This page may be out of date

We haven't updated it for a while because we're busy working on new, improved content to help you get the most out of Burp Suite. In the meantime, please note that the information on this page may no longer be accurate.

Visit our Support Center →

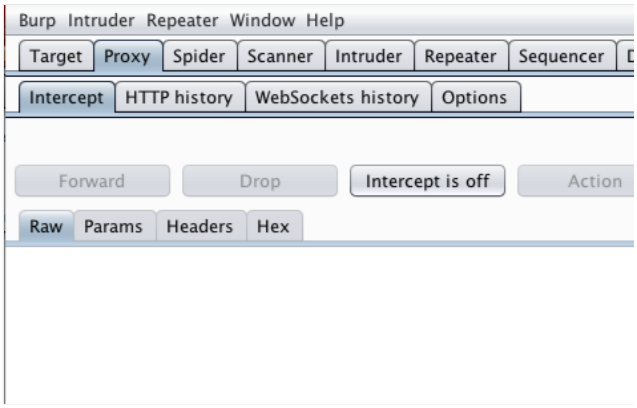
Using Burp to Test for Code Injection Vulnerabilities

Server-side code injection vulnerabilities arise when an application incorporates user-controllable data into a string that is dynamically evaluated by a code interpreter. If the user data is not strictly validated, an attacker can use crafted input to modify the code to be executed, and inject arbitrary code that will be executed by the server.

Server-side code injection vulnerabilities are usually very serious and lead to complete compromise of the application's data and functionality, and often of the server that is hosting the application. It may also be possible to use the server as a platform for further attacks against other systems.

In this example we will demonstrate how to detect code injection flaws using Burp Suite. This tutorial uses the OWASP "NodeGoat" project training tool.

Ensure "Intercept is off" in the Proxy "Intercept" tab.

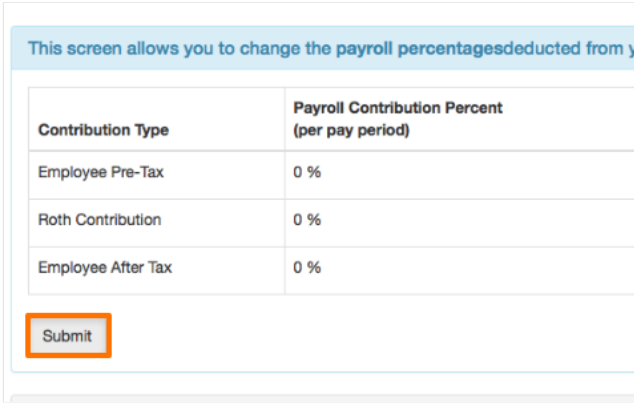


During your initial mapping of the application, you should already have identified any obvious areas of attack surface in relation to injection vulnerabilities.

Visit the web page of the application that you are testing.

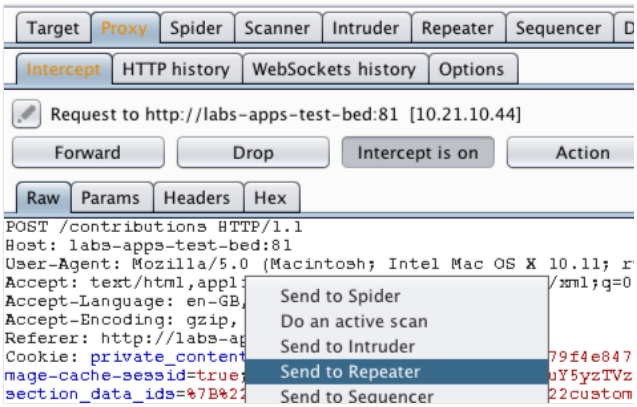
Return to Burp and ensure "Intercept is on" in the Proxy "Intercept" tab.

Now send a request to the server. In this example by clicking the "Submit" button.



The request will be captured in the Proxy "Intercept" tab.

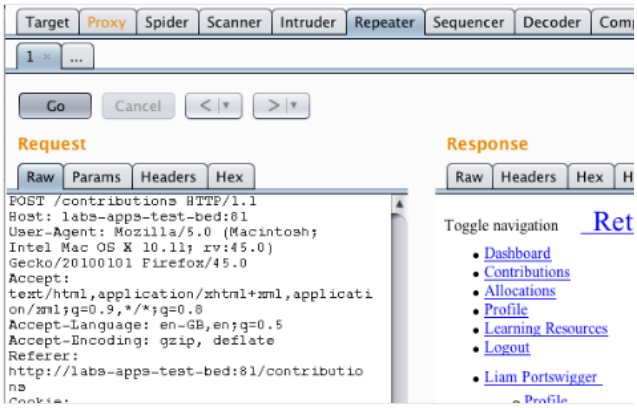
Right click anywhere on the request to bring up the context menu and click "Send to Repeater".



Here we can input various payloads in to the input field of a web application and monitor the response.

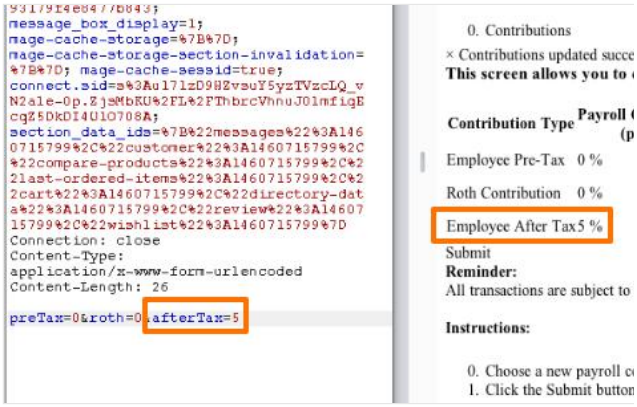
Click the "Go" button in Repeater to send the request to the server.

You can observe the response from the server in the Repeater "Response" view.



We can see that by editing the `afterTax` parameter we are able to affect the response.

The Employee After Tax contribution value is now set to 5%.



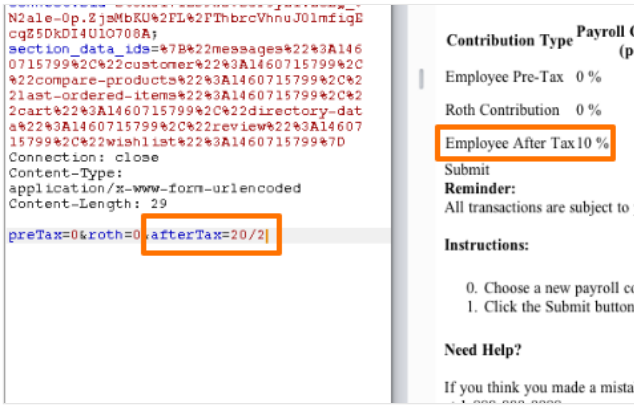
What we want to ascertain is whether the application incorporates user-controllable data into a string that is dynamically evaluated by a code interpreter.

First, we can input a calculation: 20 / 2.

We can see from the response that the application has evaluated this input.

The Employee After Tax contribution value is now set to the value of our calculation: 10%.

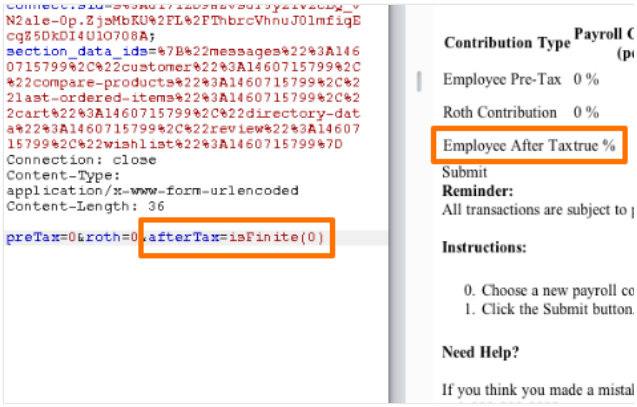
Our method of detection would alter if we were injecting into a string. We could try to break out and reopen the string using a single quotation mark followed by double quotation marks.



Next, we can attempt an input code in to our insertion point and assess whether the application processes our instruction.

The global JavaScript function `isFinite()` determines whether the passed value is a finite number.

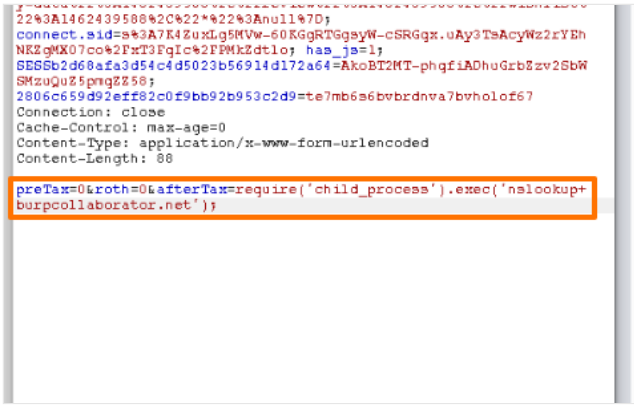
The response `true` demonstrates that user data is not strictly validated, an attacker can use crafted input to modify the code to be executed, and inject arbitrary JavaScript code that will be executed by the server.



Finally, we should attempt to demonstrate the execution of a shell command.

In this example we have used the `nslookup` command, a network administration tool for querying the Domain Name System (DNS).

If successful, the command will cause a connection with our server.



Related articles:

- What is Burp Proxy?
- Using Burp Repeater