INTRODUCTION TO

WEB EXPLOITATION

ABOUT ME

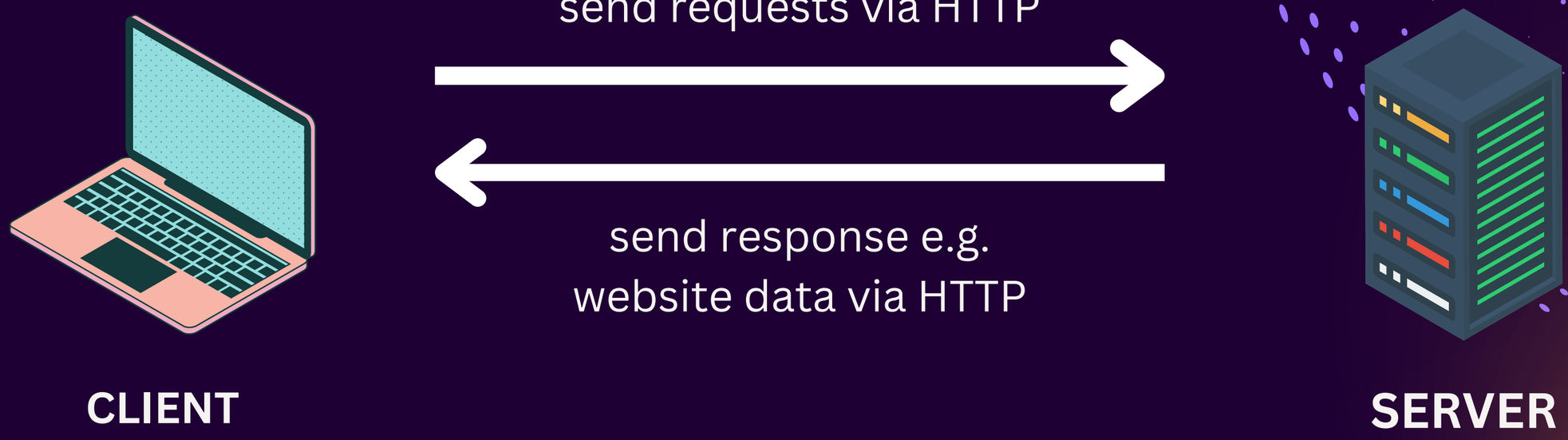
- Call me Danish!
- 2nd year computer science student specializing in Network & Security.
- not a pro ctf player :(
- Occasional Web, RedTeam CTF Player
- Linkedin - in/jerit3787
- Portfolio - www.danplace.tech
- CTF writeups - ctf.danplace.tech

CONTENT

HOW THE WEB WORKS

CLIENT-SERVER MODEL

The web operates on a client-server model. A **client** (usually a browser or script) initiates requests to a **server** which processes the requests and responds with appropriate content (HTML, JSON, files, etc.)



HOW THE WEB WORKS

CLIENT-SERVER MODEL

CLIENT

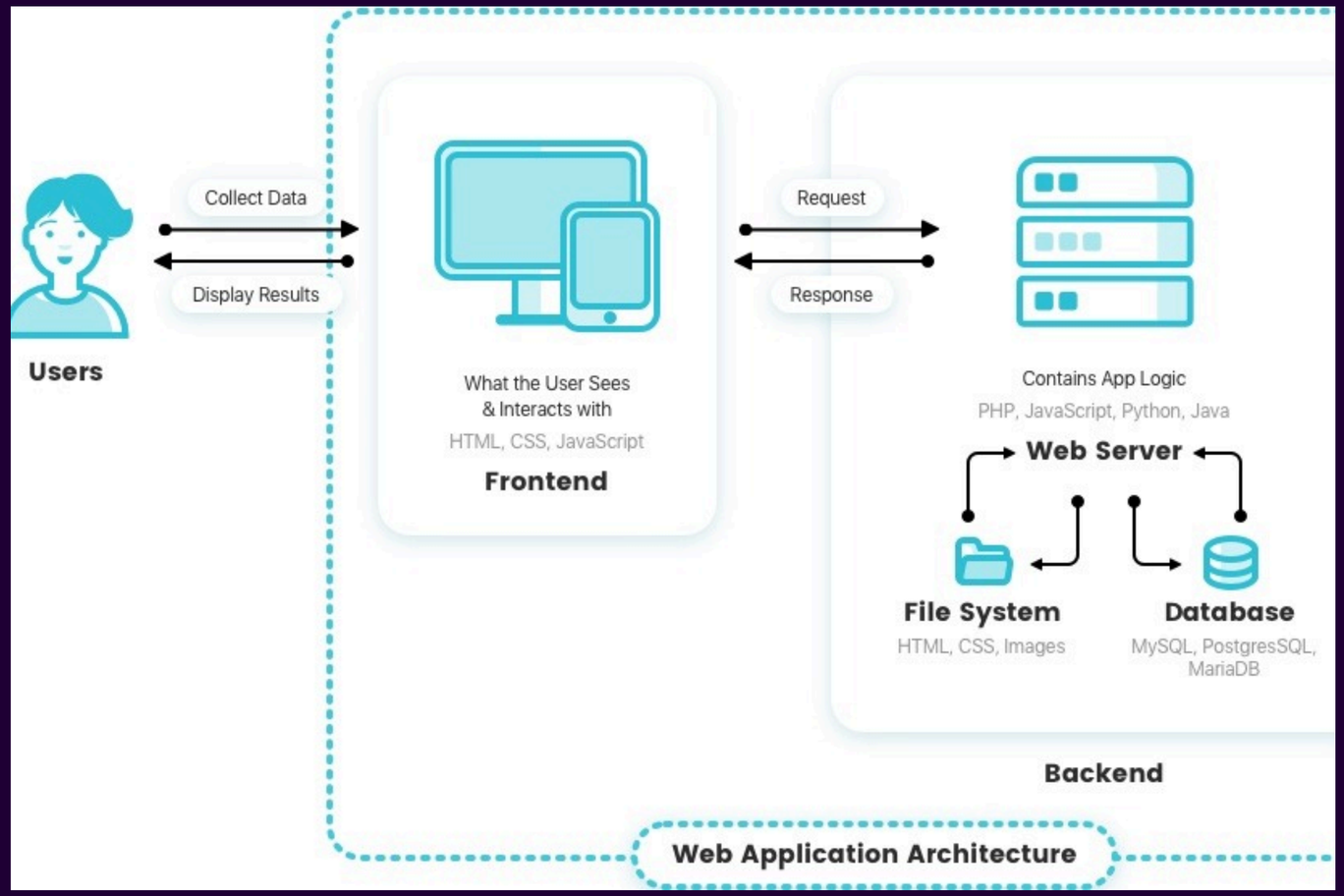
- Example: Chrome, Firefox, curl, Postman
- Role: Send HTTP requests (GET, POST, etc.)
- Often handle executing scripts (e.g. Javascript)

SERVER

- Example: Apache, Ngix serve static content or pass requests to application backends
- Application servers (PHP, Node.js, Python) handle dynamic content and buisness logic
- Handle routing, processing input, accessing database and rendering output

HOW THE WEB WORKS

CLIENT-SERVER MODEL



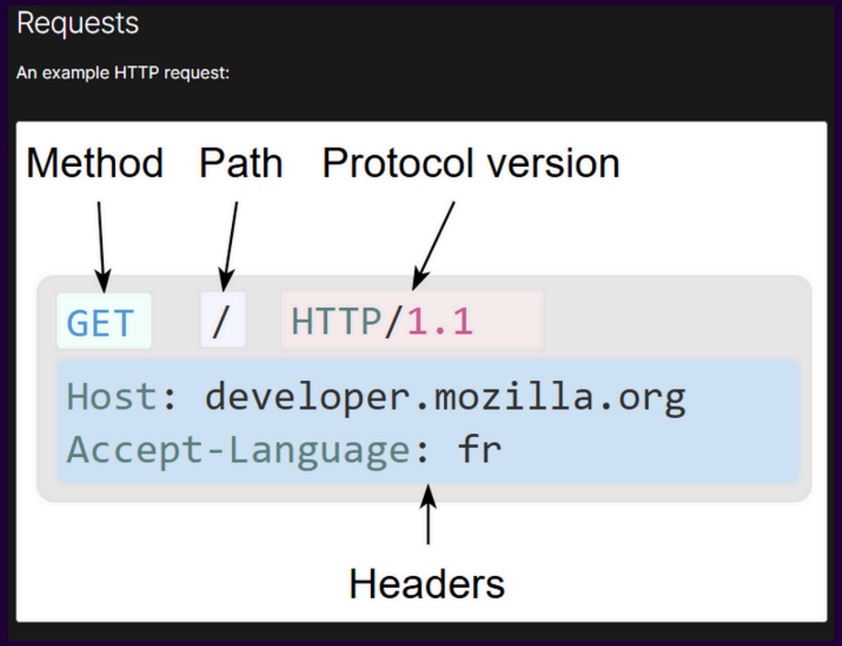
Example: Facebook Login Page

The frontend, written in HTML, CSS, and JavaScript, is what users see and interact with in their browser, like the Facebook login page, where you input your email and password. When you click the button, you generate a **HTTP Request**

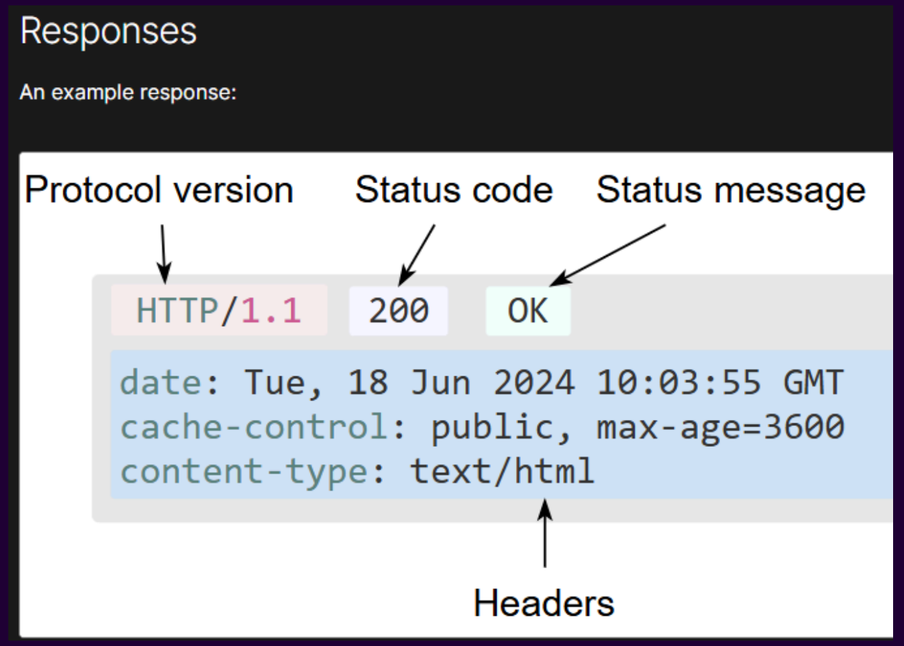
This data(HTTP message) is sent to the backend, written in languages like Python, PHP, or Java, which processes the login request by validating your credentials, checking the database for a match, and granting access if valid. The web server will return **HTTP Response** to be understand by your browser

HOW THE WEB WORKS

CLIENT-SERVER MODEL - HTTP PROTOCOL



An HTTP request is a message sent by a client (usually a web browser) to a server to request a resource, such as a web page or an **API endpoint,** typically including methods like GET, POST, PUT, or DELETE.



An HTTP response is a message sent by the server back to the client, providing the requested resource or information, and typically includes status codes (like 200 for success or 404 for not found) and the requested data.

HOW THE WEB WORKS

PROGRAMMING LANGUAGE USED

FRONTEND DEVELOPMENT

- Languages: HTML (structure), CSS (style), JS (behavior)
- Frameworks: React, Angular, Vue for building dynamic UIs

BACKEND DEVELOPMENT

- Languages: PHP, Node.js, Python (Flask/Django), Ruby and etc.
- Purpose: Handle logic, interact with databases, manage sessions and handle authentication

HOW THE WEB WORKS

DEVELOPMENT AND VULNERABILITIES

How Vulnerabilities Arise

- Speed over security in modern development
- Lack of input validation and output encoding
- Misunderstanding of complex frameworks
- Blind trust in third-party dependencies

Examples

- Login logic not checking passwords correctly → Auth bypass
- Database queries directly including user input → SQL injection
- File upload feature trusting file extensions → RCE risk

Key takeaway → Most web bugs are coding/design mistakes made by developers

INTRO TO CTF

WHAT IS CTF?

- **Capture The Flag Explained**
  - Cybersecurity competition with hidden "flags" to find
  - Challenges simulate real-world scenarios
- **CTF Formats**
  - **Jeopardy-style**: Solve static challenges for points
  - **Attack-Defense**: Teams attack others' infrastructure while defending their own
  - **King of the Hill**: Control a vulnerable box while stopping others
- **Scoring**
  - Based on difficulty, time taken, and number of solves

INTRO TO CTF

WEB CHALLENGES IN CTF

- **Web Challenges Typically Include:**
  - Exploiting logic bugs, authentication flaws, or unsafe input handling
  - Reverse engineering JavaScript or traffic
  - Accessing admin or flag endpoints
- **Required Skills**
  - Understanding of web architecture
  - Familiarity with HTTP traffic (Burp Suite, dev tools)
  - Ability to write payloads or scripts to automate exploitation
- **Common Tools**
  - Burp Suite, curl, Postman
  - Python requests, browser dev tools
  - SQLMap, XSStrike, custom scripts

INTRO TO CTF

UNDERSTANDING WEB EXPLOITATION

- **Attack Surface Overview**
  - Input fields, query strings, headers, cookies, uploads
  - Misused trust boundaries between client and server
- **Common Exploitable Behaviors**
  - Trusting user input
  - Using insecure defaults (e.g., eval, unsanitized rendering)
  - Leaking sensitive data in error messages
- **Attacker Mindset**
  - What can I control?
  - What assumptions does the server make?
  - How can I manipulate logic/data flow?

INTRO TO CTF

COMMON EXPLOITS IN WEB

1. **SQL Injection (SQLi)** – Manipulate database queries via input
2. **Cross-Site Scripting (XSS)** – Inject JS into pages viewed by others
3. **Cross-site Request Forgery (CSRF)** – Exploit logged-in users to perform actions
4. **Command Injection** – Execute OS-level commands via user input
5. **Path Traversal** – Access unintended files using ../
6. **File Upload Issues** – Upload scripts disguised as images
7. **Server-Side Template Injection (SSTI)** – Inject into server-side templates (e.g., {{7*7}})
8. **Insecure Deserialization** – Trigger object execution upon decoding
9. **Logic Bypass** – Abuse logic to skip checks or reveal data
10. **Server-Side Request Forgery** (SSRF) – Force server to fetch internal resources.
11. **Vulnerable and Outdated Components** – Use of outdated libraries or frameworks with known exploits (e.g., old versions of jQuery, WordPress, or vulnerable PHP modules)

INTRO TO CTF

COMMON EXPLOITS IN WEB

SQL INJECTION

- **Definition:** Injecting SQL queries through user input to manipulate database execution.
- **Impact**: Dump credentials, bypass auth, modify data.
- **Examples**: ' OR 1=1 --, UNION SELECT, time-based blind SQLi.
- **CTF Tips**: Look for login forms, search bars, GET parameters.

INTRO TO CTF

COMMON EXPLOITS IN WEB

SQL INJECTION

```
$username = $_POST['username'];
$password = $_POST['password'];
$sql = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
$result = mysqli_query($conn, $sql);
```

INTRO TO CTF

COMMON EXPLOITS IN WEB

CROSS-SITE SCRIPTING (XSS)

- **Definition**: Injecting malicious JavaScript into web pages viewed by others.
- **Types**: Reflected, Stored, DOM-based.
- **Impact**: Session hijack, redirect, keylogger.
- **CTF Tips**: Inspect HTML rendering, use <script>alert(1)</script>, look for unsanitized input.

INTRO TO CTF

COMMON EXPLOITS IN WEB

CROSS-SITE REQUEST FORGERY (CSRF)

- **Definition**: Forcing a logged-in user to unknowingly perform an action.
- **Impact**: Change password/email, perform transactions.
- **CTF Tips**: Look for actions without CSRF tokens or referer checks.

INTRO TO CTF

COMMON EXPLOITS IN WEB

COMMAND INJECTION

- **Definition**: Injecting OS commands via input fields.
- **Impact**: Arbitrary code execution, file read/write.
- **CTF Tips**: Try ; ls, && whoami, URL encoded payloads.

INTRO TO CTF

COMMON EXPLOITS IN WEB

PATH TRAVERSAL

- **Definition**: Navigating out of intended directories using ../
- **Impact**: Read /etc/passwd, source code disclosure.
- **CTF Tips**: Check download/view file features.

INTRO TO CTF

COMMON EXPLOITS IN WEB

FILE UPLOAD VULNERABILITIES

- **Definition**: Uploading malicious files disguised as safe ones.
- **Impact**: Remote code execution, XSS, defacement.
- **CTF Tips**: Upload .php, bypass extension checks with file.php.jpg

INTRO TO CTF

COMMON EXPLOITS IN WEB

SERVER-SIDE TEMPLATE INJECTION (SSTI)

- **Definition:** Injecting code into template engines (e.g., Jinja2).
- **Payloads**: {{7*7}}, {{config}}, {{request.args.q}}
- **CTF Tips**: Look for dynamic renderings with templates.

INTRO TO CTF

COMMON EXPLOITS IN WEB

SERVER-SIDE TEMPLATE INJECTION (SSTI)

```
from flask import Flask, request, render_template_string

app = Flask(__name__)

@app.route("/greet")
def greet():
    name = request.args.get("name", "Guest")
    template = f"Hello {name}!"
    return render_template_string(template)
```

INTRO TO CTF

COMMON EXPLOITS IN WEB

SERVER-SIDE TEMPLATE INJECTION (SSTI)

from client → /greet?name={{7*7}}

displayed to client → Hello 49!

INTRO TO CTF

COMMON EXPLOITS IN WEB

INSECURE DESERIALIZATION

- **Definition**: Supplying manipulated serialized data.
- **Impact**: Arbitrary code execution.
- **CTF Tips**: Look for base64 blobs, JWTs, pickles.

INTRO TO CTF

COMMON EXPLOITS IN WEB

SERVER-SIDE REQUEST FORGERY (SSRF)

- **Definition**: Exploit where server makes unauthorized or internal requests
- **Impact**: Access internal services (admin panels, APIs), Leak metadata from cloud services (AWS, GCP), Chain with file protocols or other bugs to gain RCE
- **How It Happens**
  - App fetches user-supplied URLs (e.g., for preview or API proxy)
  - No validation → attacker can request localhost, internal IPs, or cloud services

INTRO TO CTF

COMMON EXPLOITS IN WEB

VULNERABLE AND OUTDATED COMPONENTS

- **Definition**: Usage of components with known vulnerabilities.
- **Examples**: jQuery <3.5.0 (XSS), PHPMyAdmin, WordPress plugins.
- **CTF Tips**: Look for version leaks, known CVEs, changelogs.

INTRO TO CTF

TRENDS IN WEB EXPLOITATION

- OWASP Top 10 Highlights
  - A01: Broken Access Control
  - A03: Injection (SQLi, XSS)
  - A05: Security Misconfiguration
  - A06: Vulnerable and Outdated Components
  - A10: SSRF
- CTF Trends
  - Creative use of SSTI and chained vulnerabilities
  - Exploits using obscure or undocumented features
  - Abuse of modern JS frameworks, JWTs, APIs
  - More real-world style flaws in cloud/web environments

INTRO TO CTF

GENERAL STEPS

- Enumeration
  - URL/Parameters
  - Technologies Used
  - Source Code(Browser Dev Tools)
  - HTTP messages
- Mapping Functionality
  - Test out all functionality manually with normal response and observe
  - Understand also API interaction and parameters being used
- Identify Vulnerabilities
  - Test out different general payload and observe the differences with normal
  - response
- Craft Payload
  - Craft the payload to get the flag
  - Might need to bypass certain restrictions

INTRO TO CTF

TIPS AND TRICKS - BLACKBOX CHALLS

- Playing with HTTP Headers:
  - Modify headers like X-Forwarded-For, User-Agent, or Referer to test for trust-
  - based logic flaws or bypass restrictions.
- Exploring Random Files:
  - Perform directory brute-forcing to discover hidden or sensitive files.
  - Common targets:
    - robots.txt, .git, .conf
- Modifying Parameters in URL
- Use browser developer tools to:
  - View Network Tab, Cookies, and Hidden values, comments.
- Testing Application Functionality:
  - Experiment with different actions on the app (e.g., skipping steps, trying edge
  - cases).
  - Guess what the application might allow based on observed behavior

INTRO TO CTF

TIPS AND TRICKS - WHITEBOX CHALLS

- Review the Source Code:
  - Inspect backend code (if provided) or frontend logic via "View Source."
  - Look for hardcoded credentials, endpoints, or weak implementations.
- Check for Input Filtering:
  - Test if input sanitization or filtering is present and if it can be bypassed.
- Identify Vulnerable Keywords:
  - Look for potentially dangerous functions or patterns, such as:
  - eval() or exec() in backend code (e.g., Python, PHP).
  - unserialize() in older PHP versions.
- Understand Each Functionality:
  - Map out how functions interact with user input and external components.
- Look for exploitable chains:
  - Identify areas where chained exploits may arise, which are usually more
  - complex.

HANDS-ON LAB