МЕТОДОЛОГІЯ ЗЛОМІВ ВЕБСАЙТІВ Й ДОДАТКІВ ЗА ДОПОМОГОЮ SQL-INJECTION ТА ПРОТИДІЯ НИМ

Онищенко Юрій Миколайович
кандидат наук з державного управління, доцент, заступник декана факультету з навчально-методичної роботи факультету № 4
Харківський національний університет внутрішніх справ

Чукалов Кирило Едуардович
курсант групи Ф4-102 факультету № 4
Харківський національний університет внутрішніх справ

Гельдт Станіслав Володимирович
курсант групи Ф4-302 факультету № 4
Харківський національний університет внутрішніх справ

Каланча Андрій Андрійович
курсант групи Ф4-202 факультету № 4
Харківський національний університет внутрішніх справ

SQL ін’єкція – один з поширених способів злому сайтів та програм, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду.

Впровадження SQL, залежно від типу СКБД та умов впровадження, може дати можливість атакуючому виконати довільний запит до бази даних (наприклад, прочитати вміст будь-яких таблиць, видалити, змінити або додати дані), отримати можливість читання та/або запису локальних файлів та виконання довільних команд на сервері.

Атака типу впровадження SQL може бути можлива за некоректної обробки вхідних даних, що використовуються в SQL-запитах. Розробники додатків, що працюють з базами даних, повинні знати про таку вразливість і вживати заходів протидії впровадженню SQL ін’єкцій.

Принцип атаки.

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 (наприклад так: http://example.org/script.php?id=5), то виконується такий SQL-запит:

SELECT * *FROM news WHERE id_news = 5*

Але, якщо зловмисник передасть як параметр id рядок -1 OR 1=1 (наприклад, так: http://example.org/script.php?id=-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 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).

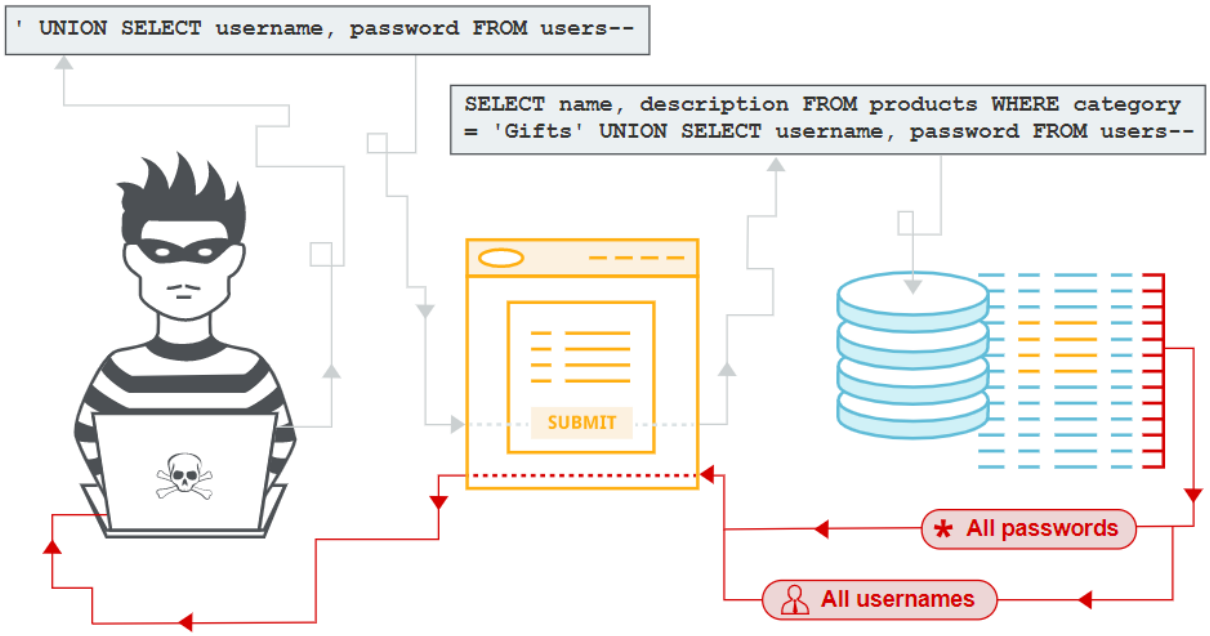

Схема роботи SQL – injection.

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 (ідентифікатор новини, яку необхідно відобразити, передається в параметрі id):

$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.

Методика атак типу впровадження SQL-коду.

На цьому етапі зловмисник вивчає поведінку скриптів сервера при маніпуляції вхідними параметрами з метою виявлення їх аномальної поведінки. Маніпуляція відбувається всіма можливими параметрами:

- *даними, переданими через методи POST і GET;*
- *значеннями [HTTP-Cookie];*
- *HTTP_REFERER (для скриптів);*
- *AUTH_USER та AUTH_PASSWORD (при використанні автентифікації).*

Як правило, маніпуляція зводиться до підстановки в параметри символу одинарної (рідше подвійної або зворотної) лапки.

Аномальною поведінкою вважається будь-яка поведінка, при якій сторінки, одержувані до і після підстановки лапок, розрізняються (і при цьому немає повідомлення про неправильний форматі параметрів).

Найчастіші приклади аномальної поведінки:

- виводиться повідомлення про різні помилки;
- при запиті даних (наприклад, новини або списку продукції) запитувані дані не виводяться взагалі, хоча сторінка відображається і т.д.

Слід враховувати, що відомі випадки, коли повідомлення про помилки, в силу специфіки розмітки сторінки, не видно в браузері, хоча і присутні в її HTML-коді.

Захист від атак типу впровадження SQL-коду.

Для захисту від цього типу атак необхідно фільтрувати вхідні параметри, значення яких будуть використані для SQL-запиту.

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 «екрануванням спецсимволів»).

Фільтрація чисельних параметрів.

У багатьох 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.

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.

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;

на Perl – DBI::quote або DBI::prepare;

на Java – клас PreparedStatement;

на C# – властивість SqlCommand.Parameters;

на PHP – MySQLi (при роботі з MySQL), PDO.

T-SQL або Transact SQL – це мова запитів, специфічна для продукту Microsoft SQL Server. Це може бути корисним для таких операцій, як отримання даних з одного рядка, вставка нових рядків та отримання кількох рядків. Це процедурна мова SQL Server.

Транзакція – одна з фундаментальних понять всіх СУБД. Суть угоди полягає у поєднанні кількох кроків на одну операцію. Внутрішні проміжні стани між кроками не видно іншим конкуруючим транзакціям, і якщо під час виконання транзакції виникає помилка, що перешкоджає завершенню транзакції, до бази даних не вносяться жодних змін.

Припустимо, є база даних, що містить баланси для кількох клієнтів та загальні баланси депозитів для філій. Наприклад, ми хочемо перевести 100,00 $ продажів від клієнта Аліси клієнту Бобу, та за для цієї операції ми робимо відправлення від рахунка Аліси до рахунку Боба, і як результат, отримуємо вже нову графу даних у вигляді вже переведених грошей. Найпростіший набір інструкцій, що виконує цю операцію, може мати такий вигляд синтаксичних конструкцій оператора оновлення транзакцій в SQL та виглядає наступним чином:

sql
Copy code
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;

де:

table_name - назва таблиці, яку необхідно оновити;

column1, column2, ... - назви стовпців, які необхідно оновити;

value1, value2, ... - нові значення, які будуть записані в відповідні стовпці;

condition - умова, яка визначає, які рядки таблиці необхідно оновити.

Приклад використання оператора оновлення транзакцій в SQL:

sql
Copy code
UPDATE employees
SET salary = 50000

```
WHERE department = 'Sales';
```

У цьому прикладі ми оновлюємо значення стовпця "salary" в таблиці "employees" для всіх рядків, які належать до відділу "Sales" і встановлюємо нове значення "50000".

Наші банкіри захочуть зробити всі ці оновлення одразу чи взагалі нічого. Це пов'язано з тим, що внаслідок системної помилки Боб отримує 100 доларів США, які не віднімаються від суми Аліси. Або може статися так, що Аліса відніме цю суму, а Боб її не отримає. Нам потрібна гарантія того, що якщо щось піде не так під час процесу оновлення та виставлення рахунків, жодних змін не буде внесено. Цю гарантію можна отримати, групуючи інструкції по оновленню транзакції. Транзакція є атомарною дією по відношенню до інших транзакцій, і або вона завершується повністю успішно, або не виконується жодна з дій, що становлять транзакцію.

Ми також хочемо переконатися, що повністю завершена та підтверджена СУБД транзакція справді зберігається і не може бути втрачена, навіть якщо після її виконання відбудеться збій системи. Наприклад, якщо ми зберігаємо кеш переказів клієнта Боба, ми не хочемо, щоб гроші клієнта Боба були втрачені внаслідок системного збою, який може статися, скажімо, коли Боб іде з банку. Традиційні СУБД гарантують, що всі оновлення, зроблені в рамках однієї транзакції, записуються в надійне сховище (тобто диск) до того, як СУБД повідомить про завершення транзакції.

SQL-ін'єкції – це дуже поширена проблема, але за іронією долі, їх також легко запобігти. SQL-ін'єкції так поширені, оскільки існує дуже багато вразливих місць, і в разі успішної ін'єкції, хакер може отримати хорошу винагороду (наприклад, повний доступ до даних у базі):

• Не використовувати динамічні запити до бази.

• Не використовувати дані користувача в запитах.

Все начебто просто, але це теорії, на практиці ж відмовитися від динамічних запитів неможливо, як і виключити введення даних користувача. Але це означає, що уникнути ін'єкцій неможливо. Є деякі прийоми та технічні можливості мов програмування, які допоможуть запобігти SQL-ін'єкції.

Що можна зробити для запобігання SQL-ін'єкціям?

Хоча рішення багато в чому залежить від конкретної мови програмування, все ж таки загальні принципи запобігання SQL-ін'єкцій схожі. Ось кілька прикладів, як це можна зробити:

✓ **Використовувати динамічні запити лише у разі потреби.**

Динамічний запит майже завжди можна замінити підготовленими виразами (prepared statements), параметризованими запитами або процедурами, що зберігаються. Наприклад, замість динамічного SQL, в Java можна використовувати PreparedStatement() з прив'язаними параметрами, в .NET можна використовувати параметризовані запити, такі як SqlCommand() або OleDbCommand() з прив'язаними параметрами, а в PHP можна використовувати PDO зі строгою типізацією параметризованих запитів (використовуючи

bindParam()). На додаток до підготовлених виразів (prepared statements), можна використовувати процедури, що зберігаються. На відміну від підготовлених виразів (prepared statements), процедури, що зберігаються, зберігаються в базі, але в обох випадках спочатку визначається SQL-запит, і в нього передаються параметри.

✓ **Перевірка введених даних у запитах.**

Перевірка введення даних менш ефективна, ніж параметризовані запити і процедури, що зберігаються, але якщо немає можливості використовувати параметризовані запити і процедури, що зберігаються, то вже краще все ж перевіряти введені дані – це краще, ніж нічого. Точний синтаксис використання перевірки введених даних залежить від бази даних.

✓ **Не сподіватися на чарівні лапки (Magic Quotes).**

Увімкнення параметра magic_quotes_gpc може запобігти деяким (але не всім) SQL-ін'єкціям. Magic quotes ніяк не останній захист, і що ще гірше, іноді вони вимкнені, про що користувач не знає або не має можливості їх ввімкнути. Саме тому необхідно використовувати код, який екрануватиме лапки.

Підсумовуючи, варто зазначити, що SQL injection (SQLi) – це метод атаки на вебдодатки, який полягає в тому, що зловмисник використовує недостатньо захищену вхідну форму на вебсайті, щоб внести шкідливий SQL-код в запит до бази даних. Це може призвести до витоку конфіденційної інформації, видалення або зміни даних у базі даних, або навіть до повного контролю над вебсайтом.

Щоб запобігти SQL-ін'єкціям, важливо користуватися параметризованими запитами, перевіряти вхідні дані на валідність та екранувати спеціальні символи. Також важливо оновлювати програмне забезпечення та вебсервери, щоб запобігти використанню відомих вразливостей.

Список літератури:

1. Вікіпедія. Штучний інтелект. URL: https://uk.wikipedia.org/wiki/Штучний_інтелект (дата звернення: 03.02.2023).
2. Balbix. Using Artificial Intelligence in Cybersecurity. URL: https://www.balbix.com/insights/artificial-intelligence-in-cybersecurity/ (дата звернення: 03.02.2023).
3. Ideamotive. 100 Artificial Intelligence Statistics For 2022: The Ultimate List. URL: https://www.ideamotive.co/blog/the-ultimate-list-of-artificial-intelligence-statistics (дата звернення: 03.02.2023).
4. IBM. IBM Security QRadar SEIM. URL: https://www.ibm.com/products/qradar-siem/addons#3071036 (дата звернення: 03.02.2023).
5. Balbix. Balbix Security Cloud. URL: https://www.balbix.com/product-overview/ (дата звернення: 03.02.2023).