SSTI

What is it?

SSTI: Server side template injection

SSTI can occur because of a technique that developers use to solve the issue of dynamic page generations. Some websites take user input or calculated values and place them in a template that is ready to be served to the site visitor.

You might wonder why templating engines exist and so did I. It turns out templating engines on the server help speed up the process of programming by using smaller tokens and syntax. Templating engines can also help us add logic and variables to our templates. While these are beautiful positives, we also have some negatives to templating engines.

The biggest negative to using server side templating engines is the security aspect. I often warn you guys that integration points are places where security vulnerabilities happen and this is one of the clearest examples i can think of. Since templating engines run on the server and sometimes use variables that may be user supplied, these engines need to grab that data from somewhere. Since templating engines are running as standalone software that take input, there are no inherent security rules defined.

All of this is very fancy language for saying that developers need to filter user data properly, or user can supply commands to the templating engine and the templating engine non-descrimitly takes that input and executes this command.

I do not want to talk down to any developer. They have my deepest respect. That being said, developers have a tendency to not implement these filtering mechanisms properly due to their nature. Often these filters are blacklist based, meaning the developer creates a config file that states what character or sequence of characters is supposed to be disallowed. If the developer forgets one of the characters that we can use to construct an SSTI attack string however, we have an entry point.

It may also occur that developers do implement the filtering systems properly but that we still have an SSTI. This can occur because the filters might not be applied to all points where the templating engine takes input from.

SSTI can occur in plaintext and in code context.

Attack strategy

The strategy we will take on this topic is very similar to the XSS strategy. We are pretty much banking on the fact that integration points make it easy to forget to apply the correct filtering.

For a full attack strategy, we want to detect SSTI, identify the templating engine being used and finally exploit it.

Detect

We want to insert our attack vector as soon as we can and into every single field that we can to make detection as likely as possible. The reason for this is simple, as you test the application, the templating engine might use any of the saved attack vectors and if the developers forgot to include the filtering on that endpoint/integration point, we have our basic entry point.

As an attack vector i would suggest you use:

${7*7}

The attack vector allows for the broadest detection possible as we use characters that are used in the most prevalent templating engines.

When the server doesn’t return our original attack vector, but instead returns 49, we have likely detected either an SSTI or a CSTI.

Identify

When we know we have an SSTI, we have to identify which templating engine is being used so we can craft a custom attack vector and prove our impact.

To identify our templating engine, we will base our strategy on the excellent research done by James Kettle for PortSwigger.

Enter the attack string:

{{7*7}}

IF you'd get 49 back,move on to the next step, IF you get your original attack vector back, the templating engine is NOT vulnerable and you should move on entirely.

Next step:

{{7'*7'}}

IF you'd get 49 back, The templating engine is Twig, move on to the exploit chapter. IF you get 7777777 back, the templating engine is Jinja2, move on the exploit. IF neither get returned there is no vulnerability, move on to the next step.

Next step:

A{*comment*}b

IF it resolves, the templating engine is Smarty. IF it doesn't resolve, move on to the next step.

Next step:

${"z".join{"ab"}}

IF it resolves, the templating engine is Mako. IF it doesn't resolve, there is probably no SSTI, move on the CSTI.

Exploit

This is HIGHLY dependant on the templating engine that is being used. Your first step will be to look up the manual of the templating engine you identified. These manuals will contain a section that details what security measures should be taken and how we can exploit them.

You can also refer to this article. It continues where this chapter stops and describes all of the exploits per templating engine.

https://portswigger.net/research/server-side-template-injection