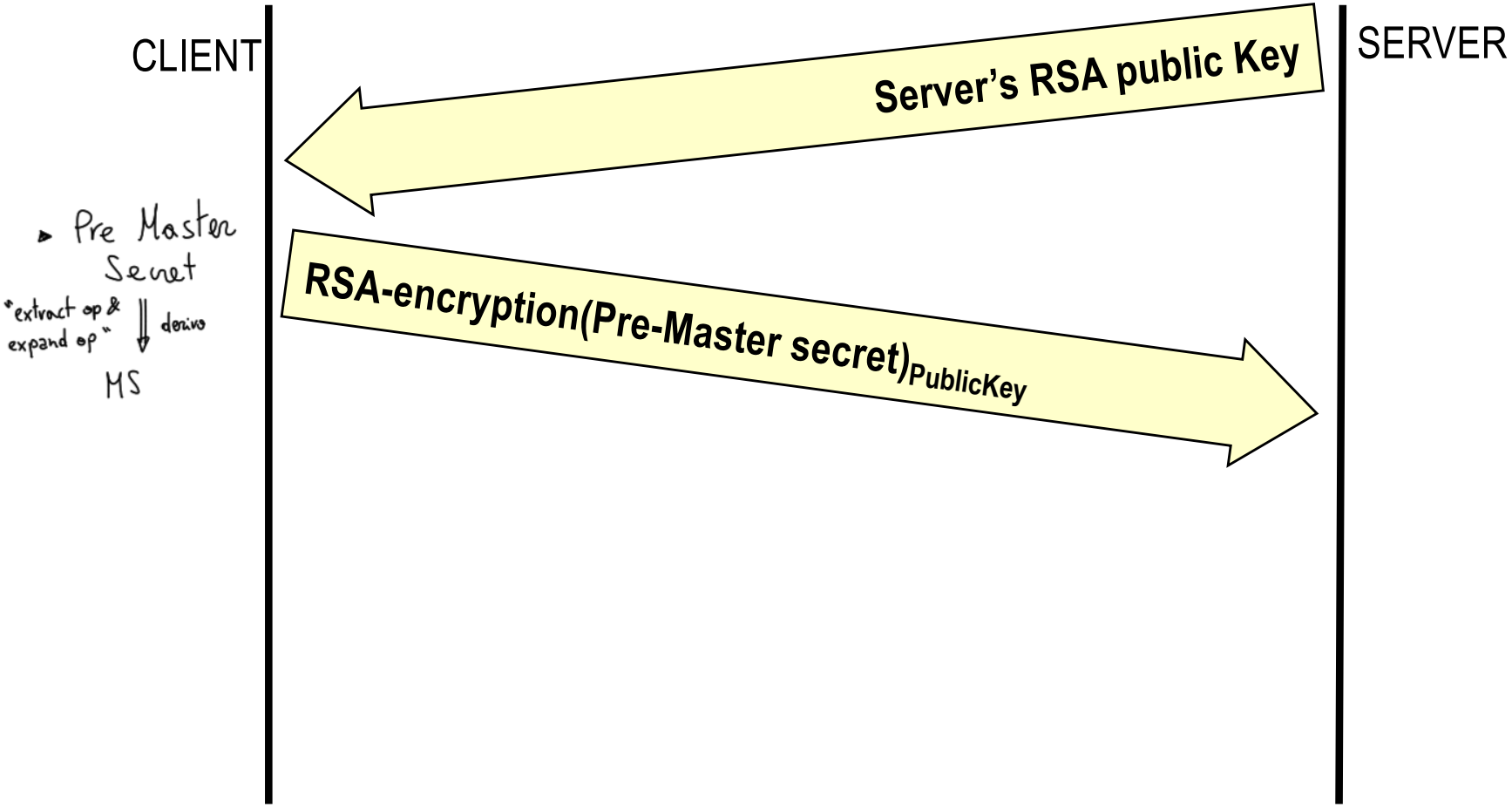
More insights on the security of RSA key transport

(perché oggi è superato?)

RSA key transport

ho solo encryption!



Is RSA robust against Chosen ciphertext attacks? (RSA no integrity)

➔ **[all following operations are assumed mod n]**

➔ **You see a ciphertext** $C=M^e$

➔ **Goal: decrypt it, i.e. find** $M = C^d$

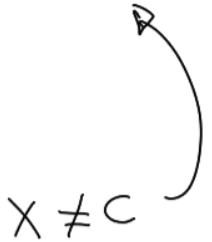
➔ **Attacker power: can access a decryption oracle**

⇨ But obviously CANNOT submit original data, otherwise trivial!

➔ **Attack:** (idea) sempre Mod n

⇨ Choose random value r

⇨ Construct NEW ciphertext $X= (r^e C)$



⇨ Ask oracle to decrypt seemingly innocuous message X

→(chosen ciphertext attack)

⇨ Oracle will return decryption of X $X^d = (r^e C)^d = r^{ed} C^d = r C^d$

⇨ Attacker cancels r and finds M! $X^d r^{-1} = r C^d r^{-1} = C^d = M$

→multiply by r^{-1} mod n

#1 Problem: RSA is malleable

"posso alterare il testo, ma senza alterarne totalmente il significato"

➔**Non-malleability:**

given an encryption c of some message m, the attacker should NOT be able to create a different ciphertext c' that decrypts to a message m' that is somehow related to m.

↳ no meaning, ma può essere relazionato a m

➔**RSA padding: should at least mitigate such problem**

⇨But we will see later on whether this is the case… ☹

#2 problem: RSA non CPA-secure: $(M)^e \bmod n = C_1$ ⎫ sono uguali!
$(M)^e \bmod n = C_1$ ⎭

come li risolvo entrambi? Se M=0 ? Se M=1 ?

RSA padding: PKCS #1 v1.5

public key criptography standard

➔ **PKCS #1 = first standard for the family PKCS = Public-Key Cryptography Standards (PKCS)**

➔ **Specifies RSA encryption and decryption operation, including basic RSA padding approach**

⇨ Also available as IETF RFC (see original RFC 2313, see also latest one: 8017)

Real RSA

2 bytes: specificano l'operazione!

"nonce", o "IV" semantic security

dopo inizio msg

difficile da alterare, devo cambiare tutto il msg, ma primi 11 bit perdono significato

Padding string: at least 8 non-zero bytes (random, to avoid otherwise trivial CPA attack)

Actual data: no more than k-11 bytes

00	02	rnd	rnd	rnd	rnd	rnd	rnd	rnd	rnd	00	dat	dat	dat	dat	dat	dat

0002 = encryption

SKIP

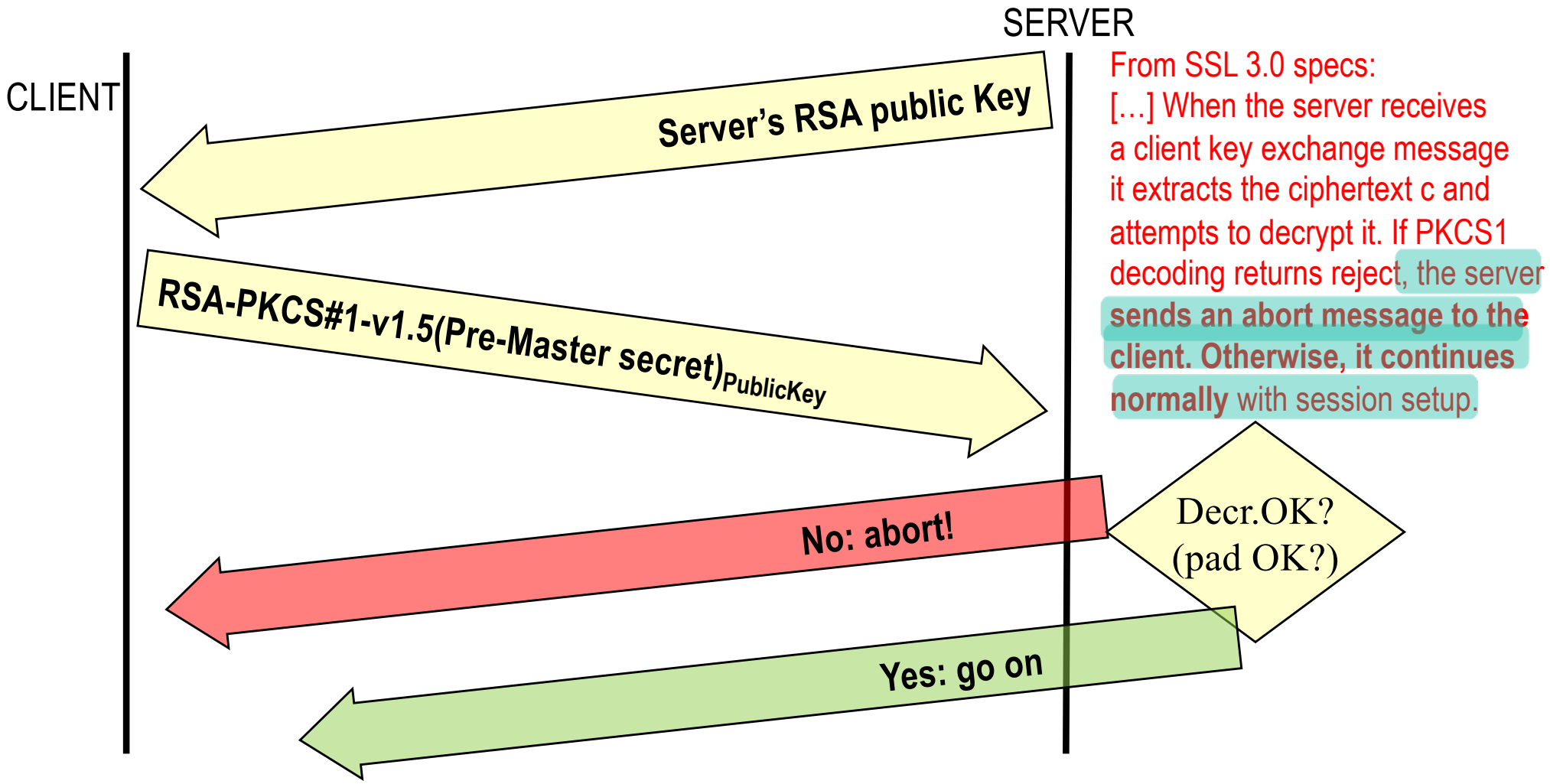
k = size of the RSA modulus in bytes – formally: k = integer value which satisfies

$$2^{8(k-1)} \leq n < 2^{8k}$$

e.g. with RSA-1024, k = 128 (in most cases) → you can encrypt up to 117 bytes

max size RSA msg

RSA key transport (until SSL v3.0 included)



OOOOPPPPSSS!!! Looks like we have a possible Padding Oracle!!!

Bleichenbacher's Oracle (nuova forma per RSA enc)

- **Daniel Bleichenbacher, 1998**
  - Adaptive Chosen Ciphertext Attack
  - Against RSA PKCS #1
- **Targets up to SSL v3.0**
  - Corrected in TLS1.0+
  - Removed in TLS 1.3 (no more RSA key transport)
- **But still around**
  - at least until 2017/18!!

Bleichenbacher's Oracle

➔**Attacker's goal: decrypt ciphertext C**

⇨$C=M^e \bmod n$ (M è real msg)

➔**Pick value r and construct new ciphertext**

⇨$C' = C\, r^e \bmod n$

$= (M\, r)^e \bmod n$ alterandolo, cambiano anche i bit 0002 iniziali?

➔**Send to SSLv3.0 server**

⇨Oracle will tell you if (M r) starts with 0002 or not!

⇨This leaks information about M!

⇨Bleichenbacker's result: with a sufficient number of attempts (order of 2^{20} with RSA-1024) you can completely decrypt M! ≈ 1 m/h

⇨Adaptive attack: pick r based on previous results!

Toy Bleichenbacher example

➔Assume a simpler Oracle

- ⇨Msg = $\log_2 n$ bits
- ⇨Oracle tells if first bit is 0 or 1. (NOT OK → 0, OK → 1) (non più 0002)
  - ➔Binary answer, nothing else
- ⇨*Real Bleichenbacher's attack: same concept, but PKCS#1 padding oracle tells if first 2 bytes are 0002, hence much more elaborate math – see original paper*

➔With order of $\log_2 n$ attempts, you can decrypt the WHOLE message!!

Assumo $M =$ "10010110" non noto!

è incluso però in $C = M^e$!!

Se faccio $C \cdot 2^e = (2M)^e$, mandato al server, esso dice se il primo bit è "0" o "1" (più complesso nella realtà)

"2" è 1° shift, 4 il 2°, 8 il 3° ...

con log N ottengo tutto il msg. (nel caso 1 bit: vero/falso).

Nella realtà devo inizialmente cercare `000`, e poi passare avanti.

Example (see math file)

➔RSA parameters

⇨p = 13, q=19, n=247 (8 bit) (p, q primi)

⇨Let's pick e=29 → $d=e^{-1} \bmod 216 = 149$

(e: public key, coprima rispetto "n"; 216 = $\phi(n) = 12 \cdot 18$)

➔Assume you see C=90, come decripto?

➔Send to server:

- $C_1 = 90 \cdot 2^{29} = (2M)^{29}$, invio al server (1° bit) = ∅
- $C_2 = 90 \cdot 4^{29} = (2^2 \cdot M)^{29}$, invio al server (2° bit) = 1

Mod[c* PowerMod[2^Range[1, 8], e, n], n] =
{20, 224, 187, 69, 180, 40, 201, 127}

➔Server returns: {0,1,1,1,0,1,1,1}

8 bit

Example (see math file)

- Sempre mod 247

➔**2M starts with 0 if**

⇨M in (0, 63) or (124,187)

sfruttо ciò che ho appreso prima

poichè c'è modulo

- (0,63) perchè: 0 x x x x x x x da 0 a 63 (64 32 . . . . 1)
- (124,187) perchè: $124 \cdot 2 = 248 \bmod 247 = 1$

➔**Of this, 4M starts with 1 if**

$C \cdot (2^e)^e$ server → 1

⇨M in (32, 61) or (156,185)

➔**Of this, 8M starts with 1 if**

⇨M in (47, 61) or (171,185)

➔**And so on, 8° attempt, i.e. 256M, will break the last tie!**

⇨binary search: linear with #bit!!

La "sicurezza di RSA" ricade sul singolo bit, se leakato è problematico!

Aftermath: as long as one bit gets known, all the ciphertext gets known as well!!

(repeat the example with the parity bit, even easier!)

Corollary: attacking parity

- ➔A shop rejects RSA-PKCS#1 encrypted requests for an even number of items
- ➔Parity side channel = parity oracle!
- ➔Same situation as before: as long as just 1 bit gets revealed, ALL plaintext is at stake!

Bleichenbacher's Oracle in the wild

➔ **Discovered in 1998**
- ⇨ Affected SSL up to v3.0
- ⇨ Corrected by TLSv1.0+

➔ **But… guess what happened next!**
side channels → pandora's box → Bleichenbacker' saga!
- ⇨ Klima et al. (CHES 2003)
- ⇨ Bleichenbacher's Attack Strikes Again: Breaking XML Encryption (ESORICS 2012)
- ⇨ Degabriele et al. (CT-RSA 2012)
- ⇨ Bardou et al. (CRYPTO 2012)
- ⇨ Cross-Tenant Side-Channel Attacks in PaaS Clouds (CCS 2014)
- ⇨ Side Channel Attacks to TLS implementations (USENIX 2014)
- ⇨ Attack to QUIC (CCS 2015)
- ⇨ DROWN (USENIX 2016)
- ⇨ Return Of Bleichenbacher's Oracle Threat (USENIX 2018)
- ⇨ The Dangers of Key Reuse: Practical Attacks on IPsec IKE (USENIX 2018)
- ⇨ The 9 Lives of Bleichenbacher's CAT - cache attacks (IEEE S&P 2019)

2019 CAT paper, coauthored by Adi Shamir:
the inventor of RSA now recommending to deprecate RSA in TLS!

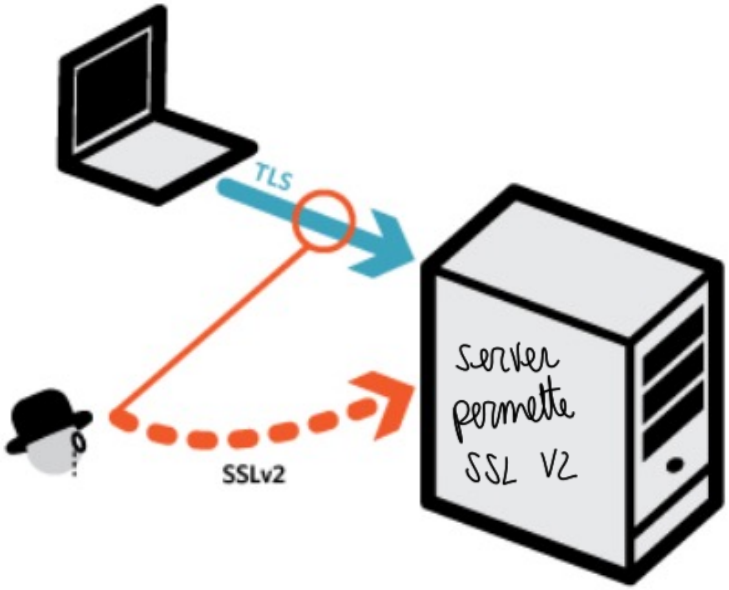
Recent examples of Bleichenbacher's Oracle in the wild

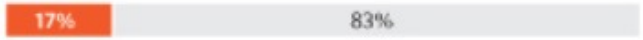
- **2016: DROWN attack to TLS**
  - Use Bleichenbacher with SSLv2 (vulnerabile!)
    - SSLv2? Are you kidding? Affects 1/3 of the sites, including major ones!
- **2018: ROBOT attack**
  - Extensive large scale analysis: quite a few poor implementations!
- **IPSEC IKE issues (2018)**
  - combine Bleichenbacher with downgrade to IKEv1.0
    - Vulnerabilities in Cisco (CVE-2018-0131), Huawei (CVE-2017-17305), Clavister (CVE-2018-8753), and ZyXEL (CVE-2018-9129)

DROWN attack (2016)

A server is vulnerable to DROWN if:

It allows both TLS and SSLv2 connections

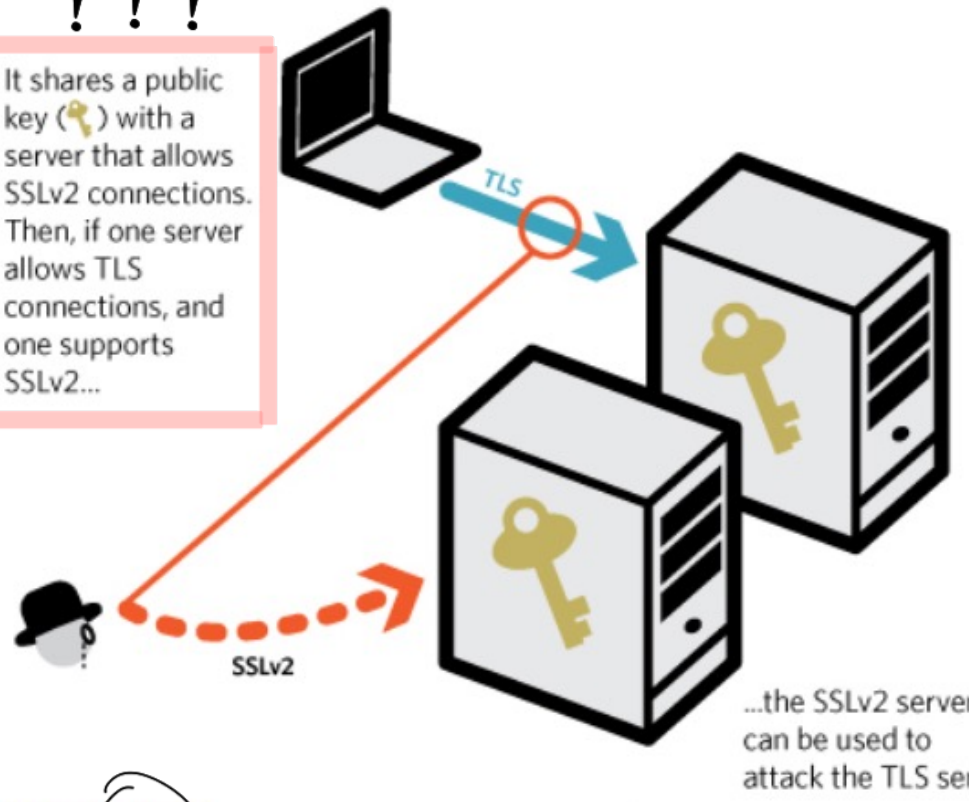


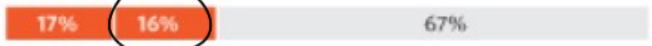


17% of HTTPS servers still allow SSLv2 connections (downgrade)

! ! !

It shares a public key (🔑) with a server that allows SSLv2 connections. Then, if one server allows TLS connections, and one supports SSLv2...





When taking key reuse into account, an additional 16% of HTTPS servers are vulnerable, putting 33% of HTTPS servers at risk

Source: https://drownattack.com/

How critical can it get?

TLS implementations: a jungle!!

lots of oracles until last year!!

Source: ROBOT 2018 USENIX SEC

Implementation	Server response		TLS flow	Oracle	Reference / ID
	Valid message	Invalid message			
Facebook					
1st vulnerability	20	47	full	strong	-
2nd vulnerability	20	TCP FIN	shortened	strong	-
F5					
Variant 1	TCP timeout	40	shortened	strong	CVE-2017-6168
Variant 2	One alert (40)	Two alerts (40)	full	strong	CVE-2017-6168
Variant 3	TCP timeout	40	shortened	weak	CVE-2017-6168
Variant 4	One alert (40)	Two alerts (40)	full	weak	CVE-2017-6168
Variant 5	20	80	full	strong	CVE-2017-6168
Citrix Netscaler					
with CBC cipher suites	Connection reset	TCP timeout	full	strong	CVE-2017-17382
with GCM cipher suites	51	TCP timeout	full	strong	CVE-2017-17382
Radware					
Radware Alteon	51	TCP reset	full	strong	CVE-2017-17427
Cisco					
Cisco ACE	20	47	full	strong	CVE-2017-17428
Cisco ASA	TCP timeout	TCP reset	full	weak	CVE-2017-12373
Erlang					
Erlang version 19 and 20	10	51	full	strong	CVE-2017-1000385
Erlang version 18	20	51	full	strong	CVE-2017-1000385
Palo Alto Networks					
PAN-OS	One alert (40)	Two Alerts (40)	full	weak	CVE-2017-17841
IBM					
IBM Domino	20	47	full	weak	(unfixed)
IBM WebSphere MQ	?	?	?	?	CVE-2018-1388
WolfSSL					
WolfSSL prior to 3.12.2	TCP timeout	Alert 0	shortened	weak	CVE-2017-13099
Bouncy Castle					
Bouncy Castle 1.58	`ChangeCipherSpec`	80	shortened	weak	CVE-2017-13098

Table 1: Overview of vulnerable implementations and affected servers found in our research. TLS alerts are referenced by their numbers: 10 (`unexpected_message`) 20 (`bad_record_mac`), 40 (`handshake_failure`), 47 (`illegal_parameter`), 51 (`decrypt_error`), and 80 (`internal_error`).

How critical can it get?

Implementation	Server response		TLS flow	Oracle	Reference / ID
	Valid message	Invalid message			
Facebook					
1st vulnerability	20	47	full	strong	-
2nd vulnerability	20	TCP FIN	shortened	strong	-
F5					
Variant 1	TCP timeout	40	shortened	strong	CVE-2017-6168
Variant 2	One alert (40)	Two alerts (40)	full	strong	CVE-2017-6168
Variant 3	TCP timeout	40	shortened	weak	CVE-2017-6168
Variant 4	One alert (40)	Two alerts (40)	full	weak	CVE-2017-6168
Variant 5	20	80	full	strong	CVE-2017-6168
Citrix Netscaler					
with CBC cipher suites	Connection reset	TCP timeout	full	strong	CVE-2017-17382
Erlang version 18	20	51	full	strong	CVE-2017-1000385
Palo Alto Networks					
PAN-OS	One alert (40)	Two Alerts (40)	full	weak	CVE-2017-17841
IBM					
IBM Domino	20	47	full	weak	(unfixed)
IBM WebSphere MQ	?	?	?	?	CVE-2018-1388
WolfSSL					
WolfSSL prior to 3.12.2	TCP timeout	Alert 0	shortened	weak	CVE-2017-13099
Bouncy Castle					
Bouncy Castle 1.58	`ChangeCipherSpec`	80	shortened	weak	CVE-2017-13098

Table 1: Overview of vulnerable implementations and affected servers found in our research. TLS alerts are referenced by their numbers: 10 (`unexpected_message`) 20 (`bad_record_mac`), 40 (`handshake_failure`), 47 (`illegal_parameter`), 51 (`decrypt_error`), and 80 (`internal_error`).

imp

See relevant talk @ USENIX Security 2018

https://www.youtube.com/watch?v=037D7YZCpSM

lots of oracles until last year!!

Source: ROBOT 2018 USENIX SEC

Countermeasures

➔ **Change RSA-PKCS#1-v1.5 into more secure padding**

- ⇨ E.g., RSA-OAEP
- ⇨ Way too major change!! Declined!

➔ **Careful implementation**

- ⇨ If padding malformed, follow up as if it were properly formed…
- ⇨ But not nearly easy!!
  - → See many more SUBTLE issues at https://eprint.iacr.org/2003/052.pdf
    - » Also shows that random generation of 48 bytes is not secure
- ⇨ In fact, many implementations messed up!

➔ **Get rid of RSA key transport…**

- ⇨ Well, this is what TLSv1.3 has done!

Take-home messages

- **Implementation is very tricky! And errors last for long**
  - Once crypto error is made (e.s. original SSL) it's hard to incrementally patch it
  - Somewhat reminds me the MAC-then-encrypt saga… (also ending with removal from TLSv1.3)
- **Even security folks might not fully grasp crypto attacks**
  - Fun slide in ROBOT's presentation
    - When talking about responsible disclosure: