



School of Computer Sciences
Semester II, Academic Session 2020/2021

CST235 (Principles of Computer Networks and Information Security)

Assignment 2 - Security Improvement

Assignment Conditions: 1. Risk Assessment and Identification.

Video Presentation: https://youtu.be/G3j_g-C_dGs

Name	Matric No
Aqilah Syahirah binti Shahabudin	148346
Muhammad Aliff Iskandar bin Razali	147651
Siti Sakinah binti Ahmad Sanusi	148421
Teoh Sin Yee	148484

Lecturer's Name
Ts. Dr. Mohd. Najwadi Yusoff

Date of Submission
27th June 2021

Table of Contents

1.0	Introduction.....	1
1.1	Background.....	1
1.2	Purpose.....	2
2.0	Summary of Findings.....	2
2.1	External Threat.....	3
2.1.1	Injection.....	3
2.1.2	Broken Authentication	5
2.1.3	Sensitive Data Exposure	7
2.1.4	Improper Input Validation	9
2.1.5	Broken Access Control.....	12
2.1.6	Cross-Site Scripting (XSS)	14
2.1.7	Insecure Deserialization	15
2.1.8	Using Components with Known Vulnerabilities	17
2.1.9	Broken Anti-Automation	20
2.2	Internal Threats.....	22
3.0	Discussion	23
4.0	Conclusion	23
	References.....	25

1.0 Introduction

To practice the security assessment and management, we have selected an E-commerce website: [OWASP Juice Shop](#) for testing:

The OWASP Juice Shop is an open-source project hosted by the non-profit Open Web Application Security Project (OWASP) and is developed and maintained by volunteers. We chose to test this system because it is known as the most sophisticated and vulnerable system.

OWASP Juice Shop is like a small online shop that sells fruit and vegetable juice and associated products.

For better customization, we have deployed the [website](#) to Heroku as a copy.

In this project, we have assessed this website based on the top 10 Vulnerabilities by OWASP including internal and external threats.

1.1 Background

Architecture overview

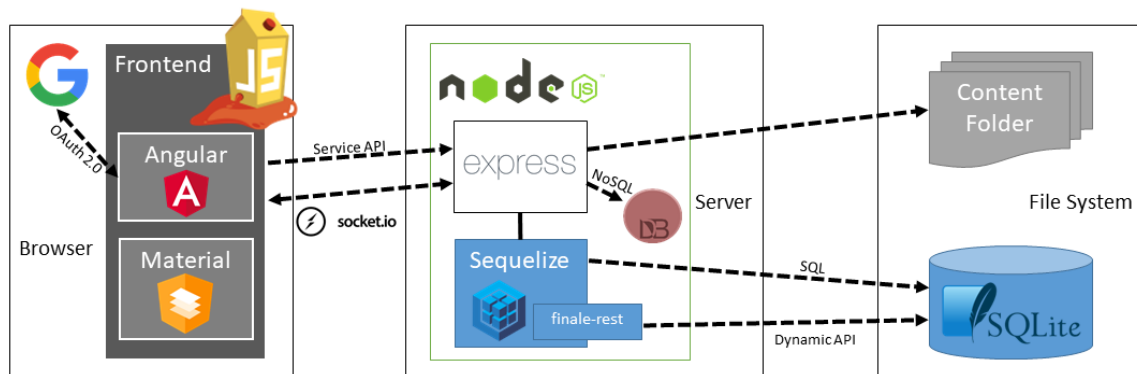
The OWASP Juice Shop is a pure web application implemented in JavaScript and TypeScript (compiled into regular JavaScript). In the frontend, the popular Angular framework is used to create a so-called Single Page Application. The user interface layout is implementing Google's Material Design using Angular Material components. It uses Angular Flex-Layout to achieve responsiveness. All icons found in the UI are originating from the Font Awesome library.

JavaScript is also used in the backend as the exclusive programming language: An Express application hosted in a Node.js server delivers the client-side code to the browser. It also provides the necessary backend functionality to the client via a RESTful API. As an underlying database, a light-weight SQLite was chosen because of its file-based nature. This makes the database easy to create from scratch programmatically without the need for a dedicated server. Sequelize and finale-rest are used as an abstraction layer from the database. This allows using dynamically created API endpoints for simple interactions (i.e., CRUD operations) with database resources while still allowing the execution of custom SQL for more complex queries.

As an additional data store, a MarsDB is part of the OWASP Juice Shop. It is a JavaScript derivative of the widely used MongoDB NoSQL database and compatible with most of its query/modify operations.

The push notifications that are shown when a challenge was successfully hacked are implemented via WebSocket Protocol. The application also offers convenient user registration via OAuth 2.0 to sign in with their Google accounts.

The following diagram shows the high-level communication paths between the client, server, and data layers:



1.2 Purpose

A security assessment must be taken because third parties' credentials and privacy are involved in this application. There are several functionalities of the website that might have a risk of exposing customers' data.

- Account registration
- Shopping cart items
- Product reviews/ Customer Feedback
- Product List
- Photo wall

2.0 Summary of Findings

In performing a detailed application risk assessment against OWASP Juice Shop, we identified several issues of concern. Throughout this report, we provide a brief description of each testing category and provide more details of negative findings.

The table below shows a breakdown of the vulnerabilities identified based on the category and severity of the risk.

External Threat: Vulnerabilities tallied by risk rating				
Testing category		High	Medium	Low
1.	Injection	1		
2.	Broken Authentication	1		
3.	Sensitive Data Exposure	1		
4.	Improper Input Validation	1		

5.	Broken Access Control		1	
6.	Cross-Site Scripting (XSS)		1	
7.	Insecure Deserialization	1		
8.	Using Components with Known Vulnerabilities		1	
9.	Broken Anti Automation		1	

2.1 External Threat

2.1.1 Injection

Injection flaws allow attackers to relay malicious code through an application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, and calls to backend databases via SQL (i.e., SQL injection). Whole scripts are written in Perl, Python, and other languages can be injected into poorly designed applications and executed. Any time an application uses an interpreter of any type, there is a danger of introducing an injection vulnerability.

Many web applications use operating system features and external programs to perform their functions. Sendmail is probably the most frequently invoked external program, but many other programs are used as well. When a web application passes information from an HTTP request through as part of an external request, it must be carefully scrubbed. Otherwise, the attacker can inject special (meta) characters, malicious commands, or command modifiers into the information. The web application will blindly pass these on to the external system for execution.

Vulnerability: Log in with the administrator's user account	
Risk	High
	The injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. The injection can sometimes lead to a complete host takeover.
Complexity	Low
	This attack only requires the ability to intercept POST parameters being sent to a server and use intruders to brute force it. This attack is trivial to carry out.
Summary: The application allows admin to log in and view all users' data. The login implementation contains a vulnerability that allows any user to access the admin's privilege by logging in using admin credentials. We can demonstrate this by performing the following steps:	

- [illegible]

4 | Page

- The preferred option is to use a safe API, which avoids using the interpreter entirely or provides a parameterized interface or migrate to use Object Relational Mapping Tools (ORMs).
- Use positive or "whitelist" server-side input validation. This is not a complete defence as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection

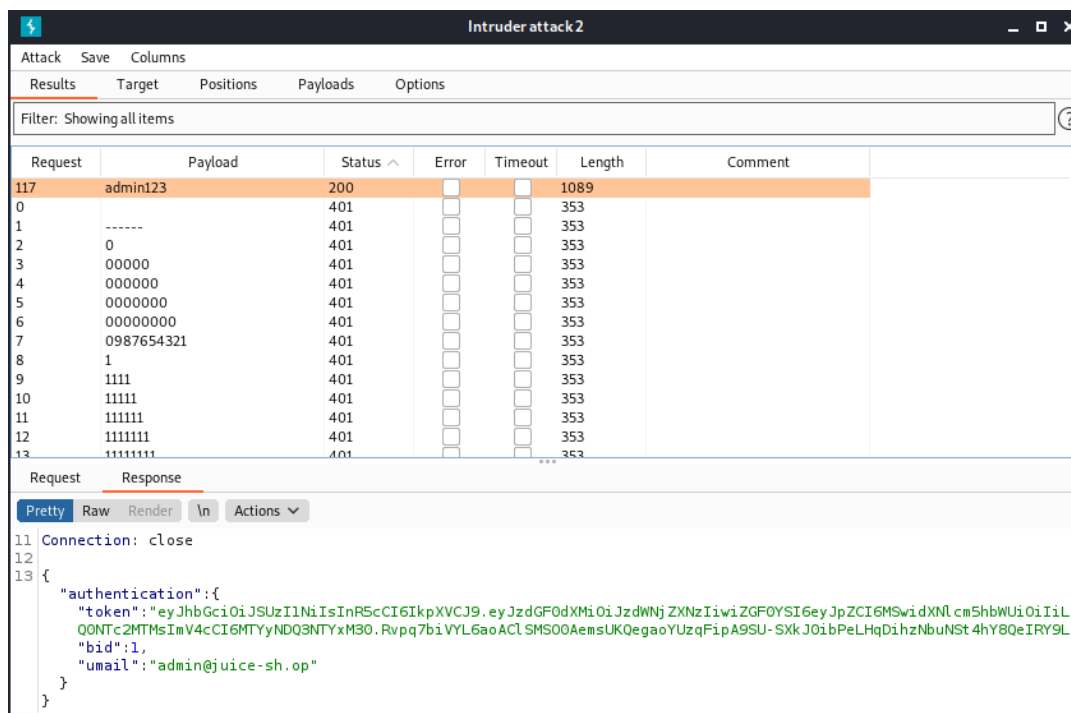
2.1.2 Broken Authentication

An attacker could compromise a legitimate user's account by hijacking their session ID or compromising their credentials. Compromising an account would allow the attacker to impersonate the legitimate user and perform actions that are only available to them. A compromised user account could allow the attacker to place unwanted orders on behalf of the compromised account, steal the account's owner's personal information, or change the credentials of the account. This would be especially problematic if the compromised account has an administrator's privileges as the attacker could compromise the entire system and disclose highly sensitive classified information.

Vulnerability: Getting account accesses by brute-forcing passwords	
Risk	High
	This attack could compromise accounts, leading to stolen personal data, alteration or loss of data, unauthorized actions, impersonation, fraud, company data breaches, or total system failure.
Complexity	Low
	This attack can be carried out with brute-forcing software and a word list to brute force accounts' passwords with known registered emails.
Summary: The application has a login form consisting of an email and password pair to log in to an account. The login form contains a vulnerability in which an attacker could input a known registered email in the email field and carry out a brute-forcing attack on the password field. In this case, we will log into the administrator's account.	

We can demonstrate this by performing the following steps:

1. Go to the system's login page.
2. Open Burp Suite and intercept HTTP requests.
3. Input the administrator's email into the email field (admin@juice-sh.op).
4. Input an arbitrary password into the password field.
5. Click on "Log in".
6. Intercept the POST request containing the email and password field.
7. Send the request to the Intruder tool.
8. Clear the marked positions.
9. Highlight the password field and add the position.
10. Add a payload for use in the attack (we used this payload:
<https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/best1050.txt>).
11. Start the attack and let it run.
12. Once the attack is finished, we can sort it by status and look for a "200" HTTP response status code to find a successful request.
13. We can see a "200" status code that indicates that an attempt successfully logged into the account.
14. We can now log into the administrator's account using the password.



The screenshot shows the 'Intruder attack 2' window in Burp Suite. The 'Results' tab is active, displaying a table of attack results. The first row, index 117, shows a successful login attempt with a status of 200 and a length of 1089. The payload for this request is 'admin123'. Below the table, the 'Response' tab is selected, showing the JSON response from the server, which includes a token and a bid.

Request	Payload	Status	Error	Timeout	Length	Comment
117	admin123	200			1089	
0		401			353	
1	-----	401			353	
2	0	401			353	
3	00000	401			353	
4	000000	401			353	
5	0000000	401			353	
6	00000000	401			353	
7	0987654321	401			353	
8	1	401			353	
9	1111	401			353	
10	11111	401			353	
11	111111	401			353	
12	1111111	401			353	
13	11111111	401			353	

```
11 Connection: close
12
13 {
  "authentication": {
    "token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMtOiJzdWNjZXNzIiwiaWF0IjZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLQONTc2MTMsImV4cCI6MTYyNDQ3NTYxM30.Rvpq7biVYL6aoACLSMS00AensUKQegaoYUzqFipA9SU-SXkJ0ibPeLHqDihzNbuNst4hy8QeIRY9L",
    "bid": 1,
    "umail": "admin@juice-sh.op"
  }
}
```

Figure 2.1.2-1 – Result of the intruder attack

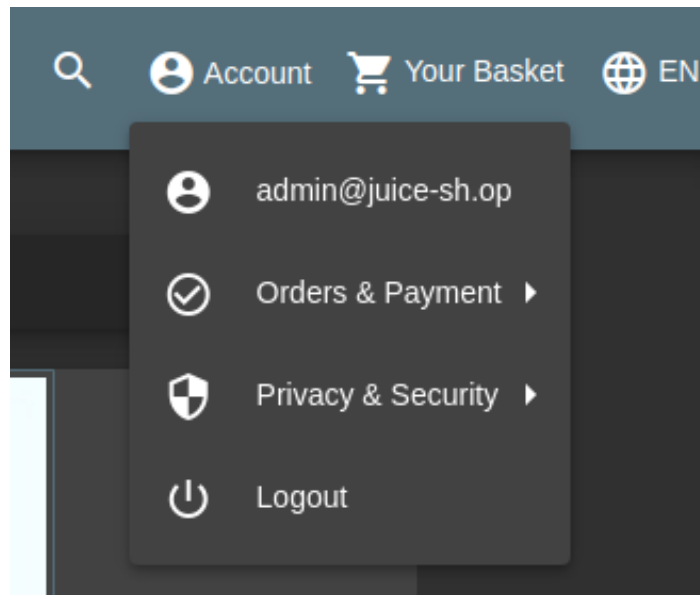


Figure 2.1.2-2 – Successfully logged in as the administrator

Recommended solution:

Several methods can mitigate this password brute-forcing method:

- Enforce a strong password policy consisting of 16 characters or more, including a combination of uppercase and lowercase characters, symbols, and numbers.
- Deny passwords that contain common words or phrases.
- Enforce the two-factor authentication (2FA) process when logging in.
- Temporarily ban the IP address of a user when they entered an incorrect password to an account multiple times, and send a notification to the security department and the account's email about the attempted breach.

2.1.3 Sensitive Data Exposure

Sensitive data is any data that should be protected from unauthorized access. Sensitive data can be exposed by a web application if its databases are not adequately protected. Some web applications have hidden directories to store their files. Although hidden, these directories and files do not require authorization to be accessed and found by an attacker through web fuzzing or URL fuzzing. URL fuzzing would brute force the web application's URL to discover hidden directories and files. Depending on what is stored, this could be a harmless note left behind by a web developer to a massive sensitive data exposure with extreme legal consequences.

Vulnerability: Accessing hidden directories and sensitive data using a URL fuzzer	
Risk	High
	This vulnerability could cause sensitive data to be exposed, which could cause other vulnerabilities to be exposed, stolen personal data, or company data breaches which could have massive legal implications.
Complexity	Low
	This attack uses a web fuzzer to brute-force URLs to find hidden directories and files in the web application.
<p>Summary:</p> <p>The web application takes a URL input from the client to get the content to be served.</p> <p>The web application contains a vulnerability whereby a URL fuzzer can brute-force URLs to find hidden directories and sensitive data.</p> <p>We can demonstrate this by performing the following steps:</p> <ol style="list-style-type: none"> 1. Open the juice shop web application in a web browser. 2. Open the URL fuzzing application (in this case, we used FFUF). 3. Copy the URL of the juice shop web application. 4. Write the command to run FFUF, using the URL we copied and using DIRB's common wordlist. 5. Replace the part of the URL with the "FUZZ" string where we would be brute-forcing the URL with the contents of the chosen word list. 6. Run the command "\$ ffuf -w /usr/share/wordlists/dirb/common.txt -u https://juice-shop.herokuapp.com/FUZZ" 7. We would see many HTTP responses with the code "200" with the same sizes as each other. 8. Run the FFUF command again with an included filter to filter out the repeating sizes of the HTTP "200" responses found previously. 9. In our case, the repeating sizes were "1834", the command "\$ ffuf -w /usr/share/wordlists/dirb/common.txt -u https://juice-shop.herokuapp.com/FUZZ -fs 1834" was executed. 10. After the URL fuzzing is completed, we can see the hidden directories located during the process. 11. The URLs can then be traversed, and the files stored there can be snooped. 	

```

:: Wordlist      : FUZZ: /usr/share/wordlists/dirb/common.txt
:: Follow redirects : false
:: Calibration   : false
:: Timeout      : 10
:: Threads      : 40
:: Matcher      : Response status: 200,204,301,302,307,401,403,405
:: Filter       : Response size: 1834

assets          [Status: 301, Size: 179, Words: 7, Lines: 11]
ftp             [Status: 200, Size: 10620, Words: 1546, Lines: 355]
promotion       [Status: 200, Size: 4933, Words: 409, Lines: 98]
robots.txt      [Status: 200, Size: 28, Words: 3, Lines: 2]
:: Progress: [4614/4614] :: Job [1/1] :: 25 req/sec :: Duration: [0:03:04] :: Errors: 0 ::

```

Figure 2.1.3-1 – The result of fuzzing the URL

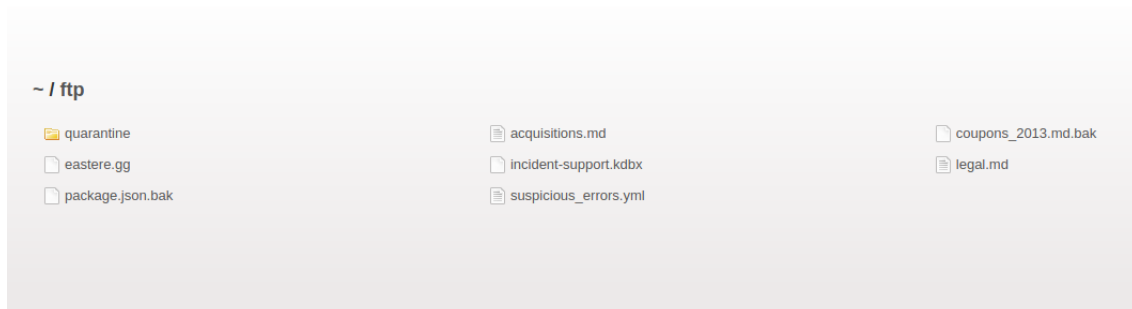


Figure 2.1.3-2 – Snooping around the /ftp directory to find sensitive data

Recommended solution:

We recommend the following steps to prevent URL brute-forcing and to protect sensitive data:

- Limit the number of requests which the client can send to the server to slow down the URL brute-forcing process.
- Disable directory listing in the web application.
- Prevent the files in the web application from being directly accessible by verifying whether the requesting user can view the file and generate a download token for the user to download the file.
- Remove sensitive files from the directories of the web application.

2.1.4 Improper Input Validation

Sometimes, a user will intentionally or unintentionally provide unexpected inputs to the web application. Some inputs can be dangerous as they could interact with the web application's components directly. If proper input validations are in place, dangerous unexpected input will be filtered out or converted into a safe input for processing. Improper input validation would cause unexpected results within the web application, which could cause system failure, data alteration, or security breaches. Improper input validation could allow an attacker to conduct a mass assignment attack. This attack involves altering a POST request where the attacker can modify or add a field of their choosing into the request. The database would receive the

request, including the injected field, and thus create a database entry based on the altered request.

Vulnerability: Registering an account with administrator privileges	
Risk	High
	This attack could create accounts with administrator privileges, leading to stolen personal data, alteration or loss of data, unauthorized actions, impersonation, fraud, company data breaches, or total system failure.
Complexity	Medium
	This attack is a form of mass assignment attack. The POST request upon registering an account is altered to include a "role" field where the registrant can assign themselves as an administrator.
Summary: The web application utilizes mass assignment functionality to receive user-inputted data from a form to create a new database entry when a new user registers. The registration form has a vulnerability whereby the attacker can conduct a mass assignment attack. The POST request is altered upon registering for an account by adding a "role" field to assign themselves as an administrator. We can demonstrate this by performing the following steps: <ol style="list-style-type: none">1. Go to the juice shop's account registration page.2. Fill out the form with your desired inputs.3. Open Burp Suite and intercept the HTTP requests.4. Click on register and check the POST request on Burp Suite.5. We can inspect the fields of the response to the POST request to see that there is a "role" field in which we are assigned as "customer"	

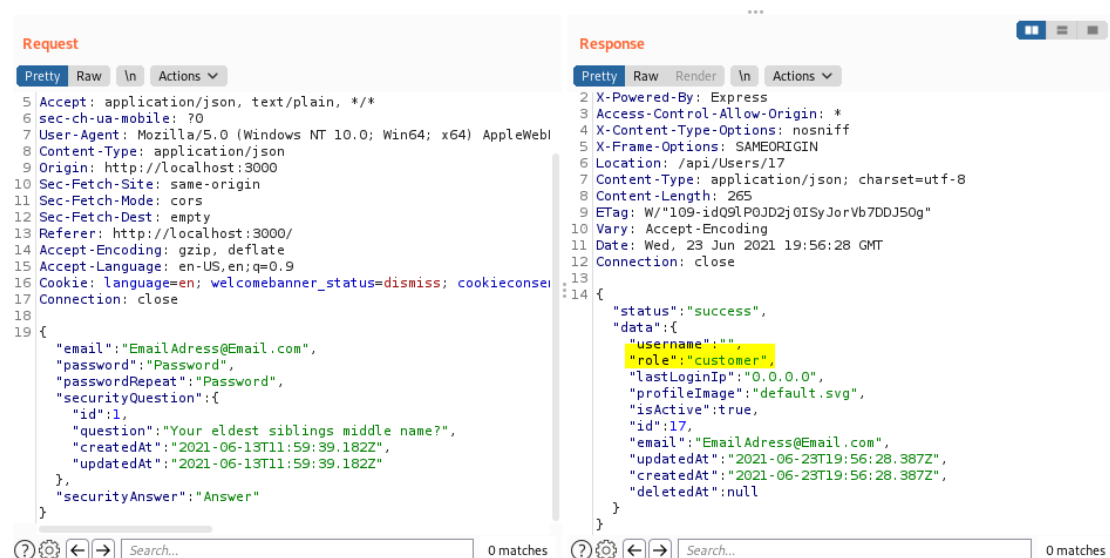


Figure 2.1.4-1 – The "role" field in the response

6. Send the POST request to the repeater tool.
7. Alter the form to include the "role" field with the input as "admin" and send the request.
8. We can see a response indicating that the request was successful and that the account was created with the "admin" role.

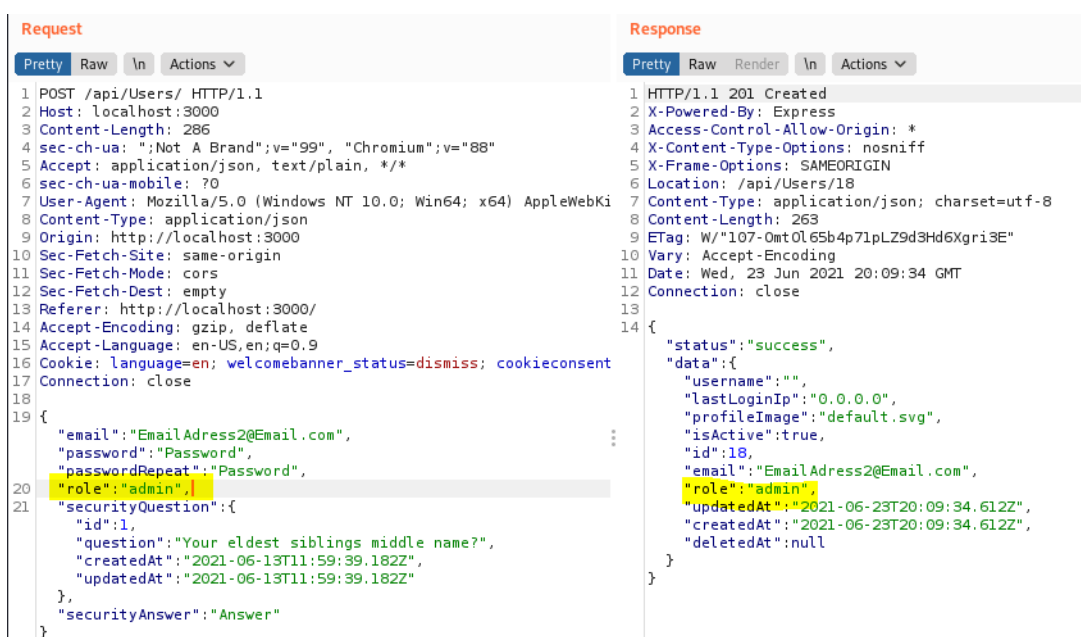


Figure 2.1.4-2 – Added "role" field to request with "admin" input and receiving a successful response

Recommended solution:

We suggest that the web application would defend from mass assignment attacks by:

- Explicitly assigning the attributes of the form fields, one by one, instead of using the mass assignment.
- Whitelisting the fields in the form, which can be in the POST request sent by the user.

2.1.5 Broken Access Control

Access control, sometimes called authorization, is how a web application grants access to content and functions to some users and not others. These checks are performed after authentication and govern what 'authorized' users are allowed to do. Broken Access Control typically led to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside of the user's limits. Common access control vulnerabilities include:

- Bypassing access control checks by modifying the URL, internal application state, or the HTML page, or simply using a custom API attack tool
- Allowing the primary key to be changed to another's user's record, permitting viewing or editing someone else's account.
- Elevation of privilege. Acting as a user without logging in or acting as an admin when logged in as a user.

Vulnerability: Post some feedback in another user's name.	
Risk	Medium
	The technical impact is attackers acting as users or administrators, or users using privileged functions, or creating, accessing, updating or deleting every record.
Complexity	Low
	This attack only requires the ability to intercept POST parameters being sent to a server and use a repeater to brute force it. This attack is trivial to carry out.
Summary: The application allows users to submit customer feedback. The feedback form implementation contains a vulnerability that allows any user to post feedback in another user's name. We can demonstrate this by performing the following steps: <ol style="list-style-type: none"> 0. Filling up the feedback form and submit anonymously 1. Capture the POST request 2. Send the captured request to the repeater 3. Alter the UserId = 3 and send the request 4. The feedback has sent out on behalf of the user who UserId = 3 	

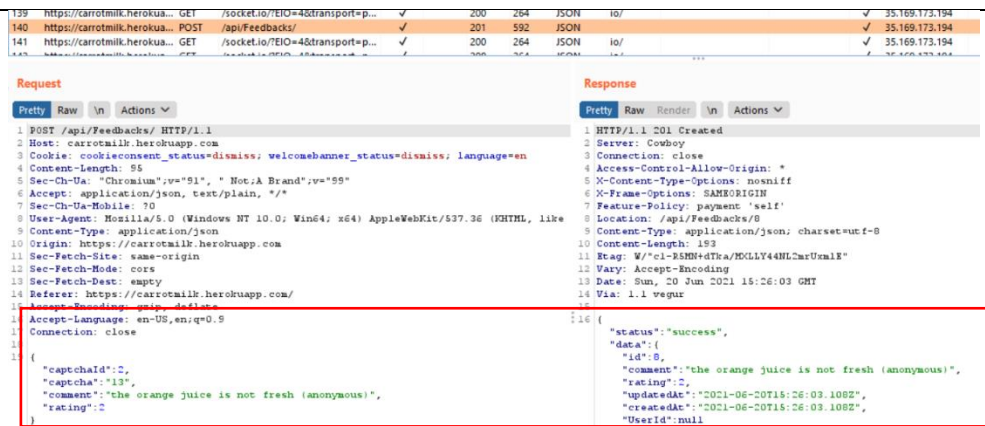


Figure 2.1.5-1 – Capture of POST request

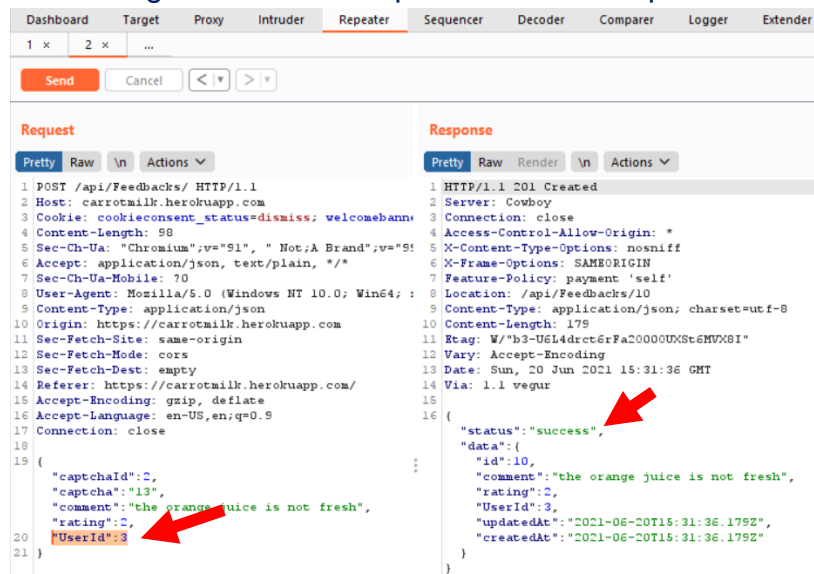


Figure 2.1.5-2 – UserId changed from NULL to 3

Recommended solution:

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker cannot modify the access control check or metadata.

- With the exception of public resources, deny by default.
- Implement access control mechanisms once and re-use them throughout the application, including minimizing CORS usage.
- Model access controls should enforce record ownership rather than accepting that the user can create, read, update, or delete any record.
- Domain models should enforce unique application business limit requirements.
- Disable web server directory listing and ensure file metadata (e.g., .git) and backup files are not present within web roots.
- Log access control failures, alert admins when appropriate (e.g., repeated failures).

- Rate limit API and controller access to minimize the harm from automated attack tooling.
- JWT tokens should be invalidated on the server after logout.
- Developers and QA staff should include functional access control unit and integration tests

2.1.6 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) is an attack where injected malicious scripts into the websites. It is a type of injection. It happened when a malicious code sends by the attacker use the web application. Usually, it can send the code from the browser side script to a different end-user. If the input validation is weak, the attacker can successfully spread the malicious code anywhere in the web application.

This attack can occur when we use XSS to send malicious codes. As a result, the browser's malicious codes cannot be detected, so it thinks it came from a trusted source and execute the codes. Commonly, the code is written in JavaScript.

Vulnerability: DOM-based XSS	
Risk	High
	DOM-based XSS attacks can result in data-stealing (user's data and cookies) and data manipulation. Also, it can inject trojan functionality and perform virtual defacement into the website.
Complexity	Low
	This attack can be performed in the front end of the website by using JavaScript. The attack happened in client-side code, which is hard to detect by the server.
Summary: The application allows the attacker to inject malicious scripts into the client-side code. The DOM-based XSS implementation contains a vulnerability that allows the website's data manipulation that seems normal to the victim. We can demonstrate this by performing the following steps: 1. On the search bar, insert the code: <code><iframe src="javascript:alert('xss')"></code> 2. Enter the code, and the pop-up message will appear.	

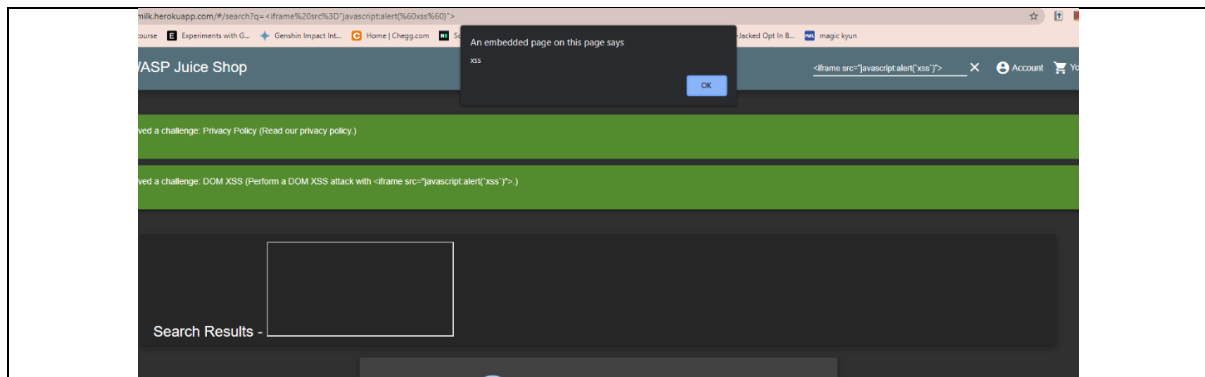


Figure 2.1.6-1 – The pop-up message appear

This can be known as an attack because it is a reflected XSS vulnerability. It can trick the victim into clicking the link that contains the payload. This attack seems normal to the user, but it will cause harm since the user thought the message comes from a trusted source.

Recommended solution:

The vulnerability can be fixed by:

- The input should be filtered. The developer should do a validation input and filter the codes that the attacker might use. The filter method should be concise and strict so the attacker cannot easily manipulate it.
- The output should be control by encoding the data. When the user-controllable data is output in HTTP responses, the output should be encoded so the trick cannot be deceived as active content. This action can be made by applying multiple codes consists of HTML, URL, JavaScript, and CSS encoding.

2.1.7 Insecure Deserialization

Serialized objects can be manipulated by the attacker when the website has the vulnerability to deserialized the user-controllable data. It could lead to passing malicious and harmful data into the website. This act is known as Insecure Deserialization. In this context, deserialization means converting the stream of bytes into serialized objects.

This attack can happen if the lack of exposure to the danger of deserializing user-controllable data. It also can occur if the developer overlooked the complexity of the website. Even though much measurement has been taken, but the user input can be easily manipulated. For example, the developer only trusts data provided by a sanitized object rather than classical user input. It takes much work, but it is achievable if the website uses languages that use a binary serialization format rather than string-based ones.

Furthermore, the website with numerous dependencies in each library can be one reason why insecure deserialization occurs. It hardly manages efficiently, which means malicious data can be invoked. It will be troublesome if the malicious data are everywhere in the classes.

Vulnerability: Changing the price tag of the website	
Risk	High
	This vulnerability allows the attacker to remote code execution. The attacker can abuse the website logic, which means they can replace the serialized object with another object even though it comes from different classes. If it is done properly, the replacement can be executed successfully. Other than that, it can lead to privilege escalation, unauthorized file access, and denial-of-service attacks. Also, this attack can be called object injection.
Complexity	High
	This attack requires a special tool to discover the flaw of the website deserialization, but it needs some skill to validate it.
<p>Summary: One example that performs insecure deserialization is changing the price tag of a product on the website. It is altering the serialized object. When it is loaded into the website, it can be used without validation. It would be successful when the attacker can check out and pay the product with a "fake" price.</p> <p>Recommended solution: There is much measurement that can be done to prevent insecure deserialization:</p> <ul style="list-style-type: none"> • The website should not accept the serialized object from an untrusted source. It can be avoided by implementing a digital signature to check the authenticity of the data. This method should be implemented before starting the deserialization process. Otherwise, it will be inefficient. • The dependencies of each library should be reduced. Also, the generic deserialization features should be avoided. The developer should create a class-specific serialization method because it will help them to control the objects. • The developer can consider using a firewall so it can detect the insecure deserialization attack. 	

2.1.8 Using Components with Known Vulnerabilities

Most of these dangers are caused by software dependencies, such as employing libraries and frameworks that were previously known to be vulnerable or that became vulnerable later due to unpatched software fixes or updates that were not deployed on time. Using Components with Known Vulnerabilities is type of risk arises when the app's components, such as libraries and frameworks, are mostly executed with full privileges. The attackers spot a weak component by utilising automatic scan tools or manually analysis of the application to find faults, and if they discover that the application uses a component that has previously been identified as vulnerable, they can quickly try to exploit that vulnerability and makes their job easier to cause a serious data loss or server takeover. To locate the dependencies, the attackers utilise fingerprinting techniques such as searching for recognised html components, creating problems, forcing browsing, and so on.

Because of its ease of exploitability, this vulnerability can cause a significant risk to the business. This vulnerability can easily overcome application security defences and operate as a pivoting point for a variety of different attacks, such as invoking a web service with full authority without giving an authorization token or executing code remotely. Injection, XSS, and failed access control are example of full range weaknesses when using vulnerable components.

Typosquatting is one of example of this attack. Typosquatting, also known as URL hijacking, is a sort of cybersquatting that targets people who mistakenly type a website address into their browser's URL field. Cybersquatters buy domain names that are slightly different from the target brand. It happens when a prominent domain name is typos, misspellings, or misunderstandings. If a person makes a typo when typing a domain name and does not realise it, they may find up on a malicious website set up by hackers.

There are at least eight kinds of typosquatting:

- Typos: Mistyped site URLs of well-known firms in the address such as "twitter.com"
- Misspelling: Misspelt domain especially it is an invented word such "goggle.com"
- Wrong domain extensions: The likelihood of typosquatting sites increases as more top-level domain (TLD) names are added. For example, typing ".com" instead of a ".org"
- Alternative spellings: Users may be confused of the abbreviated spelling of services, brand names, or items such as getphotos.com vs getfotos.com
- Hyphenated domains/combosquatting: This is dropping out or adding a hyphen to illegally send visitors to a typo-domain, such as facebook.com vs. face-book.com.
- Supplementing popular brand domains: When well-known trademarks are combined with relevant phrases, a legitimate sounding typosquatted domain name can be created, such as apple-shop.com vs apple.com.

- Pretending to be www: wwwfacebook.com vs www.facebook.com
- Abuse of Country Code Top-Level Domain (ccTLD): Directing a person who has left a letter at the wrong location away from the actual location such as twitter.cm vs twitter.com

Although not all typosquatting attempts are motivated by cybercrime, many typosquatted domain owners intentional act dishonest. Typosquatted domain can be used for:

- Install malware: The malicious website download malware or adware on visitors' devices.
- Phishing: Malicious websites are designed to look exactly like popular websites to steal personal information, login passwords, or email addresses.
- Bait and switch: The bogus website offer you something you want to buy at the correct URL, but it never sends anything to you.
- Affiliate links: Use wrong spelling to monetize the domain, send people to competitors, or redirect traffic back to the brand via an affiliate link, earning money on every click.

Owner of the website can prevent typosquatting is buy an EV SSL Certificate and Display Site Seals. Users will be able to identify who is the owner by looking at the certificate data if go through a more thorough validation process. Most EV SSL certificates also include a site seal, which demonstrates that website is secure and trustworthy. All of these visual cues will make it clear to users that they are not on a typosquatting website.

Trademarking brands is another approach to protect the website against typosquatting. This could contain the logos, taglines, and brand names. The United States Patent and Trademark Office is where to file the trademark (USPTO). The protection and remedies will be given for those in the thick of a typosquatting investigation.

Vulnerability: Inform the shop about a typosquatting trick	
Risk	High
	If an attacker can identify the vulnerable components that a particular application uses, it can be easily exploited because the exploit methods are already available on the internet, and the attacker only needs to use them. This can result complete data compromise and even server/host can take over for organisations.
Complexity	Medium
	We must inform the shop about a typosquatting trick it has been a victim of at least in v6.2.0-SNAPSHOT. (Mention the exact name of the culprit)

Summary:

This challenge is focus about security issues of library. There is no actual malice or mischief included as the typosquatter is completely harmless in this challenge. Keep in mind that, in fact, a situation like this could have severe implications and be even more difficult to spot.

We can demonstrate this by performing the following steps:

0. Access the developer's forgotten backup file and open the package.json.bak file
1. Analysing each entry in the dependencies list until will at some point get to epilogue-js.

```
J,
"dependencies": {
  "body-parser": "~1.18",
  "colors": "~1.1",
  "config": "~1.28",
  "cookie-parser": "~1.4",
  "cors": "~2.8",
  "dottie": "~2.0",
  "epilogue-js": "~0.7",
  "errorhandler": "~1.5",
  "express": "~4.16",
  "express-jwt": "0.1.3",
  "fs-extra": "~4.0",
  "glob": "~5.0",
```

Figure 2.1.8-1 – epilogue-js in the text file

2. Visit <http://localhost:3000/#/contact>
3. Submit the feedback with epilogue-js in the comment to solve this challenge

You successfully solved a challenge: Legacy Typosquatting (Inform the shop about a typosquatting trick it has been a victim of at least in v6.2.0-SNAPSHOT. (Mention the exact name of the culprit))

X

Figure 2.1.8-2 – Successful solve the challenge

Recommended solution:

When a security flaw is discovered, the development team can address one of the circumstances listed below.

1. Use tools like versions, DependencyCheck to keep track of the versions of both client-side and server-side components (e.g. frameworks, libraries) and their dependencies.
2. Remove unneeded dependencies, features, components, files, and documentation.

3. Keep an eye out for unmaintained libraries and components that don't produce security patches for previous versions. If patching isn't an option, try using a virtual patch to track, identify, and guard against the problem.
4. Only use secure URLs to obtain components from official sources. To limit the risk of a changed, malicious component being included, prefer signed packages.

2.1.9 Broken Anti-Automation

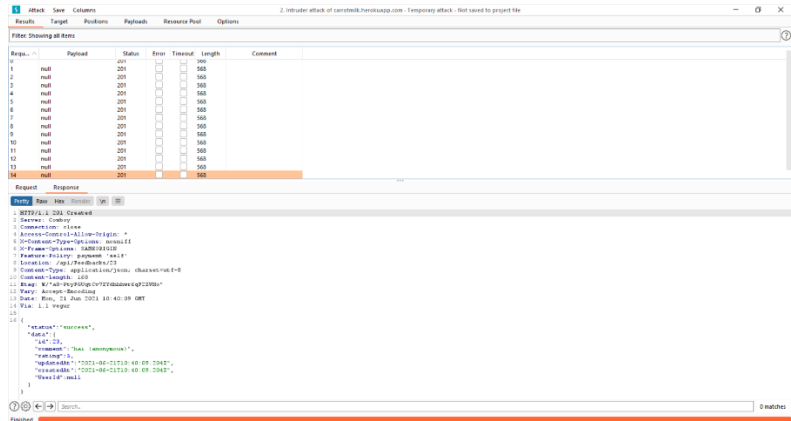
Broken Anti-Automation is happening when website fail to identify and block automated assaults could allow an attacker to conduct brute force attacks or cause service denial. All of the application's web forms must be safeguarded from automated submissions. An attacker can generate bogus accounts or overwhelm the database by automatically filling out and submitting registration forms. Spam can be sent to application administrators or users using contact and messaging forms that do not block automated form submissions. Automated password cracking applications may target login forms with poor anti-automation features.

The functionality of a web application that is frequently targeted by automation attacks includes:

- Application login forms: to guess user passwords, attackers may automate brute force login queries.
- Forms for registering for services: attackers might create thousands of new accounts instantly.
- Email forms: attackers may utilise email forms to send spam or flood a specific user's mailbox.
- Account maintenance: An attacker can launch a distributed denial of service (DDoS) attack on an application by flooding it with requests to disable or delete user accounts.
- Forms requesting account information: attackers may attempt to extract user personal information in bulk from an online application.
- Comment forms / Content Submission forms: they can be used to spam blogs, web forums, and web bulletin boards by posting spam or web-based malware automatically.

Implementing CAPTCHA methods in online applications is a typical approach for defending against automation assaults. The acronym CAPTCHA stands for 'Completely Automated Public Turing Test to Tell Computers and Humans Apart,' which defines the purpose. These early CAPTCHAs were text that had been

manipulated in some way to make bots unable to understand it. While they were initially quite successful, rapid developments in computing meant that bots could read the content. Artificial intelligence (AI) has progressed to the point that a modern "CAPTCHA bot" or "block reCAPTCHA tool" can easily overcome the test and negating their purpose.

Vulnerability: CAPTCHA Bypass	
Risk	High
	Failure to detect and block automated assaults could allow an attacker to conduct brute force attacks or cause service denial.
Complexity	Medium
	We must submit 10 or more customer feedbacks within 10 seconds.
<p>Summary: This application allow attacker to beat this automation protection.</p> <p>We can demonstrate this by performing the following steps:</p> <ol style="list-style-type: none"> 1. Set up the Burp. 2. Fill in the Customer Feedback form and submit. 3. Intercept the Feedback page POST request. 4. Send the captured request to repeater and then send it to intruder. 5. Clear the all the payload markers. 6. Select the null payload and set it to 15. 7. Start the attack using brute force. 8. Check all payload to make sure all success. 	
	
<p>Figure 2.1.9-1 – Result of intruder attack</p>	



You successfully solved a challenge: CAPTCHA Bypass (Submit 10 or more customer feedbacks within 10 seconds.) X

Figure 2.1.9-2 – Successfully submit the feedback

Recommended solution:

1. Use real CAPTCHA, limit the number of feedback comments each registered user can submit at a certain time period, and only enable registered users to provide feedback.
2. Implementing a multi-factor authentication (MFA). This method verifies that the systems are being accessed by real people. For example, someone want to log in into their account and then give them a text message containing a one-time passcode that they must enter on the website to proceed to the next step. While this strategy may be useful in secure contexts, such as banking and brokerage accounting software, it will almost certainly cause much too much user friction in the average business.
3. Apply ad fraud to protects the website from malware and human fraud
4. Use biometrics. Biometrics could be used to verify that users are genuine people and not bots. For example, use their fingerprints to verify their identification. Other types of biometrics are typing biometrics, speech recognition, and facial recognition. Biometrics may not be the ideal option for some use case as those systems also quite costly. Only few people want to hand up their biometric data to a company that sells socks.

2.2 Internal Threats

An internal threat is a threat which comes from within the organization running the system. This usually involves current or former employees, or associates who has privileged access to the system. These threats could arise from various intents such as accidentally creating a security vulnerability, consciously not adhering to security protocols, or actively conducts malicious actions against the system.

Some of the internal threats to this system are as follows:

1. Turn-cloaks. A turn-cloak is an insider who abuses legitimate credentials and privileges steals data for personal incentives or financial gains. Turn-cloaks are especially dangerous because they are familiar with the security protocols of the company or system, and has legitimate credentials to abuse for carrying out attacks.
2. Careless insiders. A careless insider is a loyal, regular employee who made a mistake which opened up opportunities for external threats to take place. This can be

caused by employees' actions such as forgetting to adhere to security protocols, mistakenly place sensitive information in an insecure database or creating weak credentials.

We suggest a few ways to counter these internal threats:

1. Monitor file access. Set up a system to monitor the transfer of files to track the file access of every employee. Any employee who accesses a large number of files in a short time or accessing unusual files which does not relate to their currently assigned tasks, might indicate that they are stealing data.
2. Review and enforce security policies. Security policies need to be constantly updated to keep up with the newest emerging internal threats. These policies should be strictly enforced upon every employee. Any security policy violation which could create opportunities for more threats, should have serious actions taken to remedy the damage and ensure more violations do not occur.

3.0 Discussion

Based on the penetration test, suitable tools should be considered to ensure that remedial actions are completed on time. While a thorough list of items that should be addressed is within the scope of this engagement, there are a few high-level items worth mentioning.

There are certain recommendations to improve the security of the website as follows:

1. Password strength validation should be implemented. As we all know, a weak password can be easily led to a cyberattack. It is important because it acts as the first barrier of defense against unauthorized intrusion.
2. Establish trust boundaries. Define your trust limits. Create logical trust boundaries on the internal network as needed. Each logical trust segment should be able to be breached without simply cascading the breach to other segments. This should include using separate administrative accounts for each segment so that a hacked system in one segment cannot be used in another.
3. Do a regular security patch. A security patch is a modification made to address a vulnerability. This corrective action will avoid successful exploitation and reduce a threat's potential to exploit a specific vulnerability.

4.0 Conclusion

OWASP Juice Shop offers various hacking challenges in which the user is expected to exploit the implicit vulnerabilities. Aside from hacker and awareness training, we can also utilise Juice Shop as a "guinea pig" website to see how effectively the tools can handle the JavaScript-heavy application frontends and REST APIs.

The objectives in doing the penetration test are:

- Identify the vulnerabilities found in the OWASP Juice Shop
- Proposed the security enhancement that can be made to secure the OWASP Juice Shop

The objectives of the penetration test were met. To prevent low levels of access control on the network and the server, it is essential to secure your site with a high-level security solution. Much effort is needed to enhance the security of a website for the security failures can be reduced.

References

- OWASP TOP 10 Vulnerabilities. OWASP. (2017). <https://owasp.org/www-project-top-ten/2017/>.
- Kimminich, B. (2014). *OWASP Juice Shop Introduction*. OWASP. <https://owasp.org/www-project-juice-shop/>.
- Kimminich, B. (2014). *Architecture overview*. Architecture overview · OWASP Juice Shop. <https://pwning.owasp-juice.shop/introduction/architecture.html>.
- A1:2017-Injection. OWASP. (2017). https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.
- Poza, D. (2020, August 20). *What Is Broken Authentication?* Auth0. <https://auth0.com/blog/what-is-broken-authentication/>.
- Security, C. (n.d.). Broken Authentication. <https://www.contrastsecurity.com/knowledge-hub/glossary/broken-authentication>.
- Kiprin, B. (2021, April 2). *Fuzzer (Sensitive Data Exposure)*. Crashtest Security. <https://crashtest-security.com/sensitive-data-exposure/>.
- Kudkar, I. (2021, January 25). *OWASP : SENSITIVE DATA EXPOSURE Attacks*. Medium. <https://medium.com/shallvhack/owasp-sensitive-data-exposure-attacks-7ef41e6b4a59>.
- Dalci, E. (2021, March 15). *CWE-20: Improper Input Validation*. Common Weakness Enumeration. <https://cwe.mitre.org/data/definitions/20.html>.
- Rope Security. (n.d.). What is a Mass Assignment Vulnerability? <https://ropesec.com/articles/mass-assignment/>.
- A5:2017-Broken Access Control. OWASP. (2017). https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.
- What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy. (2021). Retrieved from Portswigger.net website: <https://portswigger.net/web-security/cross-site-scripting>
- Insecure deserialization | Web Security Academy. (2021). Retrieved from Portswigger.net website: <https://portswigger.net/web-security/deserialization>
- A9:2017-Using Components with Known Vulnerabilities | OWASP. (2017). Retrieved June 27, 2021, from Owasp.org website: https://owasp.org/www-project-top-ten/2017/A9_2017-Using_Components_with_Known_Vulnerabilities#
- What is Typosquatting (and how to prevent it) | UpGuard. (2020). Retrieved June 27, 2021, from Upguard.com website: <https://www.upguard.com/blog/typosquatting>

Bjoern Kimminich. (2013). Broken Anti-Automation - Pwning OWASP Juice Shop. Retrieved June 27, 2021, from Gitbooks.io website: <https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/part2/broken-anti-automation.html>

Kahn, R. (2021). CAPTCHA and reCAPTCHA: How Can You Bypass It? Retrieved June 27, 2021, from Anura.io website: <https://www.anura.io/blog/captcha-and-recaptcha-how-can-you-bypass-it>

Imperva. (2019, December 29). *What Is an Insider Threat: Malicious Insider Attack Examples:* Imperva. Learning Center. <https://www.imperva.com/learn/application-security/insider-threats/>.

Peters, J. (2021, January 29). *What is an Insider Threat? Definition and Examples:* Varonis. Inside Out Security. <https://www.varonis.com/blog/insider-threats/>.