**SSTI – Updated Notes:**

SSTI – leverages many popular front-end template syntax to inject into the webpage giving us access to the underlying features. This occurs when some user input is directly added to a template rather then passed in as data. This allows us to access many features of the templating engine and often take control of the server.

This vulnerability often is very high severity and could result in RCE. Even if we don’t get RCE, we could potentially read arbitrary files or access other sensitive data.

**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.

Static templates that simply provide placeholders into which dynamic content is rendered are generally not vulnerable to server-side template injection. The classic example is an email that greets each user by their name, such as the following extract from a Twig template:

$output = $twig->render("Dear {first\_name},", array("first\_name" => $user.first\_name) );

This is not vulnerable to server-side template injection because the user's first name is merely passed into the template as data.

However, as templates are simply strings, web developers sometimes directly concatenate user input into templates prior to rendering. Let's take a similar example to the one above, but this time, users are able to customize parts of the email before it is sent. For example, they might be able to choose the name that is used:

$output = $twig->render("Dear " . $\_GET['name']);

In this example, instead of a static value being passed into the template, part of the template itself is being dynamically generated using the GET parameter name. As template syntax is evaluated server-side, this potentially allows an attacker to place a server-side template injection payload inside the name parameter as follows:

http://vulnerable-website.com/?name={{bad-stuff-here}}

Vulnerabilities like this are sometimes caused by accident due to poor template design by people unfamiliar with the security implications. Like in the example above, you may see different components, some of which contain user input, concatenated and embedded into a template. In some ways, this is similar to SQL injection vulnerabilities occurring in poorly written prepared statements.

However, sometimes this behavior is actually implemented intentionally. For example, some websites deliberately allow certain privileged users, such as content editors, to edit or submit custom templates by design. This clearly poses a huge security risk if an attacker is able to compromise an account with such privileges.

**Constructing a server-side template injection attack**

Identifying server-side template injection vulnerabilities and crafting a successful attack typically involves the following high-level process:

**Detect**

Server-side template injection vulnerabilities often go unnoticed not because they are complex but because they are only really apparent to auditors who are explicitly looking for them. If you are able to detect that a vulnerability is present, it can be surprisingly easy to exploit it. This is especially true in unsandboxed environments.

As with any vulnerability, the first step towards exploitation is being able to find it. Perhaps the simplest initial approach is to try fuzzing the template by injecting a sequence of special characters commonly used in template expressions, such as **${{<%[%'"}}%\**. If an exception is raised, this indicates that the injected template syntax is potentially being interpreted by the server in some way. This is one sign that a vulnerability to server-side template injection may exist.

Server-side template injection vulnerabilities occur in two distinct contexts, each of which requires its own detection method. Regardless of the results of your fuzzing attempts, it is important to also try the following context-specific approaches. If fuzzing was inconclusive, a vulnerability may still reveal itself using one of these approaches. Even if fuzzing did suggest a template injection vulnerability, you still need to identify its context in order to exploit it.

----------------------------------

**Plaintext context**

Most template languages allow you to freely input content either by using HTML tags directly or by using the template's native syntax, which will be rendered to HTML on the back-end before the HTTP response is sent. For example, in Freemarker, the line render('Hello ' + username) would render to something like Hello Carlos.

This can sometimes be exploited for XSS and is in fact often mistaken for a simple XSS vulnerability. However, by setting mathematical operations as the value of the parameter, we can test whether this is also a potential entry point for a server-side template injection attack.

For example, consider a template that contains the following vulnerable code:

render('Hello ' + username)

During auditing, we might test for server-side template injection by requesting a URL such as:

http://vulnerable-website.com/?username=${7\*7}

If the resulting output contains Hello 49, this shows that the mathematical operation is being evaluated server-side. This is a good proof of concept for a server-side template injection vulnerability.

Note that the specific syntax required to successfully evaluate the mathematical operation will vary depending on which template engine is being used. We'll discuss this in more detail in the [Identify](https://portswigger.net/web-security/server-side-template-injection#identify) step.

-----