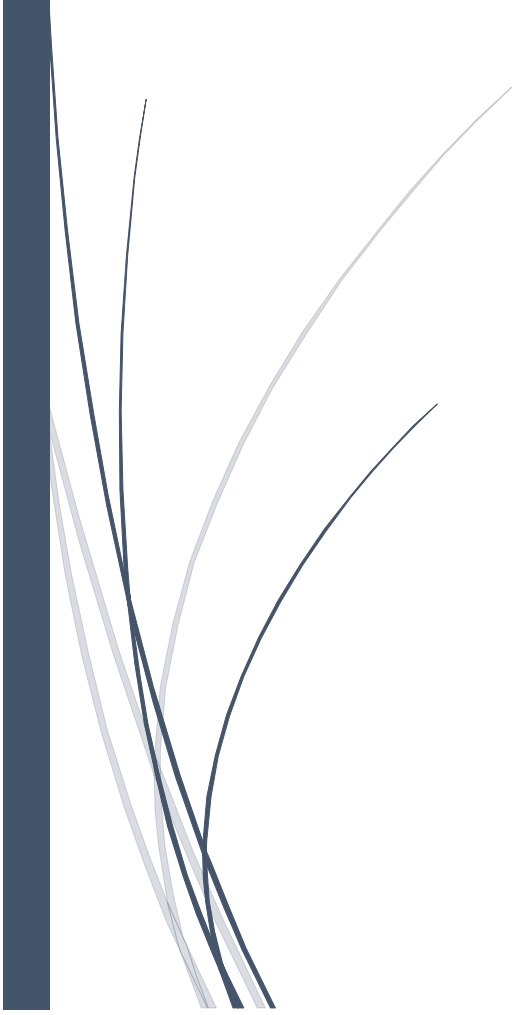
24/11/2022

Composants vulnérables et Obsolète



Membres du groupe 8

- El Hadji Malick SY
- Mouhamed FALL
- Alexandre Adiouma NDENE
- El Hadji Moussa Koulibaly
- Bocar DIALLO

Table des matières

I. C’est quoi une fonction de hachage cryptographique ? 2
II. Principes et vocabulaires ? 2
III. Conséquences de la cryptographie obsolète sur la sécurité 3
IV. Exemples d’attaques d’une cryptographie obsolète 4
1. Lucky13 4
2. BEAST (Browser Exploit Against SSL/TLS) 5
3. POODLE (Padding Oracle On Downgraded Legacy Encryption) 5

I. C’est quoi une fonction de hachage cryptographique ?

Une **fonction de hachage cryptographique** est une fonction de hachage qui, à une donnée de taille arbitraire, associe une image de taille fixe, et dont une propriété essentielle est qu'elle est pratiquement impossible à inverser, c'est-à-dire que si l'image d'une donnée par la fonction se calcule très efficacement, le calcul inverse d'une donnée d'entrée ayant pour image une certaine valeur se révèle impossible sur le plan pratique. Pour cette raison, on dit d'une telle fonction qu'elle est à sens unique.

Les fonctions de hachage sont utilisées dans la sécurité informatique pour protéger les données contre la corruption ou la falsification. Une fonction de hachage prend un bloc de données et produit une valeur de hachage de taille fixe. Les données peuvent être des fichiers, des messages ou des flux de données. Une valeur de hachage est un nombre utilisé pour représenter des données. Les fonctions de hachage sont utilisées dans de nombreuses applications telles que les signatures numériques, la vérification de l'intégrité des fichiers et la déduplication des données. Les fonctions de hachage sont un moyen de s'assurer que les données n'ont pas été falsifiées. Si même un bit de données change, la valeur de hachage change. En effet, un hachage est une représentation de données et la modification des données modifie la représentation. Les fonctions de hachage sont également utilisées pour détecter les données en double. Si deux données ont la même valeur de hachage, elles sont plus susceptibles d'être identiques.

II.Principes et vocabulaires ?

Une fonction de hachage, c'est de façon très simplifiée une fonction mathématique qui, en fonction de ce qu'on lui donne à manger en entrée, va vous sortir une chaîne de longueur fixée à l'avance : c'est ce qu'on appelle l'empreinte (ou le condensé, ou encore la somme de contrôle), une suite de caractères assez courte (tout est relatif hein) représentant le texte qu'il condense. Ces empreintes sont beaucoup utilisées pour la signature électronique, et doivent être rapides à calculer afin de rendre rapide l'identification des données.

Par exemple, pour que la sortie fasse 160 bits, avec une entrée de (par exemple) 512 bits, il faut que la fonction de hachage intègre une fonction de compression. Concrètement, le bloc que vous entrez est découpé en morceau de taille définie, et la fonction de compression est appliquée à chacun de ces blocs.

Les fonctions de hachage répondent malgré tout à un certain nombre de contraintes. On peut citer :

- la résistance aux collisions : une collision, c'est le fait que deux textes différents donnent la même empreinte, et oui bien évidemment c'est une chose à éviter…
- la résistance au préimage : calculer une empreinte via une fonction de hachage, ça doit être facile, on est d'accord. Par contre, partir de l'empreinte pour remonter au texte original, ça, c'est à éviter. On stocke parfois des empreintes de mots de passe plutôt que les mots de passe eux-mêmes, justement parce qu'en cas de piratage, il est compliqué de faire la manipulation inverse !
- la résistance au second préimage : connaissant une chaîne d'entrée et l'empreinte correspondante, il doit être compliqué de trouver une deuxième chaîne d'entrée qui donne la même empreinte (une sorte de « mix » entre les deux premiers critères, si on veut). Attention, si la résistance aux collisions implique nécessairement une résistance à la seconde préimage, cela ne touche pas à la « première » préimage

III.Conséquences de la cryptographie obsolète sur la sécurité

Les algorithmes cryptographiques sont omniprésents dans les systèmes d'information. Ils servent à sécuriser les communications, à stocker des mots de passe, ou encore à chiffrer les disques. Cependant, les spécifications qui définissent ces usages reposent souvent sur des algorithmes ou des constructions cryptographiques obsolètes, c'est-à-dire pour lesquels une

faiblesse est connue. Bien que ces faiblesses soient parfois uniquement théoriques, ou apparemment inexploitables, force est de constater que les attaques ne peuvent que s'améliorer (*Attacks always get better*). Il est donc nécessaire de mettre en place des contre-mesures qui forcent les développeurs à faire des acrobaties dans le code cryptographique.

IV.Exemples d'attaques d'une cryptographie obsolète

Plusieurs attaques ont été décrites contre ce mode au cours des années, mais la personne qui le résume le mieux est Bodo Möller, qui décrivait dès 2001 les fondements théoriques de Lucky13, BEAST et POODLE.

1. Lucky13

Cette attaque, présentée en 2013 par Paterson et al, mettait en pratique une attaque théorique. Longtemps, l'application à TLS a été considérée comme non pertinente, pour deux raisons :

- à la première erreur de déchiffrement (quelle qu'elle soit), la connexion est coupée et toute reconnexion subséquente mène à un changement des clés cryptographiques ;
- les messages envoyés sur le réseau en cas d'erreur de padding ou d'erreur de MAC sont chiffrés, et donc indistinguables pour l'attaquant.

Le premier argument peut être contourné assez facilement si l'on considère les éléments qui peuvent intéresser un attaquant : un mot de passe ou un cookie d'authentification. Comme ces éléments sont répétés à chaque nouvelle connexion, l'attaquant apprend à chaque fois un peu d'information sur le secret qu'il veut recouvrer. Ces informations partielles collectées dans les différentes connexions peuvent ensuite être agrégées.
Concernant le second argument, il est intéressant de se référer à la RFC 4346 (TLS 1.1) qui spécifie qu'une implémentation doit se protéger des attaques temporelles (*timing attacks*), en rendant le traitement des messages protégés *essentiellement* le même, que le padding soit correct ou non. Une proposition d'implémentation suit, indiquant que cette solution laisse cependant un petit canal auxiliaire concernant le temps d'exécution. C'est ce canal qui est exploité dans l'attaque Lucky 13.

À la suite de ces révélations, les développeurs des différentes piles TLS ont réécrit le code correspondant pour supprimer cette différence dans le temps d'exécution, menant en particulier à un patch sordide pour OpenSSL. Pourtant, s2n, une implémentation récente de TLS, s'est révélée vulnérable à l'attaque, à cause d'une contre-mesure incomplète.
Face à ce problème, plusieurs approches sont possibles :

- documenter l'attaque dans une spécification en laissant le développeur traiter le problème ;
- réécrire la crypto à très bas niveau pour implémenter les contre-mesures nécessaires. OpenSSL contient une telle implémentation, au prix d'un *patch* illisible. Cependant, l'écriture de code cryptographique est un exercice délicat qu'il est préférable de laisser aux experts du domaine ;
- jeter le mode *mac-then-encrypt*, soit en implémentant *encrypt-then-mac* (défini pour TLS dans la RFC 7366), soit en utilisant un mode combiné tel que GCM. Ces solutions reposent respectivement sur une extension et sur une version particulière du protocole, ce qui nuit à la compatibilité.

2. BEAST (Browser Exploit Against SSL/TLS)

En 2011, Duong et Rizzo ont publié une attaque contre TLS, intitulée BEAST. Cette attaque repose sur l'utilisation dans SSL et TLS 1.0 d'un IV implicite : en dehors du premier message, chaque message est chiffré en utilisant comme IV le dernier bloc chiffré du message précédent. Ainsi, l'IV qui va être utilisé pour le message à venir est connu d'un attaquant pouvant espionner le canal.
Or, dès 1995, Rogaway avait montré que cette connaissance pourrait permettre des attaques à clair choisi. On suppose pour cela que l'attaquant, en plus de pouvoir observer les messages chiffrés, peut choisir une partie des messages clairs. Bien que cette description semble totalement irréaliste, les hypothèses sont remplies dans un navigateur...
Il est intéressant de remarquer que la contre-mesure avait été spécifiée dès 2006 dans la RFC 4346 (TLS 1.1), bien avant que la preuve de concept soit publiée. Cependant, l'attaque ayant été considérée comme théorique, presque aucune pile TLS n'avait mis en œuvre TLS 1.1 avant 2011.

3. POODLE (Padding Oracle On Downgraded Legacy Encryption)

Les attaques exploitant un oracle de padding CBC peuvent être dévastatrices contre SSLv3, puisque le padding y est mal spécifié : seul le dernier octet du padding doit être vérifié par le

destinataire. Cette observation a permis à des chercheurs travaillant chez Google de présenter en 2014 une preuve de concept exploitant un autre type d'oracle de padding : POODLE.

Par désespoir, certains ont même recommandé l'utilisation de RC4, pourtant victime d'attaques pratiques de plus en plus efficaces depuis 2013. Cependant, la seule solution est de bannir l'usage de SSLv3, un protocole vieux de plus de 20 ans.

Ce qui est intéressant, c'est qu'en écrivant des outils pour tester la vulnérabilité, des chercheurs se sont rendu compte qu'en fait, certaines piles TLS (et non SSLv3) implémentent tout de même cette version faible du padding CBC pour les versions récentes ; on a alors parlé de POODLE-TLS.

Ainsi, le mode CBC tel qu'il est utilisé dans TLS a montré ses limites. La version actuelle du protocole, TLS 1.2, a introduit les modes combinés (et GCM en particulier) pour apporter un peu de modernité au protocole. Ces nouveaux modes seront les seuls spécifiés dans TLS 1.3, en cours de définition. Pour réellement profiter de cette avancée en termes de sécurité, il faut maintenant s'atteler à désactiver TLS 1.0 et TLS 1.1, en plus de SSLv2 et SSLv3 (ce qui devrait déjà être fait...).