Analyse de l'implémentation d'un client TLS 1.3 en utilisant l'algorithme L^*

Aina Toky Rasoamanana
Télécom SudParis
Institut Polytechnique de Paris
Email : aina.rasoamanana@telecom-sudparis.eu

Olivier Levillain
Télécom SudParis
Institut Polytechnique de Paris
Email : olivier.levillain@telecom-sudparis.eu

***Résumé*—Dans cet article, nous présontons nos résultats autour de l'inférence de machine à états d'un client TLS 1.3 de wolfSSL. Avec notre méthode, nous pouvons reproduire des vulnérabilités connues (CVE-2020-24613 et CVE-2020-12457), mais aussi, nous avons trouvé un nouveau bug qui affecte le client TLS 1.3 de toutes les versions wolfSSL jusqu'à 4.6.0. Notre travail montre que l'algorithme L* est efficace pour inférer la machine à états des implémentations des procoles cryptographiques.**

I. Introduction

SSL/TLS est un protocole cryptographique ayant pour but de créer un canal de communication sécurisé qui garantit à la fois l'authentification, la confidentialité et l'intégrité. TLS 1.3 [4], qui corrige plusieurs vulnérabilités connues, est la version la plus récente de SSL/TLS. Un exemple complet des échanges entre un client et serveur TLS 1.3 est donné à la figure 1.

Avec TLS 1.3, l'échange de clés est réalisé avec les extensions `ClientKeyShare` et `ServerKeyShare`, des extentions incluses respectivement dans les messages `ClientHello` et `ServerHello`, ce qui permet de réaliser la négociation des paramètres, l'échange de clés et l'authentification du serveur avec un seul aller-retour.

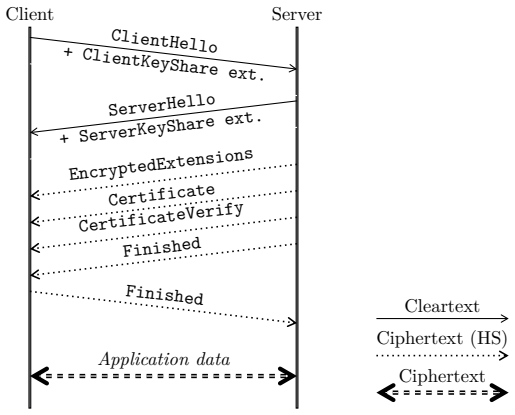

FIGURE 1. Exemple d'un Handshake Protocol de TLS 1.3

Tous les messages TLS 1.3 après le message `ServerHello` sont chiffrés. Pour que le client puisse authentifier le serveur explicitement, ce dernier envoie son certificat (dans le message `Certificate`) ainsi que la signature de tous les messages précédents (dans le message `CertificateVerify`). Le message `Finished` sert de confirmation que les deux parties utilisent la même clé.

En 2015, de Ruiter et Poll ont proposé une approche pour évaluer la machine à états de différente pile TLS (en boîte noire) en utilisant L* [3], [2]. Cette methode a été utilisée pour analyser d'autres protocoles tels que DTLS, SSH, QUIC, OpenVPN, etc. et a permis de trouver plusieurs bugs dans plusieurs implémentations.

En 2020, des chercheurs de NCC group ont trouvé une vulnérablité (CVE-2020-24613) sur l'implémentation du client TLS 1.3 de wolfSSL [1], qui permet à un attaquant de réaliser une attaque de type homme du milieu en ignorant les messages `Certificate` et `CertificateVerify`, c'est à dire, en envoyant directement le message `Finished` juste après `EncryptedExtensions`.

Pour comprendre cette vulnérabilité, nous avons inféré la machine à états du client TLS 1.3 de wolfSSL en utilisant un algorithme appelé L* [1]. L* nous a permis de reproduire cette vulnérabilité, et aussi, de trouver un nouveau bug dans la version jusqu'à 4.6.0 de wolfSSL.

Pour illustrer nos résultats, nous décrivons d'abord notre méthodologie d'analyse puis nous regardons en détails les résultats obtenus.

II. Méthodologie de l'inférence

Pour l'implémentation de notre outil d'inférence, nous utilisons `pylstar` [2] (une implémentation de l'algorithme L* en python) et `Scapy` [3] (un outil de manipulation de paquets réseaux écrit en python).

Utiliser L* pour inférer la machine à états d'une pile TLS nécessite l'existence d'un harnais capable d'abstraire et de concrétiser des messages. Pour notre cas, notre harnais serait une implémentation d'un serveur TLS 1.3 orientée message, c'est à dire, une implémentation qui permet d'envoyer des messages TLS dans n'importe quel ordre même si le message n'a aucun sens au moment où on envoie le message. Pour cela, nous avons donc ajouter des modifications dans `Scapy` pour supporter ce genre de comportement.

L* nécessite également la définition du vocabulaire que nous devons concrétiser avant d'envoyer les

1. https://www.wolfssl.com/
2. https://github.com/gbossert/pylstar.git/tree/next
3. https://github.com/pictyeye/scapy/tree/pylstar

messages à la cible (ici le client TLS 1.3). Pour notre inférence, nous avons défini les vocabulaires suivants : `ServerHello(SH)`, `EncryptedExtensions(EE)`, `Certificate(Cert)`, `EmptyCertificate (EmptyCert)`, `CertificateVerify(CertVerify)` et `Finished`.

Concernant l'interprétation des automates, nous avons abstrait les réponses aux stimuli par des vocabulaires simples tels que `Finished` et `AppData` ; et des différents alertes. Tous les messages que nous n'avons pas pu interpréter sont notés `Unknown`, et – signifie qu'on n'a rien reçu de la part de la cible après un timeout. Enfin, `EOF` signifie que la connexion est fermée.

Quand tout se passe correctement, L* infère un automate fini déterministe représentant le comportement de la cible.

III. Resultats

Dans tout ce qui suit, le chemin vert correspond au happy path et les erreurs de la machine à états sont colorés en rouge. Notons qu'à partir du moment où nous recevons les messages `Finished` et `AppData` du client en passant par des chemins rouges, nous avons un bug.

Notre premier résultat concerne la reproduction du CVE-2020-24613 et CVE-2020-12457. En ajoutant le message `ChangeCipherSpec(CCS)` dans le vocabulaire, nous avons mis en évidence ces deux vulnérabilités simultanément.

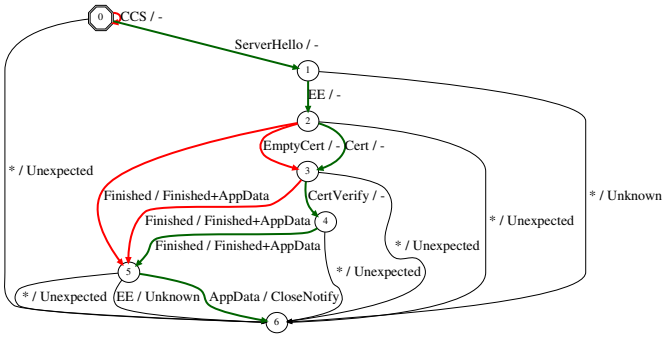

Figure 2. Machine à états du client TLS 1.3 de wolfSSL v4.4.0

La figure 2 represente la machine à états d'un client TLS 1.3 de wolfSSL v4.4.0. Elle nous montre que nous pouvons envoyer indéfiniment le message `ChangeCipherSpec`, ce qui permetrait à un attaquant de réaliser une attaque par déni de service (CVE-2020-12457). Elle nous montre également le CVE-2020-24613. En effet, la figure 2 montre qu'en ignorant les messages `Certifcate` et `CertifcateVerify`, un attaquant peut contourner l'authentification du client TLS 1.3 de wolfSSL v4.4.0.

Après la decouverte de cette vulnérabilité, wolfSSL a sorti une nouvelle version qui corrige le bug trouvé par les chercheurs de NCC group (ces derniers ont confirmé l'absence du bug dans la nouvelle version). Sauf, qu'en réalité, il y avait encore un bug très similaire à ce qu'ils

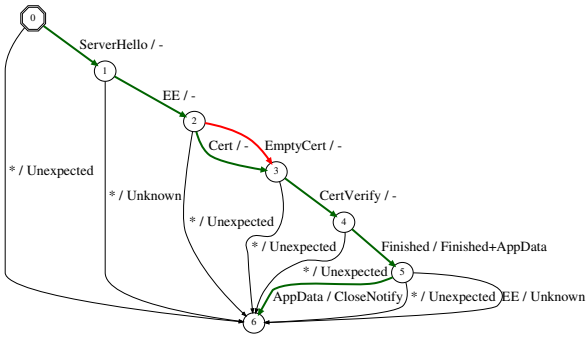

Figure 3. Machine à états du client TLS 1.3 de wolfSSL v4.6.0

ont trouvé dans les versions jusqu'à 4.6.0 de wolfSSL (voir figure 3).

En effet, un attaquant peut toujours contourner l'authentification du client TLS 1.3 de wolfSSL, en envoyant un certificat vide (`EmptyCertificate`) à la place du certificat du serveur. En revanche, un certificat généré par une autorité de certification inconnue par le client est rejeté par le client wolfSSL.

Nous pouvons remarquer ce nouveau bug (CVE-2021-3336) même dans les versions vulnérables au CVE-2020-24613 (voir figure 2). Le CVE-2021-3336 est corrigé dans wolfSSL v4.7.0.

IV. Conclusions

Dans cet article, nous avons présenté une méthodologie efficace pour retrouver des erreurs dans la machine à états d'une implémentation TLS 1.3. Elle nous a permis de reproduire des bugs du client TLS 1.3 de wolfSSL (CVE-2020-24613 et CVE-2020-12457) et de trouver un nouveau bug qui affecte toutes les versions de wolfSSL (implémentant TLS 1.3) jusqu'à 4.6.0 (CVE-2021-3336).

En plus des reproductions de bugs et la decouverte d'un nouveau bug, nous avons modifié `scapy` pour qu'il soit capable d'envoyer des messages TLS dans n'importe quel ordre qu'on veut et nous avons developpé un outil qui permet d'analyser une implémentation de TLS 1.3.

Nous prévoyons d'analyser plusieurs piles TLS avec notre outil et de systématiser nos tests pour suivre l'évolution de la machine à états d'une implémentation TLS d'une version à une autre.

Références

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2) :87–106, 1987.

[2] Joeri de Ruiter. A tale of the openssl state machine : A large-scale black-box analysis. In Billy Bob Brumley and Juha Röning, editors, *Secure IT Systems - 21st Nordic Conference, NordSec 2016, Oulu, Finland, November 2-4, 2016, Proceedings*, volume 10014 of *Lecture Notes in Computer Science*, pages 169–184, 2016.

[3] Joeri de Ruiter and Erik Poll. Protocol State Fuzzing of TLS Implementations. In *24th USENIX Security Symposium, Washington, D.C., USA*, pages 193–206, August 2015.

[4] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446 (Proposed Standard), August 2018.