Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій

Кафедра радіоелектронних і комп'ютерних систем

Реферат

з курсу «Технології захисту інформації»
на тему «Database Injection»

Виконала:
студентка групи ФеІ-31
Зьола О.П.

Викладач:
доц. Монастирський Л. С.

Львів

ЗМІСТ

ВСТУП

SQL ін'єкція — один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду. Ці «заражені» запити керують сервером баз даних веб-додатку. Зловмисники можуть використовувати вразливості SQL Injection для обходу заходів безпеки: проходити автентифікацію та авторизацію веб-сторінки чи веб-програми, отримувати вміст всієї бази даних SQL, та додавати, змінювати чи видаляти записи у базі даних.

Така атака може вплинути на будь-який веб-сайт або веб-додаток, який використовує базу даних SQL, такі як MySQL, Oracle, SQL Server тощо. Злочинці можуть використовувати його для отримання несанкціонованого доступу до конфіденційних даних: інформації про клієнтів, персональних даних, комерційної таємниці, інтелектуальної власності тощо.

Атаки SQL Injection є однією з найстаріших, найпоширеніших і найнебезпечніших вразливостей веб-додатків. Організація OWASP (Open Web Application Security Project) перераховує ін'єкції в документі OWASP Top 10 2017 як загрозу номер один для безпеки веб-додатків. Розробники застосунків, що працюють з базами даних, повинен знати про таку уразливість і вживати заходів протидії впровадженню SQL.

1. Як виконується SQL Injection атака

Щоб здійснити атаку SQL Injection, зловмисник повинен спершу знайти вразливі поля введення користувача на веб-сторінці чи веб-програмі. Веб-сторінка або веб-додаток, який має вразливість SQL Injection, використовує такий ввід користувача безпосередньо у запиті SQL. Зловмисник може створювати вхідний контент. Такий вміст часто називають ***шкідливим навантаженням*** і він є ключовою частиною нападу. Після того, як зловмисник надсилає цей вміст, у базі даних виконуються введені «заражені» команди SQL.

Розглянемо приклад використання шкідливого навантаження. Припустимо, серверне ПЗ, отримавши вхідний параметр id, використовує його для створення SQL-запиту. Розглянемо такий PHP-скрипт:

```
...
$id = $_REQUEST['id'];
$res = mysql_query("SELECT * FROM news WHERE id_news = $id");
...
```

Якщо на сервер переданий параметр id, що дорівнює 5, то виконається такий SQL-запит:

```
SELECT * FROM news WHERE id_news = 5
```

Але якщо зловмисник передасть як параметр id рядок -1 OR 1=1, то виконається запит:

```
SELECT * FROM news WHERE id_news =-1 OR 1=1
```

Таким чином, зміна вхідних параметрів шляхом додавання в них конструкцій мови SQL викликає зміну в логіці виконання SQL-запиту (в цьому прикладі замість новини із заданим ідентифікатором будуть вибрані всі наявні в базі новини, оскільки вираз 1=1 завжди істинний).

2. Підходи до створення атаки

Існують наступні підходи до створення атаки: впровадження в рядкові параметри, використання UNION, використання UNION + group_concat(), екранування хвоста запиту, розщеплення SQL-запиту та деякі інші.

2.1. Впровадження в рядкові параметри

Припустимо, серверне ПЗ, отримавши запит на пошук даних у новинах параметр search_text, використовує його в наступному SQL-запиті (тут параметри екрануються лапками) :

```
...
$search_text = $_REQUEST['search_text'];
$res = mysql_query("SELECT id_news, news_date, news_caption, news_text,
news_id_author
                    FROM news WHERE news_caption LIKE('%$search_text%') ");
```

Зробивши запит, ми отримаємо виконання такого SQL-запиту:

```
SELECT id_news, news_date, news_caption, news_text, news_id_author FROM news
 WHERE news_caption LIKE('%Test%')
```

Але, запровадивши в параметр search_text символ лапки (який використовується в запиті), ми можемо кардинально змінити поведінку SQL-запиту. Наприклад, передавши як параметр search_text значення **')** +and+ (news_id_author='1, ми змусимо виконати запит:

```
SELECT id_news, news_date, news_caption, news_text, news_id_author FROM news
 WHERE news_caption LIKE('%') and (news_id_author='1%')
```

2.2. Використання UNION

Мова SQL дозволяє об'єднувати результати декількох запитів за допомогою оператора UNION. Це надає зловмисникові можливість отримати несанкціонований доступ до даних.

Розглянемо скрипт відображення новини:

```
$res = mysql_query("SELECT id_news, header, body, author FROM news WHERE id_news
= ". $_REQUEST['id']);
```

Якщо зловмисник передасть як параметр id конструкцію -1 UNION SELECT 1,username, password,1 FROM admin, це викличе виконання SQL-запиту

```
SELECT id_news, header, body, author FROM news WHERE id_news =-1 UNION SELECT
1,username,password,1 FROM admin
```

Оскільки новини з ідентифікатором -1 завідомо не існує, з таблиці news не буде вибрано жодного запису, проте в результат потраплять записи, несанкціоновано відібрані з таблиці admin внаслідок ін'єкції SQL.

2.3. Використання UNION + group_concat()

У деяких випадках хакер може провести атаку, але не може бачити більше однієї колонки. У разі MySQL зломщик може скористатися функцією:

```
group_concat(col, symbol, col)
```

яка об'єднує кілька колонок в одну. Наприклад, для прикладу цього вище виклик функції буде таким:

```
-1 UNION SELECT group_concat(username, 0x3a, password) FROM admin
```

2.4. Екранування хвоста запиту

Скрипт

```
$res = mysql_query("SELECT author FROM news WHERE id=". $_REQUEST['id']." AND
author LIKE ('a%') ");
```

відображає ім'я автора новини згідно ідентифікатора id лише за умови, що ім'я починається з літери a, і впровадження коду з використанням оператора UNION складне.

У таких випадках, зловмисниками використовується метод екранування частини запиту за допомогою символів коментаря (/*або--в залежності від типу СКБД).

У цьому прикладі, зловмисник може передати в скрипт параметр id зі значенням-1 UNION SELECT password FROM admin/*, виконавши таким чином запит

```
SELECT author FROM news WHERE id=-1 UNION SELECT password FROM admin/* AND
author LIKE ('a%')
```

в якому частина запиту (*AND author LIKE ('a%')*) позначена як коментар і не впливає на виконання.

2.5. Розщеплення SQL-запиту

Для розділення команд в мові SQL використовується символ **;** (*крапка з комою*), впроваджуючи цей символ до запиту, зловмисник отримує можливість виконати декілька команд в одному запиті, однак не всі діалекти SQL підтримують таку можливість.

Наприклад, якщо в параметри скрипта

```
$id = $_REQUEST['id'];
$res = mysql_query("SELECT * FROM news WHERE id_news = $id");
```

зловмисником передається конструкція, що містить крапку з комою, наприклад «12;INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');» то в одному запиті будуть виконані 2 команди

```
SELECT * FROM news WHERE id_news = 12;
INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');
```

і в таблицю admin буде несанкціоновано доданий запис HaCkEr.

3. Як запобігти ін'єкції SQL

Конкретні методи запобігання залежать від підтипу атаки SQLi, від двигуна бази даних SQL та мови програмування. Однак існують певні загальні стратегічні принципи, яких слід дотримуватися, щоб захистити веб-додаток.

- Щоб захистити веб-додаток, усі, хто бере участь у створенні веб-програми, повинні знати про ризики, пов’язані з ін'єкціями SQL. Потрібно забезпечити відповідну підготовку з безпеки для всіх своїх розробників, персоналу з забезпечення якості, DevOps та SysAdmins.

- Потрібно ставитися до всіх користувацьких даних як до ненадійних. Будь-який «ввід» користувача, який використовується у запиті SQL, вводить ризик ін'єкції SQL. Поводьтеся із введеними даними аутентифікованих та/або внутрішніх користувачів так само, як ви ставитесь до публічних даних.

- Не фільтруйте введення користувачів на основі чорних списків. Розумний нападник майже завжди знайде спосіб обійти ваш чорний список. Якщо можливо, перевірте та фільтруйте введення користувачів, використовуючи лише суворі списки.

- Старіші технології веб-розробки не мають захисту SQLi. Використовуйте останню версію середовища розробки, мови та новітні технології, пов’язані з цим середовищем/мовою. Наприклад, в PHP використовуйте PDO замість MySQLi. Не намагайтеся створити захист SQLi з нуля. Більшість сучасних технологій розробки можуть запропонувати вам механізми захисту від SQLi. Використовуйте такі механізми замість того, щоб намагатися винаходити колесо. Наприклад, використовувати параметризовані запити або збережені процедури.

- SQL ін'єкції можуть бути введені вашими розробниками або через зовнішні бібліотеки/модулі/програмне забезпечення. Ви повинні регулярно сканувати свої веб-програми за допомогою сканера вразливості веб.

4. Методи запобігання SQLi

Єдиний вірний спосіб запобігти атакам SQL Injection - це перевірка введення та параметризовані запити, включаючи підготовлені оператори. Код програми ніколи не повинен використовувати введені користувачем дані безпосередньо. Розробник повинен санітувати весь вхід, а не лише дані веб-форм, наприклад форми входу. Вони повинні видалити потенційні елементи шкідливого коду, такі як одиничні лапки. Також непогано вимкнути видимість помилок бази даних на виробничих сайтах. Помилки бази даних можна використовувати з інжекцією SQL для отримання інформації про вашу базу даних.

4.1. Фільтрація рядкових параметрів

Припустимо, що код, який генерує запит (на мові програмування Паскаль), виглядає так:

```
statement:= 'SELECT * FROM users WHERE name = "' + userName + '";';
```

Щоб впровадження коду було неможливо, для деяких СКБД, в тому числі, для MySQL, потрібно брати в лапки всі рядкові параметри. У самому параметрі замінюють лапки на « \" », апостроф на « \' » , зворотну косу риску на \\ (це називається «екранувати спецсимволи»).

Це можна робити таким кодом:

```
statement:= 'SELECT * FROM users WHERE name = ' + QuoteParam(userName) + ';';
function QuoteParam(s: string) : string;
{ на вході — рядок; на виході — рядок в лапках та із заміненими спецсимволами }
var
 i: integer;
 Dest: string;
begin
 Dest:= '"';
 for i:=1 to length(s) do
   case s[i] of
     '''': Dest:= Dest +'\''';
     '"': Dest:= Dest + '\"';
     '\': Dest:= Dest + '\\';
   else Dest:= Dest + s[i];
   end;
 QuoteParam:= Dest + '"';
end;
```

для PHP фільтрація може бути такою:

```
<?php
$query = "SELECT * FROM users WHERE
user='".mysql_real_escape_string($user)."';";
?>
```

4.2. Фільтрація цілочислових параметрів

Візьмемо інший запит:

```
statement:= 'SELECT * FROM users WHERE id = ' + id + ';';
```

У цьому випадку поле `id` має числовий тип, і його найчастіше не беруть в лапки. У такому випадку допомагає перевірка – якщо змінна `id` не є числом, запит взагалі не повинен виконуватися.

Наприклад, на Delphi для протидії таким ін'єкціям допомагає код:

```
id_int:= StrToInt(id);
statement:= 'SELECT * FROM users WHERE id = ' + IntToStr(id_int) + ';';
```

у випадку помилки функція `StrToInt` викличе виняток `EConvertError`, і в його обробнику можна буде вивести повідомлення про помилку. Подвійне перетворення забезпечує коректну реакцію на числа у форматі `$132AB` (шістнадцяткова система числення).

Для PHP цей метод буде виглядати так:

```
$query = 'SELECT * FROM users WHERE id = '. intval($id);
```

4.3. Усікання вхідних параметрів

Для внесення змін в логіку виконання SQL-запиту потрібно впровадження достатньо довгих рядків. Так, мінімальна довжина такого рядка у наведених вище прикладах становить 8 символів («1 OR 1=1»). Якщо максимальна довжина коректного значення параметра невелика, то одним з методів захисту може бути максимальне усікання значень вхідних параметрів.

Наприклад, якщо відомо, що поле `id` у вищенаведених прикладах може приймати значення не більше 9999, можна «відрізати зайві» символи, залишивши не більше чотирьох:

```
statement:= 'SELECT * FROM users WHERE id = ' + LeftStr(id, 4) + ';';
```

4.4. Використання параметризованих запитів

Багато серверів баз даних підтримують можливість відправки параметризованих запитів (підготовлені вирази). При цьому параметри зовнішнього походження відправляються на сервер окремо від самого запиту або автоматично екрануються клієнтською бібліотекою. Для цього використовують

- на Delphi — властивість `TQuery.Params`;

Наприклад

```
var
 sql, param: string;
begin
 sql:= 'select:text as value from dual';
 param:= 'alpha';
 Query1.Sql.Text:= sql;
 Query1.ParamByName('text').AsString:= param;
 Query1.Open;
 ShowMessage(Query1['value']);
end;
```

- на Perl — через `DBI::quote` або `DBI::prepare`;
- на Java — через клас `PreparedStatement`;
- на C# — властивість `SqlCommand.Parameters`;
- на PHP — MySQLi (при роботі з MySQL), `PDO`.

ВИСНОВКИ

SQL - мова запитів, призначена для управління даними, що зберігаються у реляційних базах даних. Використовується для доступу, зміни та видалення даних. Багато веб-додатків та веб-сайтів зберігають усі дані у базах даних SQL. У деяких випадках команди SQL також використовуються для запуску команд операційної системи. Тому успішна атака SQL Injection може мати дуже серйозні наслідки.

SQL Injection можна використовувати різними способами, щоб викликати серйозні проблеми. За допомогою використання SQL Injection зловмисник може обійти автентифікацію, отримати доступ, змінити та видалити дані в базі даних. У деяких випадках SQL Injection навіть може використовуватися для виконання команд в операційній системі, що потенційно дозволяє зловмиснику перейти до більш згубних атак всередині мережі, що знаходиться за брандмауером.

Отже, в основному страждають:

- Конфіденційність: Оскільки в базах даних SQL зазвичай зберігаються конфіденційні дані, втрата конфіденційності є частою проблемою уразливості до SQL Injection.
- Автентифікація: Якщо для перевірки імен користувачів та паролів використовуються погані команди SQL, то стає можливо, підключитися до системи як інший користувач, який не знає паролю.
- Авторизація: Якщо інформація про авторизацію зберігається в базі даних SQL, можливо змінити цю інформацію через успішну експлуатацію вразливості до SQL Injection.
- Цілісність: Так само як це можливо для читання конфіденційної інформації, також можна внести зміни або навіть видалити цю інформацію при атаці SQL Injection.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. The OWASP Foundation. “SQL Injection” –2018. – [Cited 2020, 15 April]. – Available from : https://owasp.org/www-community/attacks/SQL_Injection

2. Refsnes Data. “SQL Injection” –2017 – [Cited 2020, 15 April]. – Available from : https://www.w3schools.com/sql/sql_injection.asp

3. “What is SQL Injection (SQLi) and How to Prevent It”–2019. – [Cited 2020, 15 April]. – Available from : https://www.acunetix.com/websitesecurity/sql-injection/

4. Friedl Steve “SQL Injection Attacks by Example”–2019. – [Cited 2020, 15 April]. – Available from : http://unixwiz.net/techtips/sql-injection.html