The (In)security of Network Protocol Implementations

Olivier Levillain

TELECOM SudParis
IP PARIS

Séminaire Sotern
9 février 2023

/me

Career

- Internship in cryptography on a hash function (2006)
- Member of the “system” lab at ANSSI (2007-2012)
- Head of the “network” lab at ANSSI (2012-2015)
- Head of the training center at ANSSI (2015-2018)
- Associate Professor at Télécom SudParis (2018-)

/me

Career

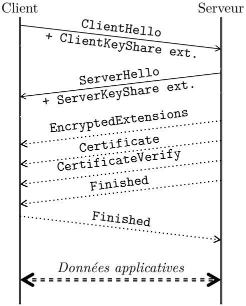
- Internship in cryptography on a hash function (2006)
- Member of the “system” lab at ANSSI (2007-2012)
- Head of the “network” lab at ANSSI (2012-2015)
- Head of the training center at ANSSI (2015-2018)
- Associate Professor at Télécom SudParis (2018-)

Research

- Contribution to the study of low-level x86 mechanisms
- PhD thesis on SSL/TLS
- Interest in programming languages
- Work on parsers and network protocol implementations

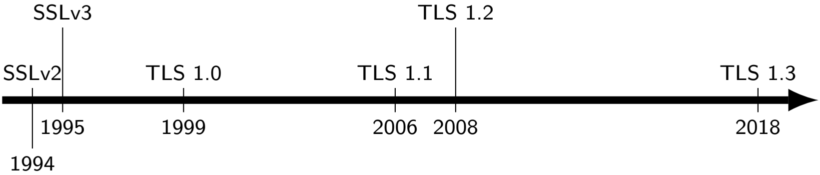
Parsing Network Messages

TLS in a Slide



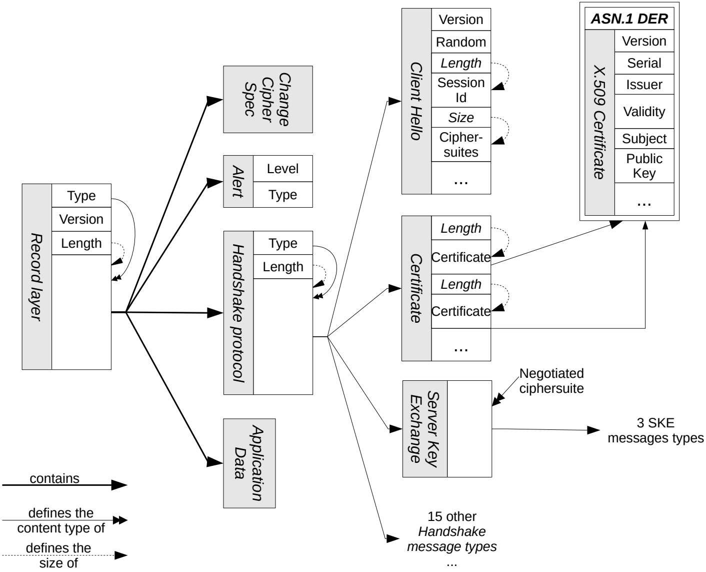
TLS Goals

- Authenticate the server
- Establish a shared secret
- Protect application data in confidentiality and integrity



More information on TLS in [PhD16] et [CRiSIS20]

Parsing TLS Messages (1/2)



Parsing TLS Messages (2/2)

But parsing TLS messages is hard

- Many complex structures, especially in Handshake messages
- Interactions with cryptographic algorithms

Parsing TLS Messages (2/2)

But parsing TLS messages is hard

- Many complex structures, especially in Handshake messages
  - OK, let's only consider record parsing, splitting and merging
- Interactions with cryptographic algorithms

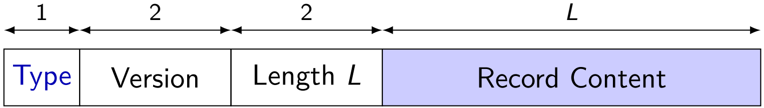
Parsing TLS Messages (2/2)

But parsing TLS messages is hard

- Many complex structures, especially in Handshake messages
  - OK, let's only consider record parsing, splitting and merging
- Interactions with cryptographic algorithms
  - OK, let's just look at the cleartext messages at the start of a connection

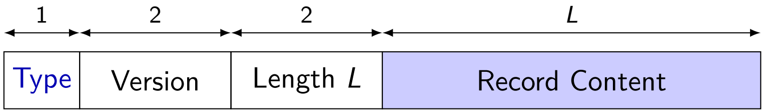
TLS Records — The Good

From SSLv3 to the latest versions of TLS, TLS messages are transported using records



TLS Records — The Good

From SSLv3 to the latest versions of TLS, TLS messages are transported using records

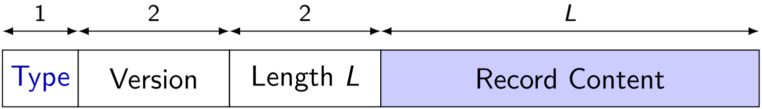


The records can transport different types of messages

- Handshake
- Alert
- ChangeCipherSpec (mostly removed with TLS 1.3)
- ApplicationData
- Hearbeat (available via an extension)

TLS Records — The Good

From SSLv3 to the latest versions of TLS, TLS messages are transported using records



The records can transport different types of messages

- Handshake
- Alert
- ChangeCipherSpec (mostly removed with TLS 1.3)
- ApplicationData
- Hearbeat (available via an extension)

How hard can it be to parse records and send them to the right handler?

TLS Records — The Bad (1/2)

Handshake messages can be longer than the record transporting them

TLS Records — The Bad (1/2)

Handshake messages can be longer than the record transporting them

- Handshake length is defined by a 24-bit field
- Record length is defined by a 16-bit field

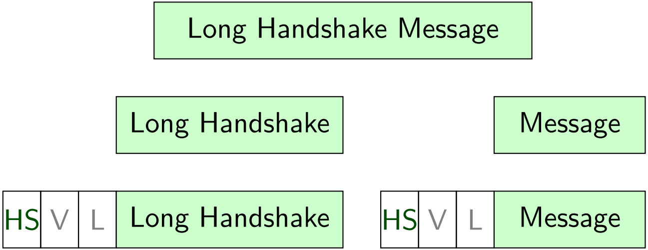
TLS Records — The Bad (1/2)

Handshake messages can be longer than the record transporting them

- Handshake length is defined by a 24-bit field
- Record length is defined by a 16-bit field
- Such messages must then be split across several records

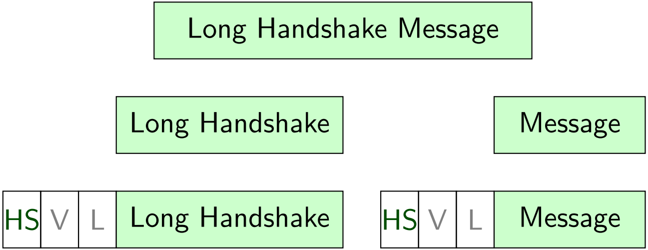
TLS Records — The Bad (1/2)

Handshake messages can be longer than the record transporting them

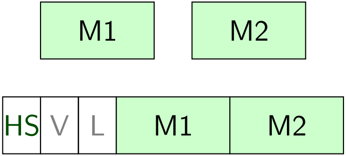


TLS Records — The Bad (1/2)

Handshake messages can be longer than the record transporting them

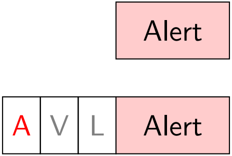


Multiple Handshake messages can also be grouped in the same record



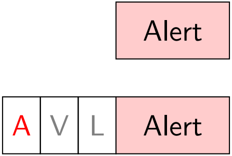
TLS Records — The Bad (2/2)

Other messages must fit exactly in one record



TLS Records — The Bad (2/2)

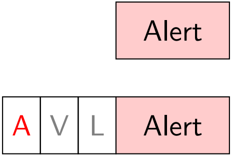
Other messages must fit exactly in one record



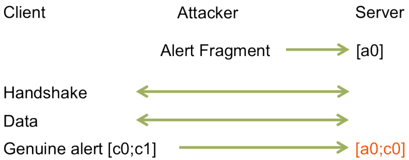
Actually, this was only specified this way recently...

TLS Records — The Bad (2/2)

Other messages must fit exactly in one record



Actually, this was only specified this way recently... following a report from the Inria Prosecco team in 2012 about a strange OpenSSL behavior



Source: `https://www.mitls.org/pages/attacks/Alert`

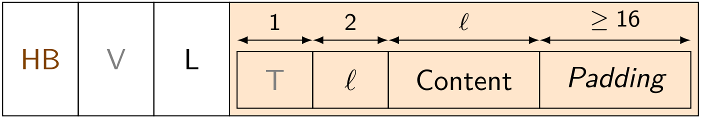
TLS Records — The Ugly (1/2)

Hearbeat messages (RFC 6520) are variable-length messages

- Keep-alive messages that should be echoed
- The variable length is for Path MTU Discovery

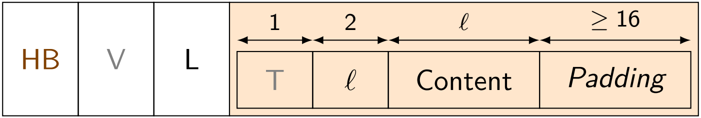
TLS Records — The Ugly (1/2)

Hearbeat messages (RFC 6520) are variable-length messages



TLS Records — The Ugly (1/2)

Hearbeat messages (RFC 6520) are variable-length messages

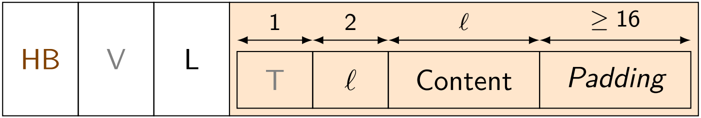


What should we do when $L < \ell + 19$?

- Reject the record

TLS Records — The Ugly (1/2)

Hearbeat messages (RFC 6520) are variable-length messages

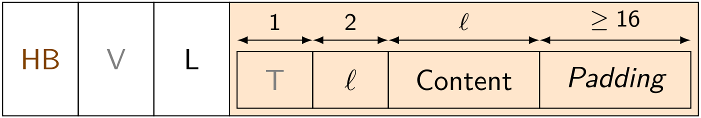


What should we do when $L < \ell + 19$?

- Reject the record
- Wait for the next record to get the complete Heartbeat message

TLS Records — The Ugly (1/2)

Hearbeat messages (RFC 6520) are variable-length messages



What should we do when $L < \ell + 19$?

- Reject the record
- Wait for the next record to get the complete Heartbeat message
- **do as if everything was OK and read beyond the end of the record**

The RFC did not clearly state that a Heartbeat record must contain **exactly one** message...

TLS Records — The Ugly (2/2)

In a TLS connection, the first message sent by the client is `ClientHello`

- It starts with the Handshake Type (1 byte)
- Then, it contains the Handshake Length (3 bytes)
- The actual `ClientHello` starts with the maximum supported version

TLS Records — The Ugly (2/2)

In a TLS connection, the first message sent by the client is `ClientHello`

- It starts with the Handshake Type (1 byte)
- Then, it contains the Handshake Length (3 bytes)
- The actual `ClientHello` starts with the maximum supported version

Record splitting before the protection is activated is *not* authenticated

- Only the stream of Handshake messages are authenticated

TLS Records — The Ugly (2/2)

In a TLS connection, the first message sent by the client is `ClientHello`

- It starts with the Handshake Type (1 byte)
- Then, it contains the Handshake Length (3 bytes)
- The actual `ClientHello` starts with the maximum supported version

Record splitting before the protection is activated is *not* authenticated

- Only the stream of Handshake messages are authenticated

OpenSSL requires to know from the first message which TLS version the client is advertizing

TLS Records — The Ugly (2/2)

In a TLS connection, the first message sent by the client is `ClientHello`

- It starts with the Handshake Type (1 byte)
- Then, it contains the Handshake Length (3 bytes)
- The actual `ClientHello` starts with the maximum supported version

Record splitting before the protection is activated is *not* authenticated

- Only the stream of Handshake messages are authenticated

OpenSSL requires to know from the first message which TLS version the client is advertizing

What happens when an attacker splits the `ClientHello` over very small chunks (less than 6 bytes) ?

- OpenSSL assumes the client version is TLS 1.0
- This can not be detected or forbidden
- CVE-2014-3511 (Downgrade Attack)

Discussions About Message Parsing

TLS record parsing

- A seemingly simple problem
- That triggered many interesting bugs

Discussions About Message Parsing

TLS record parsing

- A seemingly simple problem
- That triggered many interesting bugs

What about more complex protocols such as QUIC?

- Variable integer fields
- QUIC crypto frames can be split and contain an `Offset` fields (leading to potential reassembly issues)
- A convoluted encryption scheme

Discussions About Message Parsing

TLS record parsing

- A seemingly simple problem
- That triggered many interesting bugs

What about more complex protocols such as QUIC?

- Variable integer fields
- QUIC crypto frames can be split and contain an `Offset` fields (leading to potential reassembly issues)
- A convoluted encryption scheme

What about complex file formats such as PDF?

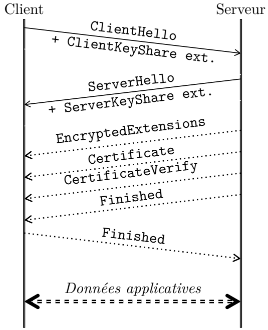
- ...

More information on QUIC in [WISTP19] and on PDF in [LangSec17]

More information on TLS in [PhD16] et [CRiSIS20]

State Machines Gone Crazy

An Example of a Problematic TLS State Machine

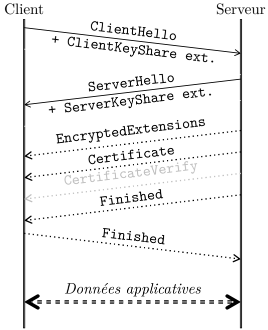


In TLS 1.3, the expected message flow is the following

- The server identifies itself (`Certificate`)
- It proves is identity (`CertificateVerify`)
- This message contains a signature requiring access to the server private key

Work with AT. Rasoamanana in the GASP project [RESSI20, ESORICS22]

An Example of a Problematic TLS State Machine



In TLS 1.3, the expected message flow is the following

- The server identifies itself (`Certificate`)
- It proves is identity (`CertificateVerify`)
- This message contains a signature requiring access to the server private key

What happens if a client accepts a connection where the `CertificateVerify` is missing?

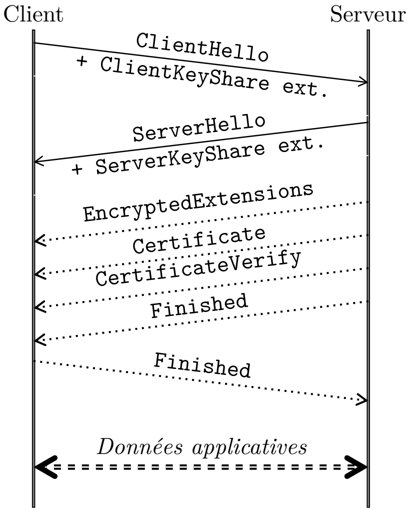
- It is not necessary anymore to know the private key to make the handshake work
- An attacker can impersonate *any server* with such a client

Work with AT. Rasoamanana in the GASP project [RESSI20, ESORICS22]

State Machine Representation

Traditional Representation

- The “serpent” diagram
- Only show the happy path



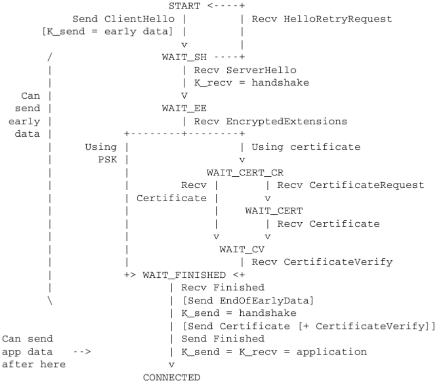
State Machine Representation

Traditional Representation

- The “serpent” diagram
- Only show the happy path

Informal State Machine

- A formalization effort
- Here, the client perspective
- Some ambiguities remain



— RFC 8446 (TLS 1.3) Appendix A

State Machine Representation

Traditional Representation

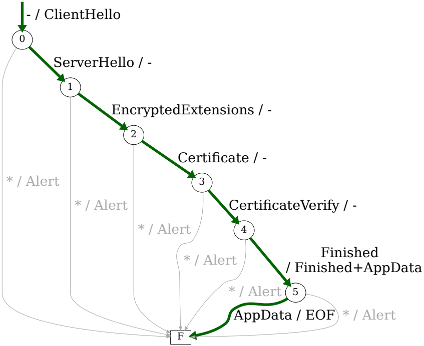
- The “serpent” diagram
- Only show the happy path

Informal State Machine

- A formalization effort
- Here, the client perspective
- Some ambiguities remain

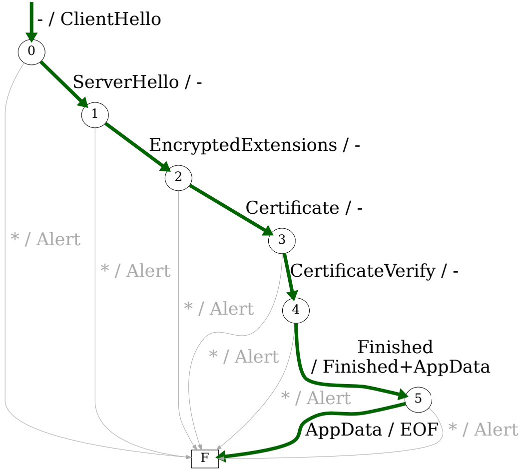
Mealy Machine

- A more formal description
- Heavier representation

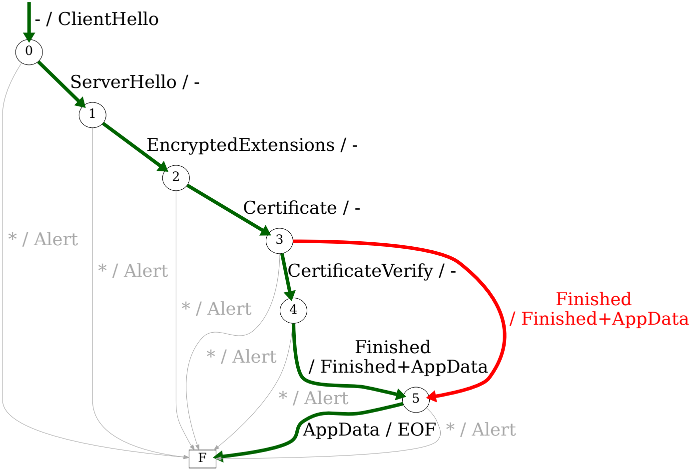


— Results from the GASP project

Highlighting CVE 2020-24613 on wolfSSL



Highlighting CVE 2020-24613 on wolfSSL



State Machine Inference

It is possible to infer the state machines from a stack in a black-box approach

- $L^\star$ algorithm (Angluin, 1987)
- Adaptation to Mealy machines used in many contexts
- State machine inference for various protocols (ex.: TLS, H2)
- *(Other approaches exist, e.g. by mutating a reference transcript)*

State Machine Inference

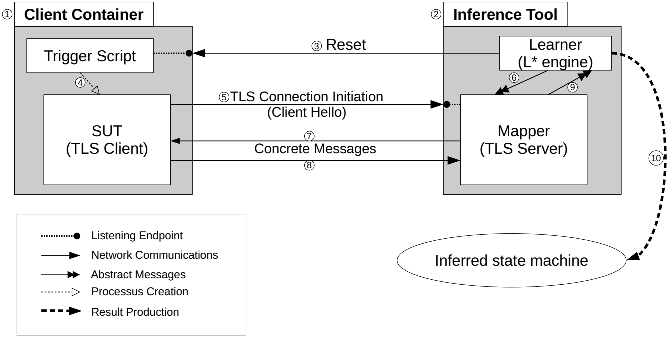
It is possible to infer the state machines from a stack in a black-box approach

- $L^\star$ algorithm (Angluin, 1987)
- Adaptation to Mealy machines used in many contexts
- State machine inference for various protocols (ex.: TLS, H2)
- *(Other approaches exist, e.g. by mutating a reference transcript)*

Application to secure communication protocols

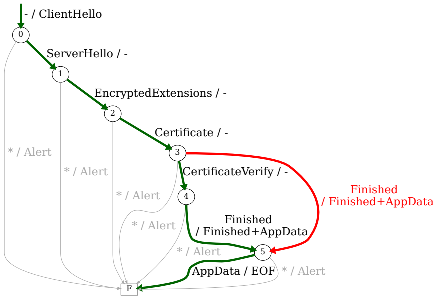
- Systematic research of authentication shortcuts
- Highlight loops in the state machine
- Exploit differences between state machines for fingerprinting purposes

Our Methodology



- $>$ 400 versions of client and server open source implementations
- OpenSSL, GnuTLS, wolfssl, NSS...

Results on TLS Stacks: Authentication Bypasses



- EarlyCCS (CVE-2014-0224) and FREAK (2015-0204) on OpenSSL detected
- **CVE-2020-24613 reproduced on wolfSSL**
- Three new CVEs on wolfSSL TLS 1.3 client and server

Results on TLS Stacks: Unexpected Loops

Stack	**Scenario**	**Messages**	**Max. Time Between Msgs**
erlang 24	1.0/1.2 Server	`NoRenegotiation` Alert or `ApplicationData`	$>$ 1 hour*
fizz 22.01.24	1.3 Client	`ChangeCipherSpec`	$>$ 1 hour
matrixssl 4.0 - 4.3	1.0/1.2 Server	`NoRenegotiation` Alert	$\approx$ 40 seconds
NSS 3.15 - 3.78	1.0/1.2 Server	`NoRenegotiation` Alert	$>$ 1 hour
OpenSSL $<$ 1.1.0	1.0/1.2 Server	Empty `ApplicationData`	$>$ 1 hour

Result on TLS Stacks: Fingerprinting (1/2)

For a given scenario (role, TLS version, option)

- Different stacks *always* produce different state machines
- Consecutive versions of the same stack can share a state machine
- Extracting distinguishing sequences leads to a fingerprinting tool
- Complementary to other fingerprinting approaches

Result on TLS Stacks: Fingerprinting (1/2)

For a given scenario (role, TLS version, option)

- Different stacks *always* produce different state machines
- Consecutive versions of the same stack can share a state machine
- Extracting distinguishing sequences leads to a fingerprinting tool
- Complementary to other fingerprinting approaches

TLS 1.3 servers can be put in 13 classes using 8 sequences

`CloseNotify`	`ClientHello Certificate`
`ClientHello Certificate`	`ClientHello Finished CloseNotify`
`ClientHello ClientHello`	`ClientHello EmptyCertificate CertificateVerify`
`ClientHello CloseNotify`	`ClientHello EmptyCertificate InvalidCertificateVerify`

Result on TLS Stacks: Fingerprinting (2/2)

Stack	**Versions**	*N*
erlang	24.0.3 - 24.2.1	9
GnuTLS	3.6.16 - 3.7.2	4
matrixssl	4.0.0 - 4.1.0	4
	4.2.1 - 4.3.0	6
NSS	3.39 - 3.40	4
	3.41 - 3.78	4
OpenSSL	1.1.1a - 1.1.1n	4
	3.0.0 - 3.0.2	4
wolfSSL	3.15.5 - 4.0.0	7
	4.1.0 - 4.6.0	7
	4.7.0 - 4.8.1	7
	5.0.0 - 5.1.1	7
	5.2.0	6

Work in Progress on SSH and OPC-UA

SSH

- A 3-stage Protocol: Transport, Authentication, Connection (overall, 30 messages)
- Natural Loops (renegotiation)
- Connection messages are complex to handle (multiple channels)
- OpenSSH, libssh, asyncssh, dropbear, wolfssh

OPC-UA

- Industrial Control Systems / SCADA
- A rather sketchy specification
- Various implementations in .Net, C, Python, Rust

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,
- we can initiate a renegotiation (`KEXINIT`)...

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,
- we can initiate a renegotiation (`KEXINIT`)...
- ask for *n* new channels (`CHANNEL_OPEN`)...

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,
- we can initiate a renegotiation (`KEXINIT`)...
- ask for *n* new channels (`CHANNEL_OPEN`)...
- and complete the renegotiation (`DH_INIT`)

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,
- we can initiate a renegotiation (`KEXINIT`)...
- ask for n new channels (`CHANNEL_OPEN`)...
- and complete the renegotiation (`DH_INIT`)
- which leads to the following answers after the last message: `DH_REPLY`, `NEWKEYS` and n times `OPEN_CONFIRM`

Challenges: Counting Parentheses

OpenSSH state machine can not be represented as a Mealy machine

- After the client authentication,
- we can initiate a renegotiation (KEXINIT)...
- ask for *n* new channels (CHANNEL_OPEN)...
- and complete the renegotiation (DH_INIT)
- which leads to the following answers after the last message: DH_REPLY, NEWKEYS and *n* times OPEN_CONFIRM

A solution to produce an approximate state machine

- Group the OPEN_CONFIRM answers as a fake OPEN_CONFIRM+ message

Challenges: Exploding State Machines

Inferring asyncssh state machine (Transport + Authentication layers)

- 5 + 5 messages in the vocabulary
- 360 states
- Problem with stacked Auth messages in the middle of a negotiation

Efficiency

Main efficiency problem with $L^\star$

- We keep waiting for the target responses
- A short timeout may lead to invalid or non-deterministic behavior
- The optimal timeout depends on the studied stack

Efficiency

Main efficiency problem with $L^\star$

- We keep waiting for the target responses
- A short timeout may lead to invalid or non-deterministic behavior
- The optimal timeout depends on the studied stack

Optimizations

- EOF is final (no need to explore sequences beyond an EOF
- Since $L^\star$ relies on a deterministic behavior, exploit the known responses
- Drastic improvement (25 times faster for a typical TLS inference)
- (Preliminary work to monitor the time wasted waiting for timeouts)

Discussions About State Machines

There is still room for improvement for most implementations

- Authentication bypasses
- Deviations from the standard
- Possible Denial of Service situations

$L^\star$ is a powerful tool

- Our approach aims at reproducibility and automation
- Work is still needed to improve the performance and tackle corner cases

More information in [ESORICS22]

Conclusion

Conclusion

Parsing messages for real-world protocols is hard

- Do not disregard the difficulty
- Encourage simple (and properly formalized) formats
- Stress test implementations

Conclusion

Parsing messages for real-world protocols is hard

- Do not disregard the difficulty
- Encourage simple (and properly formalized) formats
- Stress test implementations

State machines for real-world protocols are complex

- Fix ambiguous and incomplete specifications
- Discuss implementation choices leading to fingerprinting possibilities
- Send feedback to stack developers about deviations

Questions ?

Thank you for your attention

References

[LangSec16] *Caradoc: a pragmatic approach to PDF parsing and validation*. G. Endignoux, OL and J.-Y. Migeon. LangSec Workshop @ IEEE SSP 2016

[PhD16] *A study of the TLS ecosystem*. OL. PhD defended in 2016

[WISTP19] *Analysis of QUIC Session Establishment and its Implementations*. E. Gagliardi and OL

[CRiSIS20] *Implementations Flaws in TLS Stacks...*. OL

[RESSI20] *Le projet GASP: a Generic Approach to Secure network Protocols*. OL

[ESORICS22] *Towards a Systematic and Automatic Use of State Machine Inference to Uncover Security Flaws and Fingerprint TLS Stacks*. AT Rasoamanana, OL and H. Debar

Articles and resources available on `https://paperstreet.picty.org` and `https://gasp.ebfe.fr`