Surveillance et récupération de l'exécution des fonctions cryptographiques

Matthieu AMET
Université de Lorraine, CNRS - Loria
Nancy, France
matthieu.amet@loria.fr

Oussama BEN MOUSSA
ENSEIRB-MATMECA
Talence, France

Guillaume BONFANTE
Université de Lorraine, CNRS - Loria
Nancy, France
guillaume.bonfante@loria.fr

Sébastien DUVAL
Université de Lorraine, CNRS - Loria
Nancy, France
sebastien.duval@loria.fr

***Abstract*—L'analyse de l'exécution des programmes est un outil clé pour l'évaluation de la sécurité logicielle, en particulier dans le domaine de la cryptographie où la protection des secrets est essentielle. Cependant, les méthodes existantes d'inspection des traces d'exécution sont souvent coûteuses en ressources ou intrusives, limitant leur applicabilité à grande échelle.**

Dans ce travail, nous proposons un cadre d'analyse dynamique léger permettant d'identifier les opérations cryptographiques et d'extraire des secrets sans nécessiter d'accès au code source ni de modifications invasives du programme cible.

Nous illustrons l'efficacité de notre approche à travers deux scénarios concrets : la détection d'opérations cryptographiques dans les traces d'exécution, notamment lors d'établissements de connexions TLS, et la récupération de clés privées générées par OpenSSL RSA. Les résultats expérimentaux démontrent la pertinence de notre méthode sur diverses plateformes et configurations, soulignant son potentiel pour l'évaluation de la sécurité et les simulations d'attaque.

I. Introduction

Les attaques visant les bibliothèques logicielles critiques soulignent la nécessité d'une surveillance accrue de l'exécution des programmes. En mars 2024, la vulnérabilité CVE-2024-3094 affectant XZ Utils a mis en lumière les risques liés à l'introduction discrète de code malveillant dans des composants essentiels. Cette compromission, détectée in extremis avant son exploitation à grande échelle, pose une question fondamentale : dans quelles conditions une attaque similaire pourrait-elle passer inaperçue ?

Un attaquant capable de modifier une bibliothèque partagée sur un système cible peut introduire un comportement malveillant subtil sans perturber le bon fonctionnement apparent du programme. Lorsqu'un exécutable manipule des opérations cryptographiques, une telle modification peut permettre l'exfiltration de secrets sans accès direct au code source. Pour minimiser la détection, l'attaquant privilégiera des méthodes d'interception discrètes et à faible empreinte.

L'objectif de cette étude est d'évaluer dans quelle mesure il est possible d'extraire des secrets cryptographiques en exploitant uniquement les traces d'exécution d'un programme, sans analyse symbolique coûteuse. En s'appuyant sur des outils comme Intel Pin, nous examinons la faisabilité de cette approche dans un contexte réaliste, en prenant OpenSSL [10] comme cas d'étude en raison de son rôle central dans les communications sécurisées.

Notre contribution repose sur une méthodologie d'analyse dynamique optimisée pour isoler les opérations cryptographiques et filtrer les données non pertinentes, améliorant ainsi à la fois la furtivité et l'efficacité du processus. Nous illustrons cette approche en analysant l'exécution de `openssl genrsa`, démontrant comment les matériaux de clé privée peuvent être extraits pendant le calcul. Nos résultats soulignent les implications en matière de sécurité et ouvrent la voie à des contre-mesures adaptées contre ce type d'exfiltration.

II. Identification des parties cryptographiques dans les échanges web

Cette section décrit une approche permettant d'identifier les opérations cryptographiques sensibles au sein des traces d'exécution des connexions HTTPS, en mettant l'accent sur le protocole TLS [8]. L'objectif principal est de détecter les opérations manipulant des secrets, tels que les clés privées ou les clés de session partagées, en analysant les traces d'exécution au niveau des instructions.

A. Collecte et filtrage des traces

Pour commencer, des connexions HTTPS ont été établies à l'aide de l'outil `curl`, avec Wireshark enregistrant les détails cryptographiques échangés au cours de chaque session. Quatre configurations TLS courantes reprenant les standards [2], [9], [3], [4], [5] ont été analysées afin d'assurer une diversité des algorithmes cryptographiques :

- `TLS_AES_256_GCM_SHA384`
- `TLS_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
- `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`

Pour chaque configuration, 50 sites web populaires ont été testés, générant environ 5 Go de données de traces d'exécution par session. Un outil d'instrumentation léger a capturé les deux premiers octets (opcodes) de chaque instruction afin de réduire

la surcharge de performance tout en conservant suffisamment d'informations pour l'analyse.

Les traces ont été prétraitées afin de ne conserver que les instructions d'appel de fonction, réduisant ainsi la taille des données et se concentrant sur les régions pertinentes pour le traitement cryptographique. Cependant, ce filtrage a introduit des défis, tels que l'exécution parallèle des threads, les appels boîte noire dans le système `BIO` d'OpenSSL, et la surcharge des routines d'initialisation cryptographique. Malgré ces obstacles, les étapes clés de TLS — comme l'échange de clés publiques, la génération de secrets partagés et l'établissement d'une session sécurisée — ont été identifiées avec succès, permettant ainsi l'isolement des zones cryptographiques.

B. Identification des opérations cryptographiques

L'identification des fonctions manipulant des données sensibles était essentielle pour cette analyse. L'investigation s'est particulièrement concentrée sur les opérations impliquant les clés privées et les secrets de session. Par exemple, lors d'un échange de clés, le secret partagé dérivé est transmis à la fonction `AES` pour le chiffrement. L'analyse des traces d'exécution a permis d'identifier précisément les zones d'instructions associées à ces opérations sensibles.

L'étude a confirmé que la fonction `AES_unwrap_key` joue un rôle central dans la manipulation des clés de session. Ces opérations ont été identifiées de manière fiable dans les traces, démontrant ainsi la faisabilité de l'isolation des zones cryptographiques même dans des environnements d'exécution hautement complexes.

C. Détection automatisée par apprentissage machine

Pour automatiser l'identification des zones cryptographiques sensibles, des techniques d'apprentissage supervisé ont été employées. Les traces d'exécution ont été divisées en blocs de 200 instructions avec un fenêtre glissante de 50 instructions pour l'étiquetage. Cette méthode a assuré que les modèles disposaient d'un contexte suffisant pour apprendre les motifs associés aux opérations cryptographiques.

Trois modèles d'apprentissage machine ont été testés :

- **Réseau de Neurones Récurrents (RNN) [7] :** Avec une précision de 78,4%, les RNNs ont montré une bonne capacité à distinguer les zones non cryptographiques, mais une précision moindre pour identifier correctement les zones cryptographiques.
- **Réseau de Neurones Convolutifs (CNN) [6] :** En transformant les opcodes en matrices de pixels, les CNNs ont amélioré la détection des zones cryptographiques, atteignant une précision de 80,5%.
- **Machine à Vecteurs de Support (SVM) [1] :** Le modèle SVM a surpassé les RNNs et CNNs, avec une précision impressionnante de 99,1%. Ce résultat est attribué à la capacité des SVMs à gérer les tâches de classification binaire avec une grande précision, identifiant efficacement les zones cryptographiques et non cryptographiques.

Les résultats soulignent la robustesse des SVMs pour cette tâche, fournissant une méthode fiable et efficace pour surveiller les traces d'exécution et identifier les opérations cryptographiques sensibles.

III. Analyse de la génération de clés RSA d'OpenSSL

Cette section présente une analyse plus détaillée du processus de génération de clés RSA d'OpenSSL, en se concentrant sur l'identification des instructions spécifiques qui manipulent les secrets cryptographiques. Bien que l'ingénierie inverse soit une option, notre objectif est d'automatiser le processus.

A. Scénario d'attaque

Considérons un attaquant tentant de commettre une fraude d'identité en exploitant la bibliothèque cryptographique d'une victime. Si l'attaquant peut remplacer la bibliothèque de la victime par une version modifiée, l'identification des instructions qui manipulent les clés secrètes devient cruciale. Par exemple, si nous savons qu'une instruction particulière à une adresse connue manipule la clé secrète dans un registre spécifique (par exemple, `RAX`), celle-ci peut être interceptée pour exfiltrer le secret via un canal auxiliaire.

Nous utilisons la commande `genrsa` d'OpenSSL comme exemple concret. Cette commande génère des paires de clés RSA couramment utilisées dans des protocoles comme SSH [11] et TLS :

```
openssl genrsa -out {file} [512|1024|2048]
```

La clé privée générée peut être examinée après la génération avec :

```
openssl rsa -in {file} -noout -text
```

Nous donnant la sortie vue en Figure 1.

```
Private-Key: (512 bit, 2 primes)
modulus:
    00:bf:0b:47:dc:ab:da:dc:bc:6f:ca:6c:92:95:9e:
    e3:74:21:e8:91:b1:16:ce:e1:79:ee:a6:0c:32:14:
    f9:58:ef:95:3e:e5:df:fc:e1:73:f9:a3:0c:ac:30:
    79:ad:99:11:c5:f5:0a:3b:5e:11:cf:ca:c3:13:02:
    53:16:eb:27:fd
publicExponent: 65537 (0x10001)
privateExponent:
    00:92:32:e6:ce:97:f1:88:64:e8:44:07:ac:71:b5:
    c3:28:b7:5e:4c:48:32:45:25:c5:f2:fc:bd:6e:82:
    20:83:8e:98:50:bc:9e:87:07:1a:3f:51:e0:90:f2:
    f6:5d:41:4e:e7:29:96:d6:a4:65:40:fc:5e:7c:0f:
    48:02:a6:d5:81
prime1:
    00:f9:07:06:eb:ec:6c:11:d9:5d:f1:92:cc:40:30:
```

Fig. 1. Sortie de la commande de génération de clé RSA

Au lieu de récupérer les secrets après coup, notre objectif est de les intercepter pendant la génération de la clé. Bien que notre objectif principal soit l'exposant privé (PrivateExponent), la méthode est généralisable à d'autres composants de la clé.

L'approche fonctionne en exécutant OpenSSL sous un traceur pour capturer la trace d'exécution, tandis que l'exposant privé est extrait. Les données extraites aident à

identifier les instructions, les registres ou les zones mémoire qui manipulent la clé secrète. En utilisant ces informations, nous pouvons créer un cheval de Troie qui se comporte comme le programme OpenSSL original mais qui exfiltre la clé secrète vers un fichier externe (un "canal auxiliaire").

Nous validons les données du canal auxiliaire en les comparant au secret extrait et nous nous assurons que le cheval de Troie fournit les mêmes clés, confirmant ainsi son exactitude.

B. Vue d'ensemble

Notre approche consiste à tracer l'exécution du programme cible, à extraire les informations secrètes de la clé générée, puis à utiliser ces données pour synthétiser un cheval de Troie. Les étapes clés sont décrites ci-dessous :

1) **Collecte de trace :** L'exécution du programme est tracée pour capturer les adresses d'instruction et les valeurs des données. En restreignant la traçabilité aux bibliothèques spécifiques (par exemple, `libcrypto`), nous réduisons considérablement la taille de la trace, passant d'environ 100 Go à 1 Go.
2) **Extraction du secret :** L'exposant privé est extrait du fichier de clé généré en utilisant les outils OpenSSL standards. Cela sert de vérité de base pour identifier les instructions qui manipulent le secret.
3) **Synthèse du cheval de Troie :** En utilisant la trace et le secret extrait, nous identifions les instructions et registres pertinents associés à la manipulation du secret. Ces informations sont ensuite utilisées pour synthétiser un cheval de Troie `pintool`, qui sauvegarde le secret extrait dans un fichier séparé tout en répliquant la fonctionnalité du programme original.

Le cheval de Troie peut être exécuté comme suit :

```
pin -t openssl_backdoor.so -- \
                        openssl genrsa ...
```

Bien que l'implémentation actuelle stocke les secrets dans un fichier local, il est facile d'étendre cette approche à l'exfiltration à distance.

C. Récupération des clés privées RSA

L'analyseur est dédié à un traitement postérieur qui récupère les informations relatives à la clé. Par exemple, pour la commande `openssl genrsa` mentionnée ci-dessus, nous récupérons l'exposant privé correspondant à la clé avec la commande :

```
openssl rsa -in new_key_file.key \
                                -noout -text
```

suivie d'un simple processus de filtrage. Le résultat est stocké dans un fichier qui sera ensuite lu par le synthétiseur.

D. Analyse de trace

La phase d'apprentissage du synthétiseur fonctionne comme suit : nous prenons la véritable valeur du secret extrait par l'outil extracteur et la trace d'exécution sous forme d'entrées dans le fichier binaire `trace.bin`. Le secret est une séquence d'octets notée k, de longueur $\text{len}(k)$, tandis que les entrées de la trace sont notées S et leur taille N. Les registres enregistrés sont notés R.

Dans un premier temps, nous devons filtrer les instructions/registre qui partagent une partie du secret. Pour cela, nous construisons le dictionnaire suivant D. Ses clés sont des paires (i, r) avec $i \leq N$ et $r \in R$. À chaque clé, nous associons la plus longue sous-chaîne entre la valeur du registre et le secret. En fait, étant donné que les facteurs de longueur 1 et 2 sont trop courants, nous restreignons le dictionnaire aux séquences d'au moins 4 éléments.

Dans un second temps, nous examinons l'ensemble des instructions apparaissant dans la trace. Les instructions sont identifiées par leur adresse $a = S[i].\text{address}$. Pour une adresse a, nous calculons la liste (ordonnée) :

$$S_a = [i < N \mid a = S[i].\text{address}]$$

Autrement dit, il s'agit de la liste des entrées spécifiques à une adresse donnée a.

Une adresse a et un registre r sont considérés comme compatibles avec la clé k lorsque trois constantes α, β, γ et une liste croissante d'indices $\ell_0, \ell_1, \ldots, \ell_{\text{len}(k)} \leq \text{len}(k)$ existent, telles que pour tous $0 \leq j < \gamma$:

- $\ell_0 = 0$, $\ell_\gamma = \text{len}(k)$
- $D[S_a[\alpha + j \times \beta], r] = [k_{\ell_j}, \ldots, k_{\ell_{j+1}}]$

Autrement dit, après α applications de l'instruction à l'adresse a, chaque β-ème application de cette instruction, le contenu du registre r à la position q effectuera un balayage linéaire dans le secret. Nous nous permettons d'ignorer les dernières instructions γ de la trace. Parfois, nous avons observé que deux boucles sont nécessaires pour obtenir une adéquation correcte entre le registre et la clé, d'où le facteur β.

En résumé, identifier une séquence couvrante signifie que nous sommes capables de reconstruire la clé à partir de la valeur des registres aux instructions correspondantes.

E. Synthèse du pintool

Nous supposons avoir trouvé une séquence $\{(a_1, r), \ldots, (a_t, r)\}$ avec les paramètres respectifs. Nous pouvons alors construire un pintool qui sautera les premières occurrences des applications d'une instruction. L'oubli des dernières instructions peut être effectué via un tampon. Enfin, le comptage modulo β est (presque) immédiat.

En réalité, la principale difficulté de cette partie réside dans le choix de la "meilleure" séquence. En effet, une fois qu'une séquence est trouvée, nous avons constaté qu'il y en a en réalité beaucoup. Certaines impliqueront plus d'instructions, entraînant une perte d'efficacité en temps constant. Étant donné que le pintool repose sur l'interruption d'un très petit nombre d'instructions (généralement une seule, lorsque la séquence suivante est composée d'un singleton), l'overhead temporel est marginal.

F. Résultats expérimentaux et validation

Nous avons validé notre approche sur plusieurs plateformes (Windows, Linux, systèmes ARM) et versions d'OpenSSL (3.0

à 3.2). Dans toutes les expériences, les adresses et registres extraits contenaient systématiquement les bonnes données secrètes. Les paramètres de la clé, tels que l'exposant privé, ont été correctement récupérés pour des tailles de clés allant de 512 à 4096 bits. Il est à noter que les adresses identifiées sont restées constantes à travers les différentes tailles de clés, ce qui témoigne de la robustesse et de la généralisabilité de l'approche.

Pour les processeurs basés sur ARM, nous avons utilisé le cadre `DynamoRIO` pour répliquer le processus de traçage, obtenant des résultats comparables à ceux des systèmes x86. L'enregistrement et l'analyse des traces ont été efficaces, nécessitant environ une minute sur du matériel standard (Intel i5-1140G7).

Nous avons également testé l'outil sur différentes longueurs de clés, à savoir 512, 1024, 2048, 3096. Dans chaque cas, nous avons obtenu une réponse correcte. De plus, tous les résultats étaient compatibles : les adresses obtenues pour une clé de 512 bits étaient les mêmes pour les autres tailles. Du point de vue de l'attaquant, cela constitue une bonne nouvelle, car il n'aura pas à voler l'information avant l'attaque.

G. Discussion

Dans tous les cas mentionnés ci-dessus, les instructions identifiées comme traitant des secrets cryptographiques se trouvaient dans la bibliothèque `libcrypto`. Par exemple, l'analyse a mis en évidence des zones mémoires préfixées par `EVP_ASYM_CIPHER`, qui sont centrales dans les mécanismes cryptographiques d'OpenSSL. Nous avons observé une grande variété de registres impliqués, tels que `RAX`, `RSI`, `YMM0`, ainsi que des emplacements mémoire.

L'outil a fonctionné de manière fiable sur différentes versions de bibliothèque et architectures de processeur, démontrant sa robustesse. De plus, l'extraction de la clé et la synthèse du backdoor ont été réalisées avec un surcoût computationnel minimal, soulignant la praticité de cette approche automatisée.

H. Implications

Cette analyse met en évidence un risque de sécurité potentiel dans les logiciels cryptographiques. La capacité à automatiser l'identification des instructions manipulant des secrets cryptographiques souligne la nécessité de mettre en place des protections robustes contre de telles attaques, en particulier dans des scénarios où l'intégrité du logiciel ne peut être garantie.

IV. Conclusion

Dans ce travail, nous avons étudié la récupération de secrets cryptographiques durant l'exécution d'un programme. Nous avons démontré qu'en analysant l'exécution des programmes utilisant les fonctions OpenSSL, il est possible d'identifier où et quand les opérations cryptographiques sont effectuées. Cela permet d'isoler les parties sensibles de l'exécution en utilisant un minimum d'informations, principalement en se basant sur les opcodes des instructions, qui peuvent être capturées en temps réel lors d'une connexion internet.

Nous avons montré comment une valeur secrète, spécifiquement la clé privée lors de la génération de clés RSA avec `openssl genrsa`, peut être récupérée pendant l'exécution. Bien que notre outil ait été testé dans ce cas, il n'est pas limité à celui-ci et peut être généralisé à d'autres applications. L'outil est disponible en téléchargement sur notre site web.

Plusieurs pistes pour des recherches futures émergent de ce travail. Tout d'abord, nous avons noté qu'une grande partie des calculs sensibles d'OpenSSL se déroulent dans des "boîtes noires" qui résistent à une analyse de base. Une investigation plus approfondie pour pénétrer ces boîtes noires constitue une prochaine étape importante.

Une autre direction consiste à évaluer la quantité de données nécessaire pour récupérer des secrets cryptographiques. Bien que notre méthode montre déjà son efficacité avec un minimum de données de trace, l'analyse des fuites de canaux auxiliaires — telles que la consommation de temps, l'énergie ou les émissions électromagnétiques — pourrait offrir des vecteurs d'attaque alternatifs. Ces fuites, bien que bruyantes, sont accessibles à toute personne ayant un accès physique à l'appareil. Nous prévoyons également d'explorer s'il est possible de récupérer des secrets en analysant les traces de canaux auxiliaires, comme les poids de Hamming, avec des exigences réduites pour l'attaquant.

References

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[2] Joan Daemen and Vincent Rijmen. *The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition*. Information Security and Cryptography. Springer, 2020.

[3] PUB FIPS. 180-2. *FIPS Publication–Secure hash standard (+ Change Notice to include SHA-224)*, 2002.

[4] Ibrahim Hajjeh and Mohamad Badra. ECDHE_PSK Cipher Suites for Transport Layer Security (TLS). RFC 5489, March 2009.

[5] Jakob Jonsson and Burt Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, February 2003.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[7] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[8] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[9] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288, August 2008.

[10] The OpenSSL Project. OpenSSL: The open source toolkit for SSL/TLS. www.openssl.org, April 2003.

[11] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Connection Protocol. RFC 4254 (Proposed Standard), January 2006.