Python代码审计连载之二：SSTI

笑然 / 2016-11-10 05:14:56 / 浏览数 3913 安全技术 漏洞分析 顶(0) 踩(0)

原文地址：https://www.cdxy.me/?p=738

Template Injection

之前两篇曝光率很高的文章中指出了Flask SSTI成因及利用方式

1. exploring-ssti-in-flask-jinja2
2. exploring-ssti-in-flask-jinja2-part-ii

文中已指出利用方式，事实上使用__class__.__base__.subclasses__可以直接执行命令。

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

jinja2-sandbox

开发者可以使用它降低SSTI风险。

漏洞测试代码：

```
import sys
from jinja2.sandbox import SandboxedEnvironment

env = SandboxedEnvironment()
   print env.from_string('[Output] {}'.format(sys.argv[1] if len(sys.argv) > 1 else <empty>)).render()
```

运行时过滤器将会阻断上述exp执行，爆出安全错误。

原因在于sandbox对private-attributes进行了过滤。

jinja2/sandbox.py

```
def is_safe_attribute(self, obj, attr, value):
   return not (attr.startswith('_') or is_internal_attribute(obj, attr))
```

其中is_internal_attribute函数黑名单如下：

```
UNSAFE_FUNCTION_ATTRIBUTES = set(['func_closure', 'func_code', 'func_dict','func_defaults', 'func_globals'])

UNSAFE_METHOD_ATTRIBUTES = set(['im_class', 'im_func', 'im_self'])

UNSAFE_GENERATOR_ATTRIBUTES = set(['gi_frame', 'gi_code'])
```

黑盒

此种漏洞黑盒思路和sql注入大同小异，关于自动化攻击也可以直接照搬sqlmap，搜集整理各种模板的payload加入工具即可，也可使用一些技巧如DNS、被动式扫描等。

白盒

从白盒角度来看，Python网站出现这种漏洞情况应该不多，事实上，只有将用户可控区域传入模板渲染函数时，或者说只有当开发者将模板写到视图文件中，才可触发。

这种开发风格是明显不被认可的，目前以我的开发经验还未遇见需要在渲染之前"动态生成模板"的需求。

代码示例：

```
@app.route('/1')
def main1():
   template = '''{%% extends "base.html" %%}
{%% block body %%}
   <div class="center-content error">
       <h1>Oops! That page doesn't exist.</h1>
       <h3>%s</h3>
```

```
   </div>
{%% endblock %%}
''' % (request.url)
   return render_template_string(template, dir=dir, help=help, locals=locals)
```

而一般情况下，无论是Flask还是Django，开发者都会将模板内容写入固定文件夹，与视图代码分离，以下例子均为安全代码。

Flask:

```
@app.route('/')
def safe():
   return render_template('home.html', url=request.args.get('p'))
```

Django:

```
def home(request):
   return render(request, 'home.html', {#DATA#})

class MyView(TemplateView):
   template_name = 'home.html'
   def get_context_data():
       #DATA#
```

Format Injection

根据可控格式化字符串造成的注入点，可导致上下文环境的变量读取，进而泄露敏感内容。

- https://xianzhi.aliyun.com/forum/read/615.html
- https://virusdefender.net/index.php/archives/761/

python 2/3以下常规用法中`format_string`处可控，即可触发漏洞。

```
format_string % ()
format_string.format()
```

此外Python 3.6新增f函数可导致命令执行

```
f'''{__import__('os').system('id')}'''
```

黑盒

可添加针对该类问题的payload

```
{user.groups.model._meta.app_config.module.admin.settings.SECRET_KEY}

{user.user_permissions.model._meta.app_config.module.admin.settings.SECRET_KEY}
```

如有其他可关注的触发点及利用方式，欢迎留言

上一篇：Python代码审计连载之一：CSRF 下一篇：Python代码审计连载之三：Se...
1. 0 条回复
   - 动动手指，沙发就是你的了！