

A dimly lit desk setup featuring a laptop in the center, a desk lamp to the left, a wooden mannequin on the far left, an hourglass, a black mug, a white mug with a colorful design, a black mug, and a mouse on a mousepad to the right. The laptop screen displays a video conference interface with the date 'Nov 29, 2023' at the bottom.

Workshop on Web Security

Nov 29, 2023

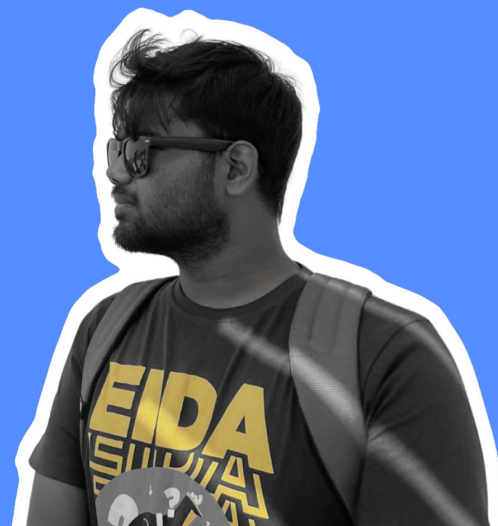
Introduction

Hi, I am **Ataf**

- BUET CSE'17 Batch, Ex-BUETSEC
- Secure Software Dev. @ OpenRefactory
 - Find & Fix Bugs in Open Source Projects
- Contributor/Member @ RevoltZero
 - Share & Practice Knowledge

GitHub : @fazledyn, @fazledyn-or

Email : rabidahamed@gmail.com



We'll Be Doing Today

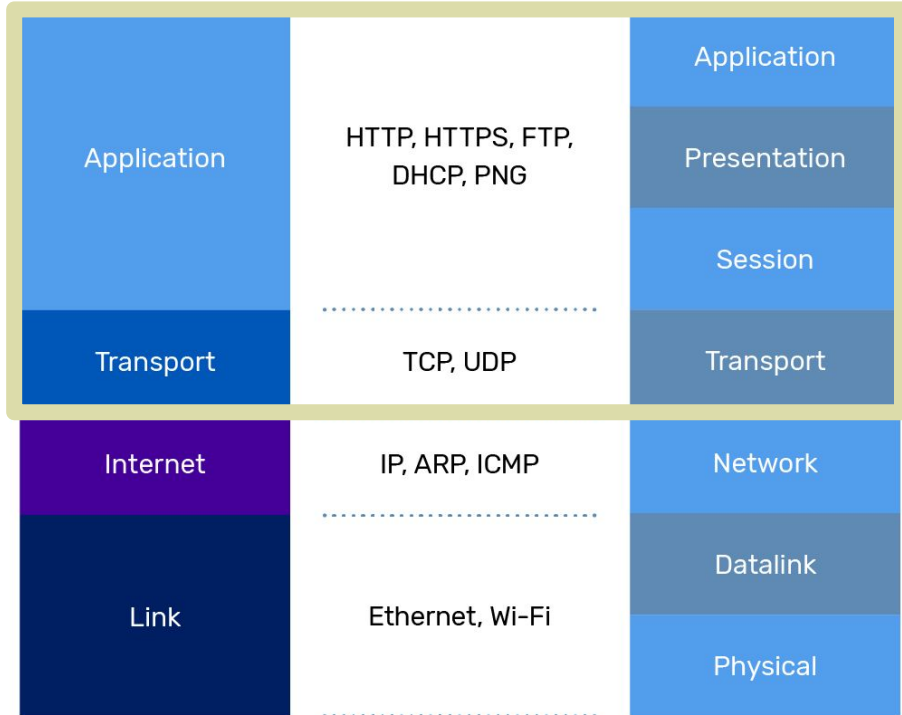
- Different Types of HTTP Requests
- The Browser Environment
- App #1: Cookie
- App #2: Flask Debug Mode
- App #3: Fuzzing
-

Network Model

TCP/IP Model

Protocols and
Services

OSI Model



For Web Security

- HTTP/HTTPS
- TCP, UDP
- FTP

of Ports

65536

0-65535

Different Types of HTTP Requests

Mainly FOUR types of request (methods)

- GET
- POST
- PUT
- DELETE

Others

- PATCH
- HEAD
- OPTIONS and so on

Read more: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

Different Types of HTTP Requests

Mainly FOUR types of request (methods)

- GET → to GET some data from server
- POST → to POST some data to server
- PUT
- DELETE

Others

- PATCH
- HEAD
- OPTIONS and so on

Read more: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

Different Types of HTTP Requests

Mainly FOUR types of request (methods)

- GET → to GET some data from server
 - Parameters are in URL
 - No payload/body
 - Limited URL length
 - https://store.com/find_products?name=Tshirt&size=XL
- POST → to SEND some data to server
 - Parameters are in URL
 - Data in payload/body
 - https://store.com/add_product
Data in payload

The Browser Environment

Let's start with a sample code -

Example #1

Let's run the code

- 1) In terminal using NodeJS
- 2) In browser

```
1
2  const fetch = require("node-fetch");
3  const url = 'https://example.com';
4
5  fetch(url)
6  .then((res) => {
7    |   return res.text();
8  })
9  .then((data) => {
10 |   console.log(data);
11 })
12 .finally(() => {
13 |   console.log("fetch completed");
14 })
15
```


The Browser Environment

Let's start with a sample code -

Example #1

Let's run the code

1) In terminal using NodeJS

2) In browser

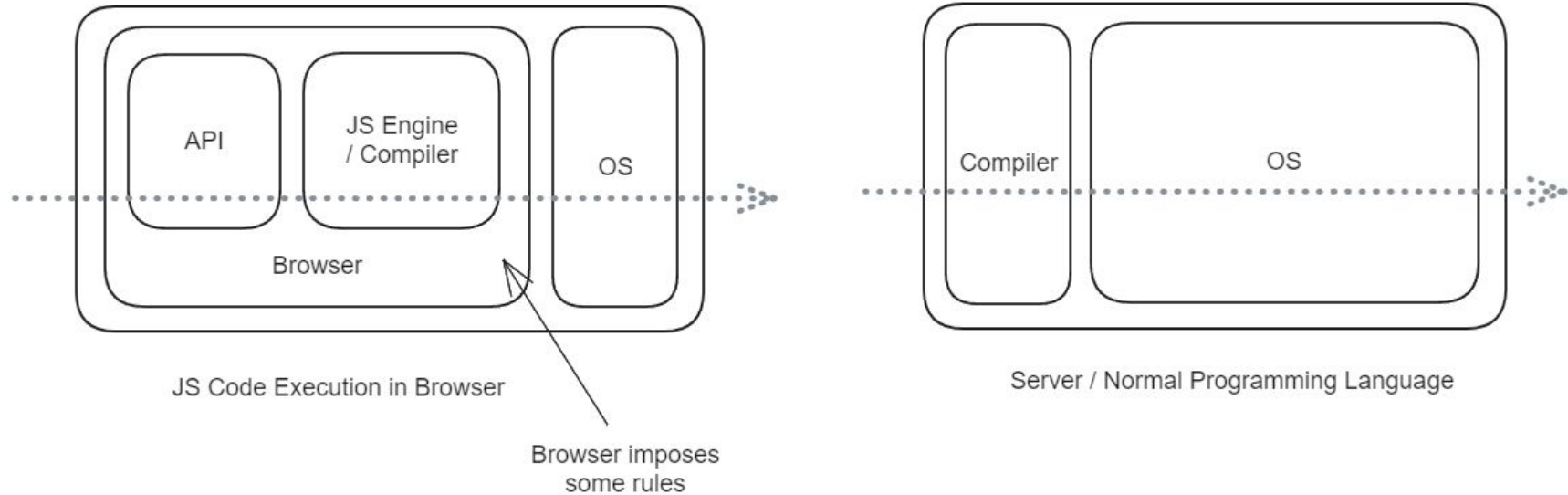
Same code but

why different output?

```
1
2  const fetch = require("node-fetch");
3  const url = 'https://example.com';
4
5  fetch(url)
6  .then((res) => {
7    |   return res.text();
8  })
9  .then((data) => {
10 |   console.log(data);
11 })
12 .finally(() => {
13 |   console.log("fetch completed");
14 })
15
```

The Browser Environment

Depends on How JavaScript Actually Works -



The Browser Environment

1) So, Different Environment = Different Output

2) But what are those rules?

- No File System Access *
- No Socket Connection
- No Threading/Subprocess
- Follow [CSP](#), [Same Origin Policy](#), etc.

3) Browser Plugins

Read More: https://infosec.mozilla.org/guidelines/web_security

The Browser Environment

Let's start with a sample code -

Example #2

Let's run these codes

```
1 import requests
2
3 url = "http://localhost:5000/"
4 res = requests.get(url)
5 print(res.text)
6
```

```
1 from flask import Flask, request
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     print("*****")
8     print(request.headers)
9     print("*****")
10    return "Hello World"
11
12 if __name__ == "__main__":
13    app.run(debug=True)
```

The Browser Environment

Let's start with a sample code -

Example #2

Let's run these codes

Different outputs here too!

```
1 import requests
2
3 url = "http://localhost:5000/"
4 res = requests.get(url)
5 print(res.text)
6
```

```
1 from flask import Flask, request
2
3 app = Flask(__name__)
4
5 @app.route("/")
6 def index():
7     print("*****")
8     print(request.headers)
9     print("*****")
10    return "Hello World"
11
12 if __name__ == "__main__":
13    app.run(debug=True)
```

Port Scanning Using `nmap`

1) Follow On Screen

a) Server 1

b) Server 2

Example App #1: Cookie

- 1) Challenge URL: <http://23.251.153.217:6000/>
- 2) Follow On Screen

Example App #1: Cookie

- 1) Challenge URL: <http://23.251.153.217:6000/>
- 2) Follow On Screen
- 3) JWT: JSON Web Tokens
 - Format: `{scheme/alg}`.`{payload/data}`.`{signature}`
 - If you find anything, try breaking it

Example App #1: Cookie

- 1) Challenge URL: <http://23.251.153.217:6000/>
- 2) Follow On Screen
- 3) JWT: JSON Web Tokens
 - Format: `{scheme/algo}.{payload/data}.{signature}`
 - If you find anything, try breaking it
- 4) Don't trust JWTs blindly! Always verify the signature!

Read more: <https://jwt.io/>

Example App #2: Flask Debug Mode

- 1) Challenge URL: <http://23.251.153.217:3000/>
- 2) Follow On Screen

Example App #2: Flask Debug Mode

1) Challenge URL: <http://23.251.153.217:3000/>

2) Follow On Screen

3) Follow these resources, solve on your own

<https://www.bengrewell.com/cracking-flask-werkzeug-console-pin/>

<https://www.youtube.com/watch?v=jwBRgaIRdgs>

Example App #3: Fuzzing

1) Challenge URL: <http://23.251.153.217:8000/>

2) Follow On Screen

- Fuzzing Tools:

- dirb
- gobuster
- nikto

Example App #3: Fuzzing

1) Challenge URL: <http://23.251.153.217:8000/>

2) Follow On Screen

- Fuzzing Tools:

- \$ dirb <http://23.251.153.217:8000> /usr/share/dirb/wordlists/common.txt

- \$ gobuster

- \$ nikto

- Difference

- `dirb` can search iteratively

- `gobuster` is multithreaded

GitHub Codespaces

A screenshot of a GitHub Codespaces environment. The interface is dark-themed. At the top, the GitHub logo and the word "Codespaces" are displayed. Below this is a code editor window showing a file named "emotion.rb" with the following code:

```
73 attr_reader :label
74
75 attr_reader :pronounceable_label
76
77 # Public: Get the Emoji that this reaction's content represents.
78 #
79 # Returns an Emoji.
80 attr_reader :emoji_character
81
82 def initialize(content:, label: nil, pronounceable_label: nil, emoji_character: nil)
83   @content = content
84   @label = label || @content
85   @pronounceable_label = pronounceable_label || @label
86   @emoji_character = emoji_character || Emoji.find_by_alias(@content)
87   @platform_enum = @pronounceable_label.gsub(" ", "").upcase
88
89   freeze
90 end
91 end
92
93 Emotion.create(content: "+1", pronounceable_label: "thumbs up")
94 Emotion.create(content: "-1", pronounceable_label: "thumbs down")
```

The left sidebar shows an "EXPLORER" view with a file tree for a project named "GITHUB". The tree includes a ".devcontainer" folder with files like "build-devcontainer.sh", "devcontainer.json", "on-create-command.sh", "post-attach-command.sh", "setup-devcontainer.sh", and "ssrh", as well as a "welcome_message.txt" file. Other folders include ".github", ".vscode", "app", "api", "assets", "components", "controllers", "helpers", "jobs", "mailers", and "models". On the far left, there are several informational cards: "CPUs up to 32 cores", a "DEBUG CONSOLE" section with "TestSynchronizer - Manual sync", and a "Memory up to 64 GB" card. At the bottom right, there is a "main" branch selector and a "tabbed-nav" section with "0 + 1 +", "0 + 6 +", and "2 + 1 +" indicators.

GitHub Codespaces

- 1) Install [GitHub CLI](#)
- 2) Install [VSCode](#) (not Codium)
- 3) Login to your account
- 4) Follow On Screen

GitHub Codespaces

- 1) Install [GitHub CLI](#)
- 2) Install [VSCode](#) (not Codium)
- 3) Login to your account
- 4) Follow On Screen

Use it as a VPS!

- Run Scripts, Fuzzers
- Host your webapps

The Hacker's Approach

- a) Perform automation, save time
 - i) nmap, fuzzers, scripts, ZAP

- b) Do it on a server than your PC
 - i) Better bandwidth
 - ii) Better performance

- c) Perform recon
 - i) [Censys](#), [Shodan](#) for finding info
 - ii) Get IP address !

The Developer's Approach

- a) Use SSL/TLS Certificates - *Free*
 - i) [ZeroSSL](#), [Cloudflare](#), etc.

- b) Use DDoS Protection
 - i) [Cloudflare](#) - *Free*
 - ii) Never expose your IP address
 - If HTTP exposed, can perform DDoS on IP
 - DDoS Protection saves only requests coming to domain

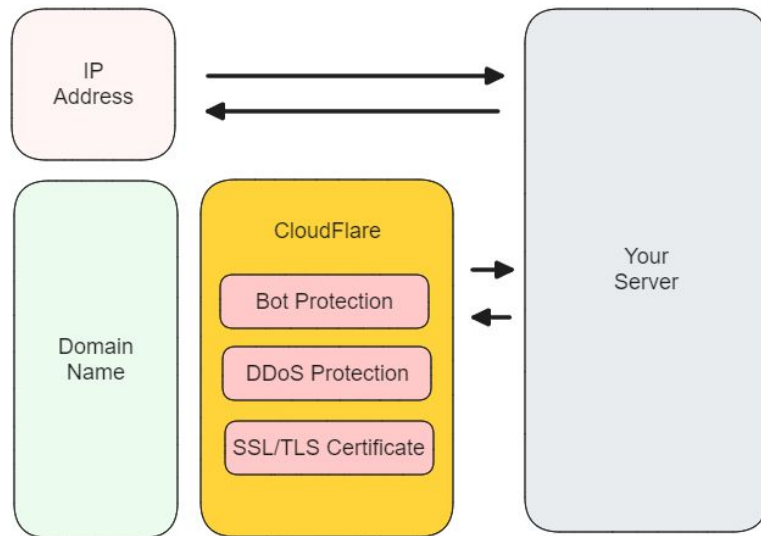
- c) Don't expose unnecessary ports. Follow [HSTS](#).

- d) Use Encryption, JWTs carefully

The Developer's Approach

b) Use DDoS Protection

- i) **Cloudflare** - *Free*
 - ii) Never expose your IP address
- If HTTP exposed, can perform DDoS on IP
 - DDoS Protection saves only requests coming to domain



Questions?

- Knock Anytime on [Discord](#)
- BUET Cyber Drill Practice Zone @ Facebook

- Resources To Study
 - [CS 253: Web Security](#) by Stanford CS
 - [PortSwigger Labs](#)
 - MDN Docs

- Contents: <https://github.com/buetsec/web-security-workshop-2023>

End Of Slide

Thank You