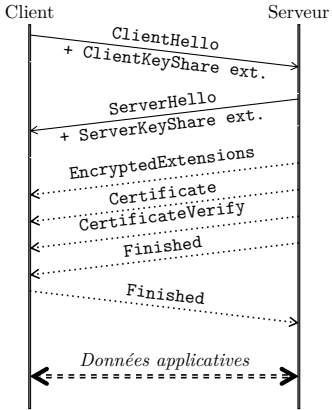
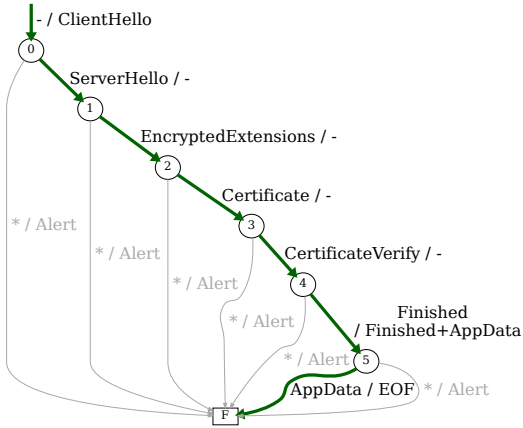
Application de l'inférence de machines à états pour la recherche de vulnérabilités dans les piles réseau

Olivier Levillain Aina Toky Rasoamanana Yohan Pipereau

SSTIC 2026

Du diagramme en serpent à un modèle formel : l'exemple de TLS 1.3





Motivation

Étude de protocoles réseau à états...

- échange de messages
- gestion d'un contexte
- les spécifications sont généralement écrites en langue naturelle

... et de leurs implémentations

- les implémentations peuvent différer
- analyse des écarts à la spécification et caractérisation d'éventuelles vulnérabilités

Motivation

Étude de protocoles réseau à états...

- échange de messages
- gestion d'un contexte
- les spécifications sont généralement écrites en langue naturelle

... et de leurs implémentations

- les implémentations peuvent différer
- analyse des écarts à la spécification et caractérisation d'éventuelles vulnérabilités

Objectif

- À partir d'une spécification
- et d'une implémentation de cette spécification,
- extraire le modèle correspondant en boîte noire, et trouver des vulnérabilités ?

Approches existantes en boîte noire

Suites de tests

- Exemple : `TLS-Anvil`
- Limitation : couverture partielle, souvent orientée vers la conformité

Fuzzing

- Exemple : `tlsfuzzer`
- Limitation : difficulté à atteindre les états internes/profonds
- Limitation : oracles de détection orientés corruption mémoire

Approches existantes en boîte noire

Suites de tests

- Exemple : `TLS-Anvil`
- Limitation : couverture partielle, souvent orientée vers la conformité

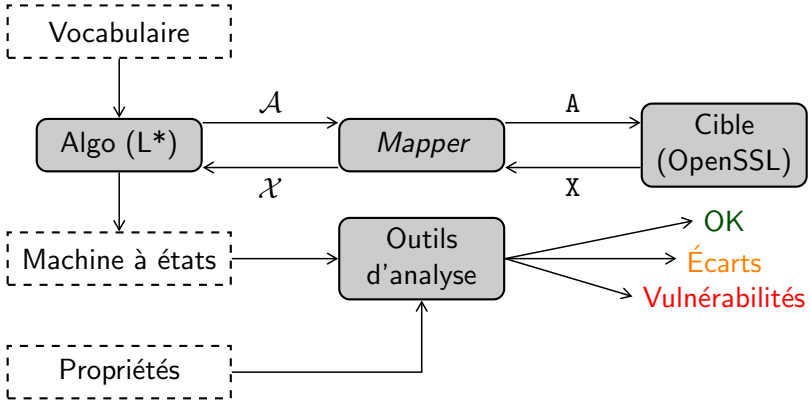
Fuzzing

- Exemple : `tlsfuzzer`
- Limitation : difficulté à atteindre les états internes/profonds
- Limitation : oracles de détection orientés corruption mémoire

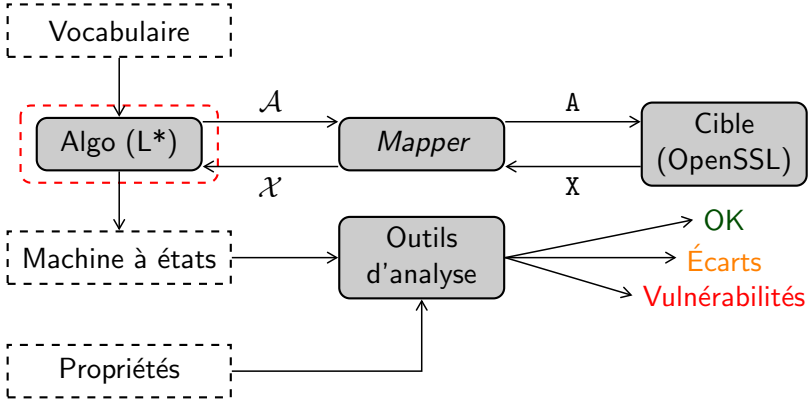
Active Automata Learning

- C'est l'objet de notre présentation !

Méthodologie / Plan

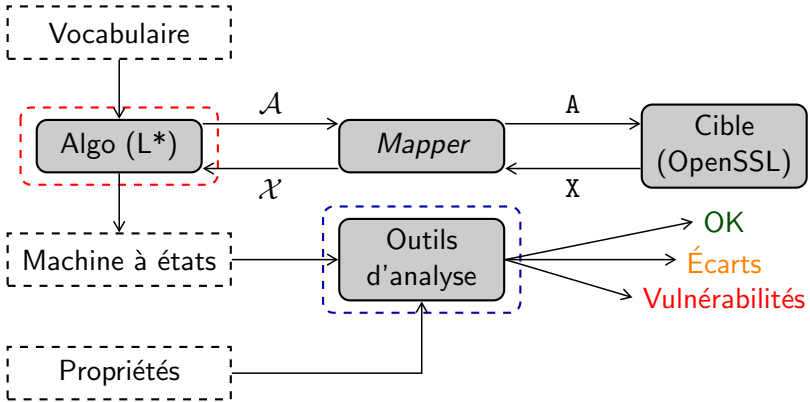


Méthodologie / Plan



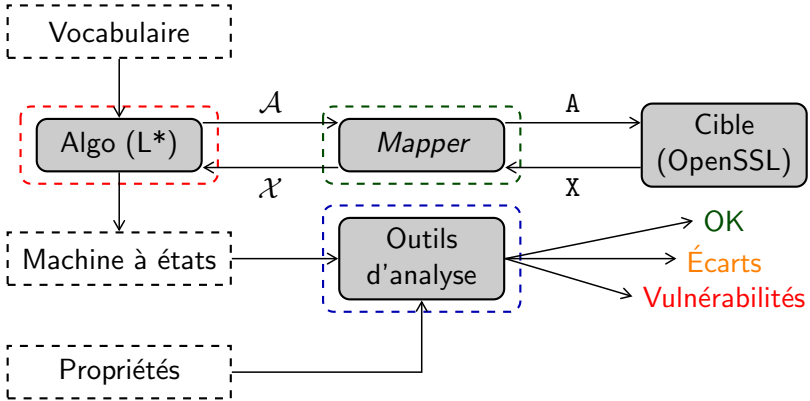
1. Présentation de l'algorithme de référence d'apprentissage d'automate

Méthodologie / Plan



1. Présentation de l'algorithme de référence d'apprentissage d'automate
2. Vérification de propriétés de sécurité des automates appris

Méthodologie / Plan



1. Présentation de l'algorithme de référence d'apprentissage d'automate
2. Vérification de propriétés de sécurité des automates appris
3. Traduction des messages concrets $\leftrightarrow$ abstraits

Démonstration

- CVE-2025-14942

1. Algorithme d'inférence : L*

- Comment apprendre l'automate d'une implémentation ?

Quelques définitions

Lettre

Un message du protocole. Ex : *ClientHello*, *ServerHello*, *EncryptedExtensions*

Quelques définitions

Lettre

Un message du protocole. Ex : *ClientHello*, *ServerHello*, *EncryptedExtensions*

Mot

Une suite de lettres. Ex : *ClientHello*, *ClientHello* · *Finished*

Quelques définitions

Lettre
Un message du protocole. Ex : *ClientHello*, *ServerHello*, *EncryptedExtensions*

Mot
Une suite de lettres. Ex : *ClientHello*, *ClientHello* · *Finished*

Vocabulaire
Un ensemble de lettres.
Ex : Vocabulaire pour target client TLS 1.3

- $E = \{SH, EE, Cert, CertVerify, Fin, AppData\}$
- $S = \{CH, Fin, AppData\}$

Qu'est-ce qu'un état ? Comment différencier deux états ?

	Entrée	Lettre de sortie
1	SH	-
2	EE	EOF
3	$SH \cdot SH$	EOF
4	$SH \cdot EE$	-
5	$EE \cdot SH$	EOF
6	$EE \cdot EE$	EOF
7	$SH \cdot EE \cdot EE$	EOF
.	...	...

Distinguer des états ?

- $w_1 = SH$
- $w_2 = SH \cdot EE$
- Pourquoi $w_1 \not\equiv w_2$?

0. TLS1.3 : SH=ServerHello, EE=EncryptedExtension, EOF=EndOfFile, -=Empty

Qu'est-ce qu'un état ? Comment différencier deux états ?

	Entrée	Lettre de Sortie
1	SH	-
2	EE	EOF
3	$SH \cdot SH$	EOF
4	$SH \cdot EE$	-
5	$EE \cdot SH$	EOF
6	$EE \cdot EE$	EOF
7	$SH \cdot EE \cdot EE$	EOF
.	...	...

Distinguer des états ?

- $w_1 = SH$
- $w_2 = SH \cdot EE$
- Pourquoi $w_1 \not\equiv w_2$?

$s = EE$

- $Sortie(w_1 \cdot s) = -$
- $Sortie(w_2 \cdot s) = EOF$

0. TLS1.3 : SH=ServerHello, EE=EncryptedExtension, EOF=EndOfFile, -=Empty

Distinguer des états

Décomposition en préfixe et suffixe

$w = p \cdot s$

- Le **préfixe** p représente le **chemin d'accès** à l'état
- Le **suffixe** s représente la **manière de le distinguer**

Conséquence

Cherchons des suffixes !

De l'organisation des observations vers la génération de l'automate

	s_1	s_2	s_3	s_4
p_1	1	2	1	3
p_2	1	1	1	3
p_3	1	1	1	3

Table d'observation

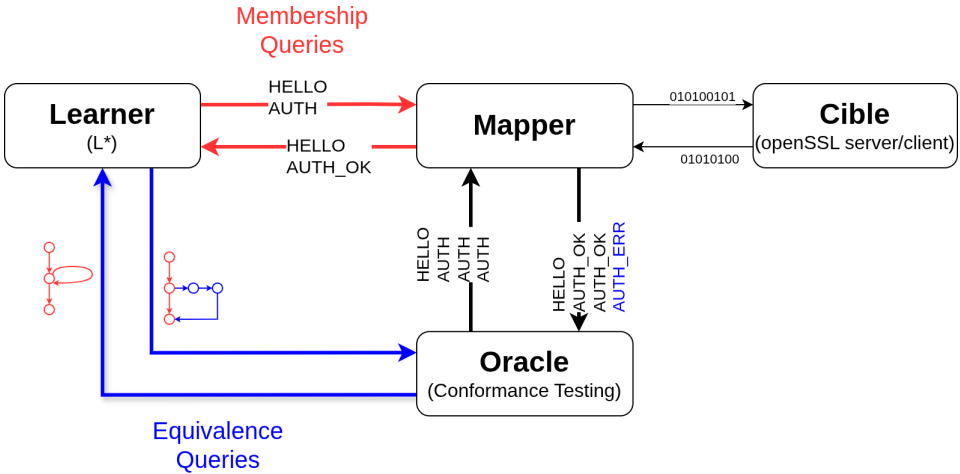
- $(1\,2\,1\,3) \neq (1\,1\,1\,3) \rightarrow p_1 \not\equiv p_2$
- $(1\,1\,1\,3) = (1\,1\,1\,3) \rightarrow p_2 \equiv p_3$

Pour les détails : L* [1]

- Quand ajouter de nouvelles colonnes/suffixes ?
- Quand ajouter de nouvelles lignes/préfixes ?
- Quel est la condition d'arrêt ?

1. Learning Regular Sets from Queries and Counterexamples, D. Angluin

Le MAT *framework* : Commun aux algorithmes d'apprentissage



MAT *framework*

Choisir l'algorithme d'inférence et son implémentation

Différents Algorithmes : L*, L#, TTT, Rivest-Schapire...

- Structure de donnée pour stocker les observations : arbre vs tableau
- Stratégie pour traiter les contre-exemples
- De meilleurs complexités temporelles pour L#, TTT, Rivest-Schapire

De multiples implémentations...

Algorithme	Langage	URL
L*	Rust	`https://gitlab.com/gaspians/rlstar`
$L\#$	Rust	`https://gitlab.science.ru.nl/sws/lsharp`
L*	Python	`https://github.com/gbossert/pylstar`
L*,L#,$L\lambda$,TTT	Java	`https://github.com/learnlib/learnlib`

Le test de conformité comme méthode d'équivalence

Objectif

- Utiliser l'automate hypothèse pour chercher des contre-exemples
- Contre-exemples = Suffixes

Méthodes Multiples

W, W-P, SPY, HSI, ADS, BDist... [2]

Complexité en symboles

- $I = \{i_1, \ldots, i_n\} : |I|^k$
- Croissance exponentielle avec taille du suffixe (k)

2. An Experimental Evaluation of Conformance Testing Techniques in Active Automata Learning, Garhewal et al.

Compromis entre exhaustivité et durée

Limiter la recherche

Un paramètre définissant une profondeur limite pour la recherche d'états manquants
Ex : $BDist(suffix\_len = 3)$, $WP(number\_states = 48)$

Compromis entre exhaustivité et durée

- Sous-estimé $\rightarrow$ États manquants
- Sur-estimé $\rightarrow$ Durée croît exponentiellement

Type de paramètre des méthodes CT

Paramètre 1 : Nombre final d'états

- Quel est le nombre total d'états dans l'automate de libssh ? openssh ? ¯_(ツ)_/¯

Paramètre 2 : BDist - Longueur maximale du suffixe [3]

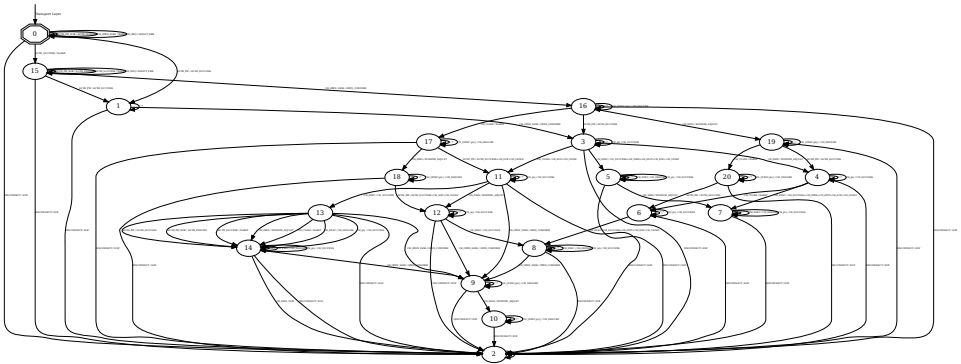
- Quelle est la longueur maximale d'un suffixe pour distinguer 2 états de libssh ? openssh ?
- En général pour les protocoles réseaux : $BDist \leq 4$.

3. DroidStar : Callback Typestates for Android Classes, Radhakrishna et al.

2. Verification

- Comment décrire les comportements corrects/incorrects ?
- Comment trouver les déviations ?

Pourquoi automatiser la vérification ?



Principes du *model checking*

Paramètres d'entrées

- **Propriété** : Une formule de logique temporelle ϕ
- **Un modèle de type automate** : $\mathcal{M}$

Deux résultats possibles

- **validation** : Tous les chemins d'exécution π of $\mathcal{M}$ vérifient la propriété ϕ : $\pi \models \phi$
- **contre-example** : Il existe un chemin π qui enfreint la propriété : $\pi \not\models \phi$

Quelques *model checkers*

LTL Model checkers

nusmv, uppaal, its-tools...

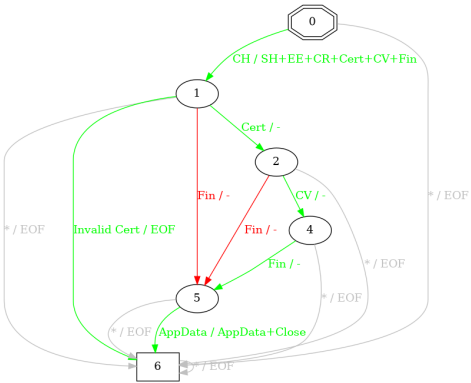
Mealy Verifier

Mealy Verifier [4] [5]

	LTL model checkers	Mealy Verifier
Expressivité du langage	Expressif	Limité
Recherche de contre-example	1	Tous
Complexité du langage	Complexe	Simple

4. https ://github.com/artfire52/Mealy-Verifier
5. Mealy Verifier : an automated, exhaustive, and explainable methodology for analyzing state machines in protocol implementations, Tran Van et al.

CVE-2022-25640 : LTL vs Mealy Verifier



Countournement d'authentification client dans wolfSSL server 5.2.0

Mealy verifier

```
CT:auth
   Cert/– | I/I
   CV/– | I/I
   Fin/–
:CT
```

LTL

```
G((input="Fin" & X(output="Empty")) ->
    Y ( (input="CV" & X(output="Empty")) &
    Y (input="Cert" & X(output="Empty"))))
```

3. Étude de cas : mapper SSH [6]

- Comment encoder un message abstrait en message concret ?
- Comment décoder un message concret en message abstrait ?

6. `https://gitlab.com/gaspians/ssh-mapper`

Challenge 1 : Quels messages sont significatifs ?

Les messages obligatoires

- Messages du chemin prévu (Happy Path)
- Réponses de la cible pendant les interactions

Des messages moins importants

- SSH_MSG_IGNORE ?
- SSH_MSG_DEBUG ?

Challenge 2 : Comment concrétiser un message abstrait ?

Des paramètres à négocier

Exemple : KexInit

- Listes d'algorithmes supportés : échange de clé, authentification, chiffrement symétrique, etc.
- Ajout de listes fournies

Différentes variantes pour un même message

Exemple : UserAuthRequest

- Plusieurs méthodes d'authentification possibles
- Besoin d'un message par variante (au moins `password`, `publickey`)

Challenge 3 : Quel sens donner à des séquences de messages sortant du cadre de la spécification ?

Exemple pour SSH

UserAuthRequest(pubkey) avant l'échange des clés

- signature ?

Idée générale

- Structure des messages
- Initialisation
- Information importante
- Logique protocolaire

Architecture du protocole SSH (RFC 4251) et choix des messages

L1 : Couche de transport (RFC 4253)

- C → S : Kexinit, DHinit, NewKeys, Disconnect, ExtensionInfo, ServiceRequest
- S → C : Kexinit, DHReply, NewKeys, Disconnect, ExtensionInfo, ServiceAccept

L2 : Couche d'authentification (RFC 4252)

- C → S : UserAuthRequest_Password, UserAuthRequest_RSA
- S → C : UserAuthSuccess, UserAuthFailure

L3 : Couche de connexion (RFC 4254)

- C → S : ChannelOpen, ChannelExec, ChannelData, ChannelClose, ChannelEof
- S → C : ChannelOpenConfirmation, ChannelSuccess, ChannelFailure

Architecture du protocole SSH (RFC 4251) et choix des messages

L1 : Couche de transport (RFC 4253)

- C → S : Kexinit, DHinit, NewKeys, Disconnect, ExtensionInfo, ServiceRequest
- S → C : Kexinit, DHReply, NewKeys, Disconnect, ExtensionInfo, ServiceAccept

L2 : Couche d'authentification (RFC 4252)

- C → S : UserAuthRequest_Password, UserAuthRequest_RSA, **UserAuthSuccess** ?
- S → C : UserAuthSuccess, UserAuthFailure

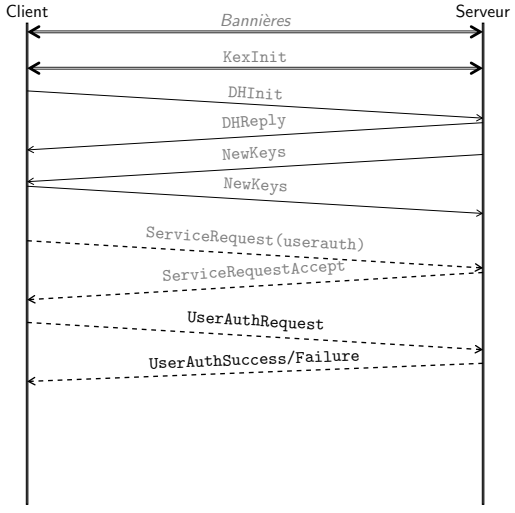
L3 : Couche de connexion (RFC 4254)

- C → S : ChannelOpen, ChannelExec, ChannelData, ChannelClose, ChannelEof
- S → C : ChannelOpenConfirmation, ChannelSuccess, ChannelFailure

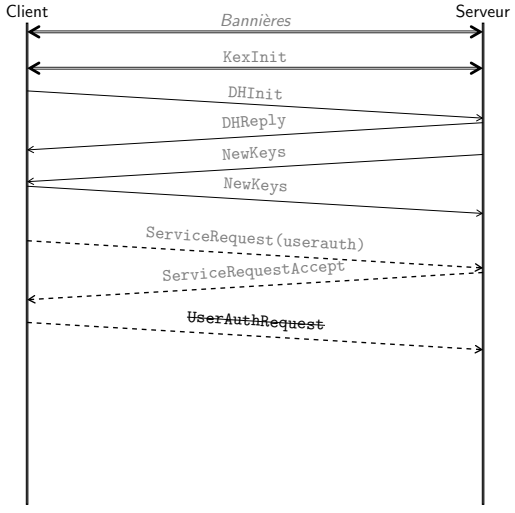
4. Analyse de vulnérabilités logiques sur des implémentations SSH

- CVE-2018-10933
- Retour sur CVE-2025-14942

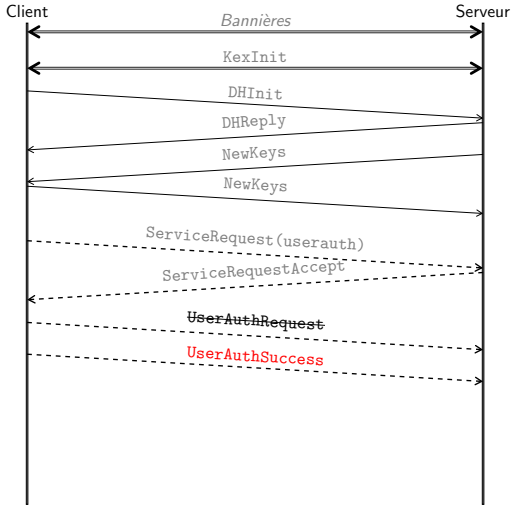
CVE-2018-10933 (vulnérabilité sur le serveur `libssh`)



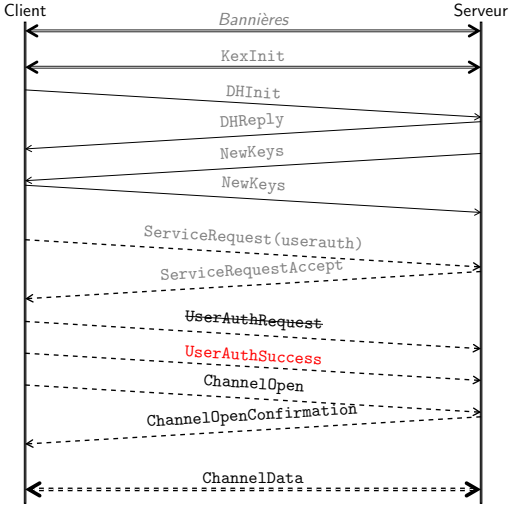
CVE-2018-10933 (vulnérabilité sur le serveur `libssh`)



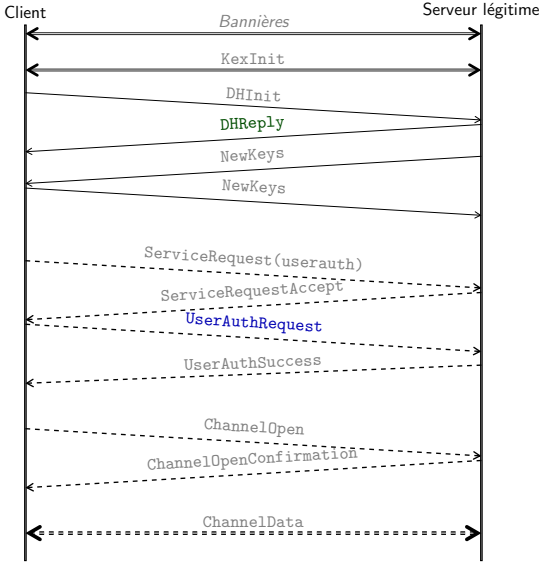
CVE-2018-10933 (vulnérabilité sur le serveur `libssh`)



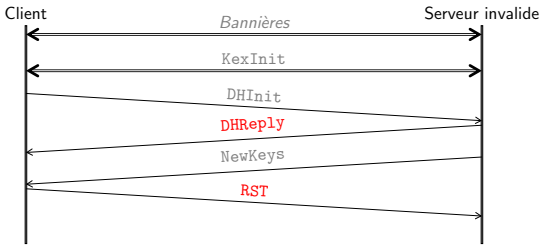
CVE-2018-10933 (vulnérabilité sur le serveur `libssh`)



CVE-2025-14942 - Early `UserAuthFailure` (serveur légitime)

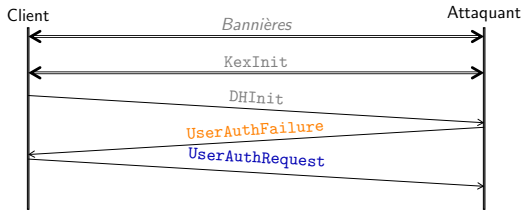


CVE-2025-14942 - Early `UserAuthFailure` (serveur invalide)



- Comme le serveur présente une clé inconnue du client
- ce dernier rejette la connexion comme attendu

CVE-2025-14942 - Early `UserAuthFailure` (attaquant)



- On saute l'authentification du serveur
- et on envoie un message `UserAuthFailure` inattendu...
- ce qui déclenche l'envoi du mot de passe par le client
- en clair, à un serveur non authentifié

Conclusion

Conclusion

Apprentissage d'automate (AAL)

- une méthode déterministe
- une analyse en boîte noire
- contrôle fin sur les approximations

Bilan

- Reproduction de vulns : CVE-2018-10933, CVE-2020-24613, CVE-2024-2873, etc.
- Nouvelles vulns : CVE-2021-3336, CVE-2022-25638, CVE-2022-25639, CVE-2022-25640, CVE-2025-14942

Remerciements

Remerciements

- *Arthur Tran Van* : Mealy Verifier, Mapper pour le protocole OPC-UA
- *Clément Parssegny* : Test de conformance BDist et outillage divers
- *Pedro Bartolomei Pandozi, Martin Horth, Mathieu Michel, Mohamed Mziou, Lorenzo Nadal Santa, Sébastien Naud, Van Nam Pham, Quentin Rabouin and Alexander Trifa*

Merci pour votre attention

Apprentissage d'automate (AAL)

- une méthode déterministe
- une analyse en boîte noire
- contrôle fin sur les approximations

Bilan

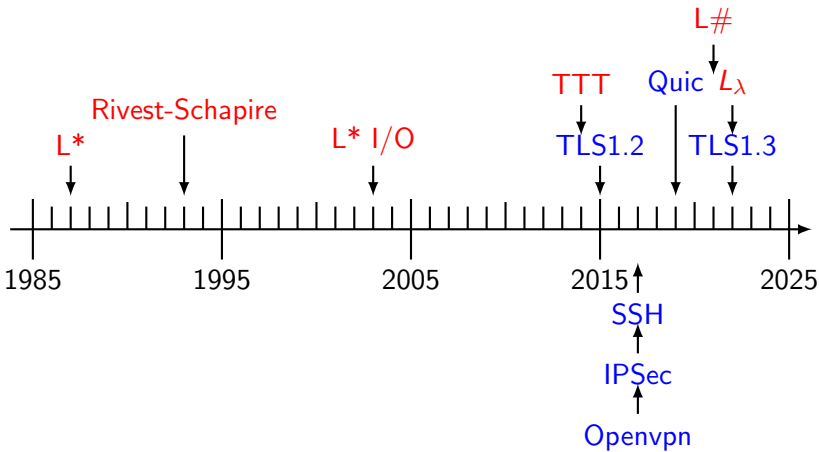
- Reproduction de vulns : CVE-2018-10933, CVE-2020-24613, CVE-2024-2873, etc.
- Nouvelles vulns : CVE-2021-3336, CVE-2022-25638, CVE-2022-25639, CVE-2022-25640, CVE-2025-14942

Questions ?

Appendix

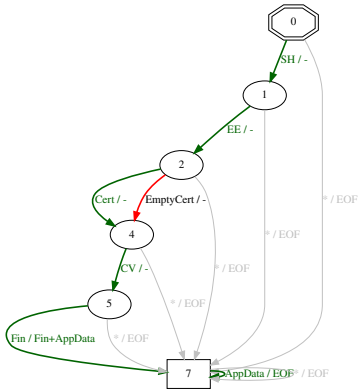
Introduction

Quelques repères autour de l'*Active Automata Learning*



L*

What is a state ? How to show they are equivalent ?

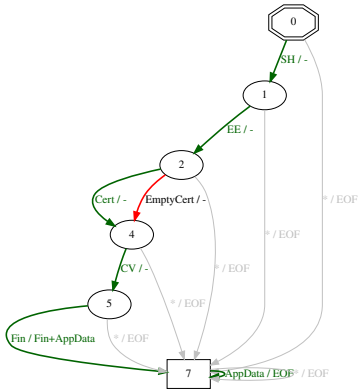


wolfSSL client 4.6.0 TLS1.3

Q - Equivalent states ?

- $w_3 = SH \cdot EE \cdot Cert$
- $w_4 = SH \cdot EE \cdot EmptyCert$
- Why $w_3 \equiv w_4$?

What is a state? How to show they are equivalent?



wolfSSL client 4.6.0 TLS1.3

Q - Equivalent states?

- $w_3 = SH \cdot EE \cdot Cert$
- $w_4 = SH \cdot EE \cdot EmptyCert$
- Why $w_3 \equiv w_4$?

Infinite lookup

- There exists no suffix which yield different output
- Suffix length ∞

Complexité

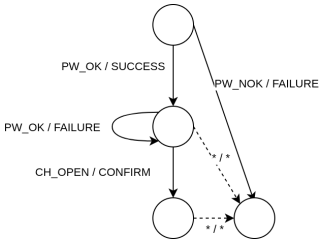
- k : taille du vocabulaire d'entrée
- n : nombre d'états
- m : contre-exemple le plus long

TTT

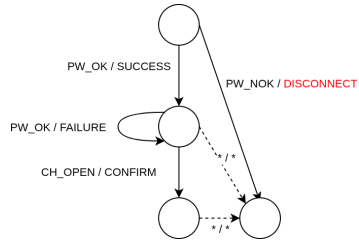
$O(kn^2 + nlog(m))$

CTT

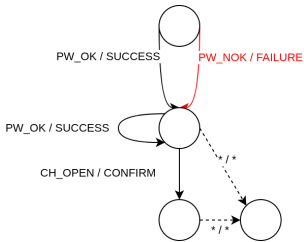
Types de faultes



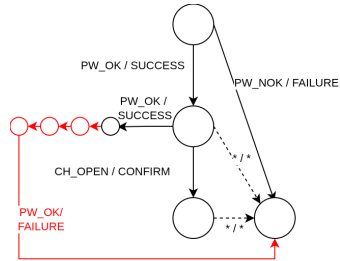
Base



Fautes de sortie (Output fault)



Fautes du prochain-état (Next-state fault)



Fautes d'états supplémentaires (extra-state)

Verification

Properties we want to express

Temporal Logic

- **linear-time** (LTL) : At all instant, a single future may exist
- **branching-time** (CTL) : Time may split into different futures

LTL BNF

BNF of LTL

$$\begin{aligned}\langle \phi \rangle \quad \models \quad & \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \\ & \mid (X\phi) \mid (F\phi) \mid (G\phi) \mid (\phi U \phi) \mid (\phi W \phi) \mid (\phi R \phi)\end{aligned}$$

BNF of PastLTL

$$\begin{aligned}\langle \phi \rangle \quad \models \quad & \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \\ & \mid (O\phi) \mid (H\phi) \mid (\phi S \phi) \mid (Y\phi)\end{aligned}$$

Properties : nuSMV temporal operators

LTL

- $\mathbf{X}p$: p is true in **next** state
- $\mathbf{F}p$: p is true in **some future** state
- $\mathbf{G}p$: p holds in **all** future time instant
- $p\mathbf{U}q$: p holds **until** a state where q holds

PastLTL (Succint + natural)

- $\mathbf{Y}p$: p holds in **the** previous state
- $p\mathbf{S}q$: p holds **since** q holds
- $\mathbf{O}p$: p holds in **one** of the past state
- $\mathbf{H}p$: p holds in **all** previous state

LTL : From cause to effect

Any OPEN must eventually imply a CLOSE

$\mathbf{G}(inp = OPEN \rightarrow \mathbf{F}inp = CLOSE))$

PastLTL : from effect to cause

Any command execution must happen after a successful authentication

$\mathbf{G}(inp = CMD \rightarrow \mathbf{O}out = AUTH\_OK))$

Mealy Verifier

Liste des opérateurs supportés

- **ST**
- **SD** : Décrit un état puit
- **RE**
- **CT** : Décrit un événement conditionnel
- **ETS**

event

Event grammar

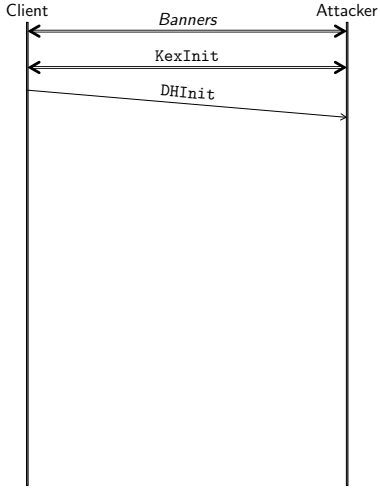
$$\begin{aligned} \langle \text{pattern} \rangle &\models \mathit{pattern}* \mid \mathit{pattern}? \\ \langle \phi \rangle &\models !\phi \mid \phi + \phi \mid \mathit{pattern} \end{aligned}$$

SD : Sink State

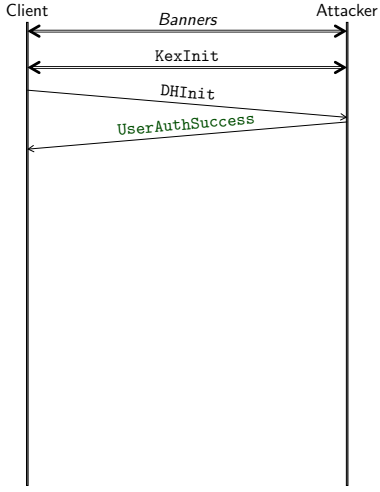
CT :auth Cert/- — I/I CV/- — I/I Fin/- :CT

Vulnérabilités

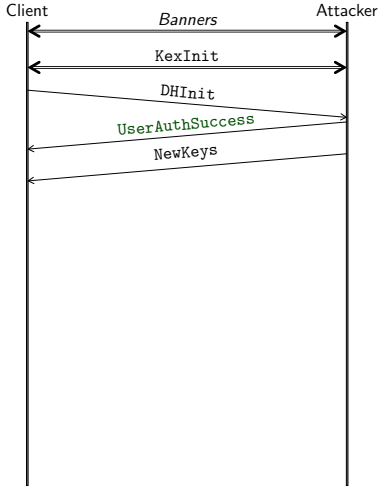
What Can I Do for You? (Early `UserAuthSuccess`)



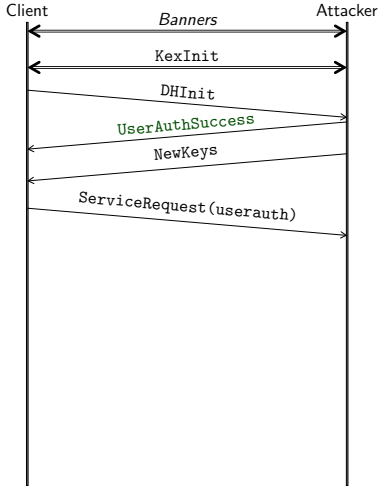
What Can I Do for You? (Early `UserAuthSuccess`)



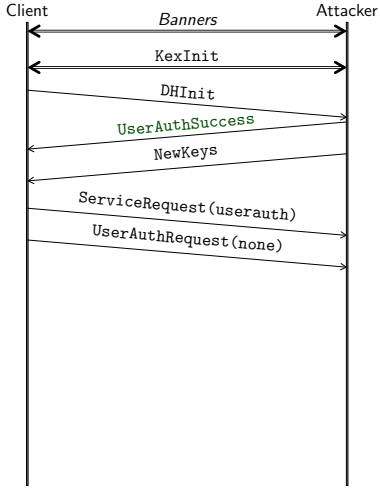
What Can I Do for You? (Early `UserAuthSuccess`)



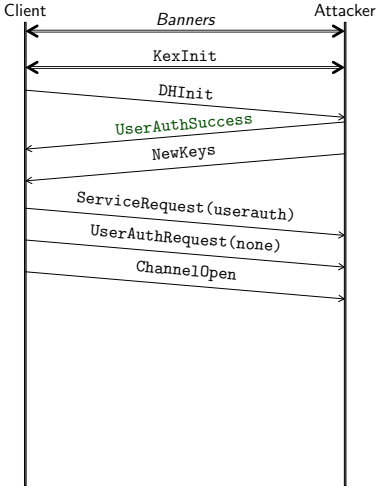
What Can I Do for You? (Early `UserAuthSuccess`)



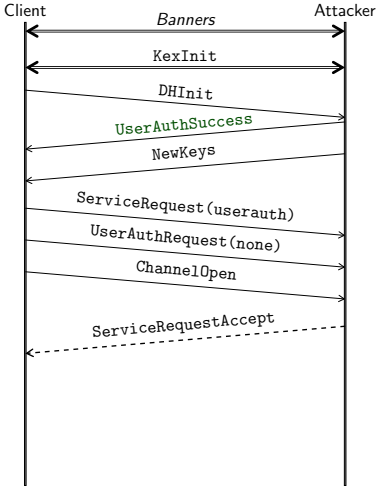
What Can I Do for You? (Early `UserAuthSuccess`)



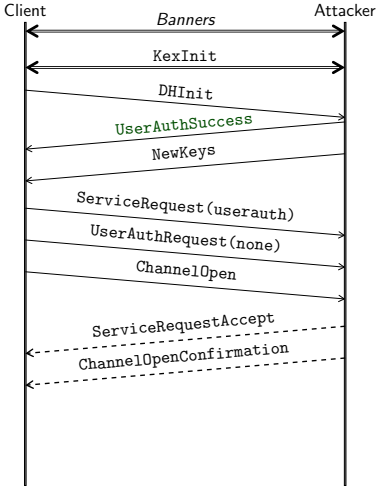
What Can I Do for You? (Early `UserAuthSuccess`)



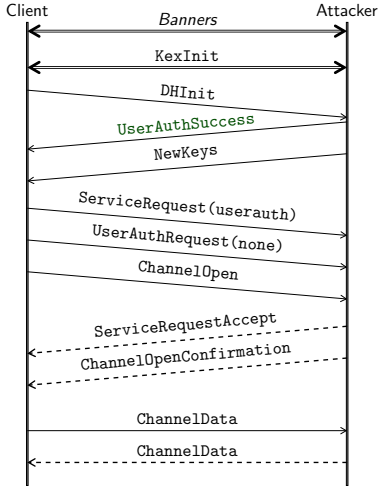
What Can I Do for You? (Early `UserAuthSuccess`)



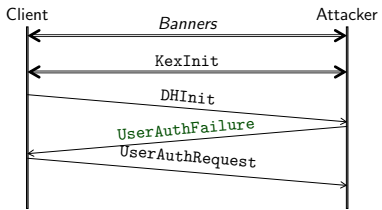
What Can I Do for You? (Early `UserAuthSuccess`)



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...