(ri=0;ri<16;ri++) { BigNumber2[ri] = RandInt();  }
BigNumber2[ri] = BigNumber2[ri] xor C0000000h;
Q= nextPrime(BigNumber2);
```

So a new seed is computed for each dword. Here again we need to write a brute-forcer. But we have the same problem how find rights seeds without performing primality test ? And how check the $4 * 2^{512}$ possibilities for each BigNumber ?

3.1 Reduce space size

GetCurrentThreadId is constant, GetTickCount and time() are based on elapsed time... but the generation is so fast that GetTickCount and time() are constants during the generation of a BigNumber, the same Seed is computed for each dword (but the Seed is different for each BigNumber because of primality testing step) and consequently _rand() has always the same value which is inferior to 07FFFh. So we have only $2^{15} * 2^{15}$ couple of seed instead of $4 * 2^{512} * 2^{512}$, with:

$$BigNumber1 = [AorC0000000h]AAAA \dots AAAA$$
$$BigNumber2 = [BorC0000000h]BBBB \dots BBBB$$

3.2 Improve

Each BigNumber has a known structure, we will use this information to guess if a seed is wrong or not. Indeed if the result of the division of the highest dwords of N by the highest dwords of $BigNumber1$ has a structure which looks like (B_or_C0000000h)BB, we can bet that this seed permits to generate P and so $Q = N/P$. Now there are only 2^{15} to check...

$$P = [AorC0000000h]AAAA \dots AAAA + \Delta_1$$
$$Q = [BorC0000000h]BBBB \dots BBBB + \Delta_2$$
$$N = BigNumber1 * BigNumber2 + \Delta_3$$
So: $N_{high}/BigNumber1_{high} = [BorC0000000h]BBBB$

1- Choose a seed (that is to say a _rand())...
2- Compute $BigNumber1_{high}$
3- If $N_{high}/BigNumber1_{high}$ doesn't have the right structure try again
4- Compute $BigNumber1$
5- Find $P = NextPrime(BigNumber1)$
6 - Compute $Q = N/P$

3.3 Example

Asprotect 1.11c is protected with Asprotect 1.11c and:

N = C7D6E57A0D1C3A9752618FB497A6E4D1DCEC39EF22318F0C
6776E429ACBC3946F2018E643746E3817C8C389EC1D18DBC0716
E2D94C5C37F691A18D13D6E6F122D03D00090AF7AAEBC5B255CE
806D00B13B27AB93F5E25676B09D01596B57AC3C2612571EE0CD
02019B87ACE4564257C710FD02A9CBB7AD8C8672586F416D536D

Using the algo described before we found, after 1s (3s to check the 2^{15} possibilities), that the right _rand() is 5454542D. So:

P = D454542D5454542D...
Q = F0F0F088F0F0F088...

And:

D = HV/nbSNxR14Tvhm4bHRVey+U+qdbHQk8Q+BPfBrY
qZYMa14KmBhtGX4flkK+gVoGGX23485UMFwdxMMux5Aw
DtEsU+ZTzlQmvNX5zEuDRVg/1jZJGc7NIBltCVy+sOt+
iVqzBnopoHPQHrNGzDkr/615Ch40ns4iIWp3i7PbRs

4 Conclusion

Funny, isn't it ? Ok, it is a very old software but it is a good exemple of how a scheme can be poorly implemented.

If you want to talk about protection, or any other interesting piece of code, feel free to contact me at ?[at]?[dot]?. See you soon for new adventures...
PS: Greets to $x \in \{TKM!, FFF, CoRE, GoD, UCF, DAMN, TMG, \ldots\}$.