Les failles de sécurité

1 - L'injection SQL

La faille SQLi, abréviation de SQL Injection, soit injection SQL en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système et pouvant en compromettre la sécurité.

Les types d'injections SQL :

- La méthode ***Blind based*** permet de détourner la requête SQL en cours sur le système et d'injecter des morceaux qui vont retourner caractère par caractère ce que l'attaquant cherche à extraire de la base de données.
- la méthode ***Error based***, qui permet de détourner la requête SQL en cours sur le système et d'injecter des morceaux qui vont retourner champ par champ ce que l'on cherche à extraire de la base de données. Cette méthode profite d'une faiblesse des systèmes de base de données permettant de détourner un message d'erreur généré par le système de base de données et préalablement volontairement provoquée par l'injection SQL pour lui faire retourner une valeur précise récupérée en base de données ;
- la méthode ***Union based***, qui permet de détourner la requête SQL en cours sur le système et d'injecter des morceaux qui vont retourner un ensemble de données directement extraites de la base de données. Cette méthode profite de certaines méthodes afin de détourner entièrement le retour de la requête SQL d'origine afin de lui faire retourner en une seule requête un important volume de données, directement récupéré en base de données.
- la méthode ***Stacked queries***, la plus dangereuse de toutes. Profitant d'une erreur de configuration du serveur de base de données, cette méthode permet d'exécuter n'importe quelle requête SQL sur le système ciblé, ce qui ne se limite pas seulement à récupérer des données comme les 3 précédentes. En effet, quand ce type de requête n'est pas désactivé, il suffit d'injecter une

autre requête SQL, et elle sera exécutée sans problème, qu'elle aille chercher des données, ou en modifier directement dans la base de données.

Comment s'en protéger ?

- ***Echapper les caractères spéciaux contenus dans les chaines de caractères.*** Avec PHP, on peut le faire via les fonctions `htmlentities()` ou `htmlspecialchars`
- ***Utilisation des requêtes préparées.*** Dans ce cas, une compilation de la requête est réalisée avant d'y insérer les paramètres et de l'exécuter, ce qui empêche un éventuel code inséré dans les paramètres d'être interprété.
- ***Utilisation des RegExp pour le contrôle des saisis des données utilisateur.*** On valide si la donnée entrée par l'utilisateur est bien de la forme souhaitée, ou profiter de fonctions de transformation spécifiques au langage.
- ***Utiliser des comptes utilisateurs SQL à accès limité*** (en lecture-seule) quand cela est possible.
- ***Utiliser des requêtes SQL à trous*** (requêtes à trous envoyées au serveur SQL, serveur à qui l'on envoie par la suite les paramètres qui boucheront les trous), ainsi c'est le SGBD qui se charge d'échapper les caractères selon le type des paramètres.

2 - La faille XSS

Le ***cross-site scripting*** (abrégé ***XSS***) est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, provoquant ainsi des actions sur les navigateurs web visitant la page. Les possibilités des XSS sont très larges puisque l'attaquant peut utiliser tous les langages pris en charge par le navigateur (JavaScript, Java...) et de nouvelles possibilités sont régulièrement découvertes notamment avec l'arrivée de nouvelles technologies comme HTML5. Il est par exemple possible de rediriger vers un autre site pour de l'hameçonnage ou encore de voler la session en récupérant les cookies.

Le principe est d'injecter des données arbitraires dans un site web, par exemple en déposant un message dans un forum, ou par des paramètres d'URL. Si ces

données arrivent telles quelles dans la page web transmise au navigateur (par les paramètres d'URL, un message posté...) sans avoir été vérifiées, alors il existe une faille : on peut s'en servir pour faire exécuter du code malveillant en langage de script (du JavaScript le plus souvent) par le navigateur web qui consulte cette page.

Les Risques

- Redirection (parfois de manière transparente) de l'utilisateur (souvent dans un but d'hameçonnage)
- Vol d'informations, par exemple sessions et cookies.
- Actions sur le site faillible, à l'insu de la victime et sous son identité (envoi de messages, suppression de données...)
- Rendre la lecture d'une page difficile (boucle infinie d'alertes par exemple).

Les types de failles XSS

- ***XSS réfléchi/non permanent.*** C'est le plus commun. Il apparaît lorsque des données fournies par un client web sont utilisées telles quelles par les scripts du serveur pour produire une page de résultats. Si les données non vérifiées sont incluses dans la page de résultat sans encodage des entités HTML, elles pourront être utilisées pour injecter du code dans la page dynamique reçue par le navigateur client.

  Avec un peu d'ingénierie sociale, un attaquant peut convaincre un utilisateur de suivre une URL piégée qui injecte du code dans la page de résultat, ce qui donne à l'attaquant tout contrôle sur le contenu de cette page.

- ***XSS stocké/permanent.*** Ce type de vulnérabilité, aussi appelé faille permanente ou du second ordre permet des attaques puissantes. Elle se produit quand les données fournies par un utilisateur sont stockées sur un serveur (dans une base de données, des fichiers, ou autre), et ensuite ré-affichées sans que les caractères spéciaux HTML aient été encodés.

Un exemple classique est celui des forums, où les utilisateurs peuvent poster des textes formatés avec des balises HTML. Ces failles sont plus importantes que celles d'autres types, parce qu'un attaquant peut se contenter d'injecter un script une seule fois et atteindre un grand nombre de victimes sans recourir à l'ingénierie sociale.

Comment s'en protéger ?

- Retraiter systématiquement le code HTML produit par l'application avant l'envoi au navigateur ;
- Filtrer les variables affichées ou enregistrées avec des caractères '<' et '>' (en CGI comme en PHP). De façon plus générale, donner des noms préfixés (avec par exemple le préfixe "us" pour user string) aux variables contenant des chaines venant de l'extérieur pour les distinguer des autres, et ne jamais utiliser aucune des valeurs correspondantes dans une chaîne exécutable (en particulier une chaîne SQL, qui peut aussi être ciblée par une injection SQL d'autant plus dangereuse) sans filtrage préalable.
- En PHP :
  - utiliser la fonction `htmlspecialchars()` qui filtre les '<' et '>' (ci-dessus) ;
  - utiliser la fonction `htmlentities()` qui est identique à `htmlspecialchars()` sauf qu'elle filtre tous les caractères équivalents au codage HTML ou JavaScript.

3 - La faille CSRF

le ***cross-site request forgery***, abrégé ***CSRF*** (parfois prononcé sea-surf en anglais) ou ***XSRF***, est un type de vulnérabilité des services d'authentification web.

L'objet de cette attaque est de transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en

avoir conscience et en utilisant ses propres droits. L'utilisateur devient donc complice d'une attaque sans même s'en rendre compte. L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

Les sites sensibles au CSRF sont ceux qui acceptent les actions sur le simple fait de l'authentification à un instant donné de l'utilisateur et non sur une autorisation explicite de l'utilisateur pour une action donnée.

Comment s'en protéger ?

- ***Demander des confirmations*** à l'utilisateur pour les actions critiques, au risque d'alourdir l'enchaînement des formulaires.
- ***Demander une confirmation de l'ancien mot de passe*** à l'utilisateur pour changer celui-ci ou changer l'adresse mail du compte.
- ***Utiliser des jetons de validité*** (ou) dans les formulaires est basé sur la création du token via le chiffrement d'un identifiant utilisateur, un nonce et un horodatage. Le serveur doit vérifier la correspondance du jeton envoyé en recalculant cette valeur et en la comparant avec celle reçue.
- ***Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions*** : cette technique va naturellement éliminer des attaques simples basées sur les images, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.
- ***Effectuer une vérification du référent dans les pages sensibles*** : connaître la provenance du client permet de sécuriser ce genre d'attaques. Ceci consiste à bloquer la requête du client si la valeur de son référent est différente de la page d'où il doit théoriquement provenir.