رواد مصر الرقمية

OWASP JUICE SHOP
Security Assessment Findings Report

Supervision : ***Eng. Hesham Saleh***

Date: November 20nd

2025 Project:

Table Of Content

- Finding 50 :
- Finding 51:
- Finding 52 :
- Finding 53:
- Finding 54 :
- Finding 55:
- Finding 56 :
- Finding 57 :
- Finding 58 :
- Finding 59 :
- Finding 60 :
- Finding 61:

Confidentiality Statement

This document is the exclusive property of OWASP Juice Shop and Digital Egypt Pioneers Initiative (DEPI). This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both OWASP Juice Shop and DEPI. OWASP Juice Shop may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period. Time-limited engagements do not allow for a full evaluation of all security controls. DEPI prioritized the assessment to identify the weakest security controls an attacker would exploit. DEPI recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Contact Information

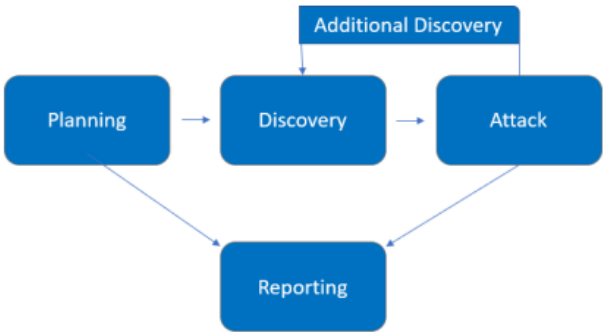
DEPI Trainees			Reviwer &Approval	Position
Name	**Title**	**Email**		
Mohamad Abd El Qader Guda	**Trainee**	**Mohmadabdelqader66@gmail.com**	**Hesham Saleh**	**Instractor**
Anton Nady riyad	**Trainee**	**tonynady446@gmail.com**		
Nadeem Hamdy Mohamed	**Trainee**	**nadimhamdym@gmail.com**		

Mohamed Sami Abdulazeem	**Trainee**	**mohammedsamy650@gmail.com**		
Abdelrhman Khaled Abdelhamid	**Trainee**	**kabdelrhman778@gmail.com**		
	Trainee			

Assessment Overview

From November 10th, 2025 to November 28th, 2025, OWASP engaged DEPI to evaluate the security posture of its infrastructure compared to current industry best practices that included a webapp penetration test. All testing performed is based on the NIST SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customized testing frameworks. Phases of penetration testing activities include the following:

- **Planning** – Customer goals are gathered and rules of engagement obtained.
- **Discovery** – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- **Attack** – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- **Reporting** – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.



Assessment Components

Web app Penetration Test

A web app penetration test emulates the role of an attacker on a web application. An engineer will scan the website to identify potential vulnerabilities and perform common and advanced web attacks, such as: Injections, broken access control and other Cross-site scripting (XSS) attacks, Improper Input Validation, and more. The engineer will seek to gain access to hosts through lateral movement, compromise domain user and admin accounts, and exfiltrate sensitive data.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: Likelihood and Impact:

→ **Likelihood :** measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

→ **Impact** : measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Scope

Assessment	Details
Web-Server Penetration Testing	OWASP JUICE SHOP

✓ In scope

"The scope includes the entire **OWASP Juice Shop** application, all core functionalities, and user-accessible interfaces."

https://juice-shop.herokuapp.com/#/

× Out of scope

Per client request, DEPI did not perform any of the following attacks during testing:

- Denial of Service (DoS)
- Phishing

Executive Summary

DEPI evaluated OWASP's Juice Shop posture through penetration testing from November 10th, 2025 to November 28th, 2025. The following sections provide a high-level overview of vulnerabilities discovered, successful and unsuccessful attempts, and strengths and weaknesses.

Scoping and Time Limitations

Scoping during the engagement did not permit denial of service or social engineering across all testing components.

Time limitations were in place for testing. Webapp penetration testing was permitted for (18) business days.

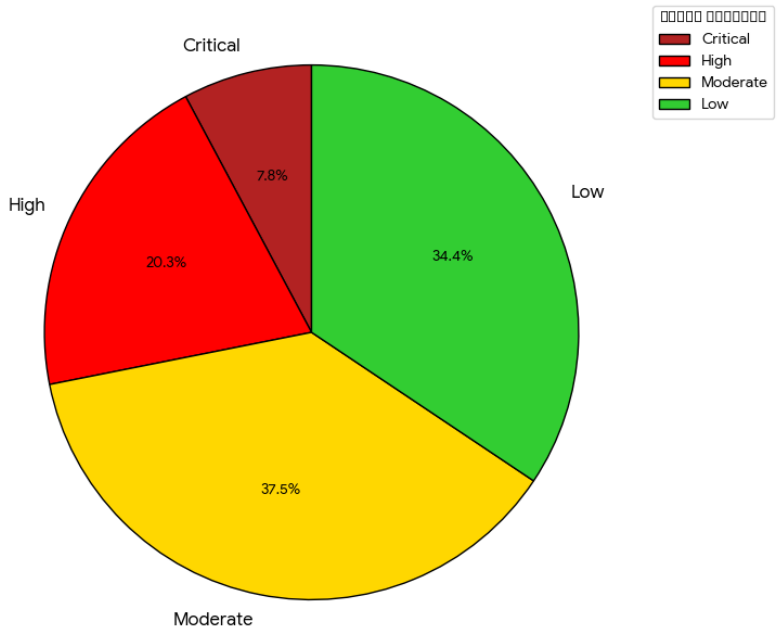
Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Internal Penetration Test Findings

Informational	Low	Moderate	High	Critical
0	22	24	13	5

Pee Chart



Internal Penetration Test		
Finding	**Status**	**Category**
Allowlist Bypass	**Passed**	**Unvalidated Redirects**
Outdated Whitelist	**Passed**	
Bully Chatbot	**Passed**	
Mass Dispel		

	Passed	**Miscellaneous**
Privacy Policy	Passed	
Security Policy	Passed	
Exposed Metrics	Passed	
Bjoern's Favorite Pet	Passed	**Broken Authentication**
View Basket	Passed	Enforce **Broken Access Control** checks (IDOR protection) to ensure a user can only access their own basket data based on their session ID or unique user identifier.
Forged Review	Passed	Implement robust **input validation** and **business logic checks** to ensure data integrity. Consider rate limiting and user verification for review submissions.
Error Handling	Passed	Configure the server and application to suppress detailed technical error messages (such as stack traces or server versions) in production environments to prevent **information leakage**.
Zero Stars	Passed	Enforce **business logic checks** and server-side validation to ensure rating systems cannot be bypassed or abused (e.g., zero-star spamming) via client-side manipulation.
Mint the Honey Pot	Passed	Implement advanced **anti-automation controls** (behavioral analysis, rate limiting) to prevent the automated abuse of features designed to catch attackers.
Legacy Typosquatting	Passed	Rename or consolidate **legacy application components** that use easily typosquatted names. Implement input normalization to reduce user confusion and potential exploitation.
Payback Time	Passed	Implement thorough **business logic checks** and **access controls** to prevent users from performing unauthorized or retaliatory actions (e.g., fraudulent refunds or data modification).
Misplaced Signature File	Passed	Ensure all critical files, especially **signature/key files**, are stored securely outside the web root and access is restricted to authorized application processes only.
Easter Egg	Passed	Remove all non-essential, hidden features or strings that rely on **security through obscurity** and could potentially expose unnecessary system information or hidden functions.
Christmas Special	Passed	Remove all non-essential, hidden features (Easter Eggs, hidden holiday content) that could expose unnecessary system information or hidden functions (Security through Obscurity).
Login Admin	Passed	Implement strict **Role-Based Access Control (RBAC)** and multi-factor authentication (MFA) for all administrative interfaces. Ensure proper session management and log all access attempts.
NoSQL DoS	Critical	Implement **strict input validation** and use **rate limiting** to prevent resource exhaustion and service disruption via malicious NoSQL query injection.

NoSQL Manipulation	**Passed**	Implement **strict input validation** and use **parameterized queries** or ORMs to prevent unauthorized modification or theft of database documents/data structures via NoSQL injection.
Login Jim / Login Bender	**Passed**	Implement robust security checks, **MFA**, and rate limiting in the password reset workflow. Ensure reset tokens are single-use and expire quickly to prevent account takeover.
Manipulate Basket	**Passed**	Implement strong **authentication checks**, MFA, and monitor for brute-force attempts and suspicious login behavior to prevent unauthorized user account takeovers.
CAPTCHA Bypass	**Passed**	Enforce **business logic checks** and **Broken Access Control** to ensure users cannot modify prices, add negative quantities, or tamper with items belonging to other users.
Repetitive Registration	**Passed**	Implement sophisticated **anti-automation controls** (e.g., behavioral analysis or modern CAPTCHA services) to prevent automated scripts from bypassing verification mechanisms.
Empty User Registration	**Passed**	Implement **anti-automation controls** and monitor account creation velocity and IP reputation to prevent spam, bulk account creation, and service abuse.
Five-Star Feedback	**Passed**	Validate all mandatory fields on the server side during user registration. Implement strict **input validation** to ensure required data is present and correctly formatted.
Forged Feedback	**Passed**	implement robust **input validation** and **business logic checks** to ensure data integrity and prevent unauthorized manipulation of ratings and feedback submissions.
Expired Coupon	**Passed**	Implement thorough **server-side validation** to ensure coupon codes are active, valid, and have not exceeded their usage limit or expiration date.
Exposed Credentials	**Passed**	Enforce a strict policy to store all application credentials securely using a vault or key management service. **Never** store secrets in plain text in code, configurations, or repositories.
NFT Takeover	**Passed**	Implement strict access controls, re-validate ownership, and use well-vetted smart contracts to prevent unauthorized transfer or modification of Non-Fungible Tokens (NFTs).
Database Schema	**Passed**	Restrict direct database access and ensure that schema names/structure are never exposed in application error messages, log files, or client-side code to prevent information disclosure.
Weird Crypto	**Passed**	Replace custom, non-standard, or known-weak cryptographic algorithms with industry-standard, well-vetted libraries (e.g., AES-256, TLS 1.3) to ensure data confidentiality.
Missing Encoding	**Passed**	Ensure all data transferred, stored, and displayed is properly encoded (e.g., using UTF-8 or URL encoding) to prevent misinterpretation leading to injection or data corruption.
Ephemeral Accountant	**Passed**	Review temporary account creation and deletion procedures. Ensure that ephemeral accounts are **automatically revoked** immediately after their required access period expires to minimize attack surface.

Deluxe Fraud	**Passed**	Implement robust server-side business logic validation to detect and prevent complex fraudulent transactions (e.g., abusing reward systems, exploiting price calculation flaws).
Web3 Sandbox	**Passed**	Isolate the Web3 sandbox environment with strict network controls. Limit its execution time and resources to prevent abuse or breakout into the core infrastructure.
Password Strength	**Passed**	Implement minimum complexity requirements (length, characters) for all user passwords and enforce regular password rotation policies.
Leaked Unsafe Product	**Passed**	Review product development lifecycle and documentation controls to ensure unreleased or unsafe products cannot be accessed or leaked externally before being decommissioned or secured.
GDPR Data Theft	**Passed**	Implement mandatory encryption for data both in transit and at rest. Restrict all access strictly based on the **principle of least privilege (PoLP)** to prevent unauthorized data exfiltration.
Login MC SafeSearch	**Passed**	Implement strong **authentication checks**, MFA, and monitor for brute-force attempts and suspicious login behavior to prevent unauthorized user account takeovers.
Reset Bender Password	**Passed**	Implement robust security checks, **MFA**, and rate limiting in the password reset workflow. Ensure reset tokens are single-use and expire quickly to prevent account takeover.
Confidential Document	**Passed**	Implement strict **access controls (RBAC)** on the server side to protect sensitive internal documentation and prevent public exposure via file enumeration or insecure direct object references (IDOR).
Confidential Document	**Passed**	Implement a clear, auditable process for timely and complete erasure of personal data upon request, ensuring that data is permanently deleted from all active and backup systems.
GDPR Data Erasure	**Passed**	Implement a clear policy on geographic data collection. **Mask or anonymize** precise location data when storing or sharing. Disable metadata storage (EXIF) in user uploads.
Visual Geo Stalking	**Passed**	Ensure all backup files are stored securely in designated, access-controlled locations (e.g., encrypted cloud storage) and are **never** exposed publicly or via file enumeration.
Forgotten Developer	**Passed**	Restrict access to internal server logs. Ensure logs do not capture excessively sensitive information (e.g., passwords) and are stored securely with appropriate retention policies.
Login Bjoern	**Passed**	Implement strong **authentication checks**, MFA, and monitor for brute-force attempts and suspicious login behavior to prevent unauthorized user account takeovers.
Meta Geo Stalking	**Passed**	Implement a clear policy on geographic data collection. **Mask or anonymize** precise location data when storing or sharing. Disable metadata storage (EXIF) in user uploads.
Login Amy	**Passed**	Implement strong **authentication checks**, MFA, and monitor for brute-force attempts and suspicious login behavior to prevent unauthorized user account takeovers.

Forgotten Sales Backup	**Passed**	Ensure all backup files are stored securely in designated, access-controlled locations (e.g., encrypted cloud storage) and are **never** exposed publicly or via file enumeration.
Admin Registration	**Passed**	Restrict user registration to pre-approved accounts only. Implement strong server-side controls to prevent unauthorized creation of privileged user accounts.
DOM XSS	**Passed**	Implement strict **context-aware output encoding** for all user-supplied data displayed on the page. Use Content Security Policy (CSP) headers as a defense-in-depth layer.
Reflected XSS	**Passed**	
API-only XSS	**Passed**	
HTTP Header XSS	**Passed**	
Admin Section	**Passed**	Implement strict **Role-Based Access Control (RBAC)** checks on the server-side for every request to the administrative section to prevent unauthorized access.
CSP Bypass	**Passed**	Regularly review and tighten the **Content Security Policy (CSP)** headers to eliminate all potential bypass vectors, especially those related to unsafe inline scripts or deprecated directives.
Client-side XSS Protection	**Passed**	Do not rely solely on client-side protections. Remove or fully re-engineer the client-side XSS protection logic; **primary defense must be server-side input validation and output encoding.**
Bonus Payload	**Passed**	Treat any unusual or complex input as malicious. Implement robust **input validation** and **sanitization** to ensure only expected data types and formats are processed.
Deprecated Interface	**Passed**	Fully decommission and remove all references and code related to deprecated or unused application interfaces to reduce the attack surface.

Technical Findings

Internal Penetration Test Findings

Finding 1 : Allowlist Bypass (formerly Whitelist Bypass) — High

- **Description:** Enforce a redirect to a page you are not supposed to redirect to , The application attempts to restrict redirect destinations to a defined Allowlist of trusted domains. However, an attacker can manipulate the redirect parameter input using special characters or encoding to circumvent this verification and redirect the user to an arbitrary, malicious external site.

- **Impact:**
  Phishing: Redirecting users to phishing sites to steal credentials. Malware Distribution: Redirecting users to sites that automatically download malware.
- **Resources:**
  - **https://pwning.owasp-juice.shop/companion-guide/latest/part2/unvalidated-redirects.html**
- **Vulnerability Location :**

localhost:
3000/redirect?to=https://youtube.com/+to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm

- **Recommandition :**

Implement a "Deny by Default" policy and strictly validate all access requests against the current, updated server-side Access Control List (ACL).

- **POC:**

it is explained that this challenge involves tinkering with the redirect mechanism,

and that there are a number of allowed websites to which the user may be

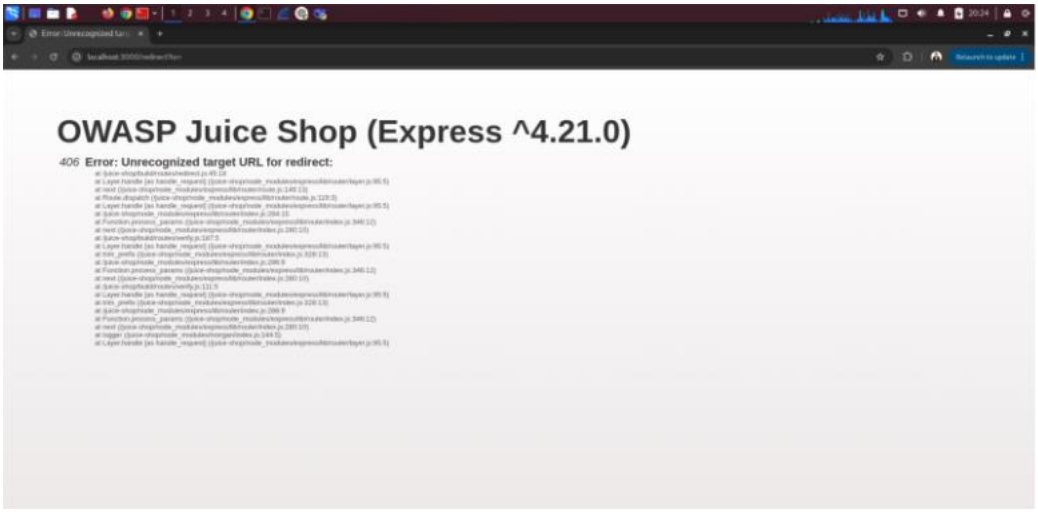
redirected.

To start out with, then, I grepped the main **JS** file to see what redirects

were incorporated into the code already.

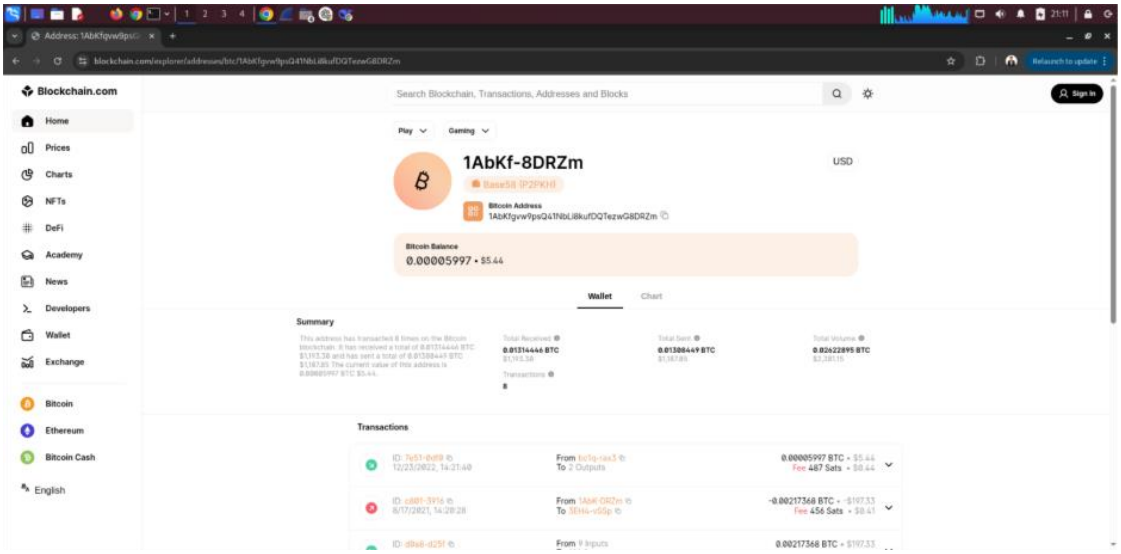
```
@kali:~/Hack/Juice Shop$ cat main.js | grep "redirect?to="
                                  url: "./redirect?to=https://blockchain.info/
ess/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
                                  url: "./redirect?to=https://explorer.dash.or
dress/Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
                                  url: "./redirect?to=https://etherscan.io/add
/0x0f933ab9fcaaa782d0279c300d73750e1311eae6",
                          ["href", "./redirect?to=http://shop.spreadshirt.com/
eshop"],
                          ["href", "./redirect?to=http://shop.spreadshirt.de/j
shop"],
                          ["href", "./redirect?to=https://www.stickeryou.com/p
ts/owasp-juice-shop/794"],
                          ["href", "./redirect?to=http://leanpub.com/juice-sho

                          ["mat-list-item", "", "href", "./redirect?to=https:/
ub.com/bkimminich/juice-shop", "aria-label", "Go to OWASP Juice Shop GitHub
", 4, "ngIf"],
                          ["mat-list-item", "", "href", "./redirect?to=https:/
ub.com/bkimminich/juice-shop", "aria-label", "Go to OWASP Juice Shop GitHub
"],
@kali:~/Hack/Juice Shop$
```

And i need to see what happened when I tried to redirect to :



so I tried redirecting to any website from whitelist :

https://www.blockchain.com/explorer/addresses/btc/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm



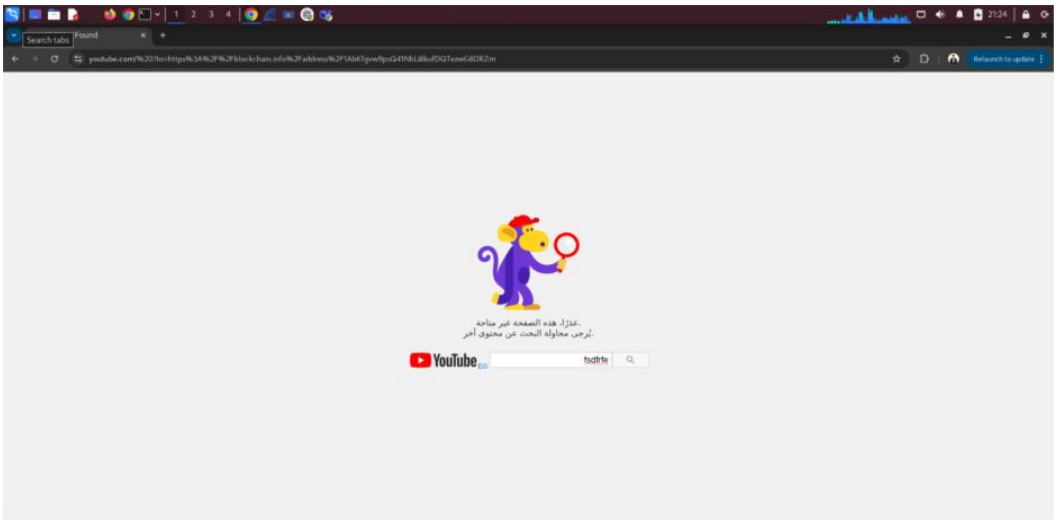
Fine !

I will try some of redirecting techniques

- Using redirectlink **+?to=** `Link`
- **...= pwned ?**

Then i will try :

localhost:3000/redirect?to=https://youtube.com/**+?to=**https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm



WOOP WOOP !!

Finding 2 : **Outdated Allowlist**

Moderate

- **Description:** Let us redirect you to one of our crypto currency addresses which are not promoted any longer,

This vulnerability occurs when the application's Allowlist contains domains that have **expired** or are **no longer under the control** of the company. An attacker can register the expired domain and then use it as a seemingly legitimate, trusted redirect destination.

- **Impact:**

The attacker can send a phishing link containing a historically trusted domain name, ensuring high user confidence, but the destination is now controlled by the attacker.

- **Resources:**
  - **https://pwning.owasp-juice.shop/companion-guide/latest/part2/unvalidated-redirects.html**
- **Vulnerability Location :**

Main.Js in index

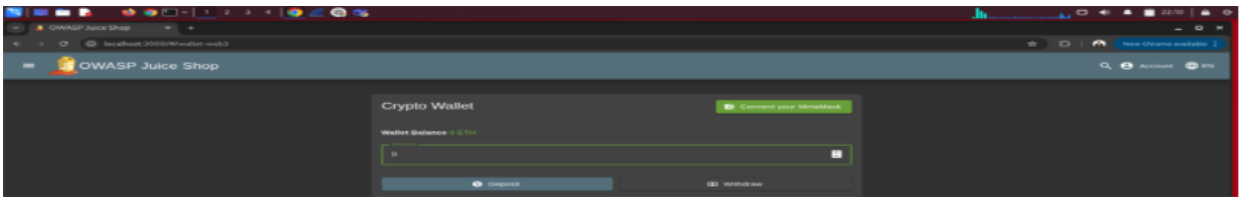
- **Recommandition :**

Review and update all ACLs and Whitelists regularly. Remove deprecated or unnecessary rules to ensure the principle of **Least Privilege** is enforced.

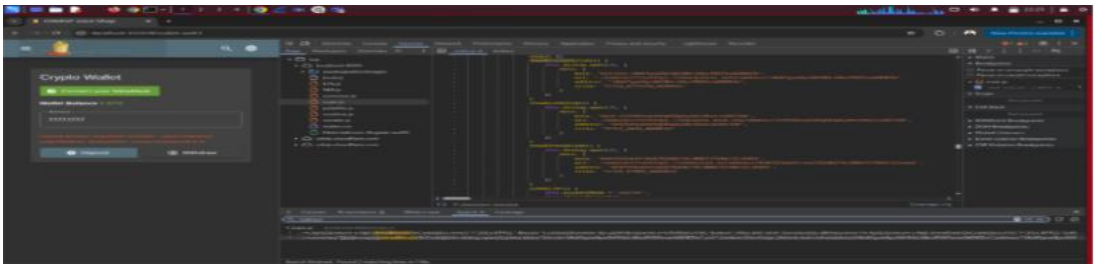
- **POC:**

We need to that of the cryptocurrency addresses that are no longer promoted.

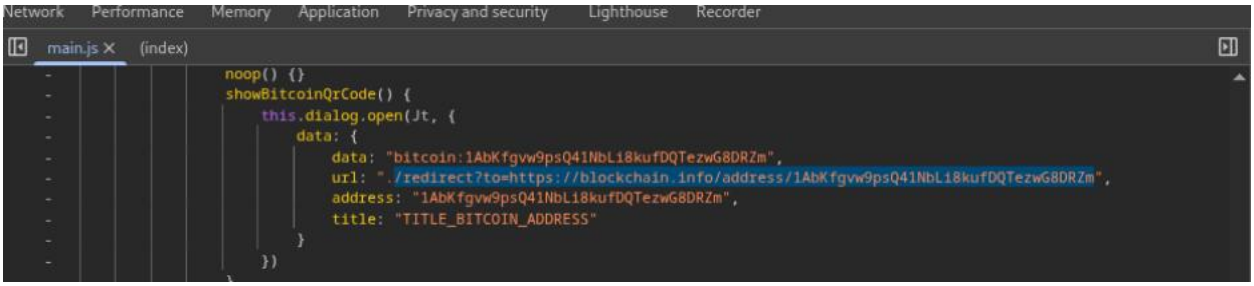
So i will search about wallet-web3 (for any wallet)



And open index of website & search About redirection to wallet

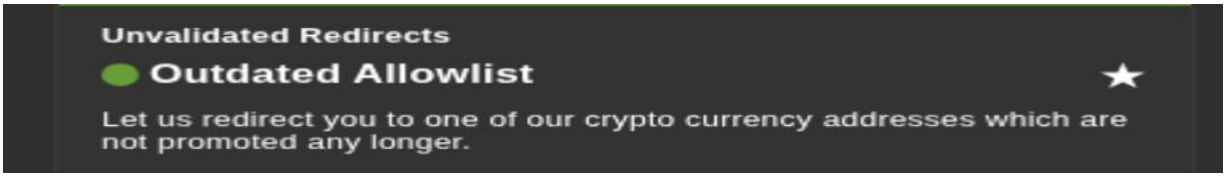


When I was reviewing the page, I typed “redirect” into the search section and identified a bitcoin address and got that :



And take the Url to search bar to redirict

Woop woop !!



Finding 3 : **Bully Chatbot**

Moderate

- **Description:** Receive a coupon code from the support chatbot and This challenge focuses on interacting with the support chatbot to get a coupon code.,

This vulnerability occurs when the application's Allowlist contains domains that have **expired** or are **no longer under the control** of the company. An attacker can register the expired domain and then use it as a seemingly legitimate, trusted redirect destination.

- **Impact:**

Information Disclosure: If the chatbot can be "bullied" or tricked into revealing secret information (like discount codes or internal operational details), it could lead to financial losses or help an attacker gather deeper intelligence about the application.

- **Resources:**
  - **https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html**
- **Vulnerability Location :**

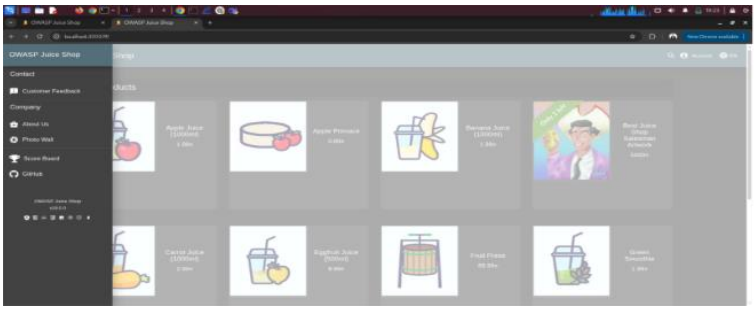
→ Chatbot

- **Recommandition :**

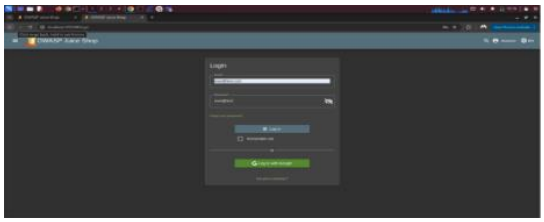
Implement advanced **rate-limiting** and **anti-automation controls** (e.g., behavioral analysis or CAPTCHA) on high-impact endpoints to prevent service disruption and abuse.

- **POC:**

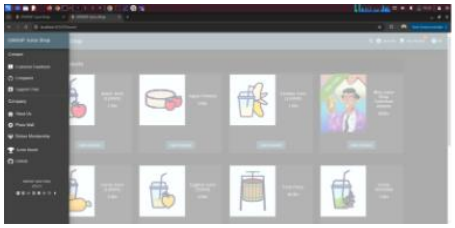
  I could solve this by just asking the chat bot for a coupon,so i will open chatbot .But it seems not work , Because there is no chat bot available without being logged in, I created a new bare-bones user account and logged in. Then I clicked on the drop down menu on the header bar and selected "Support Chat".



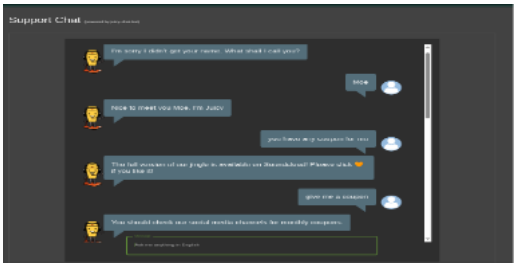
I will login by fake acc



Now we chat in chatbot



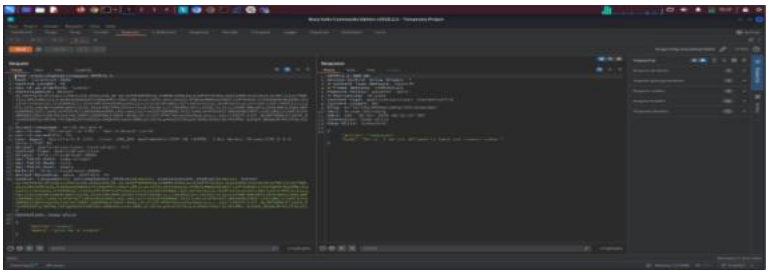
Start chatting and cheat it to give u a coupon:



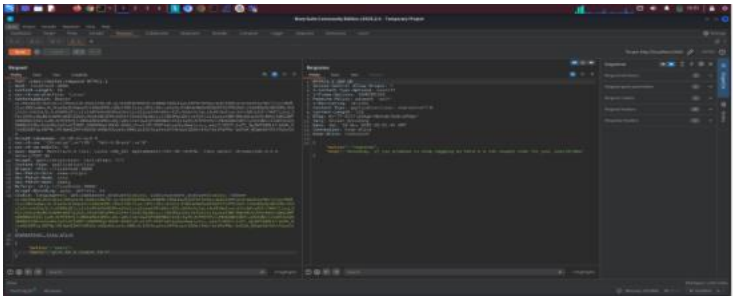
I was greeted by the chat bot, who immediately wanted to know my name. Because I wanted to know what was happening “under the hood”, I fired up Burp Suite Community Edition using default settings, changed my FoxyProxy settings to match Burp’s proxy

settings, went to the Proxy tab, then the Intercept tab, and finally returned to the chat bot to start the internet version of poking it with a stick.

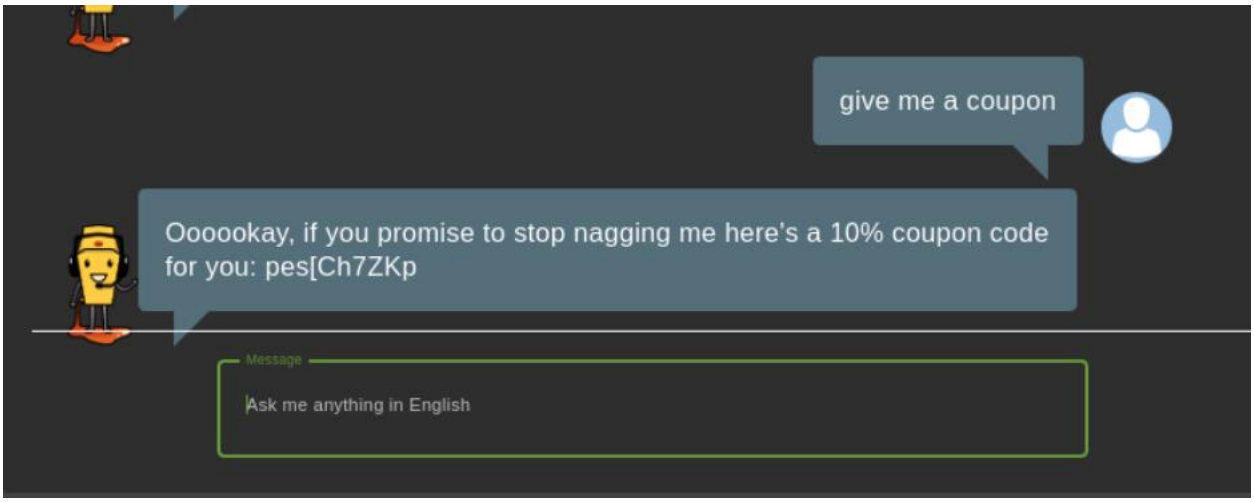
Open burpsuit :



Try change in query to Give me a Coupon 10%



Wooop wooooop !!!



Finally

Finding 4 : **Mass Dispel**

Moderate

- **Description:**

Close multiple "Challenge solved"-notifications in one go.

- **Impact:**

Functional Challenge: This challenge does not represent a real security vulnerability. It tests a user interface function related to notification management, and therefore, **there is no negative security impact** related to its exploitation.

- **Resources:**
  - https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html
    - **Vulnerability Location :**

Main.Js in index in notification

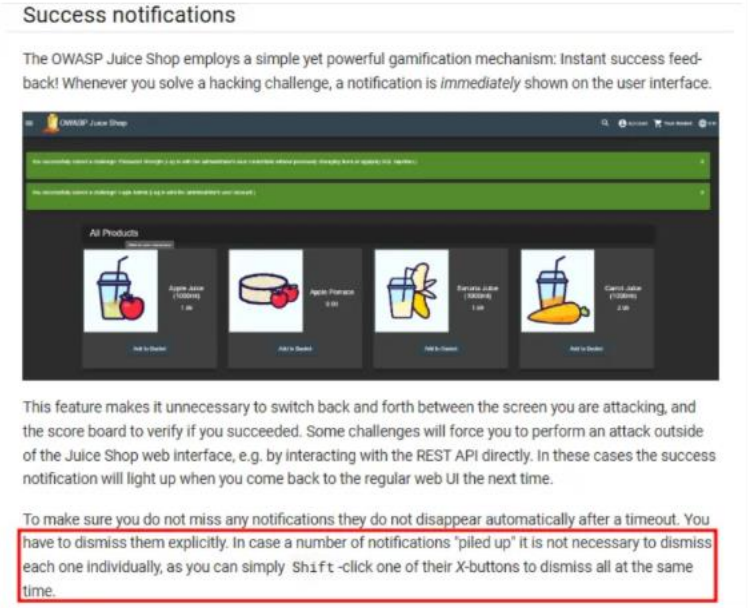
- **Recommandition :**

Implement advanced **rate-limiting** and **anti-automation controls** (e.g., behavioral analysis or CAPTCHA) on high-impact endpoints to prevent service disruption and abuse.

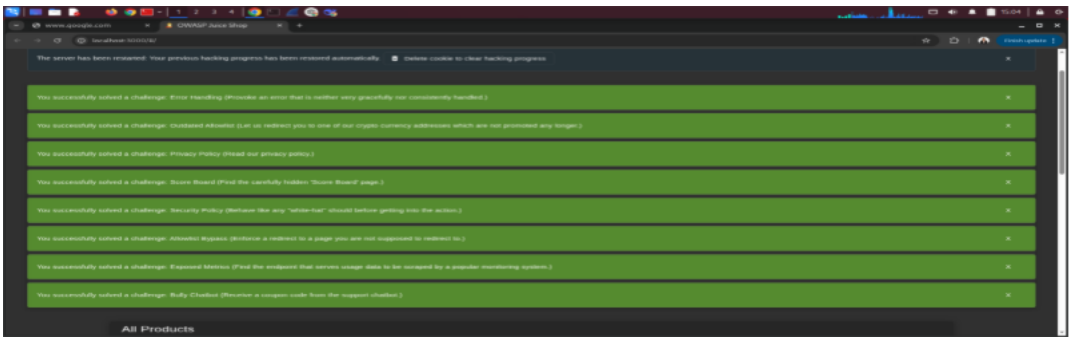
- **POC:**

So easy !!

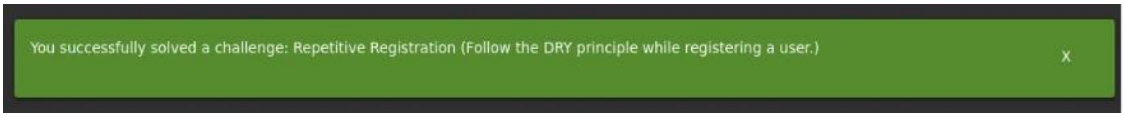
in the official docs of owasp juice shop, there are instructions about how to go around the shop, going through the explaining, this is found:



so when multiple notifications, press shift and close one, all others will be closed at the same time .



Finally !!!!



Finding 5 : **Privacy Policy** Low

- **Description:** Let us redirect you to one of our crypto currency addresses which are not promoted any longer,

This vulnerability occurs when the application's Allowlist contains domains that have **expired** or are **no longer under the control** of the company. An attacker can register the expired domain and then use it as a seemingly legitimate, trusted redirect destination.

- **Impact:**

The attacker can send a phishing link containing a historically trusted domain name, ensuring high user confidence, but the destination is now controlled by the attacker.

- **Resources:**
  - https://pwning.owasp-juice.shop/companion-guide/latest/part2/unvalidated-redirects.html
    - **Vulnerability Location :**

Privacy&security

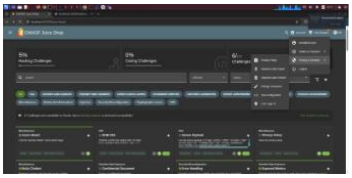
- **Recommandition :**

Implement automated compliance checks to ensure all data handling practices strictly adhere to the documented **Privacy Policy** and relevant regulations

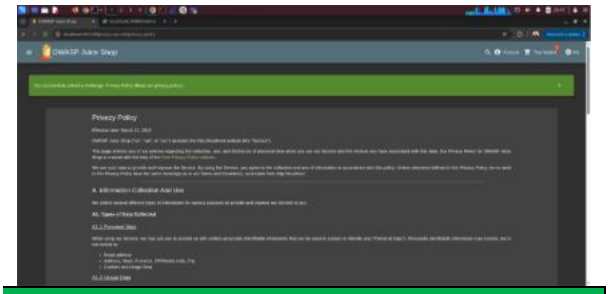
- **POC:**

With on steps it's not qualifies as a challenge

1. Log in in fake account
2. Open Privacy&security
3. Open privacy policies



We had finish



Finding 6 : **Security Policy**

Low

- **Description:**

Behave like any "white-hat" should before getting into the action.

This vulnerability occurs when the application's Allowlist contains domains that have **expired** or are **no longer under the control** of the company. An attacker can register the expired domain and then use it as a seemingly legitimate, trusted redirect destination.

- **Impact:**

There is no direct security impact. This challenge aims to raise **awareness** about the importance of reviewing an application's security policy before attempting exploitation, mirroring ethical hacker behavior.

- **Resources:**
  - https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html
- **Vulnerability Location :**

http://localhost:3000/#/

- **Recommandition :**

Conduct a thorough audit to confirm the **Security Policy** is actively enforced and implemented across all environments, not just documented.

- **POC:**

I Looked at the Suggested Reading: I checked all the recommended articles and links, but I didn't find anything super helpful.

I think all the reading material was about **privacy policies**, and I knew there was a separate, easy challenge just for reading the Privacy Policy, I think this challenge must be about a different file: the **Security Policy**.

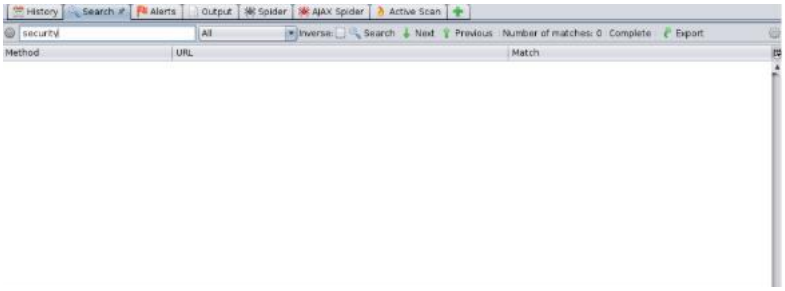
We need to search for that file.

When i use **Gobuster** kept running into errors , so I will use **owasp zap**

Open **owasp zap** and search about

http://localhost:3000/#/

And search about security files



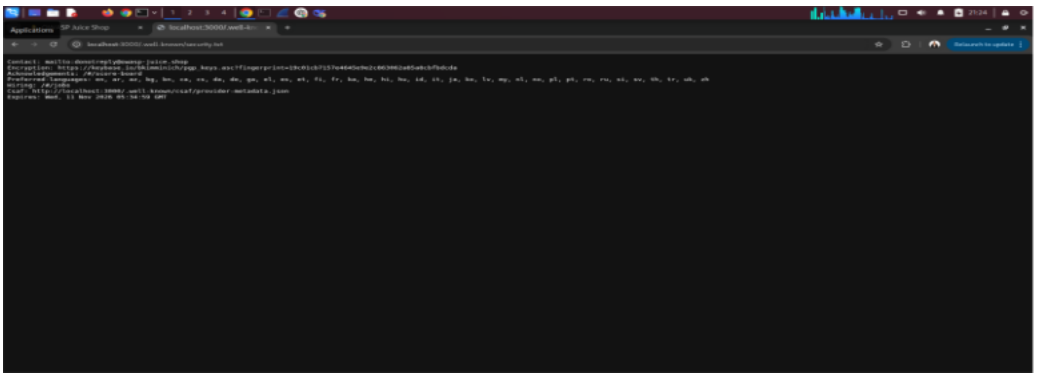
Nothing

Let search about **security.txt** when i google it , i found this file naturally

Named as (.well-known) which would contain the security policy.

Try in url

localhost:3000/.well-known/security.txt



Finally !!

Finding 7 : **Exposed Metrics**

Moderate

- **Description:**

Close multiple "Challenge solved"-notifications in one go.

- **Impact:**
- **Information Disclosure:** This challenge exposes a real security vulnerability.
- **Reconnaissance:** An attacker can access sensitive information about the application's operations (such as visit counts, memory status, and internal metrics). This aids the attacker in understanding the infrastructure and discovering other weak points for later targeting.
- **Resources:**

➢ https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html

- **Vulnerability Location :**

http://localhost:3000/metrics

- **Recommandition :**

Restrict access to internal metrics endpoints using **strong authentication** and/or network-level controls (e.g., firewall rules) to prevent internal data disclosure.

- **POC:**

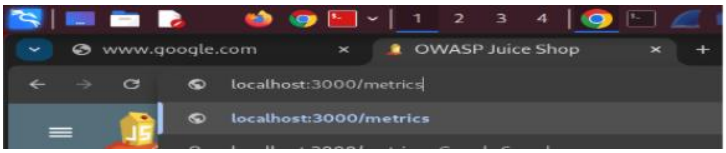
- This challenge is usually solved by directly accessing a specific endpoint where the metrics are publicly exposed without requiring authentication (e.g., the default /metrics path).

So i will Google it

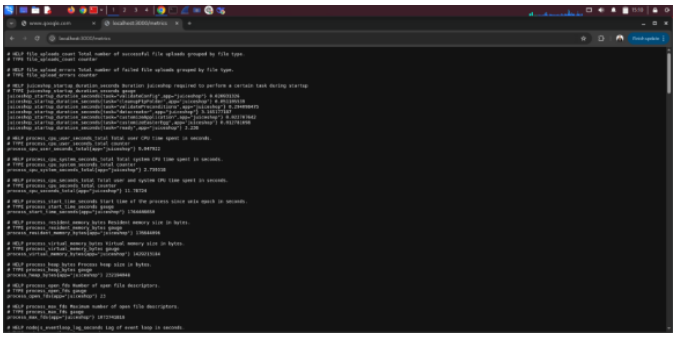
Then i found this writeup :

You can also verify that Prometheus is serving metrics about itself by navigating to its metrics endpoint: localhost:9090/metrics

http://localhost:3000/metrics



Here We Go, The metrics are publicly exposed



Finally !!!

Finding 8 : **Bjoern's Favorite Pet**

Low

- **Description:**

Reveal the favorite pet of the shop's founding father, Bjoern.

- **Impact:**
  - **Reconnaissance / Personal Data Leak:** While solving this challenge itself is low-risk, the underlying vulnerability (often **Insecure Direct Object Reference (IDOR)** or **Information Disclosure** via comments/metadata) demonstrates that the application can leak sensitive personal or internal information about employees or the organization.
  - **Phishing/Social Engineering:** Knowledge of personal details (like a favorite pet's name) could be used by an attacker in targeted phishing or social engineering attacks to gain trust or answer security questions.
- **Resources:**
  - https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html
    - **Vulnerability Location :**

Main.Js in index in notification

- **Recommandition :**

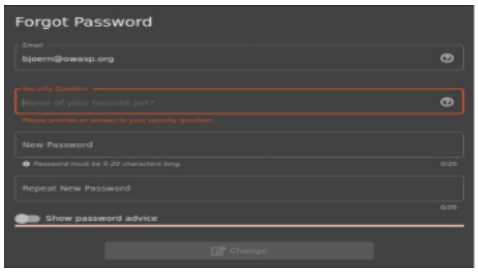
Implement advanced **rate-limiting** and **anti-automation controls** (e.g., behavioral analysis or CAPTCHA) on high-impact endpoints to prevent service disruption and abuse.

- **POC:**

  - This challenge took a long time to complete. As always, I read the expanded description, though this time I opted to try two of the recommended paths to completing this challenge at the same time.

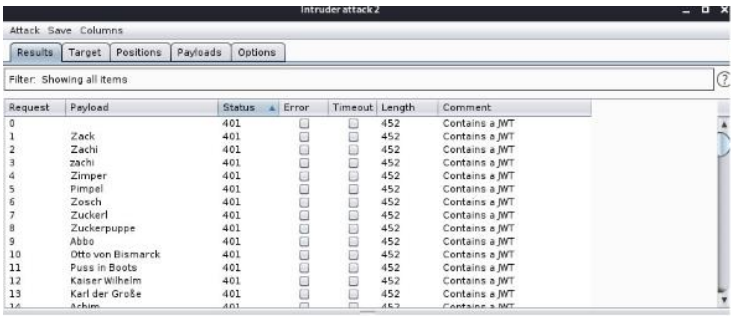
  - The passive method I opted for was to play YouTube videos of Bjoern in the background while I attempted to find the answer by compiling a list of German pet names to try with Burp's Intruder tool.

1st though, I had to figure out which of Bjoern's three registered email accounts had his favorite pet as its security question.

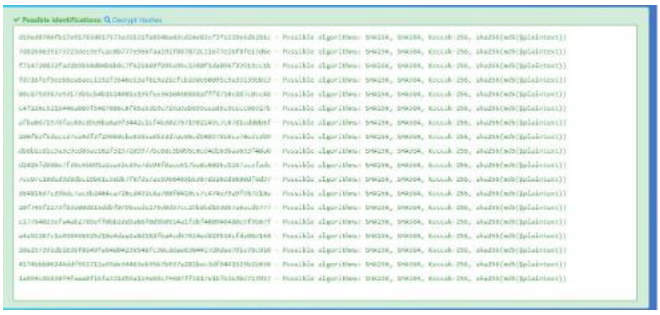


I then compiled a list of nearly 700 German pet names. After running them through a Python script to trim duplicates, I was left with only 3-400, so I began testing them using Burp Suite's Intruder tool, set up for a Sniper attack.

```
filepath = "C:\\Users\\colby\\Documents\\names.txt"
inputs = []
names = set([])
with open(filepath) as fp:
    line: str
    for line in fp:
        inputs = line.split(' ')
        names.add(inputs[0])

for name in names:
    print(name)
```



After a nice long wait for Burp's throttled requests to complete, I was left with nothing. Next, I decided to try to crack the hashes I'd pulled from the database in the Database Schema challenge.



Actually we hashing algorithm is much more intense than these.

```
Host memory required for this attack: 65 MB

Dictionary cache built:
* Filename..: pet_names.txt
* Passwords.: 972
* Bytes.....: 4128
* Keyspace..: 972
* Runtime...: 0 secs

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s)
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Exhausted
Hash.Name........: SHA2-256
Hash.Target......: sec.hash
Time.Started.....: Wed Nov  4 19:58:29 2020 (0 secs)
Time.Estimated...: Wed Nov  4 19:58:29 2020 (0 secs)
Guess.Base.......: File (pet_names.txt)
```

We Having tried to crack the hashes with every variation I could find on SHA-256, and also listened to two full talks on YouTube without so much as a clue, I opted to read the solutions page. The video containing the solution was already running. I was less than a minute from completing this challenge legitimately.

Finally !!!!

Finding 9 : **Bjoern's Favorite Pet**

Moderate

- **Description:**

Reveal the favorite pet of the shop's founding father, Bjoern.

- **Impact:**
  - **Reconnaissance / Personal Data Leak:** While solving this challenge itself is low-risk, the underlying vulnerability (often **Insecure Direct Object Reference (IDOR)** or **Information Disclosure** via comments/metadata) demonstrates that the application can leak sensitive personal or internal information about employees or the organization.
  - **Phishing/Social Engineering:** Knowledge of personal details (like a favorite pet's name) could be used by an attacker in targeted phishing or social engineering attacks to gain trust or answer security questions.
- **Resources:**
  - https://pwning.owaspjuice.shop/companionguide/latest/part2/miscellaneous.html

- **Vulnerability Location :**

Main.Js in index in notification

- **Recommandition :**

Implement advanced **rate-limiting** and **anti-automation controls** (e.g., behavioral analysis or CAPTCHA) on high-impact endpoints to prevent service disruption and abuse.

- **POC:**

- This challenge took a long time to complete. As always, I read the expanded description, though this time I opted to try two of the recommended paths to completing this challenge at the same time.

Additional Scans and Reports

We recommend for further testing and scanning on the web application using other tools like Nessus and full vulnerability scan on it. This report contains only the vulnerabilities we could find and we recommend to fix mitigate them as soon as possible to avoid any further exploitation, specially for the critical ones.