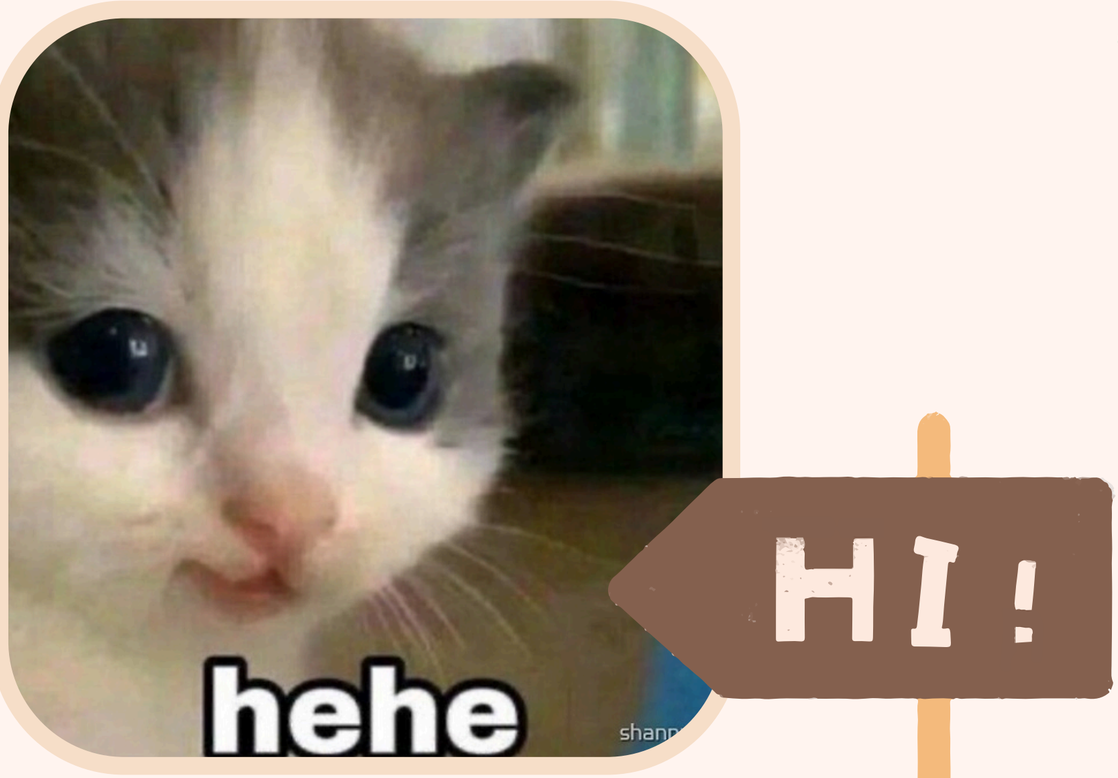
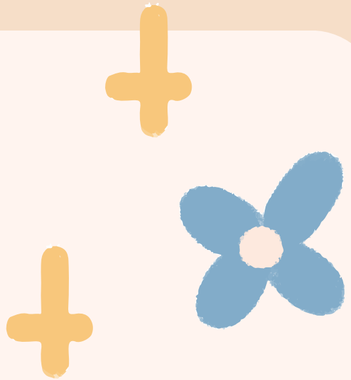
Web CTF Introduction

By JY

About me

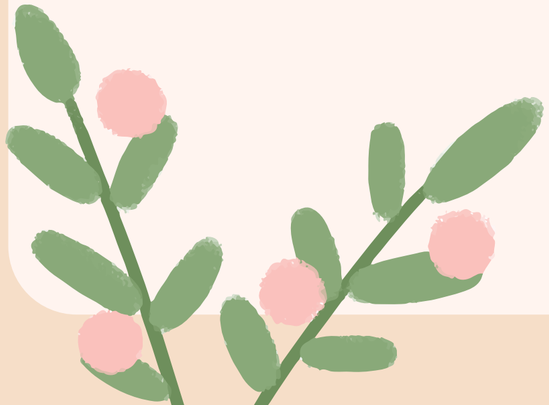
- Final Year Undergraduate in UM on Computer System and Network
- Occasional Web, RedTeam CTF Player
- MCC2023 Participant
- Linked-in(https://www.linkedin.com/in/tang-jia-yang-a13696227/)
- Blog(https://hehejy.github.io/)

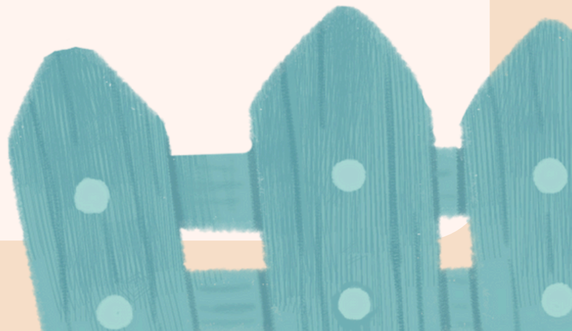




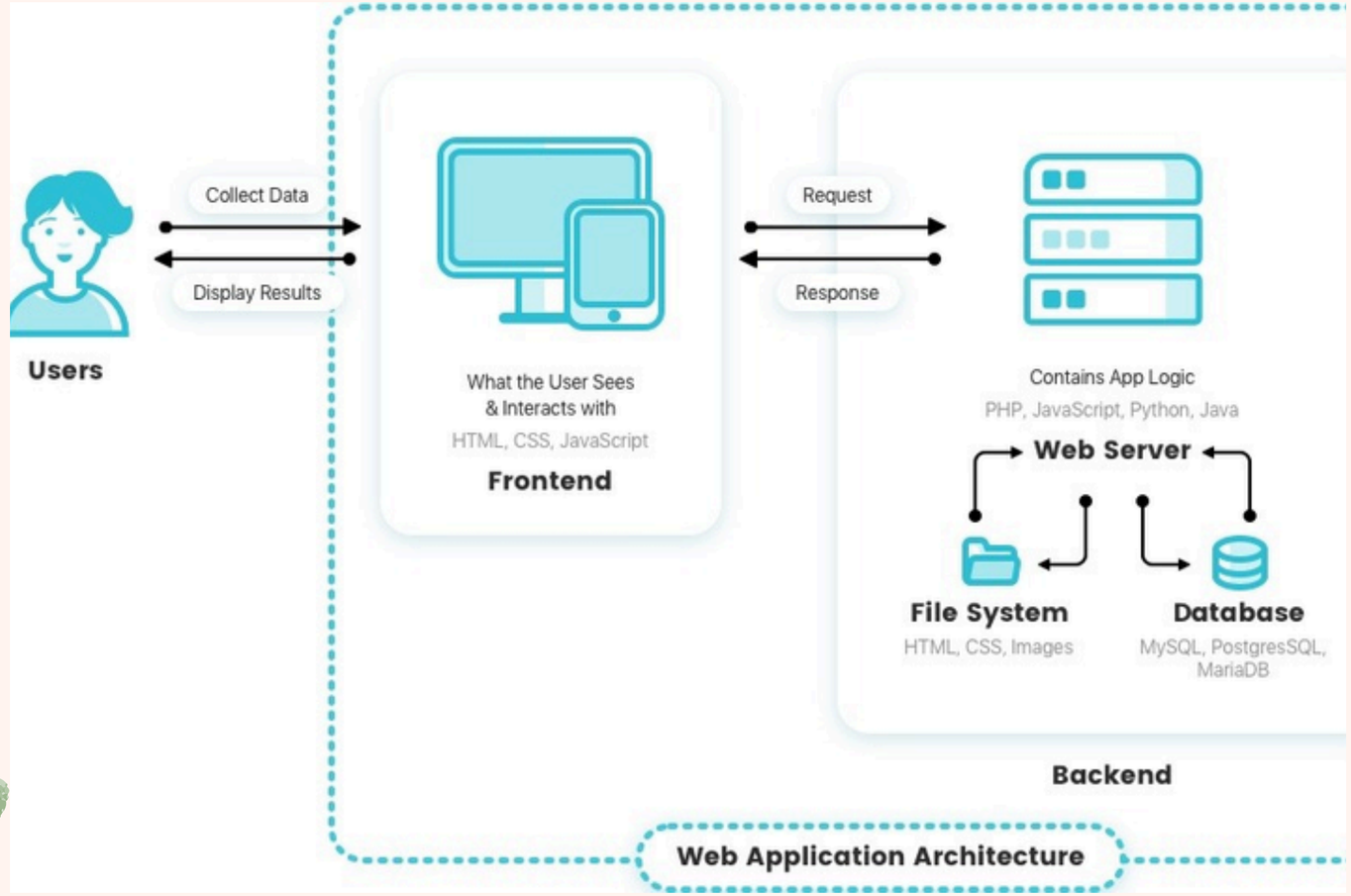
Introduction

A Web CTF (Capture the Flag) is a competitive cybersecurity event focusing on web application security. Participants solve challenges by **identifying** and **exploiting** vulnerabilities in simulated web apps to retrieve hidden "flags." It is a “must” category in most CTFs. You need **time** to grow your skills, so don’t be disappointed when getting low marks for your first few international web CTF.





Understanding the Web Application

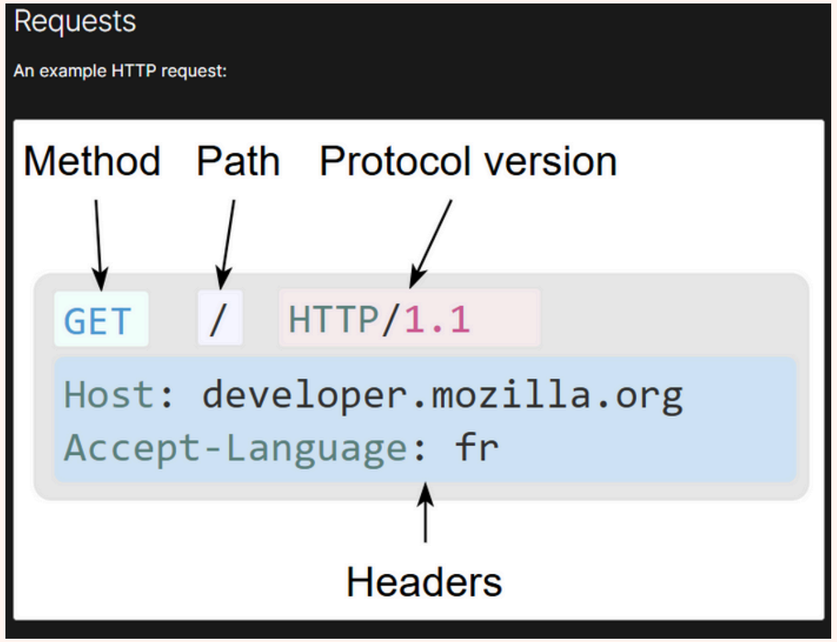


Example: Facebook Login Page

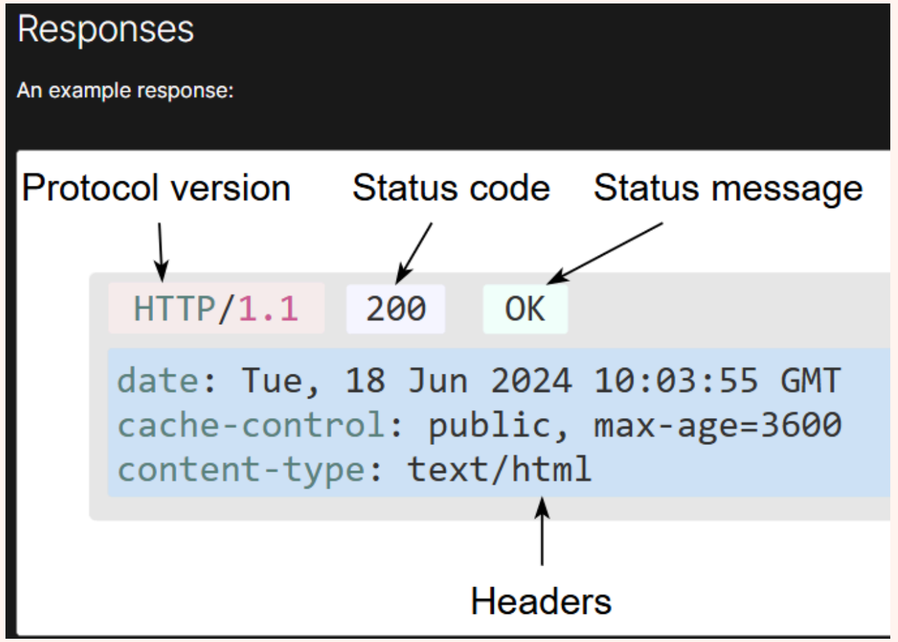
The frontend, written in HTML, CSS, and JavaScript, is what users see and interact with in their browser, like the Facebook login page, where you input your email and password. When you click the button, you generate a **HTTP Request**

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

HTTP Messages



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.

Web App Common Vuln

Broken Access Control(IDOR)

a. Example 1 : A web application allows any user to access the /admin panel by directly navigating to the URL, bypassing authentication.
b. Example 2 : A web application uses a parameter like user-id in cookies, URL query parameters, or POST data.
c. Hint
    i. Identify and test parameters or cookies, authorization header that represent user identifiers
    ii. Look for predictable patterns in user IDs
    iii. Could be reusing session cookies or bruteforcing 2FA

Web App Common Vuln

Cryptographic Failures

a. Example 1 : A web application uses a weak encryption scheme to encode sensitive data in cookies, such as user_id=1234 (Base64 encoded).
b. Example 2 : The application accepts a JWT signed with one algorithm (e.g., HS256) but verifies it using another algorithm (e.g., RS256).
c. Hint
    i. Test modifying the decoded value and re-encoding it to check for server-side validation
    ii. Look for patterns or use tools like cyberchef to test other reversible encodings (e.g., Hex, URL encoding)

Web App Common Vuln

Injection

a. Example 1 : A login page accepts user input for username and password without sanitizing the inputs.
b. Example 2 : Could be other injection type like code injection, ldap injection, xxe injection, nosql injection, template injection, directory traversal, xss
c. Hint
    i. Locate fields, headers, parameters, cookies, or file uploads where user input is accepted.(Know what your target accept first)
    ii. Try generic payloads that apply across injection
    iii. Analyze Responses(Error Type, Delays(especially blind injection) and any abnormal response)

Web App Common Vuln

Security Misconfiguration

a. Example 1 : A web server has directory listing enabled, exposing sensitive files and directories to anyone accessing them.
b. Example 2 : An application uses a misconfigured AWS S3 bucket for storage. The bucket allows public access, enabling attackers to read or write files.
c. Example 3: The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users.
d. Example 4: File Upload Vulnerability(allow some other file type to run)
e. Hint
    i. Read the source code given, use browser developer tool to see all available endpoints
    ii. Bruteforce(if allowed)
    iii. Read the manual and use related tool to solve the challenge

Web App Common Vuln

Vulnerable and Outdated Components

a. Example 1 : The target web server is running Apache Struts 2.3.15, which is vulnerable to a well-known CVE (CVE-2017-5638) that allows RCE through crafted Content-Type headers.
b. Example 2 : A web application uses an outdated version of PHP (e.g., PHP 5.6) that includes a function vulnerable to exploitation. For instance, unserialize() in older PHP versions can lead to remote code execution (RCE) if malicious serialized data is passed.
c. Hint
    i. Identify the Version/Look for unsanitized user input passed to a function
    ii. Find related CVE or blog and manual to understand why the exploit works
    iii. Craft the exploit

Web App Common Vuln

Server-Side Request Forgery

a. Example 1 : The application fetches content from a URL parameter, and internal endpoints are accessible
b. Example 2 : A web application allows users to upload files by providing a URL for remote files. The backend fetches the file using a server-side request, but it does not properly validate the requested URL. The internal services require IP-based authentication, but the SSRF allows bypassing this by connecting to internal endpoints.
c. Hint
    i. Identify input fields or parameters that accept URL
    ii. Use local IPs or well-known internal endpoints to test server behavior
    iii. Bypass Validations(Most probably)
    iv. Get response

Web App Common Vuln

Insecure Design

a. Example 1 : The application uses a weak design where a user can change their password without verifying their current password.
b. Example 2 : An e-commerce application calculates the total payment amount on the client-side and sends it to the server as part of the checkout process
c. Hint
    i. Understand the Workflow
    ii. Find the injection point
    iii. Test for Logical Flaws
    iv. Test for other possible edge cases

General Steps

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

Tips and Tricks

Blackbox

- 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.

Tips and Tricks

Whitebox

- 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 Time

How to build Your Notes

1. Login Page
   - Bruteforce, SQL Injection, Hidden Information, Insecure API, Weak Session Management, Authentication Bypass
2. Registration Page
   - Account Enumeration, Weak Password Policies, Email Verification Bypass, Default Role Assignment
3. JWT Tokens
   - Algorithm Confusion, Weak Signature Validation, Token Replay, Missing Expiry Validation
4. Search Functionality
   - SQL Injection, Reflected XSS, Stored XSS, Information Disclosure
5. File Upload
   - Unrestricted File Upload, MIME Type Validation Bypass, Path Traversal via Filename, Malicious Script Execution
6. URL Patterns
   - File Inclusion (LFI/RFI), Directory Traversal, Open Redirects, SSRF (Server-Side Request Forgery)

How to build Your Notes

1. API Endpoints
   - IDOR (Insecure Direct Object References), Rate-Limiting Issues, Missing Authentication, Improper Authorization, Broken Access Control
2. Error Messages
   - Information Disclosure, SQL Injection, Debug Mode Enabled, Stack Trace Leakage
3. Source Code Disclosure
   - Hardcoded Secrets, Sensitive Endpoints, Exposed Configuration Files, Misconfigured .git Directories
4. Password Reset
   - Token Replay, Predictable Token Generation, Email Spoofing, Weak Security Questions
5. Session Management
   - Predictable Session Tokens, Session Fixation, Insecure Cookies, Missing Logout Functionality
6. Payment or Cart Systems
   - Logic Flaws, Missing Validation on Prices, Coupon Abuse, Double Spending
7. Third-Party Integrations
   - Outdated Libraries, Unrestricted Access via API Keys, SSRF in External Services, Misconfigured CORS

How to build Your Notes

1. Headers and Cookies
   - Missing Security Headers, XSS via Cookies, Cookie Injection, Weak Content Security Policy (CSP)
2. File Downloads
   - Path Traversal, Unrestricted File Access, Arbitrary File Read, Unvalidated Redirects
3. OAuth/SSO (Single Sign-On)
   - Open Redirects, Token Leakage, Weak Validation of Issuer, CSRF in OAuth Flow
4. CMS (Content Management Systems)
   - Plugin Vulnerabilities, Unpatched CMS Versions, Directory Listing, Admin Panel Exposure
5. Custom Input Fields
   - XSS (Reflected, Stored), SQL Injection, Command Injection, Logic Flaws
6. Forgotten or Hidden Features
   - Debugging Interfaces, Test Endpoints, Backup Files, Legacy Pages
7. Networking Scenarios
   - Open Ports, Weak TLS Configurations, Default SSH Keys, SSRF Exploiting Internal Service

Thank you!

Special Mention: PEKOM, Irfan Haziq(Capang), MCC

Resource Page

https://portswigger.net/web-security/all-material (teach you all logic)
https://owasp.org/
https://academy.hackthebox.com/ (student premium subscriptions)
https://tryhackme.com/
https://github.com/juice-shop/juice-shop (playground)
https://hacker101.com
https://www.bugcrowd.com/hackers/bugcrowd-university/(real environment)
https://pentesterlab.com
https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages