Rapport SF

Sécurité applicative

Formateur : Jean-Luc Beuchat

Date : 14 novembre 2023, 12H45 à 16H00 (1ère ½ journée)

21 novembre 2023, 12H45 à 16H00 (2ème ½ journée)

Durée : 1 journée

Présents : David Guillaume, Dasek Joiakim, Cardoso Rafael, Laurent Térence, Uka Zotrim

Rapporteur : David Guillaume (1ère ½ journée), Uka Zotrim (2ème ½ journée)

Table des matières

Table des matières

Ce cours est basé sur le matériel (environnement laboratoire Docker) dont les ressources sont les suivantes :

Matériel de cours	https://gitlab.com/hesso-vs/business-information-technology/63-22-it-security/lecture-notes-and-exercises-fall-2023
Env. Docker	https://gitlab.com/hesso-vs/business-information-technology/63-22-it-security/lecture-notes-and-exercises-fall-2023/-/blob/main/Setup/README.md
Notebook 1+2	https://gitlab.com/hesso-vs/business-information-technology/63-22-it-security/lecture-notes-and-exercises-fall-2023/-/blob/main/Topics/02_Cryptography_Integrity/AsynchronousLearning/Mathematics/Jupyter/modular_arithmetic.ipynb https://gitlab.com/hesso-vs/business-information-technology/63-22-it-security/lecture-notes-and-exercises-fall-2023/-/blob/main/Topics/03_Cryptography_Confidentiality/AsynchronousLearning/Mathematics/Jupyter/primes.ipynb

Du fait que le cours se base sur le matériel ci-dessus présentant tout le support de formation (source, codes, Powerpoint, ...), ce rapport se basera davantage sur le déroulement de la formation ainsi que les spécificités vues lors de la journée de formation, renvoyant les sujets aux références.

Environnement laboratoire

Avant de commencer la session de formation, nous avons dû mettre en place l'environnement laboratoire basé sur un container Docker. Pour ce faire, il nous a fallu cloner le dépôt GIT (Configuration de Git-Lab avec un Token), installer VS Code avec son extension « Remote Container » puis lancer l'environnement.

1ère ½ journée

Sécurité et attique des sites Web

La première partie de cette ½ journée s'est orientée sur les attaques possibles sur les sites Web tels que le « clickjacking », « cross-site scripting » et autres « XSS ».

Hachage

Le hachage est un processus qui produit un résultat fixe, appelé digest, à partir d'une entrée arbitraire. Ce digest a toujours la même taille en sortie, ce qui protège la confidentialité de la longueur du mot de passe et résiste aux attaques de force brute.

Bien que l'algorithme de hachage soit largement connu, il est difficile de remonter du digest au message d'origine, nécessitant environ 2^256 opérations, ou avec une constante k*2^256.

Pour optimiser le temps, on signe directement le digest lors de la création d'un message.

Bien que deux messages puissent générer le même digest (collision), trouver un deuxième message correspondant est extrêmement complexe. Éviter les collisions simples (deux messages produisant le même digest) est une caractéristique importante du hachage.

Une collision dans le contexte du hachage fait référence à une situation où deux entrées différentes produisent le même résultat de hachage, c'est-à-dire le même digest.

Le MD5 est considéré comme cassé, il faut donc se diriger vers du SHA-2, SHA-3 qui sont OK.

SHA-2 et SHA-3 ont été conçus avec l'objectif de calculer rapidement les empreintes, ce qui peut poser des problèmes en cas d'attaque par force brute. Pour contrer cela, il est crucial de ne pas permettre qu'un mot de passe soit calculé rapidement. C'est pourquoi on ajuste le paramètre k afin de ralentir la vitesse du calcul.

L'effet avalanche est un aspect crucial à prendre en compte. Il stipule qu'un changement d'un seul bit dans l'entrée d'un algorithme de hachage doit entraîner des changements significatifs dans au moins 50% de la séquence du digest résultant. Cela garantit que de petites variations dans les données d'entrée entraînent des variations substantielles dans la sortie du hachage, renforçant ainsi la sécurité du processus de hachage.

Salage

Le salage est une technique de sécurité utilisée dans le domaine du hachage de mots de passe. L'idée fondamentale du salage est d'ajouter une information aléatoire unique à chaque mot de passe avant de le hacher. Ce procédé a pour objectif de rendre les attaques par plus complexes et coûteuses.

Le caractère déterministe du hachage présente un inconvénient majeur : deux utilisateurs ayant le même mot de passe obtiennent le même hachage. Cette vulnérabilité est atténuée par l'utilisation du salage, qui implique la génération aléatoire d'une valeur de salage unique pour chaque utilisateur. Il est essentiel de valider à la fois le hachage et le salage.

Pour vérifier l'identité d'un utilisateur, le mot de passe, le salage et la fonction d'authentification associée sont récupérés. Ensuite, ces éléments sont utilisés pour contrôler et valider l'utilisateur, renforçant ainsi la sécurité du processus d'authentification. Le salage contribue significativement à la protection contre les attaques telles que les tables arc-en-ciel (rainbow tables) et limite les risques liés à l'utilisation de mots de passe communs.

Gestion des sessions

La gestion des cookies de session est essentielle en raison de la nature « stateless » du protocole HTTP.

1. Le client envoie une requête POST de connexion au serveur.
2. Le serveur répond au POST en incluant un COOKIE dans la réponse.
3. Le client envoie une requête GET, par exemple /message, en incluant le COOKIE pour identifier la session.

Il est crucial de sécuriser les cookies de session, en particulier pour éviter que le code JavaScript côté client y accède. Un moyen de renforcer cette sécurité est d'utiliser un indicateur (flag) approprié lors de la création du cookie « httpOnly ».

HTTPS

Le protocole HTTP permet le chiffrement des données entre le client et le serveur. La fonction de hachage est utilisée pour assurer l'intégrité, la confidentialité et l'authentification des données échangées.

En matière de sécurité, il est essentiel que tous les processus soient transparents et ouvert, à l'exception de la clé privé. Le secret réside dans la clé, et la protection ne doit pas dépendre du procédé lui-même, mais plutôt de la sécurité de la clé.

L'efficacité d'une protection repose sur la robustesse de la clé, car la seule limite du reverse engineering est le temps et les ressources financières disponibles. L'objectif du chiffrement est de rendre le message inintelligible pour des tiers non autorisés.

La réalisation d'un secret parfait est possible en ayant une clé de la même longueur que le message. Dans une stratégie de sécurité optimale, chaque message devrait être associé à une nouvelle clé non réutilisée, générée par un générateur de nombres aléatoires robuste.

En résumé, la sécurité des communications repose sur un chiffrement solide, une gestion appropriée des clés et l'utilisation de pratiques garantissant l'intégrité, la confidentialité et l'authentification des données échangées.

Cryptage symétrique

Le chiffrement symétrique est une technique de cryptage où une seule clé est utilisée à la fois pour chiffrer et déchiffrer les données. Les parties communicantes partagent cette clé secrète, ce qui permet un échange sécurisé d'informations confidentielles entre elles. Les algorithmes de chiffrement symétrique sont plus rapides que leurs homologues asymétriques, mais la gestion sécurisée des clés partagées est cruciale.

Le chiffrement symétrique implique que le client et le serveur partagent la même clé pour sécuriser les communications. L'Advanced Encryption Standard (AES) est la norme courante pour cette méthode.

Il est inefficace de chiffrer deux fois avec le même algorithme, mais le chiffrement triple, avec le même algorithme, peut être efficace. Le chiffrement symétrique est rapide.

Le vecteur d'initialisation est utilisé pour introduire de l'aléatoire dans le processus déterministe du chiffrement.

Le padding consiste à ajouter des octets de remplissage.

Le "ciphertext stealing" assure que le ciphertext a la même longueur que le message.

Un processus est déterministe lorsque la longueur en entrée est égale à la valeur de sortie.

Steam Cypher

Un Stream Cipher (chiffrement par flot) est un type de chiffrement symétrique utilisé pour chiffrer les données bit par bit ou octet par octet, en continu, plutôt que par blocs comme dans le chiffrement par bloc. Il fonctionne en générant un flux de bits pseudorandom (une séquence apparemment aléatoire de bits) appelé "keystream", qui est combiné avec les données d'entrée pour produire le texte chiffré.

Le stream cipher est beaucoup plus lent que le chiffrement symétrique classique.

Diffie-Hellman Key Agreement est un protocole qui permet à deux parties de convenir secrètement d'une clé commune via un canal non sécurisé.

Le Horton Principle souligne l'importance de faire attention aux collisions, où deux entrées différentes produisent le même résultat de hachage.

En ce qui concerne l'authentification, un HMAC (Hash-based Message Authentication Code) est utilisé, mais il peut poser des problèmes dans TLS (Transport Layer Security) lorsqu'il doit être transmis au client. Cela peut nécessiter une approche différente pour garantir l'authenticité des données.

Asymétrique

En cryptographie, le terme "asymétrique" fait référence à un système où deux clés distinctes mais mathématiquement liées sont utilisées : une clé publique et une clé privée. Ces deux clés sont utilisées dans des opérations différentes, telles que le chiffrement et la signature numérique.

Dans un système de cryptographie asymétrique, la clé publique peut être partagée ou distribuée librement, tandis que la clé privée doit être gardée secrète. Un message chiffré avec la clé publique ne peut être déchiffré qu'avec la clé privée correspondante, et inversement, un message signé avec la clé privée peut être vérifié avec la clé publique correspondante.

L'utilisation de clés asymétriques offre des avantages en matière de sécurité et d'échange de clés, par rapport aux systèmes de cryptographie symétrique où une seule clé est utilisée à la fois pour le chiffrement et le déchiffrement.

L'utilisation de la cryptographie asymétrique offre une sécurité accrue, mais au prix de temps de calcul plus lents et de clés plus grandes. Les clés elliptiques sont une alternative qui atténue ces inconvénients.

Dans un échange entre client et serveur, la clé publique est partagée pour permettre l'authentification. Lors du protocole Diffie-Hellman, les messages sont signés, mais un intercepteur peut encore intercepter et simuler avec la clé publique.

Cependant, il manque une autorité de confiance dans ce processus, ce qui peut compromettre la sécurité de l'échange de clés.

Autorité de certification

Le site www.swisscom.ch utilise un certificat SSL émis par SwissSign. La chaîne de certificats comprend le certificat intermédiaire SwissSign RSA TLS OV, qui est lié à la racine de confiance SwissSign RSA TLS Root, et finalement au certificat critique SwissSign Gold. Les certificats intermédiaires sont envoyés lors du "Hello" du serveur au client.

Le certificat SwissSign Gold, bien que critique et nécessitant des changements potentiels dans le système du client, est utilisé uniquement pour signer les certificats intermédiaires. Une particularité est que la clé privée du serveur n'est pas liée à la clé AES du poste client.

Des tests avec un notebook peuvent être effectués en lançant le site en mode HTTPS pour explorer la sécurité de la connexion.

2ème ½ journée

La session a débuté par un résumé de la formation précédente, abordant d'abord les cookies et l'usage de Firefox, notamment en lien avec le protocole TLS. Trois aspects clés du TLS ont été soulignés : l'authentification via un mécanisme symétrique (clés publique et privée), l'intégrité assurée par les fonctions de hachage, et la garantie de confidentialité.
L'importance du reverse proxy et son rôle dans la sécurisation des applications web, ainsi que l'utilisation de Flask pour le développement d'applications et la gestion des bases de données, ont été discutés.

Trois niveaux de certifications SSL/TLS ont été distingués : Extended Validation, Organization Validation Certificates (nécessitant des documents bancaires et une preuve de l'existence de l'entreprise) et Domain Validation Certificates (validés par e-mail, avec des exemples comme Let's Encrypt). Il a été noté que les certificats sont généralement stockés du côté des serveurs, avec une signature asymétrique pour une meilleure sécurité, tandis que le reste utilise la cryptographie symétrique pour une exécution plus rapide.
Le processus RSA a été examiné, incluant la génération de clés (publique et privée), l'encryptage et le décryptage. Un exemple de certificat RSA a été donné : **SF-Jean-Luc\lecture-notes-and-exercises-fall-2023-main\Topics\06_CryptographicFailures\Exercises\RSA4096\cert2.pem**.

Dans le contexte du protocole Diffie-Hellman, la difficulté de choisir une clé commune, notamment en raison de l'attaque Logjam, a été soulignée. L'utilisation d'OpenSSL pour générer les paramètres du protocole Diffie-Hellman a été recommandée.
Il a été mentionné qu'il est crucial d'avoir un contrôle sur la qualité des mots de passe, car contrairement aux fonctions de hachage standard, conçues pour un déchiffrement rapide, les mots de passe nécessitent une approche qui rend le déchiffrement difficile et lent. La protection des cookies contre l'accès via JavaScript a été mise en avant, ainsi que l'importance des security headers pour prévenir les attaques via des iframes.

Les problèmes liés au Cross-site request forgery et au Broken access control ont été expliqués. Un exemple typique est la capacité de lire un document via l'URL dans un intranet, même en l'absence d'autorisations adéquates.

La question de l'injection de template permettant la lecture de fichiers a été abordée. Enfin, l'approche incorrecte du HMAC (Key || message, Sha 256, Signature) et les risques liés à la création d'un HMAC personnalisé ont été discutés, avec une référence aux notes de cours : **SF-Jean-Luc\lecture-notes-and-exercises-fall-2023-main\Topics\06_CryptographicFailures\LectureNotes\Jupyter\length_extension_attack.ipynb**.