ЛР8. SQL инъекции

SQL инъекция — один из распространённых способов взлома сайтов и программ, работающих с базами данных, основанный на внедрении в запрос произвольного SQL-кода. Внедрение SQL-кода может дать возможность злоумышленнику выполнить произвольный запрос к базе данных (например, прочитать или изменить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

В зависимости от методологии проведения атак можно выделить два типа SQL-инъекций:

- *Классические SQL-инъекции* – атаки в условиях, когда приложение получает от сервера ответ, содержащий результат запроса, или сообщение об ошибке, содержащее интересующие данные.
- *Слепые SQL-инъекции* – атаки в условиях, когда ответ сервера не содержит никакой релевантной информации, но может быть интерпретирован как логическое значение.

В зависимости от того, содержится ли данные в ответе сервера в явном виде либо косвенно – в составе сообщения об ошибке, классические SQL-инъекции можно разделить на три типа:

- SQL-инъекции на основе объединения;
- SQL-инъекции на основе последовательных запросов;
- SQL-инъекции на основе возвращаемых ошибок.

Проведение атаки с использованием классической техники эксплуатации SQL инъекций происходит с использованием оператора `UNION` или с использованием разделения SQL запросов (точка с запятой). Но не всегда уязвимость типа «SQL инъекция» возможно эксплуатировать подобным способом. В таких случаях прибегают к техникам эксплуатации уязвимости «слепым» методом.

Слепая SQL инъекция появляется в том случае, когда уязвимый запрос является некоторой логикой работы приложения, но не позволяет вывести какие-либо данные в возвращаемую страницу Web приложением. Атаки этого вида можно разделить на три подкласса:

- SQL-инъекции на основе содержимого;
- SQL-инъекции на основе возвращаемых ошибок;
- SQL-инъекции на основе времени исполнения запроса.

Аналогично классической техники эксплуатации подобных уязвимостей, слепая SQL инъекция позволяет записывать и читать файлы, получать данные из таблицы, но только чтение в данном случае осуществляется посимвольно. Классическая техника эксплуатации подобных уязвимостей (*SQL-инъекция на основе содержимого*) основывается на использовании логических выражений `true/false`. Если выражение истинно, то Web приложение вернет одно содержимое, а если выражение является ложным, то другое. Полагаясь на различия вывода при истинных и ложных конструкциях в запросе, становится возможным осуществлять посимвольный перебор каких-либо данных в таблице или в файле.

SQL-инъекция на основе возвращаемых ошибок (*Error-based blind SQL Injection*) – это самая быстрая техники эксплуатации слепых SQL-инъекций. Суть данной техники заключается в том, что различные СУБД при определенных некорректных SQL-выражениях могут помещать в сообщение об ошибке различные запрашиваемые данные (например, версию базы данных). Данная техника может использоваться в случае, когда любая ошибка обработки SQL-выражений, осуществляемая в СУБД, возвращается обратно уязвимым приложением.

Бывают такие случаи, когда помимо подавления всех уведомлений об ошибках в возвращаемой странице со стороны приложения уязвимый к инъекции SQL-запрос используется исключительно для каких-то внутренних целей и результаты выполнения запроса никак не влияют на возвращаемую страницу. Например, это может быть ведение некоторого лога посещений, различного рода внутренние оптимизации и пр. Подобные SQL инъекции относятся к третьей группе – это *Double blind (Time-based) SQL Injection* (SQL-инъекции на основе времени исполнения запроса). Эксплуатация Double Blind SQL Injection осуществляется только с использованием временных задержек при выполнении SQL-запроса, т.е. если SQL-запрос выполняется мгновенно, то это `false`, а если SQL-запрос выполнился с задержкой в N-секунд, то это `true`. В указанной технике возможно только посимвольное чтение данных.

Примеры осуществления атак.

1. Классический пример

Пусть запрос к СУБД формируется следующим образом:

```
SELECT * FROM clients WHERE client='[CLIENT]' AND password='[PASSWORD]';
```

где `[CLIENT]`, `[PASSWORD]` – данные, вводимые пользователем. Если пользователем будет введена строка «`' or 1=1#`» в поле ввода переменной `[CLIENT]`, то запрос примет следующий вид:

```
SELECT * FROM clients WHERE client='' or 1=1 #' AND password='[PASSWORD]';
```

В результате значение условия после слова `WHERE` будет всегда истинным, и данный запрос вернет все данные таблицы `clients`.

Используя выражение `UNION`, можно выполнить любой запрос БД. Необходимо лишь определить количество столбцов, возвращаемое исходным запросом. Это можно сделать простым перебором:

```
[CLIENT] = ' or 1=1 UNION SELECT 1,2,3,4,…#
```

Определив количество возвращаемых столбцов, можно вывести, например, название текущей БД и версию MySQL:

```
[CLIENT] = ' or 1=1 UNION SELECT database(),@@version,3#
```

2. Error-based blind SQL Injection

В том случае, если результат выполнения запроса не выводится пользователю, но есть возможность получения сообщений об ошибках из СУБД, применяются описанные выше слепые инъекции на основе ошибок. Например, при выполнении запроса

```
SELECT 1 AND ExtractValue(1,concat(0x5C,(select database())));
```

будет получено следующее сообщение об ошибке:

```
XPATH syntax error: '\sakila'.
```

Таким образом, используя сообщение об ошибке, можно получить любые данные из СУБД. *Максимальная длина сообщения об ошибке – 32 символа.*

3. Double blind SQL Injection

В данном случае атакующему не доступны данные об ошибках СУБД. В этом случае информация извлекается из БД с помощью задержек. Рассмотрим пример:

```
SELECT * FROM products WHERE id=1-IF(MID(VERSION(),1,1) = '5', SLEEP(15), 0).
```

В том случае, если первый символ, возвращаемый функцией `VERSION()`, равен «5», ответ на запрос придет с 15 секундной задержкой. В противном случае задержки не будет. Таким образом, в случае Double Blind SQL Injection любая информация из БД может быть получена путем перебора вариантов.

Дополнительная информация.

1. Экранирование символов

Экранирование символов — замена в тексте управляющих символов на соответствующие текстовые подстановки. В MySQL символ «\» используется как экранирующий. Например, в запросе

```
SELECT * FROM users WHERE surname = 'O\'Shea';
```

символ «\» используется для того, чтобы указать что символ «'», содержащий в строке «O'Shea» не является управляющим, а является частью текста (фамилии).

Экранирующие символы могут быть применимы при выполнении SQL инъекций в том случае, если, например, перед передачей запроса СУБД осуществляется некоторая обработка вводимых пользователями данных.

2. База данных INFORMATION_SCHEMA

`INFORMATION_SCHEMA` - это база данных, хранящаяся в каждом экземпляре MySQL, в которой хранится информация обо всех других базах данных, которые содержит сервер MySQL. База данных `INFORMATION_SCHEMA` содержит несколько таблиц только для чтения. Они на самом деле являются представлениями, а не обычными таблицами, поэтому с ними не связано никаких файлов, и невозможно установить для них триггеры. Кроме того, нет каталога базы данных с таким именем.

Таким образом, используя базу данных `INFORMATION_SCHEMA` и хранящиеся в ней представления `schemata`, `tables`, `columns`, злоумышленник может определить структуру всех баз данных, хранящихся на сервере. Например:

- Получить список всех БД на сервере:

```
SELECT schema_name FROM information_schema.schemata;
```

- Получить список всех таблиц для заданной БД:

```
SELECT table_schema, table_name FROM information_schema.tables WHERE table_schema= 'test';
```

- Получить названия всех столбцов заданной таблицы:

```
SELECT column_name FROM information_schema.columns WHERE table_schema = 'test' AND table_name='new_table'
```

Для объединения возвращаемых данных в одну строку можно воспользоваться функцией `GROUP_CONCAT`:

```
SELECT GROUP_CONCAT(SCHEMA_NAME SEPARATOR ', ') FROM information_schema.schemata;
```

Результат выполнения запроса:

```
mysql, information_schema, performance_schema, sys, sakila, world, testdb, discogs
```

3. Мультибайтовые кодировки

Для защиты от инъекций вводимые пользователем данные могут быть отфильтрованы перед выполнением запроса путем экранирования управляющих символов. Однако, в определенных ситуациях злоумышленник может обойти это ограничение - когда при взаимодействии с базой данных используются мультибайтовые кодировки.

Для кодировки GBK, например, 0xbf27 — неправильная последовательность, такого символа нет. В то же время символ 0xbf5c — есть. Теперь посмотрим на работу функции экранирования: она берет по одному байту и экранирует его, если необходимо. 0xbf — это «Ќ», 0x27 — это кавычка, ее экранируем. На выходе получается 0xbf5c27 (. '), что в MySQL воспринимается как два символа — 0xbf5c и 0x27, то есть «что-то» и кавычка. SQL-инъекция в простейшем виде будет такой: `php?id=%bf%27 OR 1=1`.

Для защиты от такого рода атак используются экранирующие функции, учитывающие настройки кодировок. Однако, в том случае, если настройка кодировки была выполнена некорректно, возможна ситуация, при которой сервер и клиент используют разные кодировки и экранирование управляющих символов не выполняется. (https://stackoverflow.com/questions/5741187/sql-injection-that-gets-around-mysql-real-escape-string).

ЛР8. Вопросы для контроля

1. Классические SQL-инъекции. Типы.
2. SQL-инъекции на основе объединения.
3. SQL-инъекции на основе последовательных запросов.
4. Слепые SQL-инъекции. Типы.
5. SQL-инъекции на основе содержимого.
6. SQL-инъекции на основе возвращаемых ошибок.
7. SQL-инъекции на основе времени исполнения запроса.
8. Способы противодействие SQL-инъекциям.
9. Подготовленные выражения.

ЛР8. Задание.

Для выполнения заданий необходимо запустить образ системы (https://drive.google.com/file/d/12pwDmSIs9KuvzPLNrY3JOWjqcrXHZyLe/view?usp=sharing) на виртуальной машине (рекомендуется использовать Virtual Box).

Необходимо

- Создать виртуальную машину, выделив как минимум 1 Gb оперативной памяти.
- В настройках, в разделе «Носители», добавить новый привод оптических дисков к контроллеру IDE, указав образ диска `sql_inj.iso`.
- В настройках сети выбрать тип подключения «Виртуальный адаптер хоста». После запуска виртуальной машины получить ее IP адрес командой `ifconfig`, и подключиться к нему с хоста.

В рамках лабораторной работы необходимо выполнить следующие задания:

1. Пример 1. Необходимо успешно пройти авторизацию.
2. Пример 2. Необходимо успешно пройти авторизацию. В данном примере на сервере осуществляется проверка числа возвращаемых запросом записей из таблицы.
3. Пример 3. Необходимо успешно пройти авторизацию. Вводимые пользователем данные фильтруются путем удаления из них символа «'».
4. Пример 4,5,6. Необходимо получить все данные из возвращаемой запросом таблицы (по умолчанию выводятся данные не всех столбцов). Ввод запроса осуществляется через строку ввода адреса веб-браузера.

5. Пример 7. Необходимо получить все данные из возвращаемой запросом таблицы для заданного пользователя. В данном случае на сервере выполняется последовательно два запроса, причем второй запрос выполняет поиск записей по полученному первым запросом значению поля `name`.
6. Пример 8. Необходимо получить список всех баз данных, хранящихся на сервере. В данном случае ввод данных для добавления записей на сервер защищен от инъекций, однако чтение данных осуществляется напрямую без фильтрации.
7. Пример 1. Определить название используемой БД, названия ее таблиц и соответствующих им столбцов. Определить содержимое таблицы, по которой осуществляется запрос в БД.
8. Пример 2. Используя технику Double Blind SQL Injection, определить количество БД на сервере (их больше 15, но меньше 30).
9. Пример 9. Необходимо успешно пройти авторизацию. В данном случае осуществляется фильтрация вводимых пользователем данных путем экранирования символов «'» и «\». Пользовательский ввод осуществляется в кодировке UTF-8.