Implementation Flaws in TLS Stacks: Lessons Learned and Study of TLS 1.3 Benefits

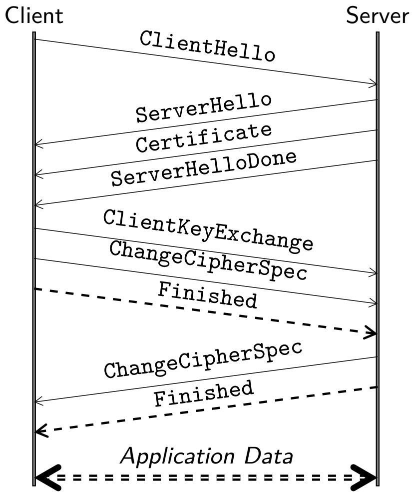
Olivier Levillain

Télécom SudParis

November 5th 2020

TLS in a nutshell

Protocol description



SSL/TLS is pervasive today

- HTTPS (many use cases)
- A generic method to secure protocols
- SSL VPN, EAP TLS...

Security goals

- Server (and optionnaly client) authentication
- Data confidentiality and integrity protection
- Anti-replay

A history of vulnerabilities

Since its inception in 1995 as SSL, the protocol has known many problems, especially since 2011

A history of vulnerabilities

Since its inception in 1995 as SSL, the protocol has known many problems, especially since 2011

- 2011: BEAST
- 2012: CRIME
- 2013: Lucky 13
- 2014: POODLE
- 2014: Heartbleed
- 2014: 3SHAKE
- 2015: FREAK
- 2015: LogJam
- 2016: DROWN

A history of vulnerabilities

Since its inception in 1995 as SSL, the protocol has known many problems, especially since 2011

- 2011: BEAST
- 2012: CRIME
- 2013: Lucky 13
- 2014: POODLE
- 2014: Heartbleed
- 2014: 3SHAKE
- 2015: FREAK
- 2015: LogJam
- 2016: DROWN

To overcome them, the IETF TLS WG started working on TLS 1.3 in 2014

The TLS 1.3 revolution

TLS 1.3, standardized in RFC 8446, brings many answers to the aforementionned problems

The TLS 1.3 revolution

TLS 1.3, standardized in RFC 8446, brings many answers to the aforementionned problems

- most obsolete cryptographic constructions were removed
  - RSA PKCS#1 v1.5
  - MD5, SHA1, RC4
  - the CBC mode

The TLS 1.3 revolution

TLS 1.3, standardized in RFC 8446, brings many answers to the aforementionned problems

- most obsolete cryptographic constructions were removed
  - RSA PKCS#1 v1.5
  - MD5, SHA1, RC4
  - the CBC mode
- the handshake phase is more secure
  - the forward secrecy is always guaranteed
  - only proper selected groups can be used in the key exchange

The TLS 1.3 revolution

TLS 1.3, standardized in RFC 8446, brings many answers to the aforementionned problems

- most obsolete cryptographic constructions were removed
  - RSA PKCS#1 v1.5
  - MD5, SHA1, RC4
  - the CBC mode
- the handshake phase is more secure
  - the forward secrecy is always guaranteed
  - only proper selected groups can be used in the key exchange
- the privacy has been enhanced
  - part of the handhsake is encrypted
  - for encrypted messages, the type is masked and the length can be padded

The TLS 1.3 revolution

TLS 1.3, standardized in RFC 8446, brings many answers to the aforementionned problems

- most obsolete cryptographic constructions were removed
  - RSA PKCS#1 v1.5
  - MD5, SHA1, RC4
  - the CBC mode
- the handshake phase is more secure
  - the forward secrecy is always guaranteed
  - only proper selected groups can be used in the key exchange
- the privacy has been enhanced
  - part of the handhsake is encrypted
  - for encrypted messages, the type is masked and the length can be padded

What about TLS implementations?

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D something else (`RC4_MD5`)

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D something else (`RC4_MD5`)

Actually, it is easy to explain

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA` (`0x002f`)

B `ECDH-ECDSA-AES128-SHA`

C an alert

D something else (`RC4_MD5`)

Actually, it is easy to explain

- a ciphersuite is represented by a 16-bit integer
- for almost a decade, all suites had their first byte equal to 00

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA` (0x002f)

B `ECDH-ECDSA-AES128-SHA` (0xc005)

C an alert

D something else (`RC4_MD5`) (0x0005)

Actually, it is easy to explain

- a ciphersuite is represented by a 16-bit integer
- for almost a decade, all suites had their first byte equal to 00

A word on existing SSL/TLS stacks

What should a client expect when they propose the following ciphersuites: `AES128-SHA` et `ECDH-ECDSA-AES128-SHA`?

- A `AES128-SHA` (0x002f)
- B `ECDH-ECDSA-AES128-SHA` (0xc005)
- C an alert
- D something else (`RC4_MD5`) (0x0005)

Actually, it is easy to explain

- a ciphersuite is represented by a 16-bit integer
- for almost a decade, all suites had their first byte equal to 00
- why bother to inspect this byte?

Implementation Flaws

Common programming errors

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

- CVE-2002-0862 (and CVE-2011-0228): `BasicConstraints` checks (missing check) in Windows (and iOS)

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

- CVE-2002-0862 (and CVE-2011-0228): `BasicConstraints` checks (missing check) in Windows (and iOS)
- CVE-2014-1266: Apple's `goto fail` (dead code)

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

- CVE-2002-0862 (and CVE-2011-0228): `BasicConstraints` checks (missing check) in Windows (and iOS)
- CVE-2014-1266: Apple's `goto fail` (dead code)
- CVE-2014-0092: GnuTLS' `goto fail` (logic error)

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

- CVE-2002-0862 (and CVE-2011-0228): `BasicConstraints` checks (missing check) in Windows (and iOS)
- CVE-2014-1266: Apple's `goto fail` (dead code)
- CVE-2014-0092: GnuTLS' `goto fail` (logic error)
- CVE-2014-0160: OpenSSL's Heartbleed (buffer overread)

Common programming errors in TLS

Selection of classical programming errors in SSL/TLS stacks

- CVE-2002-0862 (and CVE-2011-0228): `BasicConstraints` checks (missing check) in Windows (and iOS)
- CVE-2014-1266: Apple's `goto fail` (dead code)
- CVE-2014-0092: GnuTLS' `goto fail` (logic error)
- CVE-2014-0160: OpenSSL's Heartbleed (buffer overread)
- CVE-2014-6321: WinShock (buffer overflow) in Windows

Focus on GnuTLS' `goto fail` (CVE-2014-0092)

The bug allows an attacker to circumvent client-side checks regarding server certificates (source: `lwn.net`, March 2014)

> *The `check_if_ca` function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

Focus on GnuTLS' `goto fail` (CVE-2014-0092)

The bug allows an attacker to circumvent client-side checks regarding server certificates (source: `lwn.net`, March 2014)

> *The `check_if_ca` function is supposed to return true (any non-zero value in C) or false (zero) depending on whether the issuer of the certificate is a certificate authority (CA). A true return should mean that the certificate passed muster and can be used further, but the bug meant that error returns were misinterpreted as certificate validations.*

A similar bug was found in OpenSSL... in 2008 (CVE-2008-5077)!

> *The fix replaces a `if (!i)` with a `if (i<=0)`, where `i` is returned by a function checking a certificate which was interpreted as a boolean without taking into account other values corresponding to error codes*

Lessons learned

Observations

- all major TLS stacks have experienced problems
- sometimes, similar bugs have resurfaced in different stacks several years apart
- we should not put all the blame on the developers

Lessons learned

Observations

- all major TLS stacks have experienced problems
- sometimes, similar bugs have resurfaced in different stacks several years apart
- we should not put all the blame on the developers

Possible solutions

- better test suites (including negative checks, ideally shared between implementations)
- better programming languages and tools

Lessons learned

Observations

- all major TLS stacks have experienced problems
- sometimes, similar bugs have resurfaced in different stacks several years apart
- we should not put all the blame on the developers

Possible solutions

- better test suites (including negative checks, ideally shared between implementations)
- better programming languages and tools

What about TLS 1.3?

- with regards to these particular bugs, not much...

Implementation Flaws

Parsing bugs

Parsing bugs

Parsing complex structures can trigger memory management issues in low-level languages such as C

Parsing bugs

Parsing complex structures can trigger memory management issues in low-level languages such as C

Beyond them, parsing errors can lead to confusion on the interpreted value

- CVE-2009-2408: Null characters in Distinguished Names (ASN.1 Strings) in Firefox (and others)
- CVE-2014-1568: NSS/CyaSSL/PolarSSL Signature Forgery (ASN.1 Length Encodings)

Parsing bugs

Parsing complex structures can trigger memory management issues in low-level languages such as C

Beyond them, parsing errors can lead to confusion on the interpreted value

- CVE-2009-2408: Null characters in Distinguished Names (ASN.1 Strings) in Firefox (and others)
- CVE-2014-1568: NSS/CyaSSL/PolarSSL Signature Forgery (ASN.1 Length Encodings)
- CVE-2014-3511: OpenSSL downgrade attack (Record splitting)
- 2013: the Alert attack (Record boundaries) in OpenSSL (and others)
- *CVE-2014-0160: OpenSSL's Heartbleed (Record boundaries)*

Focus on the Signature Forgery attack (BERserk)

An RSA signature is a signature over an ASN.1 structure embedding the actual hash value

Focus on the Signature Forgery attack (BERserk)

An RSA signature is a signature over an ASN.1 structure embedding the actual hash value

Several implementations exhibited great laxism in the way ASN.1 messages are parsed

- arbitrary length encoding
- integer overflow in length values

This led to the ability for an attacker to forge ASN.1 messages for an arbitrary hash value, accepted by major implementations (including NSS)

Focus on the Signature Forgery attack (BERserk)

An RSA signature is a signature over an ASN.1 structure embedding the actual hash value

Several implementations exhibited great laxism in the way ASN.1 messages are parsed

- arbitrary length encoding
- integer overflow in length values

This led to the ability for an attacker to forge ASN.1 messages for an arbitrary hash value, accepted by major implementations (including NSS)

Reminder: the ASN.1 message should be DER-encoded, which is a strict set of encoding rules

Lessons learned

Observations

- complex structures lead to complex code and to bugs
- corner cases need to be explicited (and tested)
- when dealing with security, the Postel law is dangerous

Lessons learned

Observations

- complex structures lead to complex code and to bugs
- corner cases need to be explicited (and tested)
- when dealing with security, the Postel law is dangerous

Possible solutions

- better languages, tools and tests
- more formal specifications
- when applicable, prefer reconstructing a value rather than parsing and validating it

Lessons learned

Observations

- complex structures lead to complex code and to bugs
- corner cases need to be explicited (and tested)
- when dealing with security, the Postel law is dangerous

Possible solutions

- better languages, tools and tests
- more formal specifications
- when applicable, prefer reconstructing a value rather than parsing and validating it

What about TLS 1.3?

- not much
- … but some cases have been described and disambiguated

Implementation Flaws

The real impact of obsolete cryptography on security

Bleichenbacher (1/2)

RSA PKCS#1 v1.5

- RSA encryption requires a padding scheme
- how should we handle an invalid padding after decryption?

Bleichenbacher (1/2)

RSA PKCS#1 v1.5

- RSA encryption requires a padding scheme
- how should we handle an invalid padding after decryption?

Bleichenbacher attack (1998)

- main idea: send altered versions of a target encrypted message and observe the server behaviour
- if the attacker can distinguish a valid from an invalid padding, he can gather information on the plaintext
- this can be applied to TLS: the so-called “Million Message Attack”

Bleichenbacher (2/2)

The attack resurfaces in 2014

- in Java, a padding error triggers an exception
- so, to avoid a timing attack, one must redevelop the mechanism
- a TLS developer has to choose between modularity and security

Bleichenbacher (2/2)

The attack resurfaces in 2014

- in Java, a padding error triggers an exception
- so, to avoid a timing attack, one must redevelop the mechanism
- a TLS developer has to choose between modularity and security

... and in 2016

- DROWN *(Decrypting RSA with Obsolete and Weakened eNcryption)*
- attacking SSLv2 to recover a TLS pre-master secret

Bleichenbacher (2/2)

The attack resurfaces in 2014

- in Java, a padding error triggers an exception
- so, to avoid a timing attack, one must redevelop the mechanism
- a TLS developer has to choose between modularity and security

... and in 2016

- DROWN *(Decrypting RSA with Obsolete and Weakened eNcryption)*
- attacking SSLv2 to recover a TLS pre-master secret

... and in 2017 with ROBOT (Return Of Bleichenbacher's Oracle Threat)... and in 2018 with CAT

Similar issues in the symmetric domain

Dangerous or fragile constructions also exist in the Record Protocol (which protects the application data with symmetric cryptography):

- RC4
- the CBC mode used with the MAC-then-Encrypt paradigm
- invalid GCM nonce reuse breaking the integrity protection

Similar issues in the symmetric domain

Dangerous or fragile constructions also exist in the Record Protocol (which protects the application data with symmetric cryptography):

- RC4
- the CBC mode used with the MAC-then-Encrypt paradigm
- invalid GCM nonce reuse breaking the integrity protection

There again, developers must make hard choices to ensure compatibility while keeping their code maintainable and secure...

Lessons learned

Observations

- obsolete crypto can put developers in an impossible situation where they have to choose between modularity, compatibility and security
- the situation always gets worse with time (a.k.a Schneier's "Attacks always get better")

Lessons learned

Observations

- obsolete crypto can put developers in an impossible situation where they have to choose between modularity, compatibility and security
- the situation always gets worse with time (a.k.a Schneier's "Attacks always get better")

Possible solutions

- drop obsolete and fragile constructions as soon as possible
- constrain the standard so that only correct uses are compliant

Lessons learned

Observations

- obsolete crypto can put developers in an impossible situation where they have to choose between modularity, compatibility and security
- the situation always gets worse with time (a.k.a Schneier's "Attacks always get better")

Possible solutions

- drop obsolete and fragile constructions as soon as possible
- constrain the standard so that only correct uses are compliant

What about TLS 1.3?

- only modern robust constructions have been standardized
- *the only remaining legacy crypto is the RSA PKCS#1 v1.5 signature scheme used in certificates*

Implementation Flaws

The consequences of complex state machines

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

- Certificate client authentication (using elliptic curves) relies on two messages

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

- Certificate client authentication (using elliptic curves) relies on two messages
- `Certificate`, with the certificate chains
  - it contains the used elliptic curve
  - in particular, the size S of the underlying field

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

- Certificate client authentication (using elliptic curves) relies on two messages
- `Certificate`, with the certificate chains
  - it contains the used elliptic curve
  - in particular, the size S of the underlying field
- `CertificateVerify`, which contains a signature over the previous Handshake messages
  - this signature contains the coordinates of a point, of size l
  - SChannel did not verify that the l bytes would fit in the previously allocated memory area for S bytes...

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

- Certificate client authentication (using elliptic curves) relies on two messages
- `Certificate`, with the certificate chains
  - it contains the used elliptic curve
  - in particular, the size S of the underlying field
- `CertificateVerify`, which contains a signature over the previous Handshake messages
  - this signature contains the coordinates of a point, of size l
  - SChannel did not verify that the l bytes would fit in the previously allocated memory area for S bytes...

But certificate client authentication is optional and rarely used?

WinShock

WinShock is a simple buffer overflow in Microsoft's stack... but is it?

- Certificate client authentication (using elliptic curves) relies on two messages
- `Certificate`, with the certificate chains
  - it contains the used elliptic curve
  - in particular, the size S of the underlying field
- `CertificateVerify`, which contains a signature over the previous Handshake messages
  - this signature contains the coordinates of a point, of size l
  - SChannel did not verify that the l bytes would fit in the previously allocated memory area for S bytes...

But certificate client authentication is optional and rarely used?

In the default setting, all vulnerable servers nevertheless interpeted unsollicited messages, making them exploitable in practice

Shaky state machines

Most TLS stacks proved to be vulnerable to such issues in state machines, where invalid paths could be triggered, sometimes leading to security flaws

Shaky state machines

Most TLS stacks proved to be vulnerable to such issues in state machines, where invalid paths could be triggered, sometimes leading to security flaws

- CVE-2014-0224: EarlyCCS (skip key installation) in OpenSSL
- CVE-2014-6593: Early Finished (server impersonation) in JSSE and CyaSSL

Shaky state machines

Most TLS stacks proved to be vulnerable to such issues in state machines, where invalid paths could be triggered, sometimes leading to security flaws

- CVE-2014-0224: EarlyCCS (skip key installation) in OpenSSL
- CVE-2014-6593: Early Finished (server impersonation) in JSSE and CyaSSL
- Skip Verify (client impersonation) in Mono stack, CyaSSL and OpenSSL

Shaky state machines

Most TLS stacks proved to be vulnerable to such issues in state machines, where invalid paths could be triggered, sometimes leading to security flaws

- CVE-2014-0224: EarlyCCS (skip key installation) in OpenSSL
- CVE-2014-6593: Early Finished (server impersonation) in JSSE and CyaSSL
- Skip Verify (client impersonation) in Mono stack, CyaSSL and OpenSSL
- CVE-2015-0204: FREAK (server impersonation) in OpenSSL, Apple SecureTransport and Microsoft SChannel and many others

Lessons learned

Observations

- all major impementations did not correctly handle TLS state machine
- maybe SSL/TLS is too complex?

Lessons learned

Observations

- all major impementations did not correctly handle TLS state machine
- maybe SSL/TLS is too complex?

Possible solutions

- test all possible message sequences
- write crystal clear specifications, including a reference automaton
- drop support for old/useless/dangerous versions/options

Lessons learned

Observations

- all major impementations did not correctly handle TLS state machine
- maybe SSL/TLS is too complex?

Possible solutions

- test all possible message sequences
- write crystal clear specifications, including a reference automaton
- drop support for old/useless/dangerous versions/options

What about TLS 1.3?

- on one hand, TLS 1.3 simplified the messages
- ... but 0 RTT mode is a complex beast in TLS 1.3
- ... but TLS 1.3 added fake messages to accomodate middleboxes
- ... and we might have to live at least with TLS 1.2 for some time

Conclusion and Take away messages

Complexity leads to insecurity

Implementation flaws can happen at different levels

Specifications can help avoid complications

- better and unamibguous message formats
- up-to-date cryptographic primitives
- simple and formally-defined state machines

This would lead the standard to constrain implementers

Better languages, tools and methodologies

Several bugs could be avoided by using modern development tools

- modern programming languages
- strict compilers and static analysers
- tests, tests, tests

Beyond TLS 1.3

TLS 1.3 improved some implementation aspects

- but it also created new complexities
- and previous versions are far from gone

The IETF is currently standardizing QUIC

- a new secure transport layer on top of UDP
- reusing TLS 1.3
- with very complex constructions...

Questions?

Thank you for your attention

@pictyeye
olivier.levillain@telecom-sudparis.eu
https://paperstreet.picty.org/yeye