



# Practical demonstration - Main application hacking

[Introduction](#)

[Owasp Juice Shop](#)

[Installation](#)

[Judging our target](#)

[Exploring the application](#)

[Setting up burp suite](#)

[Manually walking the application](#)

[SQLi](#)

[Testing for IDORs](#)

[Testing for vertical privilege escalation](#)

[The login section](#)

[CSRF](#)

[Exploring the requests](#)

[LFI/RFI](#)

[OS Command injection](#)

## Introduction

When we hunt, it's important to look at every target in its own right. We are going to look at the OWASP juice shop. In this demonstration you will be shown all the topics we went over and which parameters we will be using to test. Not all of our tests will lead to existing issues but still we **Have** to do all these tests. We are no longer practicing right now, this is bug bounties.

## Owasp Juice Shop


### Installation

Before we can hack on the OWASP juice shop, we ofcourse need to install it first. I chose to use herokuapp for this as it's free but you can also pick the docker image.

First navigate to <https://dashboard.heroku.com/apps> if you need to create an account, do that first. You will now see the option to create a new app, select this and pick the option to create a new app.

Next navigate to <https://elements.heroku.com/buttons/bkimminich/juice-shop> and click the "Deploy to heroku" button.


Add-ons Buttons Buildpacks

 **OWASP Juice Shop**  
by [bkimminich](#)

[Deploy to Heroku](#)  
[View on GitHub](#)

Recent Deploys: 1,051  
Stars: 4,329  
Forks: 2,997

GitHub Readme.md



# OWASP Juice Shop

owasp/flagship-project release v12.6.1 Follow 3.7k Follow r/owasp\_juiceshop 160

CI/CD Pipeline passing test coverage 89% maintainability A technical debt 2% cii best practices gold GitHub 4.3k

Contributor Covenant v2.0 adopted

The most trustworthy online shop out there. (@dschadow) – The best juice shop on the whole internet! (@shehackpurple) – Actually the most bug-free vulnerable application in existence! (@vanderaj) – First you 😂😂 then you 😂 (@kramse) – But this doesn't have anything to do with juice. (@coderPatros' wife)


OWASP Juice Shop is probably the most modern and sophisticated insecure web application! It can be used in security trainings, awareness demos, CTFs and as a guinea pig for security tools! Juice Shop encompasses vulnerabilities from the entire OWASP Top Ten along with many other security flaws found in real-world applications!

[Juice Shop Screenshot Slideshow](#)

For a detailed introduction, full list of features and architecture overview please visit the official project page: <https://owasp-juice.shop>

Pick a unique name for your app, it doesn't matter which it is but it has to be unique.

Create New App


 Deploy your own  
[OWASP Juice Shop](#)  
Probably the most modern and sophisticated insecure web application  
[bkimminich/juice-shop#master](#)

App name

✓

**thexssratjuiceshop** is available

Choose a region


 United States


[Add to pipeline...](#)

[Deploy app](#)

Click the "Deploy app" button , the app will now start deploying. It will take a while before the app is deployed so give it some take, sit back and make yourself a good coffee or tea with a nice piece of cheese.

Create New App



Deploy your own  
[OWASP Juice Shop](#)  
Probably the most modern and sophisticated insecure web application  
 [bkimminich/juice-shop#master](#)


---

**App name**

✓

**thexssratjuiceshop** is available

**Choose a region**

 United States

---

Add to pipeline...

---

Deploy app

---

Create app✓

---

Configure environment•••

---

Build app

---

Run scripts & scale dynos

---

Deploy to Heroku

## Create New App



Deploy your own

[OWASP Juice Shop](#)

Probably the most modern and sophisticated insecure web application

[bkimminich/juice-shop#master](#)

App name



thexsratjuiceshop is available

Choose a region

United States

Add to pipeline...

Deploy app

Create app



Configure environment



Build app [Show build log](#)



Run scripts & scale dynos



Deploy to Heroku



Your app was successfully deployed.

Manage App

View

We can now view our app 😊

## Judging our target

We recognize this is a webshop so to hack this target properly we will have to make a small investment. Since this is a webshop we want to test for at least the following functions:

- Registration
- Login
- Buying an item
- Possibly returning an item
- Our wallet functionality if exists
- Logic flaws
- XSS
  - Stored
  - Reflected

- Basket functionality
- Adresses
- IDORs
- CSRF
- Broken access control if we can get to admin functions

This is an initial judgement and we might add to this as we explore the website and find more functionality.

## Exploring the application

First of all we need to know what functionality exists before we can start attacking our target properly. We need to fill up our site map in burp and we need to be able to explore the paramatirised requests. To do this, we need to set up burp properly first. This includes setting up our scope and setting the options that we need. In this course we will use burp suite but feel free to use any other MiTM proxy with the same functionality.

## Setting up burp suite

It really helps to have burp suite pro, you don't have to but the fact that you can save a project is a major plus for me. I can only hunt in bursts of 1 to 3 hours so i have to revisit my target often. This means two things.

- I have to take very dilligent notes so i don't retest things 10 times needlessly and so that i make sure i do test all of my functionality. Part of this documentation is the "Judging our target" section.
- If u can set up my project settings in burp suite, save them and reload them whenever you want, that is a major plus. The biggest part of any activity is getting yourself to do it and if you can skip part of the setup, that will help you get started. If you are doing something time seems to fly but if you are sitting in your sofa it takes tremendous power to get yourself up and go hunting. Anything you can do to make this easier is a major win.



First i like to setup my scope, make sure you add the propper URLs that are in scope.

The screenshot shows the Burp Suite interface with the 'Target' tab selected. The 'Scope' sub-tab is active, and the 'Use advanced scope control' checkbox is checked. Below this, there are two sections: 'Include in scope' and 'Exclude from scope'. Each section contains a table with columns for 'Enabled' and 'Prefix', and a list of control buttons (Add, Edit, Remove, Paste URL, Load ...).

Target Scope

Define the in-scope targets for your current work. This configuration affects the behavior of tools throughout the suite. The easiest way to configure scope is to browse menus in the site map to include or exclude URL paths.

Use advanced scope control

Include in scope

Enabled	Prefix
---------	--------

Exclude from scope

Enabled	Prefix
---------	--------

Hackerone has configuration files for burp you can download.

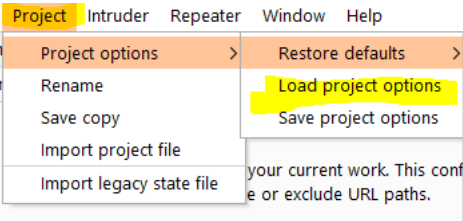
Scopes

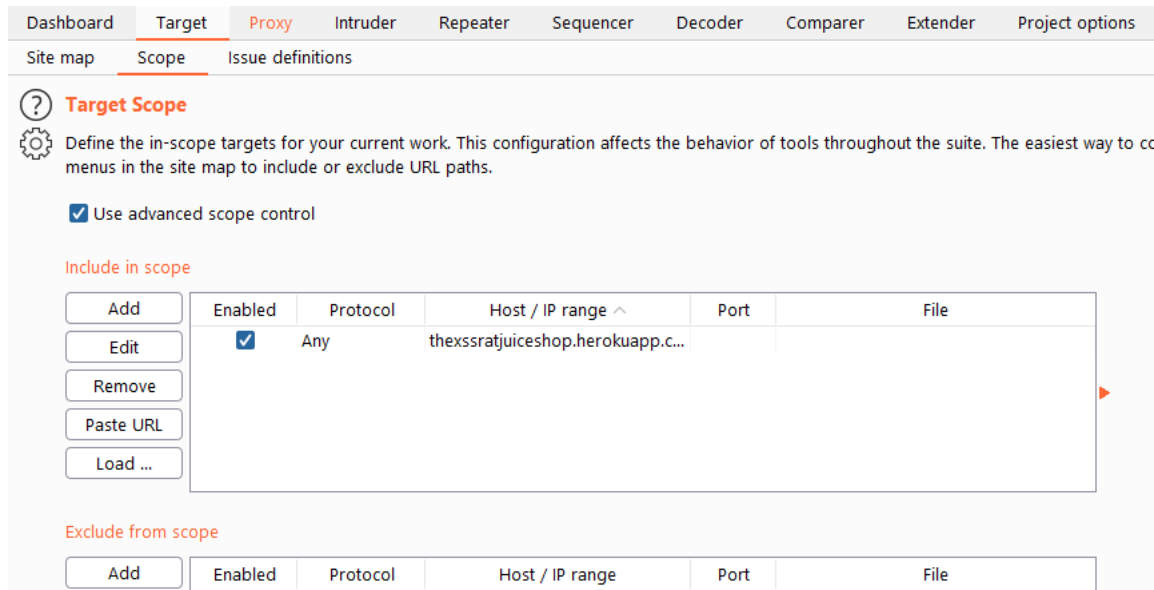
In Scope

Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible
Domain		Critical	Ineligible

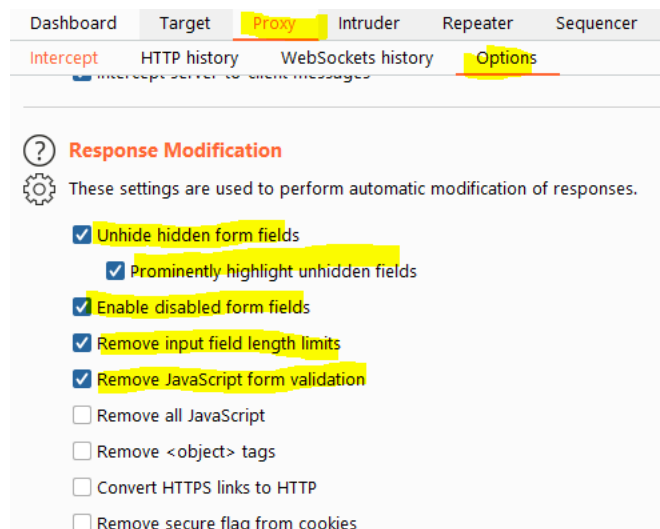
[Download Burp Suite Project Configuration file \(18 URLs\)](#) [View changes](#) Last updated on March 10, 2021.

You can then import this file via the project options import functionality under "Project > Project options > Load Project options"





After setting up our scope we will move on to setting up our proxy options. I always configure several options to make it easier for myself to see things like hidden fields and to remove any javascript validation from my responses.



This will make any hidden fields easier to see as it will unhide them and draw a big red square around them. If you see this big red square you know you are looking at a hidden field.



Now that we have burp suite set up in the background we can start exploring our application.

### Manually walking the application

Just because we are manually walking the application does not mean we should not be hacking. This is the most important phase in bug bounties and most of you will know it as the recon phase.

In this phase we want to get to know our application. We want to start by exploring the functionality and as we do that we want to take note of our privilege levels. Even though it might not seem like it, since we don't have access to the admin functionality (yet), but there are different levels of privileges.

- Unauthenticated accounts (not logged in)
- Authenticated accounts

As we hack our application, there might be more levels we can add to this such as administrators.

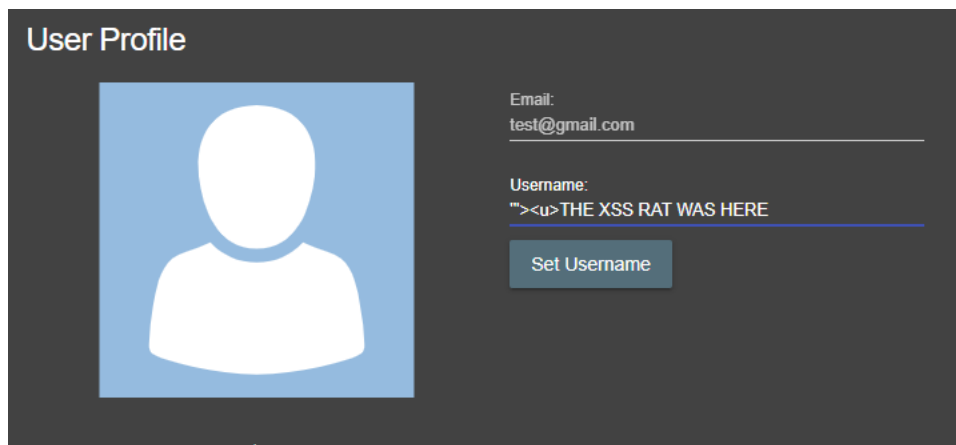
I myself use excel to make a quick mindmap but you can use whatever tool suits you best.

	Authenticated	Unauthenticated	Admin??
View items	Green	Green	?
Add items to cart	Green	Red	?
Buy items	Green	Red	?
View reviews	Green	Green	?
Add reviews to items	Green	Red	?
...	?	?	?

When i register my account, I register using an attack vector that automatically tests for JS XSS, HTML injection and HTML tag attribute injection.

```
'"><u>THE XSS RAT WAS HERE
```

I use this attack vector wherever possible when attacking my target.

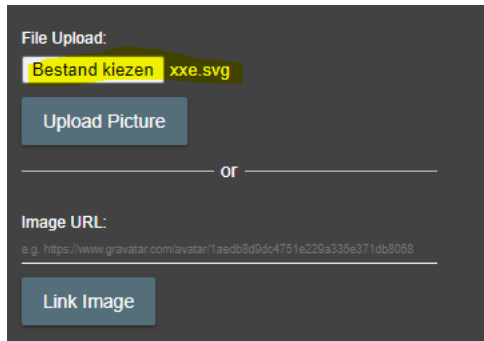


This way, whenever the application uses my username anywhere, i am automatically testing for all of the described attack vectors. If my username is reflected in the JS context anywhere, the "" will try to break out of the context it is being reflected into. If the username is reflected in a tag, like it is in the picture above. In this picture, when we save the username, we can see that we have broken out of the VALUE attribute of the input tag. From here, we can try to insert our own javascript.

```
<input class="form-control mdl-textfield__input" id="username" type="text" name="username" value="'><u>THE XSS RAT WAS HERE" style="color: #FFFFFF;" placeholder="e.g. SuperUser" aria-label="Text field for the username"> == $0
```

I will try this WHEREVER i can, this includes addresses, nicknames, ...

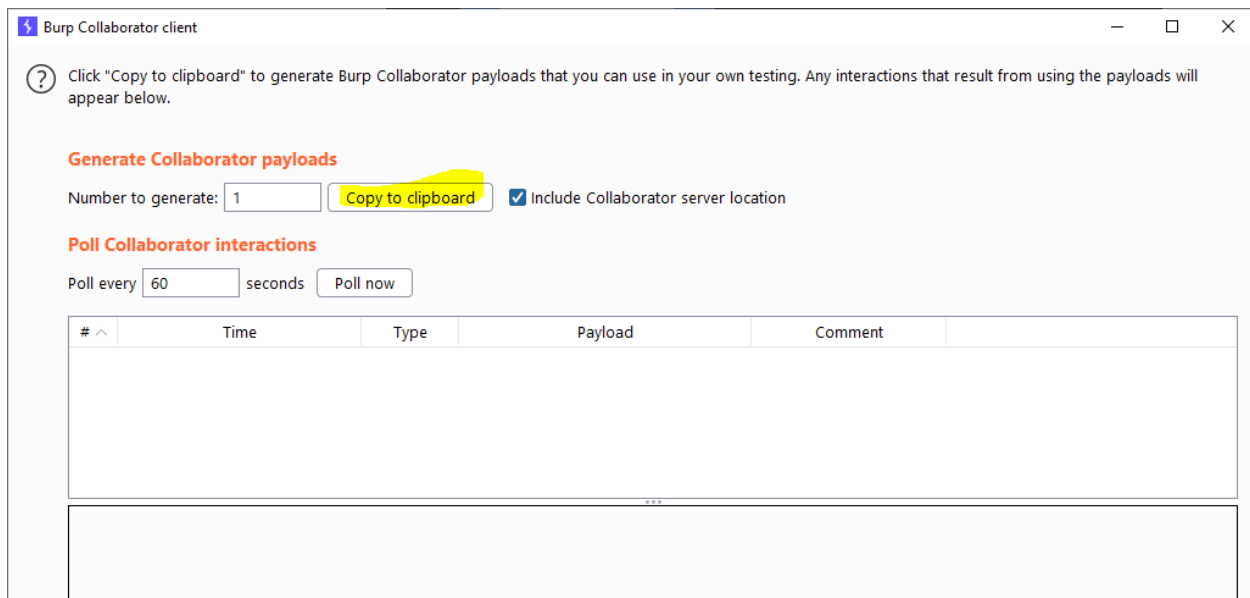
On the profile page however i can still see some more things i can test for. I see a profile picture so i can test for XXE via SVG here.



I also see a link option here. This link will resolve to a picture which gives the option for SSRF. To test for this, I start my burp collaborator and grab a URL that I can insert into this field. Running a public burp collaborator server is a premium option and only available in the paid version of burp. If you don't have the paid version of burp you can use:

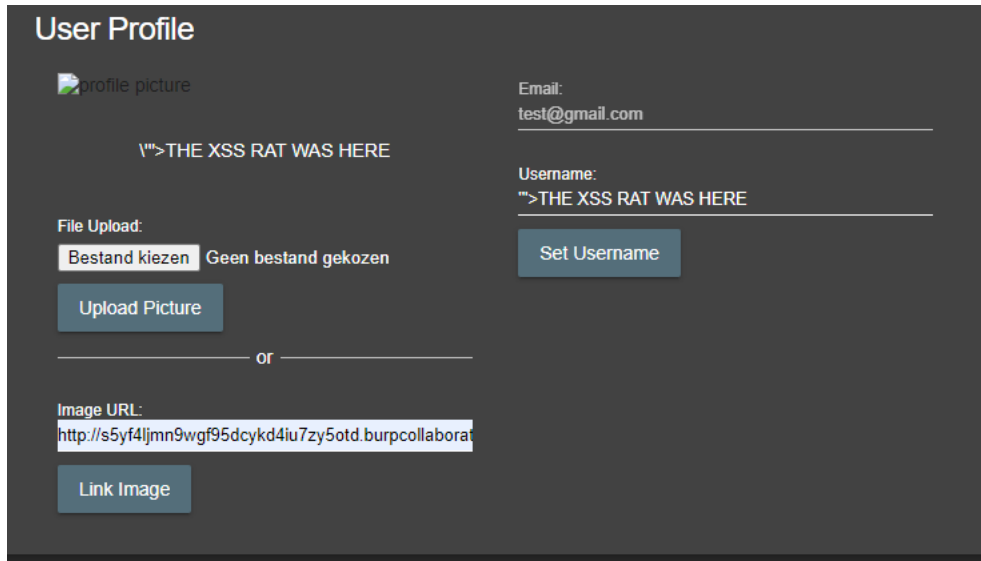
- Your own webserver
  - If you don't configure this properly, you can only capture HTTP requests
- A public burp collaborator
  - This may suffer from availability issues as it's shared between all users

Whichever option we go for, we need to copy our payload to the clipboard and paste it in our URL field that the server tries to resolve.



We might need to put HTTP:// in front of our URL if the server checks for syntax.

<http://s5yf4ljmn9wgf95dcyk4iu7zy5otd.burpcollaborator.net>



If you a lot of DNS requests coming into your burp collaborator but no HTTP requests, than there probably is no way to pull off SSRF is there is a possible egress filter in place. Egress filters can stop certain types of outgoing traffic. As of this writing OWASP juice shop does not have any SSRF vulnerabilities but if we'd find one here, we would continue on our SSRF path.

Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

**Generate Collaborator payloads**

Number to generate:    Include Collaborator server location

**Poll Collaborator interactions**

Poll every  seconds

#	Time	Type	Payload	Comment
1	2021-Mar-10 20:19:51 UTC	DNS	s5yf4ljmn9wgf95dcyk4iu7zy5otd	
2	2021-Mar-10 20:19:51 UTC	DNS	s5yf4ljmn9wgf95dcyk4iu7zy5otd	
3	2021-Mar-10 20:19:51 UTC	DNS	s5yf4ljmn9wgf95dcyk4iu7zy5otd	
4	2021-Mar-10 20:19:51 UTC	DNS	s5yf4ljmn9wgf95dcyk4iu7zy5otd	
5	2021-Mar-10 20:19:51 UTC	HTTP	s5yf4ljmn9wgf95dcyk4iu7zy5otd	

...

Description	Request to Collaborator	Response from Collaborator
	The Collaborator server received an HTTP request.	
	The request was received from IP address 34.242.128.128 at 2021-Mar-10 20:19:51 UTC.	

**My saved addresses**

"<u>THE XSS RAT WAS HERE	"<u>THE XSS RAT WAS HERE, "<u>THE XSS RAT WAS HERE, "<u>THE XSS RAT WAS HERE, 13265	"<u>THE XSS RAT WAS HERE	<input type="button" value="Copy"/>	<input type="button" value="Delete"/>
--------------------------	---	--------------------------	-------------------------------------	---------------------------------------



```

POST /api/resetPassword HTTP/1.1
Host: ferretshop.herokuapp.com
Connection: close
sec-ch-ua: ";Not A Brand";v="99", "Chromium";v="88"
Accept: application/json, text/plain, */*
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVhbnZlIiwiaWF0Ij0iJmVzZXJlIjoiaWwiZW1haWwiOiJ0
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://ferretshop.herokuapp.com/
Accept-Encoding: gzip, deflate
Accept-Language: nl-NL,nl;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: language=en; welcomebanner_status=dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWVhbnZlIiwiaWF0Ij0iJmVzZXJlIjoiaWwiZW1haWwiOiJ0
Content-Length: 24

{email:"victim@gmail.com",
email:"attacker@gmail.com"}

```

This might prompt the "GenerateLink" server to generate a password reset link for the first email address but the "SendLink" server might send the link to the attacker.

## CSRF

Whenever i see a CSRF token, i will try to replace it:

- with an empty parameter (CSRF=)
- with a parameter of the same restrictions (like same length and alphanumeric) (CSRF=247545dfs1)
- CSRF=1
- A CSRF token that does not belong to that account

I can use the tools "Match and replace" or "Autorepeater" for this although the "Autorepeater" extensions seems to have broken with the latest burp update though this might get fixed later on.

See the tools section.

## Exploring the requests

Now comes the fun part, we are going to look at all the requests and parameters in their own right. To do this we need to go back to burp suite and look at our site map. This has been filling up in the background while we click around.

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options

Site map Scope Issue definitions

Logging of out-of-scope Proxy traffic is disabled [Re-enable]

Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

Filter by request type

- Show only in-scope items
- Show only requested items
- Show only parameterized requests
- Hide not-found items

Filter by MIME type

- HTML
- Script
- XML
- CSS
- Other text
- Images
- Flash
- Other binary

Filter by status code

- 2xx [success]
- 3xx [redirection]
- 4xx [request error]
- 5xx [server error]

Folders

- Hide empty folders

Filter by search term

Regex

Case sensitive  Negative search

Filter by file extension

Show only:

Hide:

Filter by annotation

- Show only commented items
- Show only highlighted items

[Show all] [Hide all] [Revert changes]

polyfills-es5.js

Now we can see the parameterised requests, we don't really care about the static requests.

The screenshot displays a REST client interface for the URL `https://ferretshop.herokuapp.com`. The left sidebar shows a tree view of API endpoints, with `api/Address/` selected. The right pane shows the 'Contents' tab with a table listing the request details:

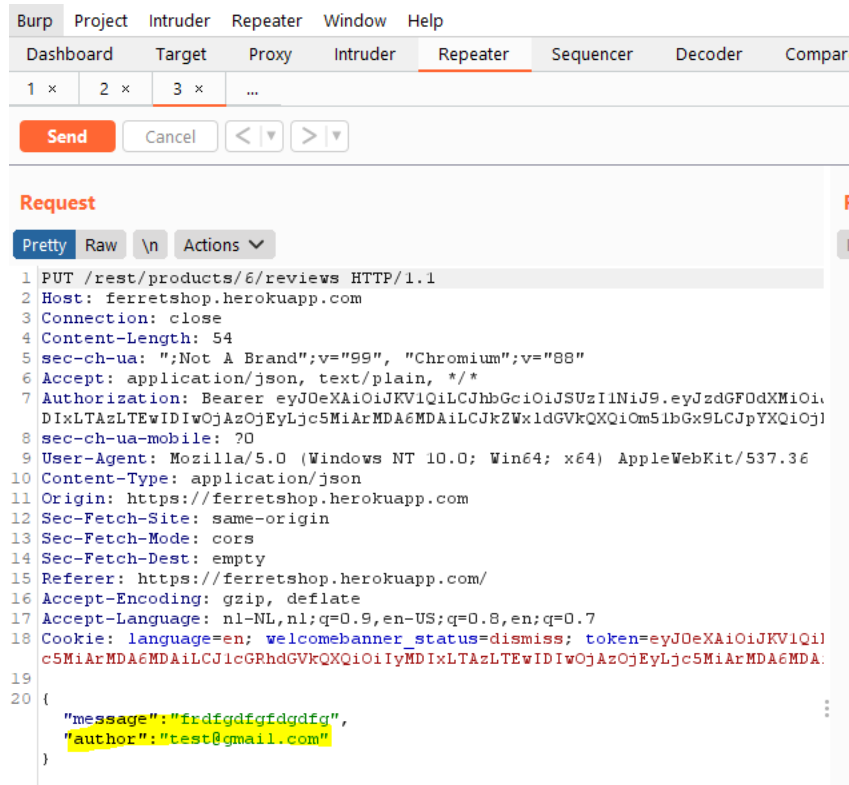
Host	Method	URL	Params
https://ferretshop.herokuapp.com	POST	/api/Address/	

Below the table, the 'Request' tab is active, showing a raw HTTP request with headers and a JSON body. The JSON body contains XSS payloads:

```
{
  "country": "'\><u>THE XSS RAT WAS HERE'",
  "fullName": "'\><u>THE XSS RAT WAS HERE'",
  "mobileNum": "654654656",
  "zipCode": "13265",
  "streetAddress": "'\><u>THE XSS RAT WAS HERE'",
  "city": "'\><u>THE XSS RAT WAS HERE'",
  "state": "'\><u>THE XSS RAT WAS HERE'"
}
```

I am going to look at all of these requests and see if i can find some parameters i can manipulate that i should not be able to manipulate such as:

- {userType:"User"} > {userType:"Admin"}
- {accountType:"Basic"} > {accountType:"Advanced"} (might be more expensive)
- {rating:5} > {rating:-5000} (might be able to negatively affect the rating of a video on youtube)
- ... Use your imagination



In this case for example, we can change the author of a review which should not be possible.

## LFI/RFI

Whenever a parameters shows that it is grabbing a file from the local file system or whenever it seems to be from a remote location, i will try either LFI or RFI respectively. See those sections to learn more about them.

- GET /avatar.php?file=image1.png
- GET /avatar.php?file=s3.bucket.org/image1.png

LFI/RFI in and off- itself is usually not that impactful. If we find this issue we should try to find files on the system that we should not be able to access like private pictures of other users or we should try to include a file which executes remote code execution. This can cause use to create a reverse shell allowing us access on the server. If you ever achieve this, you should stop and report, don't explore a production server with the risk of seeing data you should not or crashing the production server.

## OS Command injection

This is the last vulnerability type i check for. The only way to check for this issue type is to fuzz all of those parameters with your fuzzing list that you created in the OS command injection section. To do this I use the intruder tool that's built into burp.

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy **Intruder** Repeater Sequencer Decoder Comparer Extender Project options User options Authorize AutoRepeater

1 x 2 x ...

Target Positions **Payloads** Options

**?** **Payload Sets** Start attack

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 169  
 Payload type: Simple list Request count: 1,859

---

**?** **Payload Options [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste (base)))))))

Load ... (base)|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0...

Remove (base)| ping -n 30 127.0.0.1 |

Clear (base)& ping -i 30 127.0.0.1 &

(base)& ping -n 30 127.0.0.1 &

(base); ping -c 5 127.0.0.1 ;

(base)%0a ping -i 30 127.0.0.1 %0a

Add

Add from list ...