Snake Oil Crypto:

How I stopped to worry and started to love crypto

Team CIRCL

2019/12/06

D4 project

Jean-Louis Huynen

Outline

- Use-Case: RSA,
- First Hands-on: Understanding RSA,
- Snake-Oil-Crypto: a primer,
- Second Hands-on: RSA in Snake-Oil-Crypto,
- D4 passiveSSL Collection,
- Interactions with MISP.

Understanding RSA

RSA Basics

Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman in 1977:

- asymmetric crypto system,
- can encrypt and sign,
- messages are big numbers,
- encryption is basically multiplication of big numbers,
- creates a *trapdoor permutation*: turning x in y is easy, but finding x from y is hard.

RSA - Use with openssl

- **Hands-on**:

  ~/hands–on/UsingRSA

- Decrypt message.bin
- generate a new private key,
- generate the corresponding public key,
- use this new key to encrypt a message,
- use this new key to decrypt a message.

RSA "BY HAND"

run: sage rsa.sage at the folder's root:

```
PlainText is: 1234567890
p = random_prime(2^32) = 2312340619
q = random_prime(2^32) = 2031410981
n = p*q = 4697314125248937239
phi = (p−1)*(q−1) = 4697314120905185640
e = random_prime(phi) = 2588085603940229747
d = xgcd(e,phi)[1] = −2102894211931680277
Does d*e == 1?
 mod(d*e, phi) = 1
CipherText y = power_mod(x, e, n) = 1454606910711062745
Decrypted CT is: 1234567890
```

With only one key

Several potential weaknesses:

- Key size too small: keys up to 1024 bits are breakable given the right means,
- close p and q,
- unsafe primes, smooth primes,
- broken primes (FactorDB, Debian OpenSSL bug).
- signing with RSA-CRT (instead of RSA-PSS)

With a set of keys

Several potential weaknesses:

- share moduli: if $n_1 = n_2$ then the keys share p and q,
- share p or q,

In both case, it is trivial to recover the private keys.

Breaking small keys[1]

- **Hands-on**:

  ```
  ~/hands-on/SmallKey
  ```

- what is the key size of smallkey?
- what is n?
- what is the public exponent?
- what is n in base10?
- what are p and q?

Let's generate the private key: using p, then using q.

[1] https://www.sjoerdlangkemper.nl/2019/06/19/attacking-rsa/

- **Hands-on**:

```
~/hands-on/ClosePQ
```

- use Fermat Algorithm[2] to find **both p and q:**

```
def fermatfactor(N):
  if N <= 0: return [N]
  if is_even(N): return [2,N/2]
  a = ceil(sqrt(N))
  while not is_square(a^2-N):
    a = a + 1
  b = sqrt(a^2-N)
  return [a - b,a + b]
```

[2] http://facthacks.cr.yp.to/fermat.html

Researchers have shown that several devices generated their keypairs at boot time without enough entropy[3]:

```
prng.seed(seed)
p = prng.generate_random_prime()
// prng.add_entropy()
q = prng.generate_random_prime()
n = p*q
```

Given n=pq and n' = pq' it is trivial to recover the shared p by computing their **Greatest Common Divisor (GCD)**, and therefore **both private keys**[4].

"They cracked cracked about 13000 of them"

[3] Bernstein, Heninger, and Lange: `http://facthacks.cr.yp.to/`
[4] `http://www.loyalty.org/~schoen/rsa/`

- **Hands-on**:

```
~/hands-on/SharedPrimeFactor
```

- Read README.txt, you have a challenge to solve :
  - the *answers* folder should be left alone for now,
  - *scripts* contains scripts that may be useful to solve the challenge,
  - *attempts* may hold your attempt are generating private keys.
  - *bgcd-bd.sage* contains Daniel J. Berstein's algorithm for computing RSA collisions in batches.

Hands-on: Exploiting Weaknesses in RSA
– at bigger scale –

Snake Oil Crypto[5] - Problem Statement

We reckon that IoT devices **are often the weakest devices** on a network:

- Usually the result of cheap engineering,
- sloppy patching cycles,
- sometimes forgotten–not monitored (remember the printer sending sysmon?),
- few hardening features enabled.

We feel a bit safer when they use TLS, but we what you now know about RSA, should we?

[5] https://github.com/d4-project/snake-oil-crypto

Snake Oil Crypto - GCD

In Snake-Oil-Crypto we compute GCD[6] between:

- between certificates having the same issuer,
- between certificates having the same subject,
- on keys collected from various sources (PassiveSSL, Certificate Transparency, shodan, censys, etc.),
- python + redis + postgresql [7]

"Check all the keys that we know of for vendor X"

[6] using Bernstein's Batch GCD algorithm

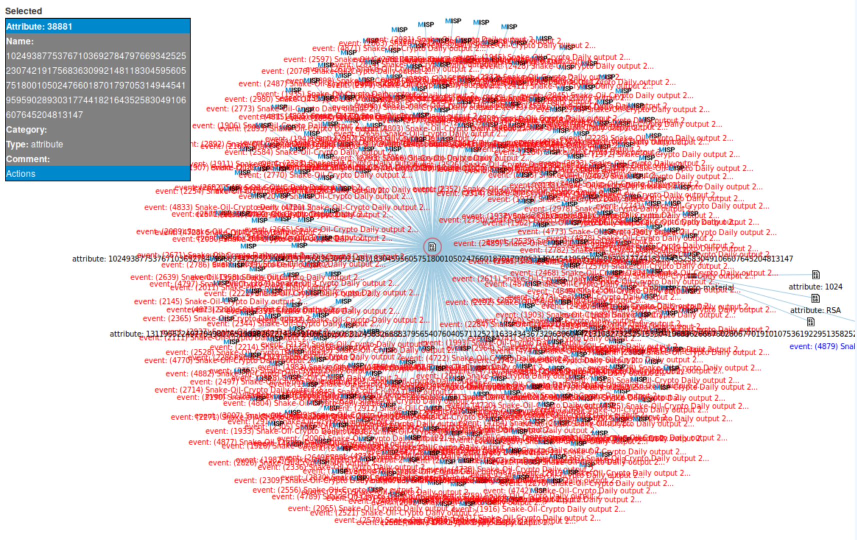
[7] `https://github.com/D4-project/snake-oil-crypto/`

Snake Oil Crypto - GCD

Quick Demo:

- Let’s check how strong are the RSA keys in our database...
- check some results on https://misp-eurolea.enforce.lan
- how bad can it be?
- do you find some vendors we should notify?

Snake Oil Crypto - MISP feed



The MISP feed:

- **Allows** for checking automatic checking by an IDS on hashed values,
- **contains** thousands on broken keys from a dozen of vendors,
- **will be accessible upon request (info@circl.lu).**

In the future:

- **Automatic** the vendor checks by performing TF-IDF on x509's subjects,
- **automatic** vendors notification.

- **Hands-on**:

```
~/hands–on/TLSinspection
```

- open stripped.pcap
- what is the admin password?
- bummer, it's encrypted,
- what is the admin password?

D4 - full chain demo.

- ✓ sensor-d4-tls-fingerprinting [8]: **Extracts** and **fingerprints** certificates, and **computes** TLSH fuzzy hash.
- ✓ analyzer-d4-passivessl [9]: **Stores** Certificates / PK details in a PostgreSQL DB.
- snake-oil-crypto [10]: **Performs** crypto checks, push results in MISP for notification
- lookup-d4-passivessl [11]: **Exposes** the DB through a public REST API.

[8] github.com/D4-project/sensor-d4-tls-fingerprinting
[9] github.com/D4-project/analyzer-d4-passivessl
[10] github.com/D4-project/snake-oil-crypto
[11] github.com/D4-project/lookup-d4-passivessl

Get in touch if you want to join/support the project, host a passive SSL sensor or contribute

- Collaboration can include research partnership, sharing of collected streams or improving the software.
- Contact: info@circl.lu
- https://github.com/D4-Project - https://twitter.com/d4_project