Les implémentations TLS dans tous leurs états

Olivier Levillain

ESIEA Secure Edition
17 juin 2023

@pictyeye

Parcours

- Stage en cryptographie sur une fonction de hachage (2006)
- Membre du labo « système » de l'ANSSI (2007-2012)
- Responsable du labo « réseau » de l'ANSSI (2012-2015)
- Responsable du centre de formation de l'ANSSI (2015-2018)
- Maître de conférence à Télécom SudParis (2018-)

@pictyeye

Parcours

- Stage en cryptographie sur une fonction de hachage (2006)
- Membre du labo « système » de l'ANSSI (2007-2012)
- Responsable du labo « réseau » de l'ANSSI (2012-2015)
- Responsable du centre de formation de l'ANSSI (2015-2018)
- Maître de conférence à Télécom SudParis (2018-)

Recherche

- *Contributions à l'étude des mécanismes bas-niveau x86*
- Thèse sur SSL/TLS
- *Étude sur les langages de programmation*
- Travail sur les *parsers* et les implémentations de piles réseau
- Analyse de vulnérabilités logicielles

TLS 101

SSL/TLS : un pilier de la sécurité d'Internet

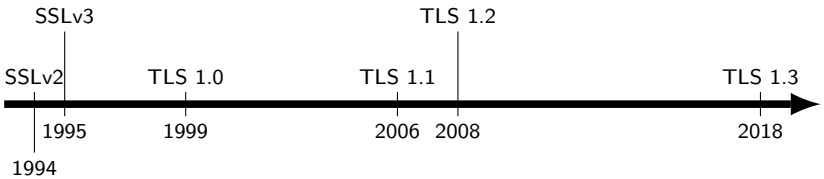
- Le schéma `https://` inventé par Netscape en 1995
  - début du commerce en ligne
- Omniprésence de SSL/TLS aujourd'hui
  - HTTPS, bien au-delà du commerce en ligne
  - Une méthode générique pour sécuriser d'autres protocoles (SMTP, IMAP, LDAP...)
  - VPN SSL
  - EAP TLS

SSL/TLS : un pilier de la sécurité d'Internet

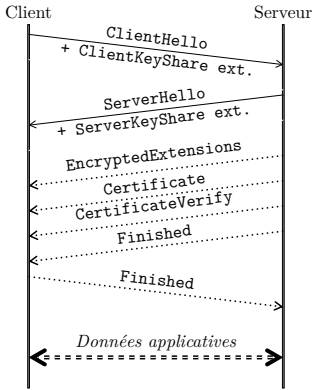
- Le schéma `https://` inventé par Netscape en 1995
  - début du commerce en ligne
- Omniprésence de SSL/TLS aujourd'hui
  - HTTPS, bien au-delà du commerce en ligne
  - Une méthode générique pour sécuriser d'autres protocoles (SMTP, IMAP, LDAP...)
  - VPN SSL
  - EAP TLS
- SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*) ?

SSL/TLS : un pilier de la sécurité d'Internet

- Le schéma `https://` inventé par Netscape en 1995
  - début du commerce en ligne
- Omniprésence de SSL/TLS aujourd'hui
  - HTTPS, bien au-delà du commerce en ligne
  - Une méthode générique pour sécuriser d'autres protocoles (SMTP, IMAP, LDAP...)
  - VPN SSL
  - EAP TLS
- SSL (*Secure Sockets Layer*) ou TLS (*Transport Layer Security*) ?



TLS 1.3 en une planche



Objectifs de TLS

- Authentification du serveur
- Établissement d'un secret partagé
- Protection des données applicatives en confidentialité et en intégrité

Plus d'information sur TLS dans [PhD16] et [CRiSIS20]

Sélection de vulnérabilités affectant TLS

Sélection de vulnérabilités affectant TLS

- Erreurs de spécification
  - confusion entre différentes sessions (*Triple Handshake*, 2014)
  - attaques cross-protocoles

Sélection de vulnérabilités affectant TLS

- Erreurs de spécification
  - confusion entre différentes sessions (*Triple Handshake*, 2014)
  - attaques cross-protocoles
- Problèmes liés à la cryptographie
  - mauvais aléa (Paquet `openssl` Debian, 2009)
  - primitives faibles (Bleichenbacher, 1998 / RC4, 2013)
  - paramètres trop petits (clés RSA de 512 bits)

Sélection de vulnérabilités affectant TLS

- Erreurs de spécification
  - confusion entre différentes sessions (*Triple Handshake*, 2014)
  - attaques cross-protocoles
- Problèmes liés à la cryptographie
  - mauvais aléa (Paquet `openssl` Debian, 2009)
  - primitives faibles (Bleichenbacher, 1998 / RC4, 2013)
  - paramètres trop petits (clés RSA de 512 bits)
- Failles dans les *parsers*
  - *buffer overread* (*Heartbleed, 2014*)
  - *buffer overflow* (*Winshock, 2014*)
  - bugs dans l'interprétation de certificats (confusion liés aux caractères nuls, 2009)

Sélection de vulnérabilités affectant TLS

- Erreurs de spécification
  - confusion entre différentes sessions (*Triple Handshake*, 2014)
  - attaques cross-protocoles
- Problèmes liés à la cryptographie
  - mauvais aléa (Paquet `openssl` Debian, 2009)
  - primitives faibles (Bleichenbacher, 1998 / RC4, 2013)
  - paramètres trop petits (clés RSA de 512 bits)
- Failles dans les *parsers*
  - *buffer overread* (*Heartbleed, 2014*)
  - *buffer overflow* (*Winshock, 2014*)
  - bugs dans l'interprétation de certificats (confusion liés aux caractères nuls, 2009)
- Erreurs de logiques dans les piles protocolaires
  - court-circuit dans la vérification d'une signature (`goto fail`, 2014)
  - chemin inattendu dans la machine à états (*EarlyCCS*, 2014 / FREAK, 2015)

Sélection de vulnérabilités affectant TLS

- Erreurs de logiques dans les piles protocolaires
  - chemin inattendu dans la machine à états (*EarlyCCS*, 2014 / FREAK, 2015)

Sélection de vulnérabilités affectant TLS

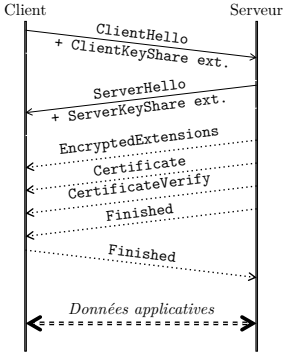
- Failles dans les *parsers*
  - *buffer overflow* (*Winshock, 2014*)
- Erreurs de logiques dans les piles protocolaires
  - chemin inattendu dans la machine à états (*EarlyCCS*, 2014 / FREAK, 2015)

Sélection de vulnérabilités affectant TLS

- Erreurs de spécification
  - attaques cross-protocoles
- Problèmes liés à la cryptographie
  - primitives faibles (Bleichenbacher, 1998)
- Failles dans les *parsers*
  - *buffer overflow* (*Winshock, 2014*)
- Erreurs de logiques dans les piles protocolaires
  - court-circuit dans la vérification d'une signature (`goto fail`, 2014)
  - chemin inattendu dans la machine à états (*EarlyCCS*, 2014 / FREAK, 2015)

Analyse des machines à états TLS

Exemple d'une machine à état déficiente

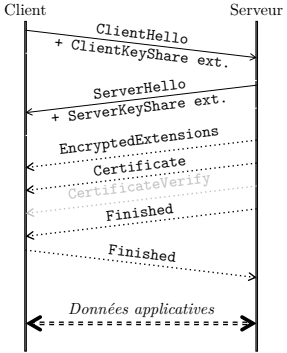


Dans TLS 1.3, en fonctionnement normal

- le serveur se présente (`Certificate`)
- il prouve son identité (`CertificateVerify`)
- ce message contient une signature qui requiert la clé privée du serveur

Travaux avec A. Rasoamanana dans le cadre du projet GASP [RESSI20, ESORICS22]

Exemple d'une machine à état déficiente



Dans TLS 1.3, en fonctionnement normal

- le serveur se présente (`Certificate`)
- il prouve son identité (`CertificateVerify`)
- ce message contient une signature qui requiert la clé privée du serveur

Que se passe-t-il si un client accepte une connexion où le `CertificateVerify` a été omis ?

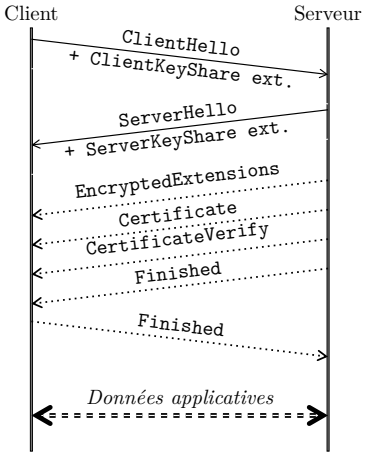
- il n'est plus nécessaire d'avoir la clé privée pour valider la négociation
- un attaquant peut usurper *n'importe quel serveur* vis-à-vis de ce client

Travaux avec A. Rasoamanana dans le cadre du projet GASP [RESSI20, ESORICS22]

Représentation de la machine à état du client

Représentation traditionnelle

- diagramme en « serpent »
- présentation du *happy path*



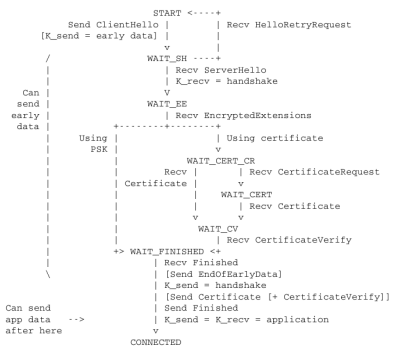
Représentation de la machine à état du client

Représentation traditionnelle

- diagramme en « serpent »
- présentation du *happy path*

Machine à états informelle

- effort de formalisation
- point de vue du client ici
- description encore ambiguë



— RFC 8446 (TLS 1.3) Annexe A

Représentation de la machine à état du client

Représentation traditionnelle

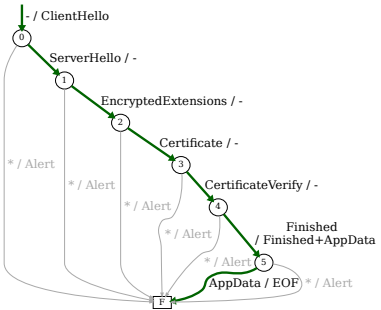
- diagramme en « serpent »
- présentation du *happy path*

Machine à états informelle

- effort de formalisation
- point de vue du client ici
- description encore ambiguë

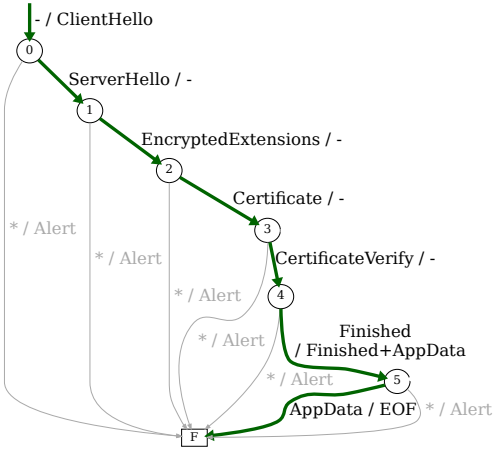
Machine de Mealy

- description formelle possible
- représentation plus lourde

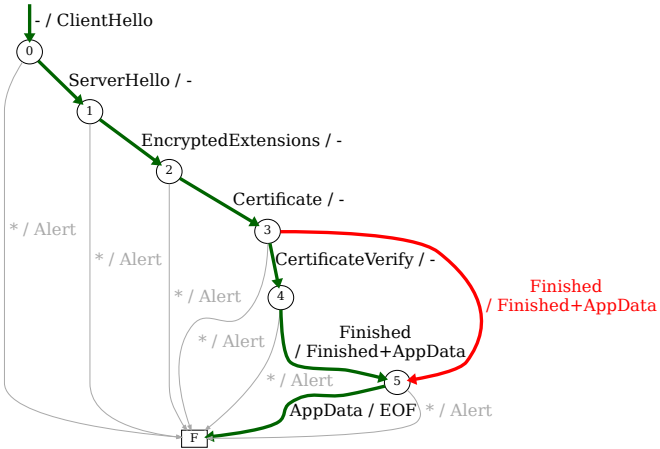


— Résultats du projet GASP

Mise en évidence de la CVE 2020-24613 sur wolfSSL



Mise en évidence de la CVE 2020-24613 sur wolfSSL



Inférence de machine à états

Possibilité d'inférer la machine à états d'une implémentation en boîte noire

- algorithme $L^\star$ décrit par Angluin en 1987
- adaptation aux machines de Mealy utilisée dans différents contextes
- inférence de machines à états de divers protocoles (ex. : TLS, H2)
- *(il existe d'autres approches, par exemple en altérant un transcript de référence par des mutations)*

Inférence de machine à états

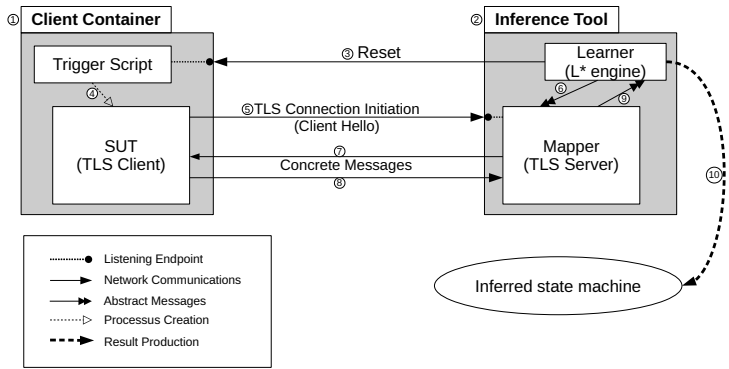
Possibilité d'inférer la machine à états d'une implémentation en boîte noire

- algorithme $L^\star$ décrit par Angluin en 1987
- adaptation aux machines de Mealy utilisée dans différents contextes
- inférence de machines à états de divers protocoles (ex. : TLS, H2)
- *(il existe d'autres approches, par exemple en altérant un transcript de référence par des mutations)*

Application à TLS

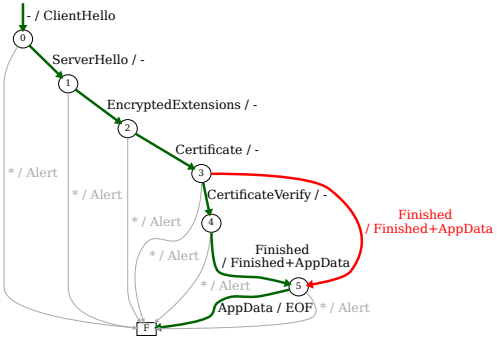
- recherche de court-circuits de manière systématique
- recherche de boucles dans la machine à états
- utilisation des différences entre les machines à états inférées pour faire du *fingerprinting*

Méthodologie d'analyse



- Plus de 400 clients et serveurs *open source* testés
- De nombreuses versions d'OpenSSL, GnuTLS, wolfSSL, NSS, etc.

Contournement de l'authentification



- Détection d'EarlyCCS (CVE-2014-0224) et de FREAK (2015-0204) sur OpenSSL
- **Reproduction de la CVE-2020-24613 sur wolfSSL**
- Trois nouvelles CVE sur l'implémentation TLS 1.3 de wolfSSL

Boucles inattendues

Pile	Scénario	Messages	Durée max. entre 2 mess.
erlang 24	Serveur 1.0/1.2	Alerte `NoRenegotiation` ou `ApplicationData`	$>$ 1 heure*
fizz 22.01.24	Client 1.3	`ChangeCipherSpec`	$>$ 1 heure
matrixssl 4.0 - 4.3	Serveur 1.0/1.2	Alerte `NoRenegotiation`	$\approx$ 40 secondes
NSS 3.15 - 3.78	Serveur 1.0/1.2	Alerte `NoRenegotiation`	$>$ 1 heure
OpenSSL $<$ 1.1.0	Serveur 1.0/1.2	`ApplicationData` vide	$>$ 1 heure

Fingerprinting (1/2)

Pour un scénario donné (rôle, version TLS, options)

- différentes piles produisent *toujours* des machine à états différentes
- des versions consécutives d'une même pile peuvent partager un automate
- extraire les séquences séparant les automates permet une prise d'empreinte
- approche complémentaire à d'autres outils de *fingerprinting*

Fingerprinting (1/2)

Pour un scénario donné (rôle, version TLS, options)

- différentes piles produisent *toujours* des machine à états différentes
- des versions consécutives d'une même pile peuvent partager un automate
- extraire les séquences séparant les automates permet une prise d'empreinte
- approche complémentaire à d'autres outils de *fingerprinting*

Les serveurs TLS 1.3 étudiés peuvent être séparés en 13 classes avec 8 séquences

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

Fingerprinting (2/2)

Pile	Versions	Nombre d'états
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

Extension à d'autres protocoles

Travaux en cours sur SSH et OPC-UA

TLS 1.3 est un protocole relativement simple et bien spécifié

- machines à états attendues avec 4-6 états
- une dizaine de messages dans le vocabulaire

Travaux en cours sur SSH et OPC-UA

TLS 1.3 est un protocole relativement simple et bien spécifié

- machines à états attendues avec 4-6 états
- une dizaine de messages dans le vocabulaire

SSH

- un protocole à 3 étages : *Transport*, *Authentication*, *Connection* (30 messages en tout)
- existence « naturelle » de boucles complexes (renégociation)
- la couche *Connection* est complexe (canaux multiples)
- piles étudiées : OpenSSH, libssh, asyncssh, dropbear, wolfSSH

Travaux en cours sur SSH et OPC-UA

TLS 1.3 est un protocole relativement simple et bien spécifié

- machines à états attendues avec 4-6 états
- une dizaine de messages dans le vocabulaire

SSH

- un protocole à 3 étages : *Transport*, *Authentication*, *Connection* (30 messages en tout)
- existence « naturelle » de boucles complexes (renégociation)
- la couche *Connection* est complexe (canaux multiples)
- piles étudiées : OpenSSH, libssh, asyncssh, dropbear, wolfSSH

OPC-UA

- un protocole issu du monde des systèmes industriels / SCADA
- une spécification parfois bancale
- piles étudiées : différentes implémentations en .Net, C, Python, Rust

Une machine à états infinie

La machine à états d'OpenSSH ne peut pas être représentée par une machine de Mealy

- dans la dernière couche (*Connection Layer*),

Une machine à états infinie

La machine à états d'OpenSSH ne peut pas être représentée par une machine de Mealy

- dans la dernière couche (*Connection Layer*),
- on peut initier une renégociation (`KEXINIT`)...
- demander l'ouverture de n nouveaux canaux (`CHANNEL_OPEN`)...
- puis terminer la renégociation (`DH_INIT`),

Une machine à états infinie

La machine à états d'OpenSSH ne peut pas être représentée par une machine de Mealy

- dans la dernière couche (*Connection Layer*),
- on peut initier une renégociation (`KEXINIT`)...
- demander l'ouverture de n nouveaux canaux (`CHANNEL_OPEN`)...
- puis terminer la renégociation (`DH_INIT`),
- ce qui donne lieu à la séquence de messages suivante : `DH_REPLY`, n fois `NEWKEYS` `OPEN_CONFIRM`

Une machine à états infinie

La machine à états d'OpenSSH ne peut pas être représentée par une machine de Mealy

- dans la dernière couche (*Connection Layer*),
- on peut initier une renégociation (`KEXINIT`)...
- demander l'ouverture de *n* nouveaux canaux (`CHANNEL_OPEN`)...
- puis terminer la renégociation (`DH_INIT`),
- ce qui donne lieu à la séquence de messages suivante : `DH_REPLY`, *n* fois `NEWKEYS` `OPEN_CONFIRM`

Un automate fini ne pouvant compter les parenthèses, une solution est d'approximer la réponse

- en groupant les réponses `OPEN_CONFIRM` avec un message factice `OPEN_CONFIRM+`

Des machines à états qui explosent

Inférence de la machine à états d'asyncssh (couches *Transport* + *Authentication*)

- 5 + 5 messages dans le vocabulaire
- 360 états

Des machines à états qui explosent

Inférence de la machine à états d'asyncssh (couches *Transport* + *Authentication*)

- 5 + 5 messages dans le vocabulaire
- 360 états

L'inférence des trois couches mène à des machines à états de plus de mille états

- au-delà d'une analyse difficile
- la question de la légitimité d'une telle machine à états se pose

Conclusion

Conclusion

Les machines à états des protocoles réseau peuvent être complexes

- les spécifications sont rarement complètes et formellement définies
- les écarts peuvent mener à des failles de sécurité...
- ou permettre d'identifier et de séparer les implémentations

Conclusion

Les machines à états des protocoles réseau peuvent être complexes

- les spécifications sont rarement complètes et formellement définies
- les écarts peuvent mener à des failles de sécurité...
- ou permettre d'identifier et de séparer les implémentations

Problème malheureusement classique

- l'implémentation *réagit* aux messages reçus...
- ... sans restreindre l'ensemble des messages licites a priori
- nombreuses vulnérabilités depuis 2014 sur TLS

Conclusion

Les machines à états des protocoles réseau peuvent être complexes

- les spécifications sont rarement complètes et formellement définies
- les écarts peuvent mener à des failles de sécurité...
- ou permettre d'identifier et de séparer les implémentations

Problème malheureusement classique

- l'implémentation *réagit* aux messages reçus...
- ... sans restreindre l'ensemble des messages licites a priori
- nombreuses vulnérabilités depuis 2014 sur TLS

Propositions

- utiliser une représentation formelle comme spécification
- ne pas multiplier les chemins acceptables

Questions ?

Merci pour votre attention

Références

[PhD16] *A study of the TLS ecosystem*. OL. Soutenance de thèse, 2016

[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*. A. Rasoamanana, OL et H. Debar

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