



# Recover a RSA private key from a TLS session with Perfect Forward Secrecy

(Marco Ortisi – 2016)



#### About me

- Netizen and IT Security enthusiast since 1996
- Penetration Tester since 1999
  - ✓ In love with buffer overflow flaws
  - ✓ I adore exotic vulnerabilities
  - Senior/Principal Penetration tester? Just a curious guy
- Get in touch with me! marco.ortisi@segfault.it



#### ∩ black hat USA 2016



#### About the topic

- Imagine you can get a server private key by sniffing TLS traffic or interacting through the network with a TLS service. Does it look quite exotic?
- Done via a side channel attack
- Side Channel Attack from Wikipedia
  - "...any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms (compare cryptanalysis). For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information, which can be exploited to break the system..."



### Roadmap

- **≻**Introduction
- > First demo
- ➤ More insight into the attack
- >Second demo
- ➤ Greetings and questions



### The attack and roots with the past

(1996) Arjen Lenstra demonstrated that the usage of the so-called CRT (Chinese Remainder Theorem) optimization put the RSA implementations at great risk (aka private key leakage) if a fault occurred during the computation of a digital RSA signature. (https://infoscience.epfl.ch/record/164524/files/nscan20.PDF)



### What is a RSA signature

#### RSA encryption

- ✓ public key is used to encrypt a message
- ✓ private key is used to decrypt that message

#### RSA signing

- ✓ private key is used to sign a value (actually it is an encryption operation)
- ✓ public key is used to decrypt that value



#### The attack and roots with the past

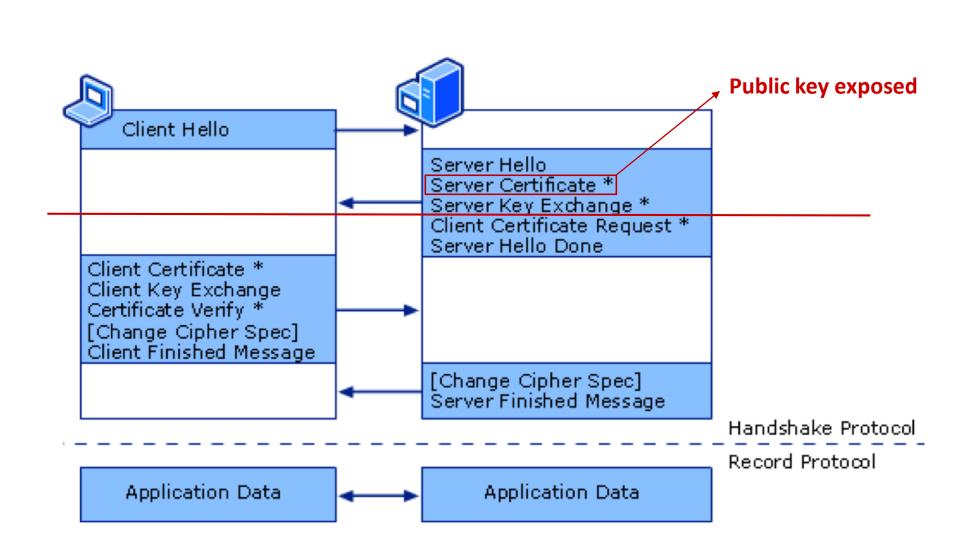
- 1996) Arjen Lenstra demonstrated that the usage of the so-called CRT (Chinese Remainder Theorem) optimization put the RSA implementations at great risk (aka private key leakage) if a fault occurred during the computation of a digital RSA signature. (https://infoscience.epfl.ch/record/164524/files/nscan20.PDF)
- (200x?) Attack conjectured as possible on smartcards if someone has physical access to the device and can disrupt the math behind the RSA operations

### The attack and roots with the past

- (2001) OpenPGP Attack (http://eprint.iacr.org/2002/076.pdf).
  - a) get a local copy of file containing the encrypted private key;
  - b) tamper with it in order to introduce faulty bits;
  - c) <u>capture a single message subsequently signed with the modified encrypted</u> private key (for example an email);
  - d) enjoy your leaked private key ©
- (2015) by targeting TLS, Florian Weimer (Red Hat) unveiled the attack can have remote impacts.

https://people.redhat.com/~fweimer/rsa-crt-leaks.pdf

#### 1 black hat USA 2016





## Recover a RSA private key: Prerequisites



• (a) Presence of a RSA signature calculated using the RSA-CRT optimization...

• (b) The signature must be applied on values known by the attacker...

• (c) Generated signature faulty/miscalculated...



#### What if the attack is successful?

- Private key is exposed
- The real server can be impersonated
- Man-in-The-Middle attack can be performed without alarming the legitimate clients



#### What if the attack is successful?





# DEMO TIME (PART 1)

(enter High Voltage!)



From: http://support-public.cfm.software.dell.com/33164\_sonicos\_5.8.4.2\_releasenotes.pdf

#### Resolved issue Issue ID

A specialized RSA-CRT attack can cause private key leakage in relatively rare cases. This security vulnerability has not been publically disclosed and it is very difficult to perform this attack. To be cautious, Dell SonicWALL recommends that customers upgrade firmware.

Occurs when the SonicOS management interface or a port on the firewall is accessed using SSL, and the following conditions are met:

- A highly sophisticated tool is used to harvest this vulnerability; this tool is not available to the general public
- The Enable Hardware RSA option is enabled in the internal SonicOS settings (by default this option is disabled, in which case the firewall is not vulnerable)

166825



From: http://support-public.cfm.software.dell.com/33164\_sonicos\_5.8.4.2\_releasenotes.pdf

#### Resolved issue ID

A specialized RSA-CRT attack can cause private key leakage in relatively rare cases. This security vulnerability has not been publically disclosed and it is very difficult to perform this attack. To be cautious, Dell SonicWALL recommends that customers upgrade firmware.

166825

Occurs when the SonicOS management interface or a port on the firewall is accessed using SSL, and the following conditions are met:

- A highly sophisticated tool is used to harvest this vulnerability; this tool is not available to the general public
- The Enable Hardware RSA option is enabled in the internal SonicOS settings (by default this option is disabled, in which case the firewall is not vulnerable)



#### A Little Recap



• (a) Presence of a RSA signature calculated using the RSA-CRT optimization...

• (b) The signature must be applied on values known by the attacker...

• (c) Generated signature faulty/miscalculated...



### (a) RSA Signature with RSA-CRT

- The modular exponentiations required by RSA are computationally expensive
- RSA-CRT introduced a less expensive way to do RSA operations (decryption and signing)
- RSA-CRT is used by default in almost every known crypto library out there (openssl, OpenJDK, libgcrypto, PolarSSL, etc...)
- Condition (a) is normally satisfied



## (b) Signature calculated on known values

- We define "X" as the value to be signed
- Padding can influence the final "shape" of "X" before being signed and make this unpredictable...
- ...but with SSL3.0, TLS 1.0, 1.1 and 1.2 the padding scheme (a variant of PKCS1.5) is fully deterministic (not randomized) and then predictable



## (b) PKCS 1.5 Padding

#### Payload:

0D3F8FF87A4D697E73FE86077FD1D10C4ECC59797E759EDD89931B 2208B8044CB4A1B96A

#### Padded Payload (RSA 2048 bit):



## (b) Signature calculated on known values

- We define "X" as the value to be signed
- Padding can influence the final "shape" of "X" before being signed and make this unpredictable...
- ...but with SSL3.0, TLS 1.0, 1.1 and 1.2 the padding scheme (a variant of PKCS1.5) is fully deterministic (not randomized) and then predictable
- Data signed must be known. This is ALWAYS satisfied in our attack if we carefully target the <u>right ciphersuites</u>

## (c) Presence of faulty signature

- We identify a faulty RSA signature with the letter "Y"
- Events causing the generation of a faulty digital RSA signature can't be predicted but they are out there
- Induced by the same vectors like in a typical bit-squatting attack:
  - ✓ CPU overheating
  - ✓ RAM errors
  - ✓ massive exposure of hardware to solar rays
  - ✓ etc...



## The right ciphersuite...

- Prerequisite (b) Signature calculated on known values
- The condition (b) is ALWAYS satisfied in our attack if we carefully target the <u>right ciphersuites</u>

### The right ciphersuite...

- RSA\_WITH\_AES\_256\_CBC\_SHA
  - $\checkmark$  **RSA** = Authentication + Key Exchange
  - $\checkmark$  **AES** = Symmetric algorithm used to encrypt data
  - ✓ **CBC** = Mode of operation
  - ✓ SHA = Hashing algorithm to avoid data tampering
- RSA private key leaked = all TLS sessions compromised (current, past and future ones)
- The "Certificate" message contains a Signature created with the private key of CA
  - statically embedded inside the certificate (not generated on the fly)
  - the condition (b) is not satisfied

### The right ciphersuite...

- DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA
  - ✓ **DHE** = Key Exchange
  - ✓ **RSA** = Only used for Authentication
  - ✓ [the rest is the same as previously mentioned]
- The key exchange is done using private/public keys generated on the fly
  - Compromission of a private key breaks only that specific encrypted session, not all the previously established
  - This is called "Perfect Forward Secrecy"



## PFS fits REALLY perfectly to us

RSA signature appended onto a TLS Server Key Exchange Message

```
♣ Handshake Protocol: Server Key Exchange

        Handshake Type: Server Key Exchange (12)
        Length: 521

■ Diffie-Hellman Server Params

           p Length: 128
           p: d67de440cbbbdc1936d693d34afd0ad50c84d239a45f520b...
           g Length: 1
           g: 02
           Pubkey Length: 128
           Pubkey: 230274659a7683fa4dd86cba367ea687675309f0b60d8477...
          Signature Length: 256
          Signature: 9dbac58a9055498f7bf1254074ac14c74ec46f3e0506164c...
Secure Sockets Layer
0110 01 00 9d ba c5 8a 90 55 49 8f 7b f1 25 40 74 ac
                                                            .....U I.{.%@t
      14 c7 4e c4 6f 3e 05 06 16 4c 37 c0 70 43 f1 50
      4f c2 8d 66 ec dd b2 67 cd 46 78 09 57 77 56 de
0130
      ad 12 55 11 92 8b c0 86 ed b9 0e 6b 44 a2 ba 31
0140
      3a da 4b a7 7b c9 2e 09 f8 4f 99 7b ed 6b c9 97
0150
      57 72 9f fd 21 ea ef d1 30 0b 61 e0 13 1e 6a 6e
                                                             Vr..!... 0.a...jr
Frame (463 bytes)
           Reassembled TCP (530 bytes)
      Diffie-Hellman server signature (ssl.handshake.sig), 256 byte
```

## PFS fits REALLY perfectly to us

- A RSA signature is appended on a TLS Server Key Exchange Message
- The values on which it is calculated are well known, padding scheme is deterministic
  - ✓ Client Random struct (*Client Hello Message*)
  - ✓ Server Random struct (*Server Hello Message*)
  - ✓ Server Params struct (Server Key Exchange Message)
- DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA and in general
   [EC]DHE\_RSA\_WITH\_ANY\_ANY\_ANY ciphersuites are good for us
  - > The condition (b) is satisfied this time







TLS Client Hello (PFS ciphersuites only negotiation)

```
# Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 198
    Version: TLS 1.2 (0x0303)

# Random
    GMT Unix Time: May 30, 1981 07:53:42.0000000000 ora legale Europa occidentale
    Random Bytes: 6179c141c844786767bd4867051955676853c5ea74dcc122...

Session ID Length: 0
    Cipher Suites Length: 30
    Cipher Suites (15 suites)
    Compression Methods Length: 1
```







#### **TLS Server Hello**

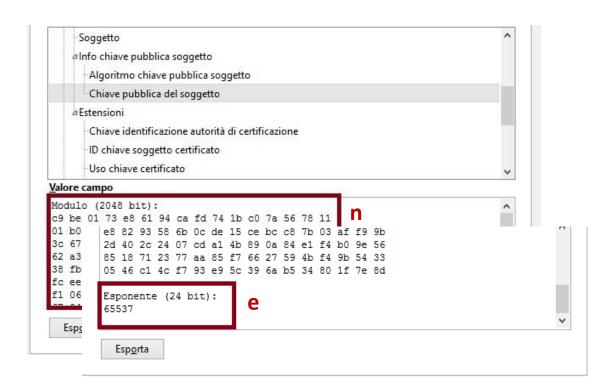
# Handshake Protocol: Server Hello
Handshake Type: Server Hello (2)
Length: 70
Version: TLS 1.0 (0x0301)

# Random
GMT Unix Time: Feb 10, 2016 19:16:19.000000000 ora solare Europa occidentale
Random Bytes: 0ddbab1877d6d8d51474dfa833b2c2ed3b05516194e65b18...
Session ID Length: 32
Session ID: df27d09ed3c26a6b61d93ae0a47bd6444abc9a1548b61fc0...
Cipher Suite: TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (0x0039)
Compression Method: null (0)

















```
# Handshake Protocol: Server Key Exchange
Handshake Type: Server Key Exchange (12)
Length: 521
# Diffie-Hellman Server Params

p Length: 128
p: d67de440cbbbdc1936d693d34afd0ad50c84d239a45f520b...
g Length: 1
g: 02
Pubkey Length: 128
Pubkey: 230274659a7683fa4dd86cba367ea687675309f0b60d8477...
Signature Length: 256
Signature: 9dbac58a9055498f7bf1254074ac14c74ec46f3e0506164c...
```









# Handshake Protocol: Server Key Exchange Handshake Type: Server Key Exchange (12) Length: 521

▲ Diffie-Hellman Server Params

p Length: 128

p: d67de440cbbbdc1936d693d34afd0ad50c84d239a45f520b...

g Length: 1

g: 02

Pubkey Length: 128

Pubkey: 230274659a7683fa4dd86cba367ea687675309f0b60d8477...

Signature Length: 256

Signature: 9dbac58a9055498f7bf1254074ac14c74ec46f3e0506164c...

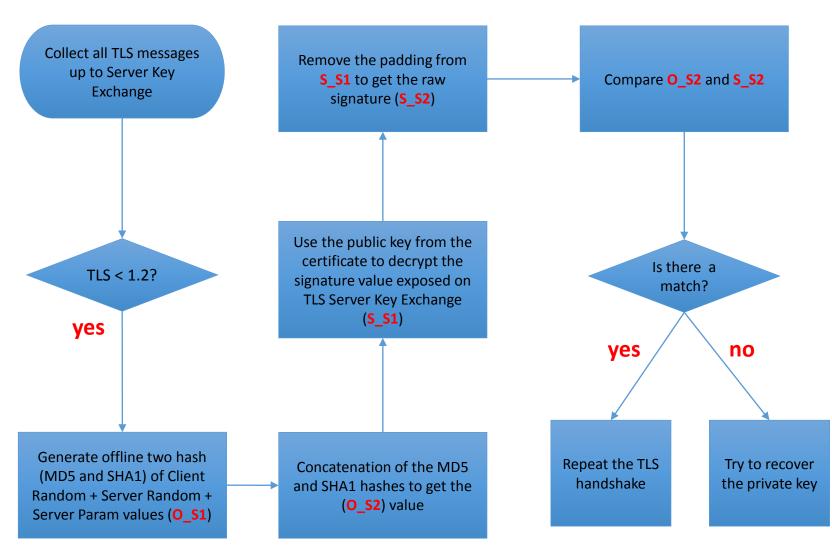
Client Random Struct (Client Hello Message)
Server Random Struct (Server Hello Message)
Server Param Struct (Key Exchange Message)

## Attacking TLS abusing PFS

- Because of "(c) Presence of faulty signature", the attack can be carried out only when a RSA signature is faulty. How to determine that?
- Looking for faulty signatures:
  - ✓ We know padding scheme (PKCS 1.5) and the values on which the server calculates its signature
    - Client Random struct (*Client Hello Message*)
    - Server Random struct (Server Hello Message)
    - Server Params struct (Server Key Exchange Message)
  - ✓ We know the clear-text message on which the signature is applied
  - ✓ We have the public key of server and can decrypt the signature from the TLS Server Key Exchange
  - ✓ If there is a mismatch between the decrypted signature and the expected plain-text value, we have found what we are looking for

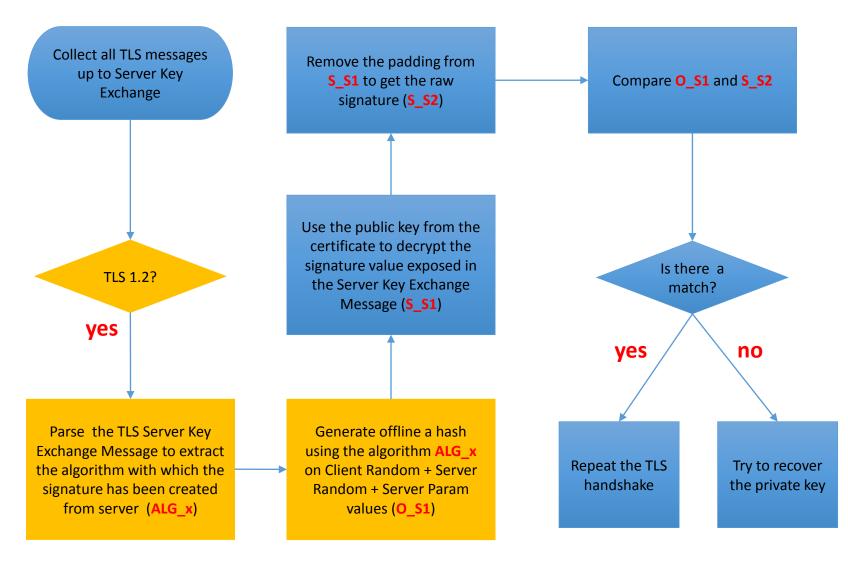


#### How to check if a digital signature is invalid? (TLS < 1.2)





#### How to check if a digital signature is invalid? (TLS 1.2)





## What else?

How RSA works

- RSA-CRT optimization
- 2<sup>nd</sup> DEMO



#### How RSA works

 $encryption = c = m^e \mod n$  $decryption = m = c^d \mod n$ 

- **c** = ciphertext
- m = message to encrypt
- n, e = public key
- e = little exponent (usually a value such as 3 or 65537)
- $\mathbf{n}$  = big semiprime number ( $\mathbf{p} * \mathbf{q}$ ) ( $\mathbf{p}$  and  $\mathbf{q}$  are said "prime factors")
- d = inverse\_mod(e, (p-1) \* (q-1)) private key
   mathematically tied with n
- getting "p" and "q" a private key can be recovered because "e" is already a public information

#### RSA: Rule 1

$$n = p * q$$
  
 $d = inverse\_mod(e,(\underline{p-1}) * (\underline{q-1}))$ 

Gaining a prime factor of  $\underline{n}$  (whatever of the two) we can determine very easily the other one and recover the private key

- **n** = 77 (the public key into the certificate)
- **p** = 7 (leaked prime factor)
- **d** = 5

## RSA: Rule 1

$$n = p \times q$$
  
 $d = inverse\_mod(e,(p-1) \times (q-1))$ 

Gaining a prime factor of  $\underline{n}$  (whatever of the two) we can determine very easily the other one and recover the private key

- **n** = 77 (the public key into the certificate)
- **p** = 7 (leaked prime factor)
- $q = 11 \rightarrow 77 / 7$

- 77 / 11 = 7 (p)
- 7 x 11 = 77 (n)



## **RSA-CRT**

• RSA-CRT (CRT stands for Chinese Remainder Theorem)

• With this optimization the RSA calculation is broken down into two smaller parts

## **RSA-CRT**

- Precompute the following values:
  - >qInv = (1/q) mod p
  - $\rightarrow$  dP = d (mod p 1)
  - $\triangleright dQ = d \pmod{q-1}$
- ...and next calculate:
  - >m1 = c^dP mod p
  - > m2 = c^dQ mod q
  - >h = qlnv.(m1 m2) mod p
  - >m = m2 + h.q

If during m1 or m2 calculation there is a fault, a faulty RSA signature is generated and one prime factor can be leaked with this formula:

## **RSA-CRT**

- Precompute the following values:
  - >qInv = (1/q) mod p
  - >dP = d (mod p 1)
  - $\triangleright dQ = d \pmod{q-1}$
- ...and next calculate:
  - >m1 = c^dP mod
  - > m2 = c^dQ  $\sim$  out
  - >h = In (m) m2) mod p
  - >**m** = 1**m** + h.q

If during m1 or m2 calculation there is a fault, a faulty Rop Signature is generated and one prime factor can be leaked with this form it:

$$gcd(y^e - x, n)$$

## RSA-CRT: Rule 2

We can leak a prime factor of n from a faulty signature when RSA-CRT is used for signing:

$$gcd(y^e - x, n)$$

- y = faulty/miscalculated signature (this can be taken directly from the TLS **Server Key Exchange** message)
- **e** = public exponent (found inside the **Certificate**)
- x = the original value before to be signed (it is padded but PKCS 1.5 padding scheme is deterministic)
- n = public key (p \* q) (found inside the Certificate)
- When one of the prime factors is known...
   <u>derived\_prime\_factor</u> = n / leaked\_prime\_factor
- When leaked\_prime\_factor and derived\_prime\_factor are known, p and q are known
   d = inverse mod(e,(prime\_factor\_P 1) \* (prime\_factor\_Q 1))

## RSA-CRT: Rule 2

We can leak a prime factor of n from a faulty signature when RSA-CRT is used for signing:

$$gcd(y^e - x, n)$$

- y = faulty/miscalculated signature (this can be taken directly from the S Server Key Exchange message)
- e = public exponent (found inside e ertifica )
- x = the original value of force by sign of (it is paided but PKCS 1.5 padding scheme is deterministic)
- n = public ke (p q) side the **Certificate**)
- When one or the prime factors is known...
   derived\_prime\_factor = n / leaked\_prime\_factor
- When leaked\_prime\_factor and derived\_prime\_factor are known, p and q are known
   d = inverse mod(e,(prime\_factor\_P 1) \* (prime\_factor\_Q 1))



# DEMOTIME (PART 2)

(enter Piciolla...)



## The bottom line

- IF:
  - you have a piece of software linked to a vulnerable crypto library using RSA-CRT &&
  - that crypto library does not verify the correctness of each RSA signature generated &&
  - environmental factors occur (CPU overheating, RAM error, etc...) causing the miscalculation of a RSA signature

THEN you might be in trouble...



## The Fix

 A few of crypto libraries allow users to disable RSA-CRT (<u>not</u> <u>convenient due to performance issues</u>)

- Most vendors have recently issued a patch to address this problem
  - ✓ Double-checks the correctness of RSA signatures without RSA-CRT optimization:

```
if (y^e = x (mod n))
  then signature_valid
```

## Vulnerable crypto libraries

- □PolarSSL < 2.1.1, 1.3.13 and 1.2.16: *MBEDTLS\_RSA\_NO\_CRT* can be defined to disable RSA-CRT but this option is off by default)
- □**libgcrypt** < 1.6.3 (equivalent to *CVE-2015-5738*)
- □Nettle < 3.1: used by GnuTLS
- □**Java SE** < 5.0u81, 6u91, 7u76, 8u40, and **JRockit** < R28.3.5 (*CVE-2015-0478*)
- **LEMC RSA BSAFE Micro Edition Suite** (MES) 4.0.x and 4.1.x before 4.1.5, **RSA BSAFE Crypto-C Micro Edition** (CCME) 4.0.x and 4.1.x before 4.1.3, **RSA BSAFE Crypto-J** before 6.2.1, **RSA BSAFE SSL-J** before 6.2.1, and **RSA BSAFE SSL-C** before 2.8.9 (*CVE-2016-0887*)



# Vulnerable crypto libraries (2)

- □OpenSSL <= 0.9.7 and \*potentially\* between 1.0.2 and 1.0.2d because of CVE-2015-3193 only on x86 64 architectures + custom versions  $\Box$ **Go** crypto library < 1.6.2 □Cryptlib up to latest 3.4.3 (CRYPT OPTION MISC SIDECHANNELPROTECTION would prevent the attack but it is set to false by default) **☐wolfSSL** (formerly **CyaSSL**) < 3.6.8 (*CVE-2015-7744*)  $\Box$ Libtomcrypt < 2.00 **□Eldos SecureBlackbox** < 13.0.280 and 14.0.281 **□MatrixSSL** < 3.8.3
- □Openswan up to latest version 2.6.47 vulnerable when not compiled with NSS



## Device types mainly affected

#### □ Embedded devices

- ✓ Various network appliances (firewalls, routers, etc...)
- ✓ Consumer / SOHO devices
- ✓ SSL Accelerators
- ✓ VPN Concentrators
- ✓ TLS Reverse Proxies
- **√** ...



## Affected devices

- □ **FORTINET** (Series 300 / FortiGate < 5.0.13 / 5.2.6 / 5.4.0 observed as vulnerable)
- □ **Dell** (SonicWALL< SonicOS 6.1.1.12 found affected)
- □F5 (Traffix SDC affected)
- □ZTE ZXSEC Firewall (affected models US2640B, US2630B, US2620B)
- □LANCOM wireless devices (version 8.84) <- apparently silently patched since 2014
- □D-Link-DCS-933L Surveillance camera
- □ **HILLSTONE NETWORKS** (SG-6000 Firewall)

- **□**CITRIX
- **□**NORTEL
- **□QNO**
- **□**Viprinet
- **□BEJY**
- **□**Alteon



## But remember...

•"...a piece of software linked to a vulnerable crypto library using RSA-CRT..."

- Wait! Is this problem only related to TLS, right?
- No! PFS and RSA are used A LOT in IPSEC VPN (IKE), SSH, etc...



# IKEv1 (Internet Key Exchange)

- Two modes for Phase 1 (Authentication of peers and Negotiation of SA)
  - ✓ Main
  - ✓ Aggressive
- It is believed that only by sniffing the network traffic it is not
  possible to recover a private key (just as happens for TLS) and an
  active approach is requested.



## IKEv1 Phase 1 Main Mode (Signature Auth)

<sup>\*</sup> Indicates payload encryption



## IKEv1 Phase 1 Aggressive Mode (Signature Auth)





#### SPECIAL THANKS



Frank Bosman for his nice drawings

http://turniphead.deviantart.com

#### Florian Weimer (Red Hat)

https://access.redhat.com/blogs/766093/posts/1976703 https://pagure.io/symboldb/tree/tls-experiment



## PLEASE VISIT SICILY...





Recover a RSA private key from a TLS session with Perferct Forward Secrecy

## QUESTIONS?

High Voltage & Piciolla – http://www.segfault.it/tools/ Marco Ortisi (2016) Blog – http://www.segfault.it/

thanks!