# Server-Side Template Injection (SSTI)

## Understanding the Difference between XSS and SSTI Attacks

Web applications are essential tools for businesses and individuals alike, but they can also be susceptible to various security vulnerabilities. Two common types of vulnerabilities are Cross-Site Scripting (XSS) and Server-Side Template Injection (SSTI). While both involve injection attacks, they target different components of a web application and can have distinct consequences.

|  |  |
| --- | --- |
| XSS | SSTI |
| Malicious code injected into web app. | Malicious code injected into templates (server-side of the web app). |
| Code executed in the victim’s browser. | Code executed on the server. |
| Attack targets:   * The client-side of the application. * Aims to manipulate user interactions. | Attack targets:   * Manipulate the app’s rendering process. * Achieve remote code execution. |

## What are Templates?

Templates are files that contain placeholders for dynamic data, such as HTML files with variables like {{greeting}}. The backend app replaces these placeholders with actual data, creating dynamic content to display on the web page.

## Exploiting SSTI

A diagram of a computer

Description automatically generated

### 3.1 Detect

To detect SSTI, try injecting special characters like $, {{, <%= %> into input fields and observe any differences in the server's response. Mathematical operations like ${7/0} can also help identify the vulnerability.

|  |  |  |
| --- | --- | --- |
| ${} | {{}} | <%= %> |
| ${7/0} | {{7/0}} | <%= 7/0 %> |
| ${foobar} | {{foobar}} | <%= foobar %> |
| ${7\*7} | {{7\*7}} | `` |
| #{7\*7} | \*{7\*7} |

Otherwise, you'll need to manually test different language-specific payloads and study how they are interpreted by the template engine. A common way of doing this is to inject arbitrary mathematical operations using syntax from different template engines. You can then observe whether they are successfully evaluated. To help with this process, you can use a decision tree like the following:

A diagram of a computer program

Description automatically generated

You should be aware that the same payload can sometimes return a successful response in more than one template language. For example, the payload {{7\*'7'}} returns 49 in Twig and 7777777 in Jinja2. Therefore, it is important not to jump to conclusions based on a single successful response.

### Identify

A diagram of a computer program

Description automatically generated

Find the template engine being used by causing errors with payloads like ${foobar}. Errors may reveal the template engine.

### 3.3 Explore & Exploit

Read the template engine's documentation and find available functions and objects. Attackers can then craft payloads to exploit vulnerabilities, like arbitrary object creation or remote code execution.

## How is that exploitable?

If attackers manage to inject malicious code that executes on the server, they can gain control over it. For instance, injecting OS commands like {{ import os; os.system("ls") }} can allow them to list files on the server.

## Prevention

* Jinja’s WTF forms provide extra protection by sanitizing input, escaping characters, and preventing malicious code execution.
* Avoid user-modifiable templates whenever possible.
* Use "logic-less" template engines, like Mustache, to separate logic from presentation.
* If you must use dynamic templates, sandbox untrusted code to limit its capabilities.
* Consider deploying the template environment in a secure container, like Docker, to add an extra layer of protection.
* By following these steps, you can better understand and protect your web application from SSTI attacks. Always prioritize security to keep your users and data safe.

## Try it!

https://ssti.secure-cookie.io/

## References

- https://secure-cookie.io/attacks/ssti/

- https://book.hacktricks.xyz/pentesting-web/ssti-server-side-template-injection

- https://portswigger.net/web-security/server