электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 05, май 2016

УДК 004.056.5

Уязвимости веб-приложений: SQL-инъекции, PHP-инъекции и XSS-инъекции

Сатова М.В., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Маслеников К.Ю., *студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Ревунков Г.И., к.т.н., доцент
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»
revunkov@bmstu.ru

В современном развивающемся мире большую роль отводят информационно-коммуникационным технологиям. Почти каждая организация имеет необходимую ей информационную систему. Информационные системы строятся исходя из предметной области, поэтому они отличаются друг от друга выполняемыми функциями, архитектурой и способами реализации.

При проектировании системы решают две главные задачи [5]:

1) разработка базы данных, для хранения информации;
2) разработка графического интерфейса пользователя клиентского приложения.

Широкое распространение для решения первой задачи получили реляционные СУБД – хорошо отлаженные, развитые, сопровождаемые системы, которые поддерживают стандарт SQL-92. К такому типу можно отнести такие системы, как Oracle, Informix, Sybase, DB2, MS SQL Server и др.

Разработка интерфейса – наиболее трудоемкая задача, отнимающая много времени у разработчиков. В последние годы появились средства визуальной разработки приложений, которые упростили задачу разработки графического интерфейса пользователя.

Большинство подобных систем представляют собой веб-ресурс, доступ к которому может быть осуществлен посредством глобальной сети Интернет. Поэтому одна из наиболее важных проблем функционирования таких систем – обеспечение информационной безопасности, главный принцип которой лежит в анализе потенциальных угроз, проведении мер по их устранению и последующем анализе состояния безопасности системы.

Согласно ГОСТ Р 50922-2006 «Защита информации. Основные термины и определения» условием реализации угрозы безопасности обрабатываемой в системе информации может быть недостаток или слабое место (уязвимость) системы. Атака злоумышленников как раз и активирует ту или иную уязвимость, которая присутствует в системе.

Рассмотрим некоторые виды уязвимости систем как с точки зрения СУБД, так и графической оболочки (интерфейса) пользователя.

Первым примером уязвимости является SQL-инъекция – это атака, при которой злоумышленник производит вставку вредоносного кода в строки, передающиеся на сервер СУБД для синтаксического анализа и выполнения.

Данная уязвимость является критической и состоит в том, что веб-приложение некорректно проверяет данные от пользователя, которые в дальнейшем используются для генерации SQL-запросов к базе данных. Злоумышленник получает возможность работать с SQL-сервером в обход логики приложения.

SQL-инъекций можно классифицировать следующим образом [3]:

1) по месту нахождения в запросе (инъекции в строковом параметре и в числовом параметре);
2) по способу внедрения инъекции (обычные и «слепые»).

Причины возникновения SQL-инъекций разнообразны, но можно выделить следующие основные причины:

1. Небезопасная конфигурация СУБД.

Разработчики приложений не всегда являются специалистами в области СУБД. Результатом этого может стать недостаточный уровень информационной безопасности и наличие уязвимостей в разрабатываемом продукте.

В качестве примера можно выделить СУБД Oracle, в которой присутствует большое количество уязвимых SQL-процедур, часть из которых доступна для выполнения непривилегированным пользователям. Также серьезной проблемой является

наличие избыточных привилегий у учетной записи, используя которую сервер приложений работает с СУБД.

2. Динамическое построение SQL-запросов.

Динамические SQL-запросы просты и удобны в использовании, однако именно они могут стать причиной SQL-инъекций. В случае, если в момент передачи запроса СУБД на выполнение входные данные недостаточно проверяются и не кодируются приложением, может возникнуть ситуация, когда они могут быть интерпретированы СУБД как дополнительные команды. Этого можно избежать, используя параметризованные запросы, так как передаваемые параметрами данные не будут интерпретированы как команды и выполнены.

3. Некорректная обработка типов, специальных символов и исключений.

Злоумышленник может узнать детали реализации приложения (язык разработки приложения, тип и версию СУБД), а также извлечь конфиденциальную информацию, если приложения, работающие с БД, неправильно обрабатывают исключительные ситуации и отображают пользователю сообщения о возникших исключениях.

В СУБД существуют специальные символы. Например, символ одинарной кавычки (‘) в SQL-запросе означает начало или конец строки, а символ комментария (*/) обозначает конец комментария. Если приложение некорректно обрабатывает специальные символы, содержащиеся во входных параметрах, то запрос, может быть модифицирован.

Многие разработчики приложений полагают, что фильтрация одинарных кавычек (‘) спасает от SQL-инъекций. Однако, этот метод защиты работает в случае SQL-инъекций только в строковых параметрах. При этом сохраняются инъекции в параметрах других типов, например, в числовых.

Этапы внедрения SQL-инъекции.

На рис. 1 представлен пример механизма внедрения SQL-инъекции, рассмотрим каждый этап более подробно.

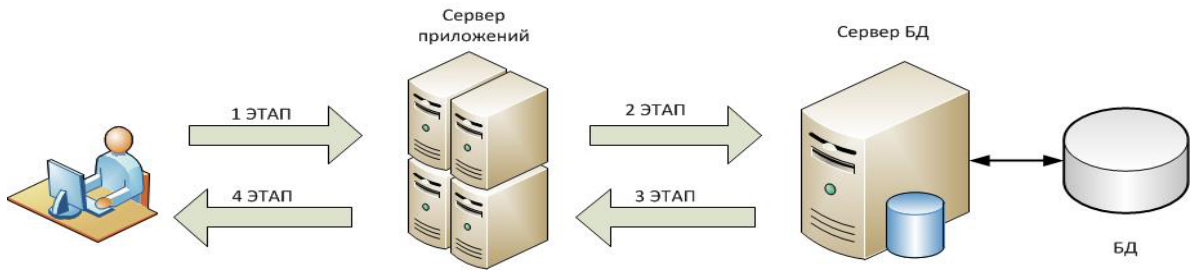

Рис. 1. Внедрение SQL-инъекции

В рассматриваемом примере будем полагать, что аутентификация в веб-приложении осуществляется с помощью веб-формы, обрабатываемой следующим кодом:

```
SQLQuery = "SELECT Username FROM Users WHERE
Username = '" & strUsername & "' AND Password = '" & strPassword & "'"
strAuthCheck = GetQueryResult(SQLQuery)
```

В этом случае разработчики непосредственно используют переданные пользователями значения strUsername и strPassword для создания SQL-запроса.

1 этап. Злоумышленник отправляет запрос в веб-приложение, в котором в качестве передаваемых параметров на форму аутентификации служат следующие параметры:

Login: ' OR "='

Password: ' OR "='

2 этап. Сервер приложений формирует SQL-запрос:

```
SELECT Username FROM Users WHERE Username = '' OR ''='' AND Password ='' OR
''=''
```

Вместо сравнения имени пользователя и пароля с записями в таблице Users, данный запрос сравнивает пустую строку с пустой строкой. Результат подобного запроса всегда будет равен True.

3 этап. Сервер БД передает данные серверу приложения.

4 этап. Злоумышленник входит в систему от имени первого пользователя в таблице.

Существуют универсальные техники применения SQL-инъекций, которые могут быть использованы атакующим вне зависимости от СУБД (MySQL, MS SQL Server, Oracle и др.) или используемого языка разработки приложения. Перечислим основные техники эксплуатации SQL-инъекций:

1. Error-based SQL-инъекция используется, когда пользователю отображается сообщение о возникшем исключении. Это дает злоумышленнику возможность построчно считывать информацию из таблицы БД.
2. Использование Union SQL-инъекции заключается в применении оператора UNION – объединяет результаты выполнения двух или более запросов SELECT. Данная техника заключается в добавлении необходимого запроса SELECT при помощи оператора UNION к первоначальному запросу. Атакующему необходимо определить количество аргументов (а также выяснить, сколько среди них ненулевых) и их типы в SQL-запросе и передать нужный SQL-запрос, возвращающий строку.

3. Если приложение корректно обрабатывает исключения или пользователю не отображается результат выполнения запроса, а значит невозможно использовать Union и Error-based SQL-инъекцию, есть возможность повлиять на работу приложения, модифицируя запрос. Это называется Blind SQL-инъекцией. Для этого составляется SQL-выражение, которое при истинном значении не нарушает логику работы приложения. При ложном значении возникает некорректное поведение в работе web-приложения: могут возвращаться неполные данные или страницы могут неправильно отображаются. При слепой инъекции видны не извлекаемые данные, а только ответ сервера.
4. Time-based SQL-инъекция очень похожа на Blind SQL-инъекцию, но анализируется время отклика сервера СУБД. При ложном значении SQL-запроса время отклика незначительно, а при истинном значении составляет несколько секунд. Используя данную технику, можно посимвольно извлекать данные из таблиц БД.
5. В случае, если ни одна из вышеперечисленных техник эксплуатации SQL-инъекций не применима, можно воспользоваться Out-bound техникой. Для этого необходим удаленный сервер, который находится под контролем злоумышленника, либо доступ к директории на сервере СУБД. Злоумышленник направляет вывод результата SQL-запроса на удаленный сервер, используя протоколы DNS, HTTP, SMTP, или осуществляет запись в файл, который расположен в доступной для злоумышленника директории на сервере СУБД.

Для успешного внедрения SQL-инъекций необходимо знать следующие параметры: способ передачи данных в веб-приложении, тип и версию СУБД, имя текущего пользователя, назначенные роли и системные привилегии.

В зависимости от назначенных текущему пользователю ролей и привилегий злоумышленник может попытаться извлечь хэши паролей пользователей СУБД, получить структуру БД и конфиденциальные данные или выполнить команды ОС, а также доступ к функциональным возможностям СУБД, а в некоторых случаях – к операционной системе сервера, на котором функционирует СУБД.

Следующий пример уязвимости – PHP-инъекции. Большинство интерфейсов современных веб-приложений написано на PHP (hypertext preprocessor) – это распространенный язык программирования общего назначения с открытым исходным

кодом. PHP сконструирован специально для ведения Web-разработок и его код может внедряться непосредственно в HTML [2].

PHP-инъекция – это одна из формы взлома веб-сайтов, работающих на PHP, который заключается в выполнении постороннего кода на стороне сервера приложений. При успешной инъекции злоумышленник может выполнить произвольный (потенциально опасный) PHP-код на сервере. PHP-инъекция становится возможной, если входные параметры принимаются и используются без проверки.

Приведем пример PHP-инъекции:

```
<?php
...
$module = $_GET['module'];
include ($module.'.php');
...
?>
```

Этот скрипт уязвимый, т.к. к содержимому переменной $module просто прибавляется «.php» и по полученному пути подключается файл.

Существуют следующие способы защиты от PHP-инъекций:

- проверка на посторонние символы в переменной $module;
- проверка, присвоения одного из допустимого значения переменной $module;
- прописывание каждого значения через оператор if;
- использование оператора switch.

Наиболее распространенная и одна из наиболее разрушительных уязвимостей в веб-приложениях – это XSS-инъекция (Cross Site Scripting) – инъекция HTML-кода в уязвимую страницу или межсайтовый скриптинг. Данная инъекция кода реализуется с помощью разнообразных способов ввода информации. Любое веб-приложение, которое выводит информацию, полученную от пользователя, может быть подвержено атакам этого вида.

Возможность успешного проведения данной операции возникает при отсутствии фильтрации значений переменных, полученных от пользователя, на наличие сценариев (скрипты JavaScript, VBScript, Perl, PHP) или опасных HTML- тегов.

Выделяют следующие типы XSS-инъекций, которые представлены на рис. 2.

Рис. 2. Типы XSS-инъекций

Постоянный (хранимый) XSS возможен, когда злоумышленнику удается внедрить на сервер вредоносный код, выполняющийся в браузере каждый раз при обращении к оригинальной странице. Классическим примером этой уязвимости являются форумы, на которых разрешено оставлять комментарии в HTML формате без ограничений.

Другими словами, хранимый XSS возникает, когда разработчики осуществляют некорректную фильтрацию при сохранении входных данные в БД на сервере или в при записи этих данных в файлы, а затем выводят эти данные в браузер пользователю.

Непостоянный (отраженный) XSS имеет место, когда данные, предоставляемые веб-клиентом в строке запроса или HTML форме, используются для генерации ответа клиенту без обработки этих данных.

XSS в DOM-модели возникает на стороне клиента во время обработки данных внутри JavaScript сценария. Данный тип XSS получил такое название, поскольку реализуется через DOM (Document Object Model) – не зависящий от платформы и языка программный интерфейс, позволяющий программам и сценариям получать доступ к содержимому HTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

Таким образом, XSS-инъекции возникают непосредственно внутри JavaScript сценария.

XSS-инъекция используется злоумышленником для перехвата значений cookie различных пользователей всех уровней (в том числе, и администратора системы), изменений настроек, размещение ложной информации и рекламы, хищение токенов форм

для проведения CSRF атак и т.д. В некоторых случаях инъекция позволяет обойти механизм аутентификации, а также позволяет выполнить произвольную команду на стороне сервера.

Тем не менее, существуют следующие способы защиты от XSS-инъекций, которые условно можно разделить на две группы: защита на стороне сервера и защита на стороне клиента [1].

К мерам защиты на стороне сервера относятся:

- кодирование управляющих HTML-символов, JavaScript, CSS и URL перед отображением в браузере;
- кодирование входных данных;
- регулярный ручной и автоматизированный анализ безопасности кода;
- указание кодировки для каждой веб-страницы до каких-либо пользовательских полей;
- обеспечение безопасности cookies, которая может быть реализована путем ограничения домена и пути для принимаемых cookies, установки параметра HttpOnly, использованием SSL.

Безопасность на стороне клиента может обеспечиваться при выполнении следующих рекомендаций:

- регулярное обновление браузера до новой версии;
- установка расширений для браузера, которые будут проверять поля форм, URL, JavaScript и POST-запросы, и, если встречаются скрипты, применять XSS-фильтры для предотвращения их запуска.

Рассмотренные виды уязвимости встречаются во всех веб-приложениях не зависимо от разработчика системы. Значительно помочь снизить степень их появления может системное проектирование, анализ возможных проблем безопасности и их решение на всех этапах разработки, а также детальное тестирование готового приложения.

Список литературы

[1] Герасименко А.В., Баранов А.А. Анализ атаки типа межсетевой скриптинг // Научный вестник НГУ. 2013, вып 3. Режим доступа: http://ir.nmu.org.ua/bitstream/handle/123456789/1791/анализ%20атаки%20типа%20межсетевой%20скриптинг.pdf?sequence=1 (дата обращения 25.02.2016).

[2] Жуков Ю.В. Основы веб-хакинга: нападение и защита. СПб.: Питер, 2011. 176 с.

[3] Егоров М. Выявление и эксплуатация SQL-инъекций в приложениях // Защита информации. Inside. 2011, вып. 2. Режим доступа: http://www.cnpo.ru/doc/echelon-sql.pdf (дата обращения 25.02.2016).

[4] Курейчик К.П., Трушкевич А.А. Основы классификации угроз информационной безопасности информационно-измерительных систем удаленной обработки данных // Доклады БГУИР. 2011, вып. 5. Режим доступа: http://libeldoc.bsuir.by/bitstream/123456789/1987/1/Основы%20классификации%20угроз.PDF (дата обращения 25.02.2016).

[5] Петров В.Н. Информационные системы. СПб.: Питер, 2003. 688 с.

[6] Постников В.М., Гребенников Н.А. Технология обработки запросов пользователей в СУБД ORACLE // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2001. № 2. Режим доступа: http://vestnikprib.ru/catalog/it/hidden/485.html (дата обращения 25.02.16).

[7] Роганов В.А., Кузнецов А.А., Матвеев Г.А., Осипов В.И. Адаптивный анализ надежности паролей при помощи гибридных суперЭВМ // Программные системы: теория и приложения. 2015, вып. 4. Режим доступа: http://skif.pereslavl.ru/psi-info/rcms/rcms-publications/2015-rus/psta2015_4_139-156.pdf (дата обращения 24.02.2016).

[8] Самохвалов Э.Н., Ревунков Г.И., Гапанюк Ю.Е. Генерация исходного кода программного обеспечения на основе многоуровнего набора правил // Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение. 2014. № 5. Режим доступа: http://vestnikprib.ru/articles/527/527.pdf (дата обращения 22.02.16).

[9] Тецкий А.Г. Исследование методов получения содержимого базы данных с помощью SQL-инъекций // Открытые информационные и компьютерные интегрированные технологии. 2014. Вып. 66. Режим доступа: http://www.khai.edu/csp/nauchportal/Arhiv/OIKIT/2014/OIKIT66/p188-191.pdf (дата обращения 25.02.2016).