GASP: a Generic Approach to Secure network Protocols

Olivier Levillain Aina Toky Rasoamanana

RESSI 2026
27 mai 2026

Plan

Plan

Motivation

- Network protocols are defined in specs such as RFCs
- They are written in English (and not in a formal language)
  - ambiguities
  - incomplete specifications

Motivation

- Network protocols are defined in specs such as RFCs
- They are written in English (and not in a formal language)
  - ambiguities
  - incomplete specifications

In particular, studying TLS in the 2010's was interesting

- first IPv4-scale scans of the HTTPS ecosystem
- state machine issues affecting all major TLS stacks (e.g. FREAK)
- other high-profile security bugs such as Heartbleed

GASP: a Generic Approach to Secure network Protocols

General Idea:

- generalize tools and techniques from the TLS ecosystem
- to a wide variety of protocols (SSH, DNS, BGP, H2...)

GASP: a Generic Approach to Secure network Protocols

General Idea:

- generalize tools and techniques from the TLS ecosystem
- to a wide variety of protocols (SSH, DNS, BGP, H2...)

Project type: ANR JCJC (Jeune Chercheur Jeune Chercheuse)
Principal Investigator: Olivier Levillain (Samovar, Télécom Sudparis)
Consortium: Télécom SudParis
HR: Aina Toky Rasoamanana, PhD (defended in June 2023)
Friends of the project: ANSSI, Loria, Radboud University
TRL: 3-5
Dates: 2019-2024

Research Directions

- Internet scans and study of the ecosystems
- Derivation of implementations
- State-machine inference to test stacks

Research Directions

- Internet scans and study of the ecosystems
  - development of platforms for TLS and SSH stacks
- Derivation of implementations

- State-machine inference to test stacks

Research Directions

- Internet scans and study of the ecosystems
  - development of platforms for TLS and SSH stacks
- Derivation of implementations
  - focus on parser generators
- State-machine inference to test stacks

Research Directions

- Internet scans and study of the ecosystems
  - development of platforms for TLS and SSH stacks
- Derivation of implementations
  - focus on parser generators
- State-machine inference to test stacks
  - *we focus on this direction in the remaining of the presentation*

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D the answer D (`RC4_MD5`)

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

A `AES128-SHA`

B `ECDH-ECDSA-AES128-SHA`

C an alert

D the answer D (`RC4_MD5`)

Guess what, we can explain it:

- a ciphersuite is a 16-bit integer
- for many years, all valid suites started with a null byte (`00 XX`)
- why bother comparing the most significant byte?

Interlude: Why Real-World Stacks ~~Suck~~ are *Interesting*

What does a TLS server answer to a client proposing `AES128-SHA` and `ECDH-ECDSA-AES128-SHA`?

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

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

C an alert

D the answer D (`RC4_MD5`) (`0x0005`)

Guess what, we can explain it:

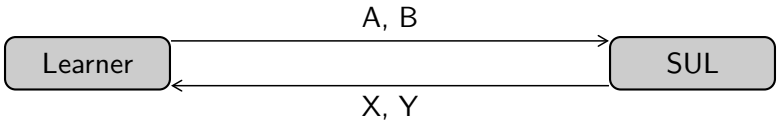
- a ciphersuite is a 16-bit integer
- for many years, all valid suites started with a null byte (`00 XX`)
- why bother comparing the most significant byte?

Plan

Examples of State Machine Flaws

- EarlyCCS (CVE-2014-0224)
- FREAK (CVE-2015-0204)
- WinShock (CVE-2014-6321)
- `libssh` Client Authentication Bypass (CVE-2018-10933)
- WolfSSL Server Authentication Bypass (CVE-2020-24613)

Methodology: Pipeline



SUL = System Under Learning (the stack being analyzed)

Process

Methodology: Pipeline

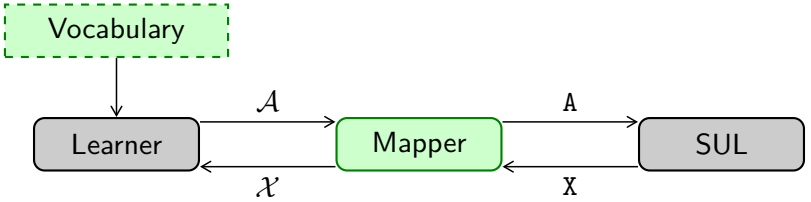
Real interactions require an intermediate component

- sending messages one at a time
- using concrete messages (bytes on the wire)

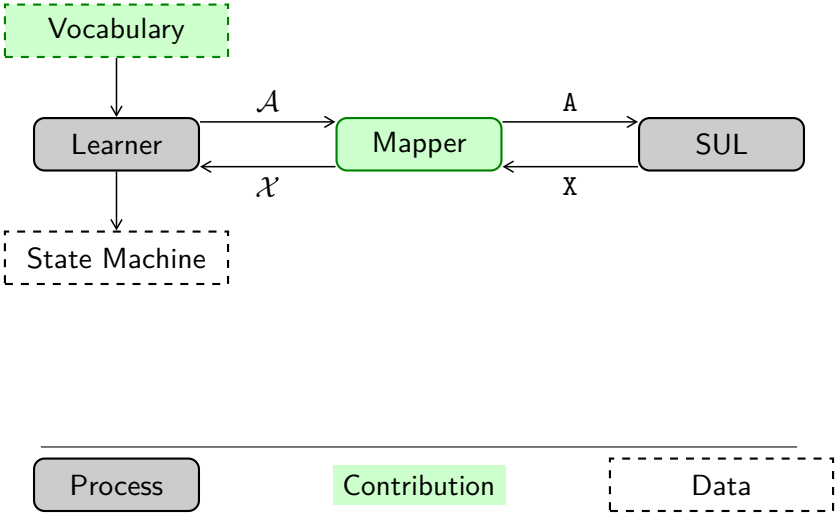
Process

Contribution

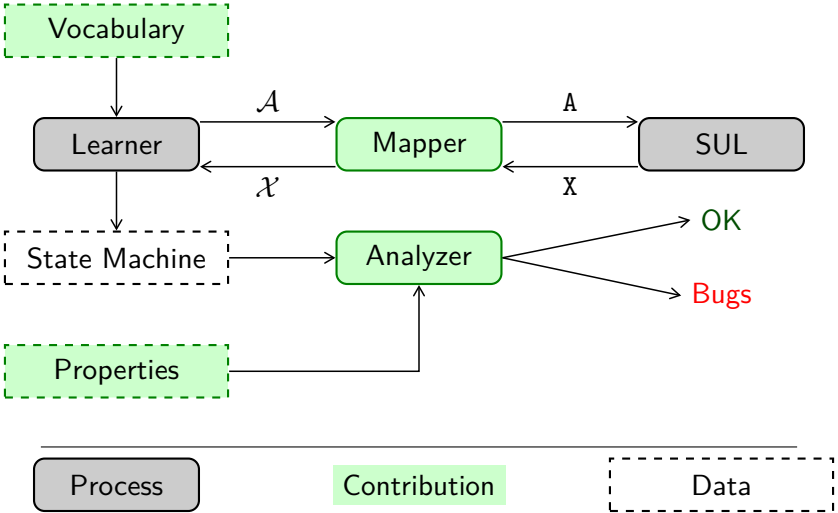
Methodology: Pipeline



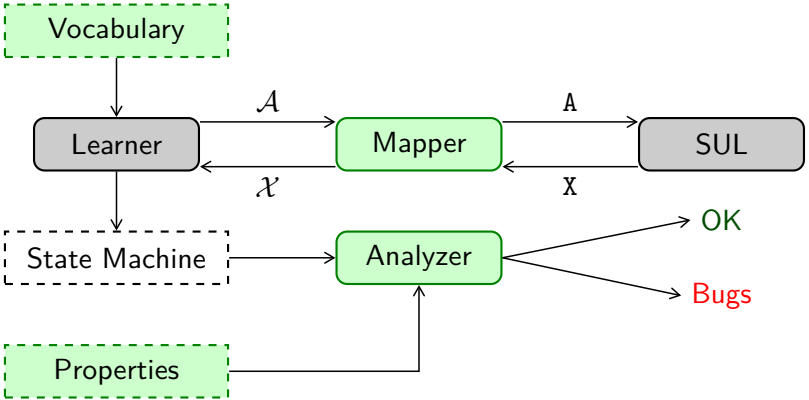
Methodology: Pipeline



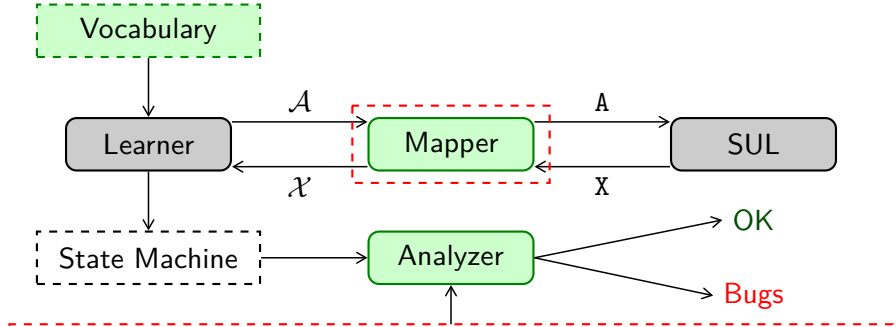
Methodology: Pipeline



Methodology: Pain Points



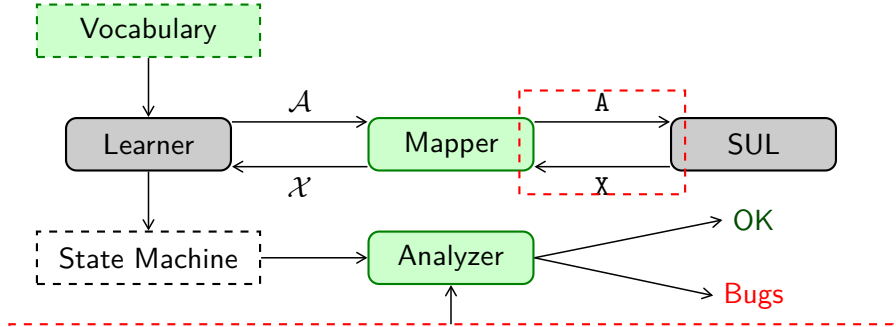
Methodology: Pain Points



Writing a new mapper for each protocol

- we (almost) need a protocol implementation...
- which is flexible and robust
- where every choice is ideally clear and explicit

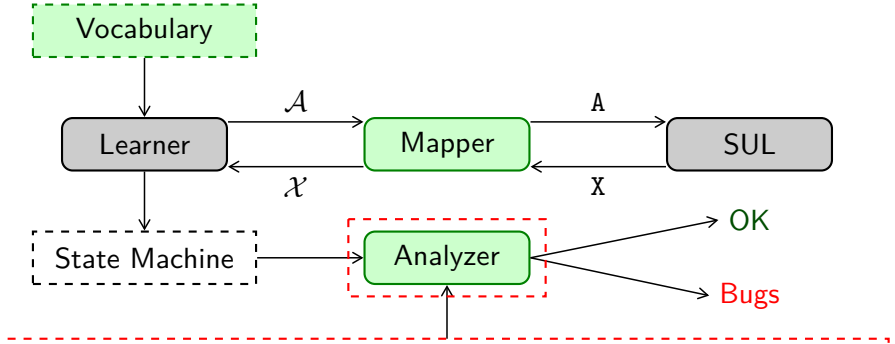
Methodology: Pain Points



Handling timeouts

- we do not know when SUL is done talking
- low values lead to missed messages
- high values lead to inefficiency

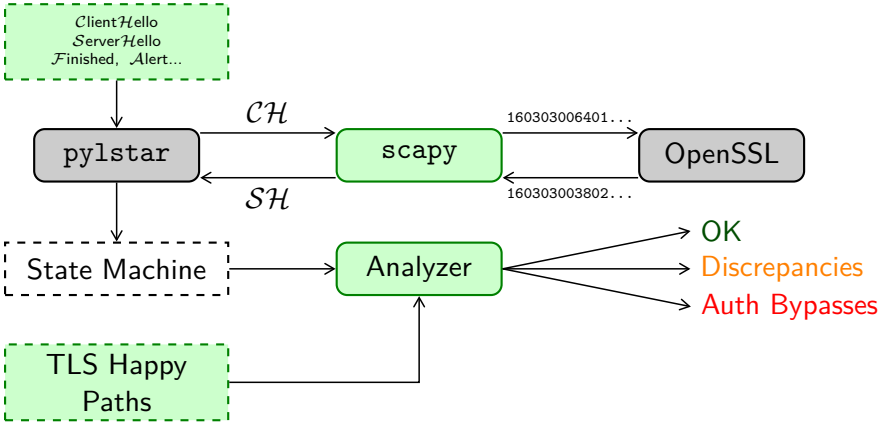
Methodology: Pain Points



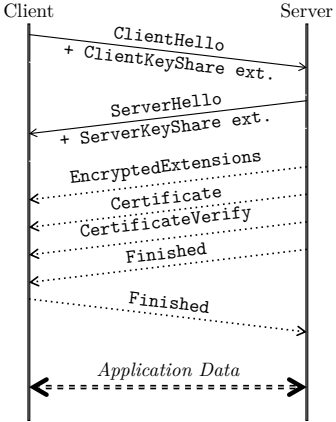
Analyzing the state machines

- the resulting automata can be huge and hard to read
- need for automatic tools for adhoc properties

Application to TLS: Authentication Bypasses [ESORICS22]



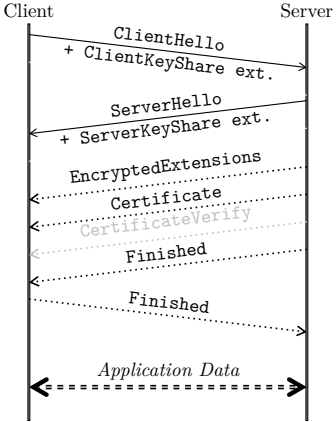
CVE-2020-24613: The Flaw



In a normal TLS 1.3 message flaw

- the server presents its (`Certificate`)
- it proves its identity (`CertificateVerify`)
- this message contains a signature (requiring the private key)

CVE-2020-24613: The Flaw

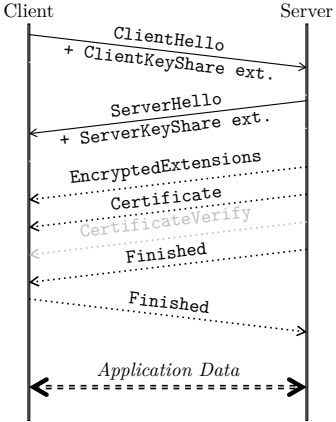


In a normal TLS 1.3 message flaw

- the server presents its (`Certificate`)
- it proves its identity (`CertificateVerify`)
- this message contains a signature (requiring the private key)

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

CVE-2020-24613: The Flaw



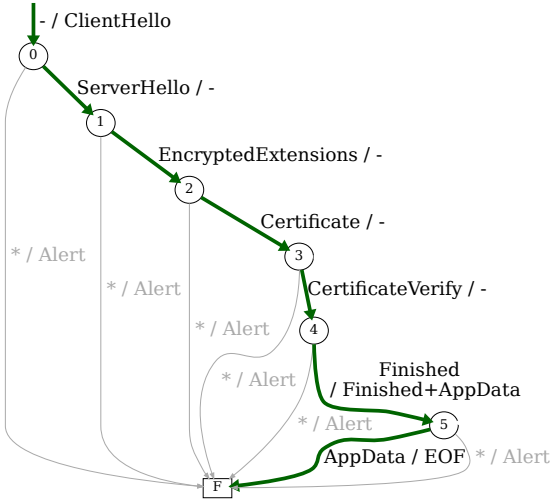
In a normal TLS 1.3 message flaw

- the server presents its (`Certificate`)
- it proves its identity (`CertificateVerify`)
- this message contains a signature (requiring the private key)

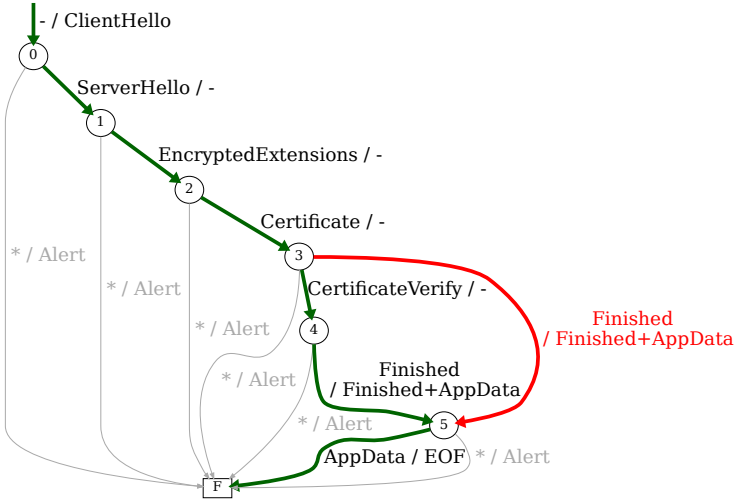
What happens if a client accepts a connection missing the `CertificateVerify`?

- the private key is not necessary anymore for a successful handshake
- an attacker can impersonate *any server* to such a client

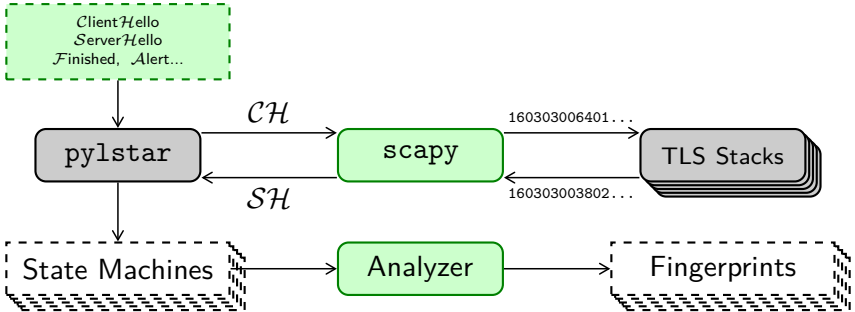
CVE-2020-24613: A Better Representation



CVE-2020-24613: A Better Representation



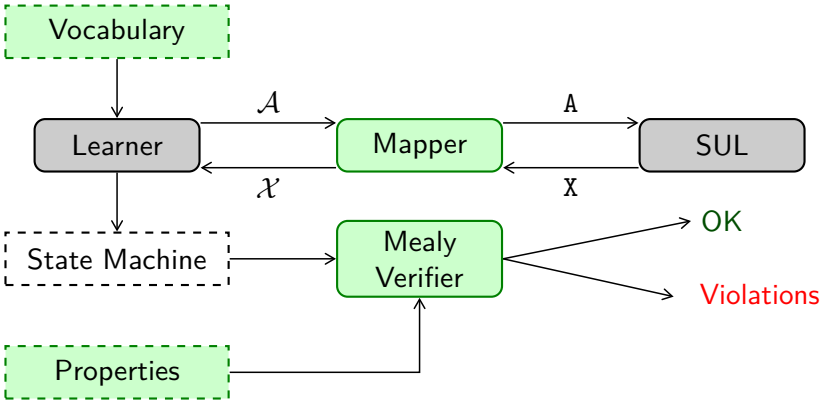
Application to TLS: Fingerprinting [ESORICS22]



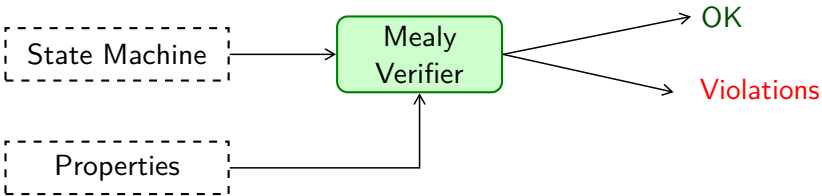
TLS stacks actually vary in their behavior

- the distinguishing sequences lead to fingerprints
- possibly more robust than other techniques

Application to OPC-UA: The Mealy Verifer [ARES24]



Application to SSH: The Mealy Verifer [ARES24]



Reproduction of existing results

- reuse of the inferred state machines
- transposition of the properties
- more precise results (thanks to the exhaustiveness)

Plan

SSH in a Nutshell (1/2)

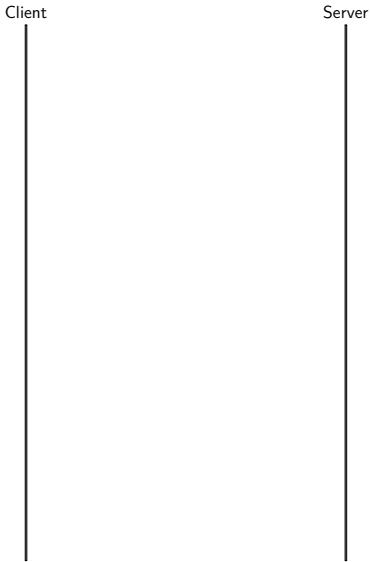
SSH in a Nutshell (1/2)

- SSH means SSH-2 in this presentation

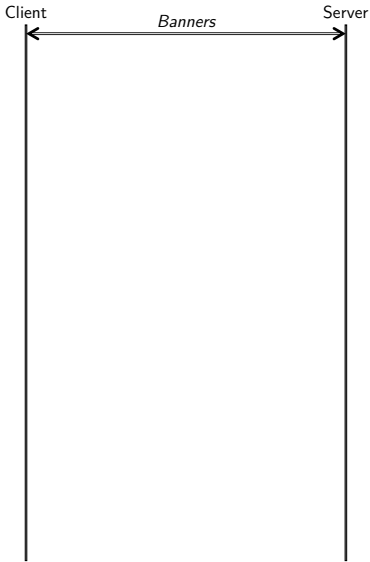
SSH in a Nutshell (1/2)

- SSH means SSH-2 in this presentation
- SSH is a 3-layer protocol
  - Transport: key exchange, server authentication and channel protection
  - User Authentication
  - Connection: multiplexing application data channels

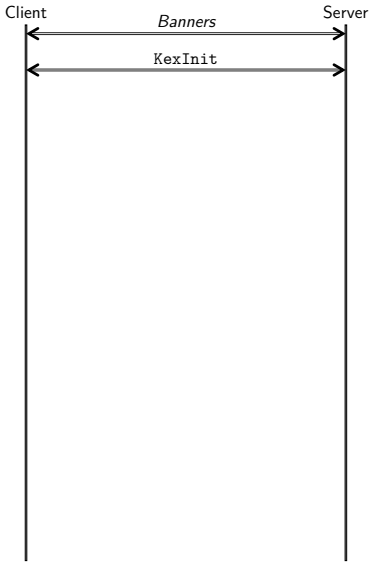
SSH in a Nutshell (2/2)



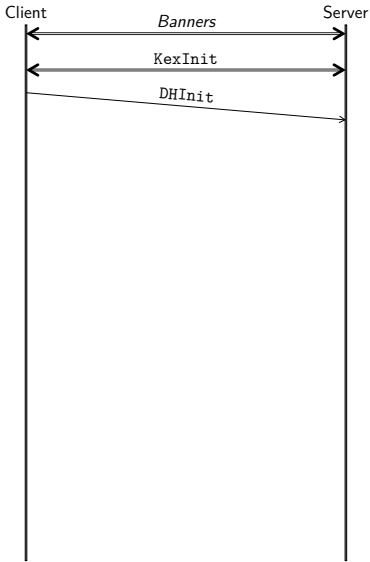
SSH in a Nutshell (2/2)



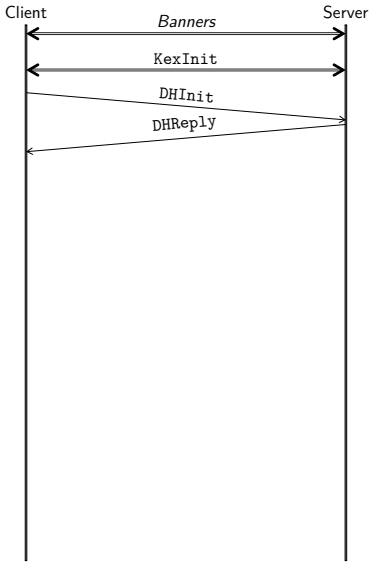
SSH in a Nutshell (2/2)



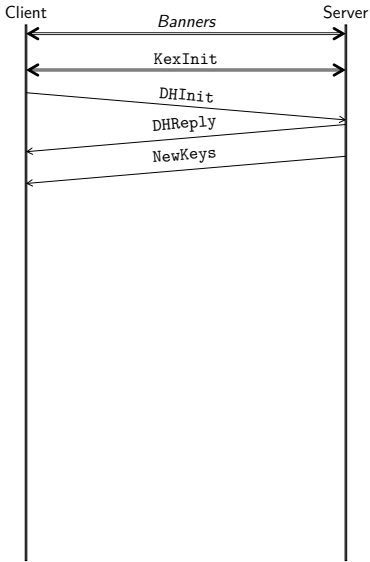
SSH in a Nutshell (2/2)



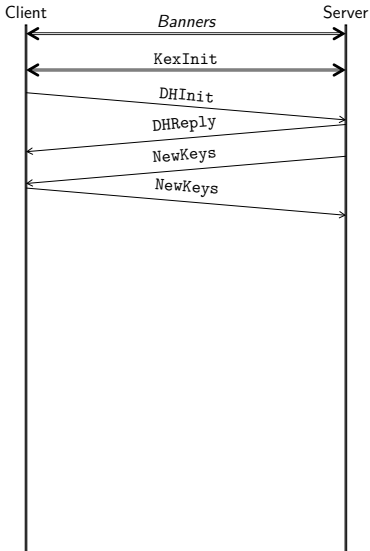
SSH in a Nutshell (2/2)



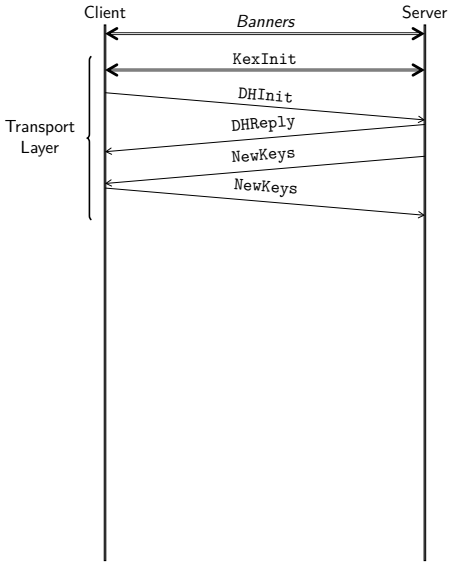
SSH in a Nutshell (2/2)



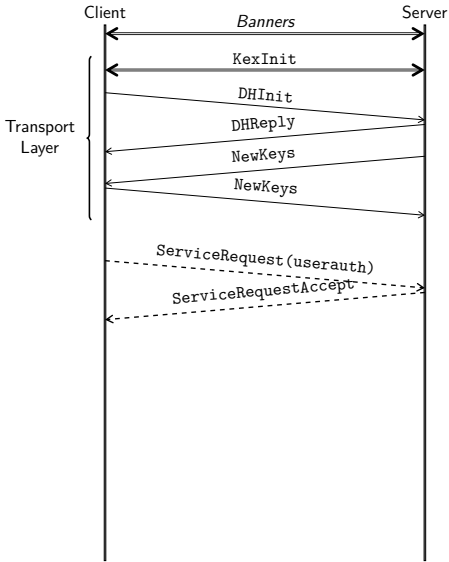
SSH in a Nutshell (2/2)



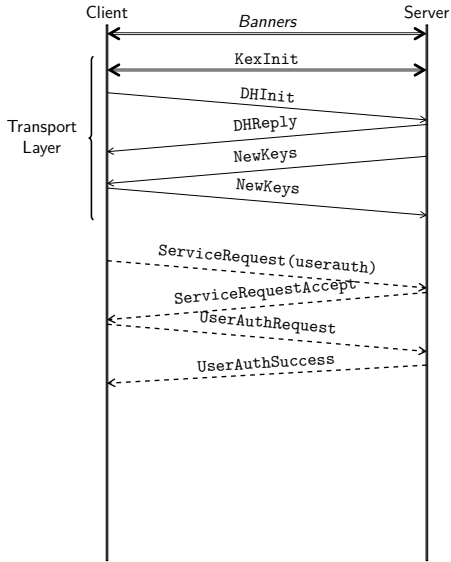
SSH in a Nutshell (2/2)



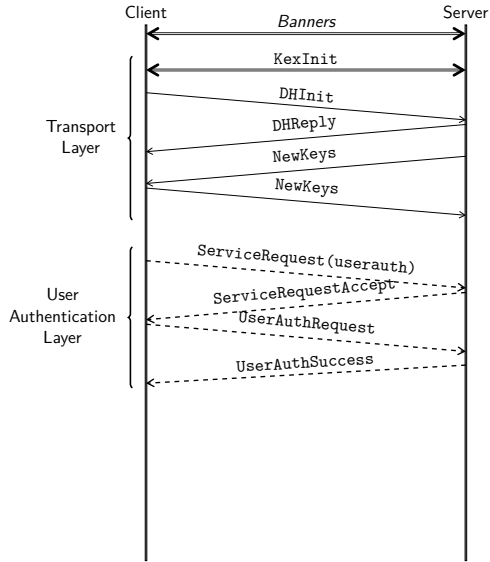
SSH in a Nutshell (2/2)



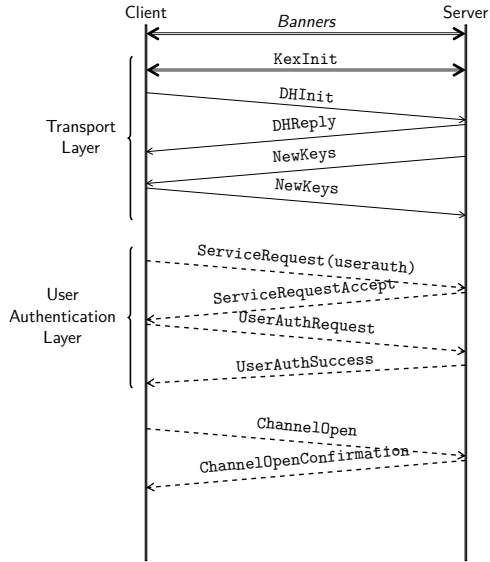
SSH in a Nutshell (2/2)



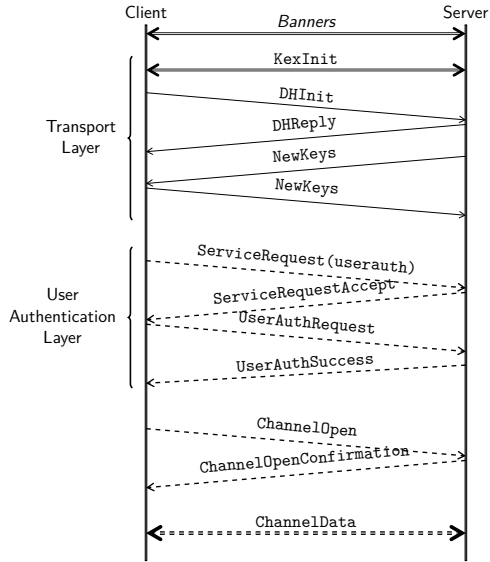
SSH in a Nutshell (2/2)



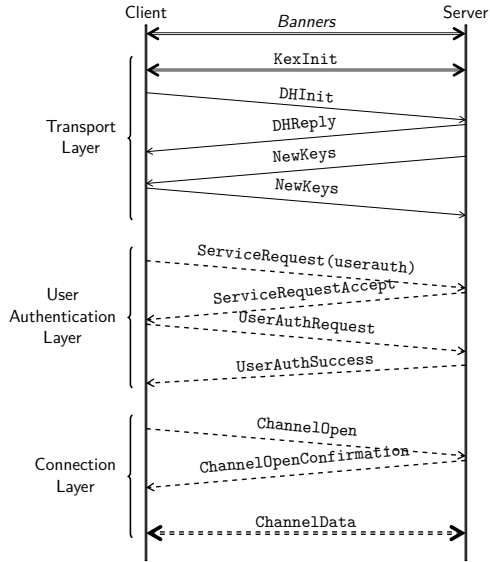
SSH in a Nutshell (2/2)



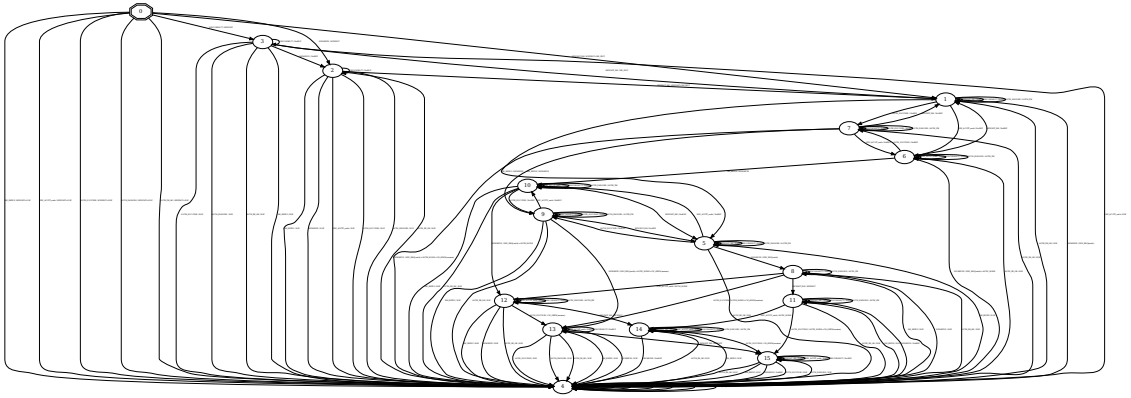
SSH in a Nutshell (2/2)



SSH in a Nutshell (2/2)

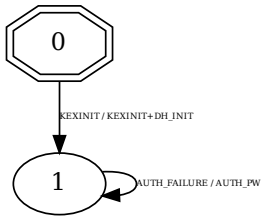


Case Study



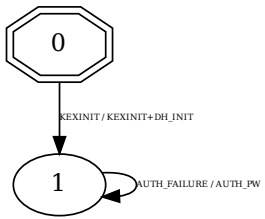
- Client state machine
- Transport + UserAuthentication layers
- 8 input messages, 16 states

Papers, please? (Early `UserAuthFailure[password]`)



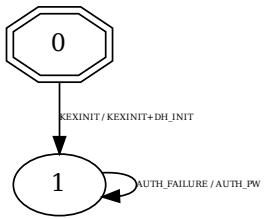
- If a rogue server starts the handshake...
- and sends an early `UserAuthFailure` with the `password` method...

Papers, please? (Early `UserAuthFailure[password]`)



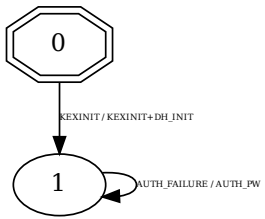
- If a rogue server starts the handshake...
- and sends an early `UserAuthFailure` with the `password` method...
- the vulnerable client happily sends its password (`AUTH_PW`)

Papers, please? (Early `UserAuthFailure[password]`)



- If a rogue server starts the handshake...
- and sends an early `UserAuthFailure` with the `password` method...
- the vulnerable client happily sends its password (`AUTH_PW`)
- Problem: there was no server authentication

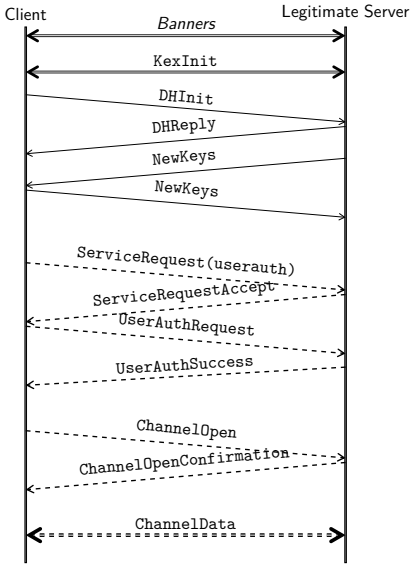
Papers, please? (Early `UserAuthFailure[password]`)



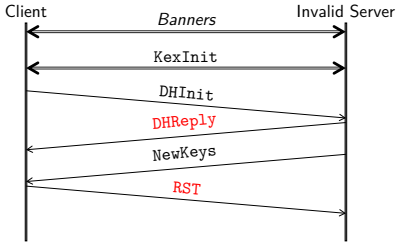
- If a rogue server starts the handshake...
- and sends an early `UserAuthFailure` with the `password` method...
- the vulnerable client happily sends its password (`AUTH_PW`)
- Problem: there was no server authentication

Demo?

Early UserAuthFailure (legitimate)

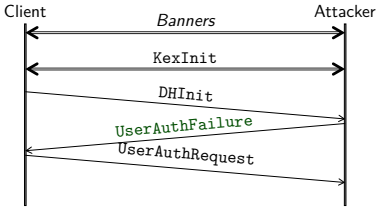


Early `UserAuthFailure` (`invalid`)



- Since the server presents an unknown public key,
- the client rejects the connecion

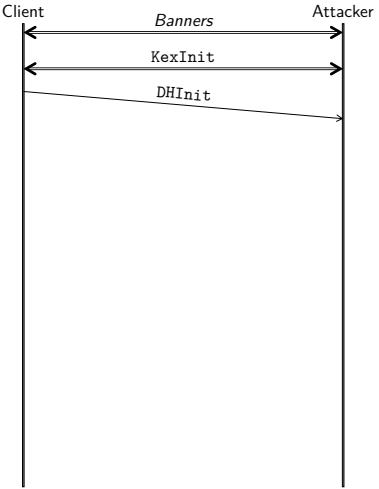
Early `UserAuthFailure` (attacker)



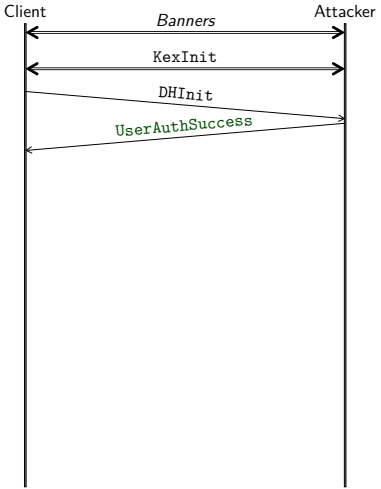
- We skip the server authentication...
- and send a bogus `UserAuthFailure` message...
- which triggers the client to send its password,
- in cleartext, and to an unauthenticated server

- We can share the PoC if you are interested (e.g. if you have to exploit a wolfSSH client...)

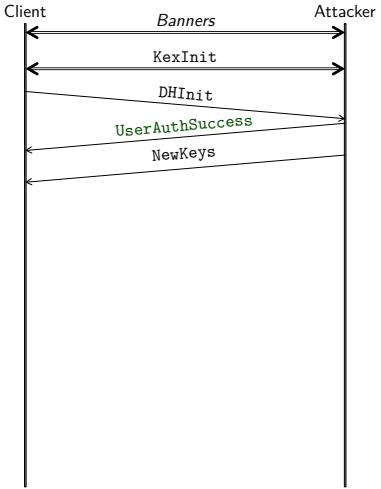
What Can I Do for You? (Early `UserAuthSuccess`)



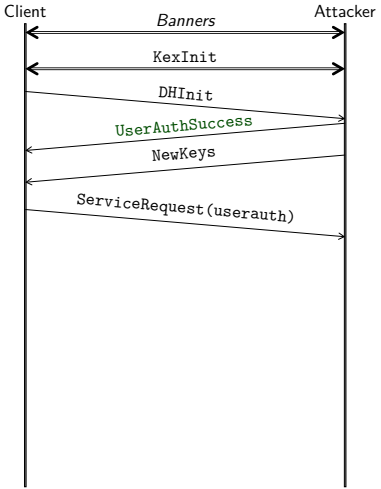
What Can I Do for You? (Early `UserAuthSuccess`)



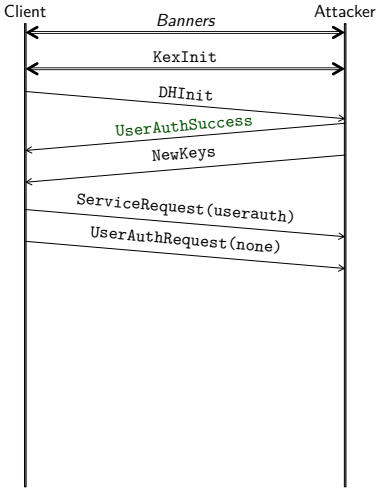
What Can I Do for You? (Early `UserAuthSuccess`)



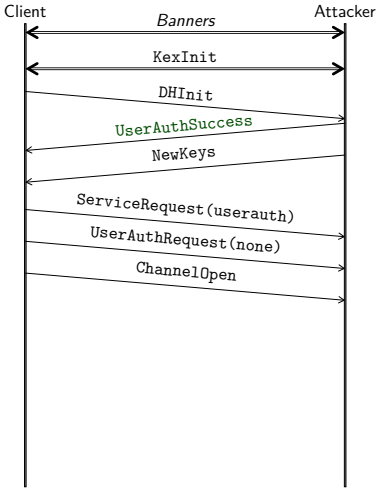
What Can I Do for You? (Early `UserAuthSuccess`)



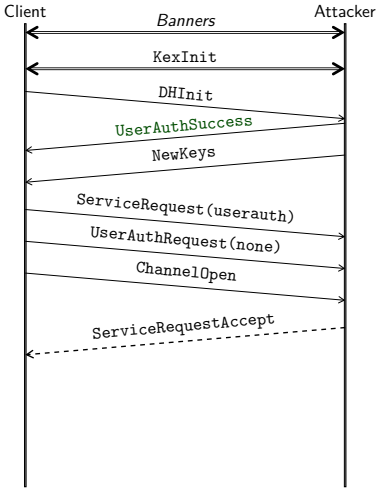
What Can I Do for You? (Early `UserAuthSuccess`)



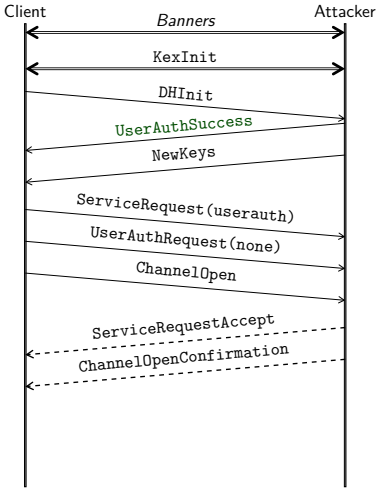
What Can I Do for You? (Early `UserAuthSuccess`)



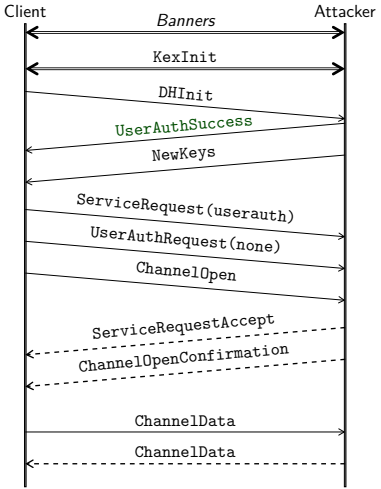
What Can I Do for You? (Early `UserAuthSuccess`)



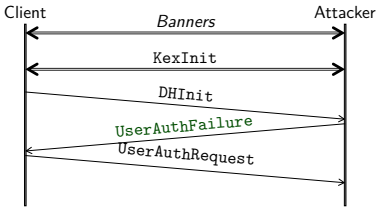
What Can I Do for You? (Early `UserAuthSuccess`)



What Can I Do for You? (Early `UserAuthSuccess`)



Can I Have an Autograph? (Early `UserAuthFailure[publickey]`)



- We skip the server authentication...
- and send a bogus `UserAuthFailure` message (with the `publickey` method)...
- which triggers the client to sign an empty context with its private key
- Such a signature could be used against other buggy servers...

Plan

GASP has ended, what's next?

Short-term

- work on our tools and platforms
- results to publish on SSH and Fingerprinting
- unpublished papers on Greybox Inference, Adaptive Learning

GASP has ended, what's next?

Short-term

- work on our tools and platforms
- results to publish on SSH and Fingerprinting
- unpublished papers on Greybox Inference, Adaptive Learning

Long-term

- more protocols (QUIC, RDP)
- more efficient inferences (adaptive learning, system-level tricks)
- more efficient analyses/visualisation

GASP has ended, what's next?

Short-term

- work on our tools and platforms
- results to publish on SSH and Fingerprinting
- unpublished papers on Greybox Inference, Adaptive Learning

Long-term

- more protocols (QUIC, RDP)
- more efficient inferences (adaptive learning, system-level tricks)
- more efficient analyses/visualisation

Longer-term

- towards automatic mappers
- extension to other domains (syscalls, APIs)

Questions

Thank you for your attention

Acknowledgements

- ANR project GASP
- CIEDS project CERES
- IMT Carnot GINS
- PhD: Aina Rasoamanana, Arthur Tran Van, Clément Parssegny
- Postdoc: Yohan Pipereau
- Interns: Eva Gagliardi, Sébastien Naud, Martin Horth, Lorenzo Nadal Santa, Quentin Rabouin, Mohamed Mziou, Mathieu Michel, Van Nam Pham, Alexander Trifa, Pedro Bartolomei Pandozzi, Jean Chavasse-Frétaz

References

[ESORICS22] *Towards a Systematic and Automatic Use of State Machine Inference to Uncover Security Flaws and Fingerprint TLS Stacks.* A. Rasoamanana, OL et H. Debar, ESORICS 2022

[ARES24] *Mealy Verifier: An Automated, Exhaustive, and Explainable Methodology for Analyzing State Machines in Protocol Implementations.* A. Tran Van, OL et H. Debar, ARES 2024

Articles et ressources disponibles sur `https://paperstreet.picty.org` et `https://gasp.ebfe.fr`