🔐 Vulnerability Assessment Report – Pet Store

Submitted by: Abhishek Prasad (Certified Ethical hacker)
Target URL: https://petstore.octoperf.com/

👤 About Me

I am Abhishek Prasad, a cybersecurity enthusiast with CEH certification and a keen interest in web application security and bug bounty hunting. I'm currently applying for a security internship to gain hands-on experience in identifying and reporting real-world vulnerabilities.

🎯 Objective

This report presents the findings from a black-box vulnerability assessment of the Pet Store web application. The goal was to identify security flaws, demonstrate their impact, and suggest mitigation strategies.

🛠️ Scope & Tools

- **Scope:** Public features of https://petstore.octoperf.com/
- **Methodology:** Manual testing + tools
- **Tools Used:** Burp Suite, SQLMap, browser dev tool.

📌 Focus Areas

- OWASP Top 10 vulnerabilities
- Session management flaws
- Authentication and token weaknesses
- Client-side and API security

🛡 Vulnerability 1: Account Takeover via Predictable Credentials

Severity: High ●

⍰ Description:

The application allows users to log in using **extremely weak or predictable username and password combinations** (e.g., yash:yash, user:user). These credentials directly grant access to **existing user accounts**, such as yashwant, without any registration, verification, or ownership validation.

This implies that:

- Accounts are created using guessable usernames and passwords.
- There is **no uniqueness check or ownership binding** between credentials and identity.
- Weak or reused credentials across accounts are exploitable.

🔥 **Exploitable Scenario:**

An attacker can gain unauthorized access to other users' data (such as transaction history, profile info) using trivial credential guessing:

Example 1:

Username: yash
Password: yash
→ Logs into the account of yashwant from Odisha (includes personal data and transactions).

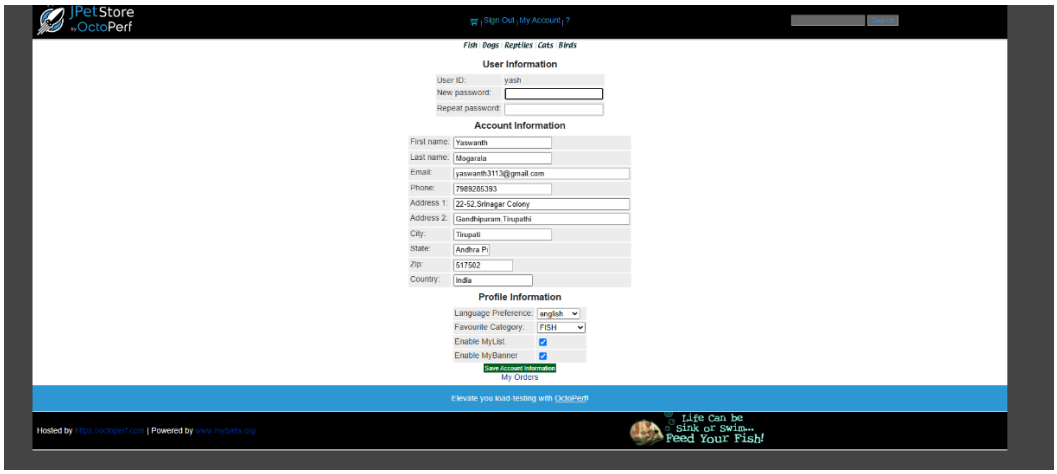
Example 2:

Username: user
Password: user
→ Logs into another user's account without validation.

The attacker can automate this attack using Burp Suite or a custom brute-force script.

🛠 Recommended Fix:

- ✅ Enforce **strong password requirements** (minimum length, complexity).
- ✅ Implement **rate-limiting and account lockout** after several failed login attempts.
- ✅ Ensure that **each account is uniquely associated with a verified user**.
- ✅ Prevent **reuse of weak default credentials** (e.g., user:user, admin:admin).
- ✅ Monitor and log suspicious login behavior for detection.



🛡 Vulnerability 2: Brute Force Login Without Rate Limiting

Severity: High ●

⍰ Description:

The login endpoint at /actions/Account.action lacks any form of **rate limiting or brute-force protection**. An attacker can send an **unlimited number of login attempts** without being blocked, throttled, or challenged with a CAPTCHA or delay.

This allows an attacker to:

- Attempt **credential stuffing** with leaked usernames/passwords.
- Use tools like **Burp Intruder**, **Hydra**, or custom scripts to perform high-speed brute-force attacks.
- Compromise accounts using **dictionary or weak password attacks**.

🔥 **Exploitable Scenario:**

Using **Burp Suite Intruder (Community Edition)**, the attacker sends hundreds of login attempts using predictable usernames and passwords (e.g., user:user, test:test, admin:admin, etc.).

Despite repeated failures, the server:

- Returns a valid response without any delay.
- Does **not block the IP** or enforce a CAPTCHA.
- Does **not log out** previous sessions or notify users.

This makes brute-force attacks trivial.

🛠 **Recommended Fix:**

- ✓ Implement **rate limiting** (e.g., max 5 attempts per minute per IP/user).
- ✓ Add **CAPTCHA** after a certain number of failed login attempts.
- ✓ Implement **account lockout** or **temporary suspension** after repeated failures.
- ✓ Log and monitor suspicious login activity.
- ✓ Apply **IP-based throttling** and **progressive delays** on failed logins.

🛡 Vulnerability 3: Lack of CSRF Protection on Order Submission

Severity: High ●

🔍 Description:

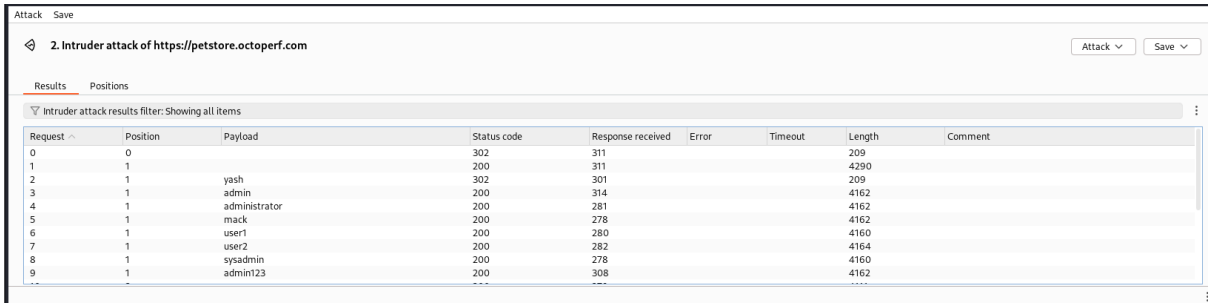
The order placement (POST /actions/Order.action) endpoint lacks anti-CSRF tokens or origin checking. This allows attackers to craft malicious requests that trigger transactions on behalf of logged-in users.

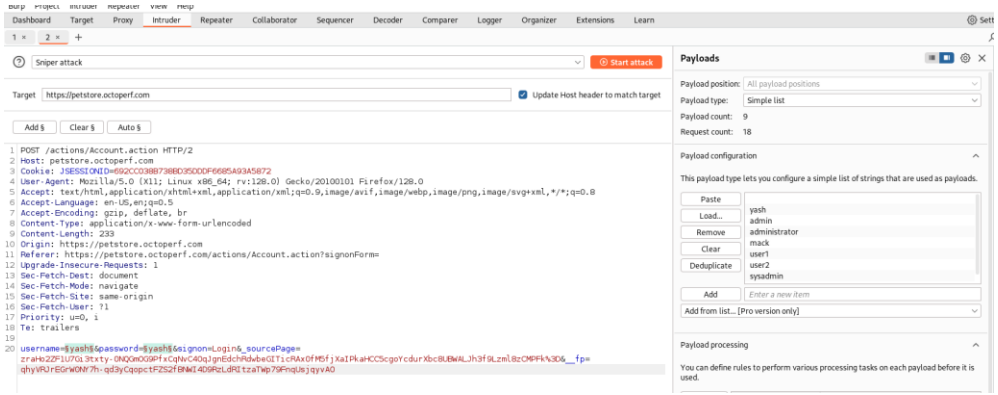
🔥 Exploitable Scenario:

An attacker hosts a page with an auto-submitting form that posts an order. If a victim is logged in, it places an unintended order using the victim's session cookie.

🛠 Recommended Fix:

- Implement CSRF tokens for all state-changing requests.
- Use SameSite cookie attributes.
- Check Origin and Referer headers server-side.





🛡 Vulnerability 4: Insecure Handling of Credit Card Data (Unmasked & Unvalidated)

Severity: Critical ●

□ Description:

The checkout form accepts and processes credit card data **without validation**, and displays the **full card number in plain text** across user-facing pages. This behavior indicates a **lack of input sanitization, format checking**, and **data masking**, which are essential for secure handling of payment information.

While this is a demo environment, if the same implementation existed in production, real users' card data would be exposed — leading to **severe risks of financial theft, PCI DSS non-compliance**, and legal liability.

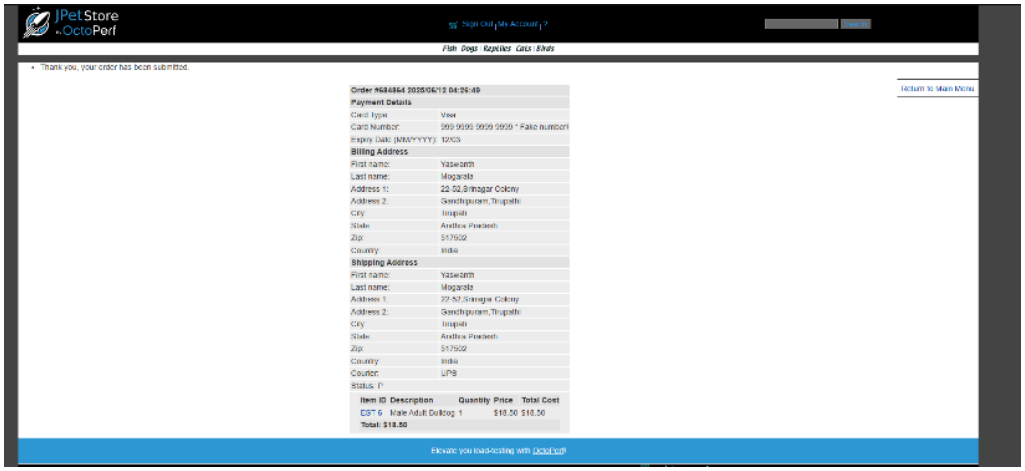
🔥 Exploitable Scenario:

An attacker can:

- Submit fake or malformed card numbers (e.g., 9999 9999 9999 9999) and proceed through checkout.
- If real card data were used, view the **entire credit card number and expiry** directly in their session or account page.
- Perform card stuffing, abuse unvalidated inputs, or extract payment data for fraud.

🛠 Recommended Fix:

- ✓ **Validate** credit card input (format, length, Luhn checksum).
- ✓ **Mask** card numbers in UI and session storage (e.g., **** **** **** 1234).
- ✓ **Do not store** card data unless absolutely necessary — if so, ensure **PCI DSS compliance**.
- ✓ Use **tokenization** or a **secure third-party payment gateway** to handle transactis

- 
JPetStore
OctoPerf
Sign Out | My Account | ?
Search
Fish Dogs Reptiles Cats Birds
Thank you, your order has been submitted.
Return to Main Menu
Order #6846864 2025/06/12 04:26:49
Payment Details
Card type: Visa
Card Number: 999 9999 9999 9999 * Fake number!
Expiry Date (MM/YYYY): 12/03
Billing Address
First name: Yaswanth
Last name: Mogarala
Address 1: 22-52,Srinagar Colony
Address 2: Gandhipuram,Tirupathi
City: Irrupati
State: Andhra Pradesh
Zip: 517502
Country: India
Shipping Address
First name: Yaswanth
Last name: Mogarala
Address 1: 22-52,Srinagar Colony
Address 2: Gandhipuram,Tirupathi
City: Irrupati
State: Andhra Pradesh
Zip: 517502
Country: India
Courier: UPS
Status: P
Item ID Description Quantity Price Total Cost
EST-6 Male Adult Bulldog 1 $18.50 $18.50
Total: $18.50
Elevate your load-testing with OctoPerf!

🛡 Vulnerability 5: Session Fixation

Severity: High ●

❓ Description:

The application fails to generate a **new session ID** after a successful login. Instead, it reuses the existing unauthenticated session. This behavior allows attackers to **"fix" a session ID before login** and hijack the user's session once they authenticate.

🔍 Evidence:

- **Session ID Before Login**: 7FAAD3AD9B5A37CC5455E47D1079CEC5
- **Session ID After Login**: 7FAAD3AD9B5A37CC5455E47D1079CEC5

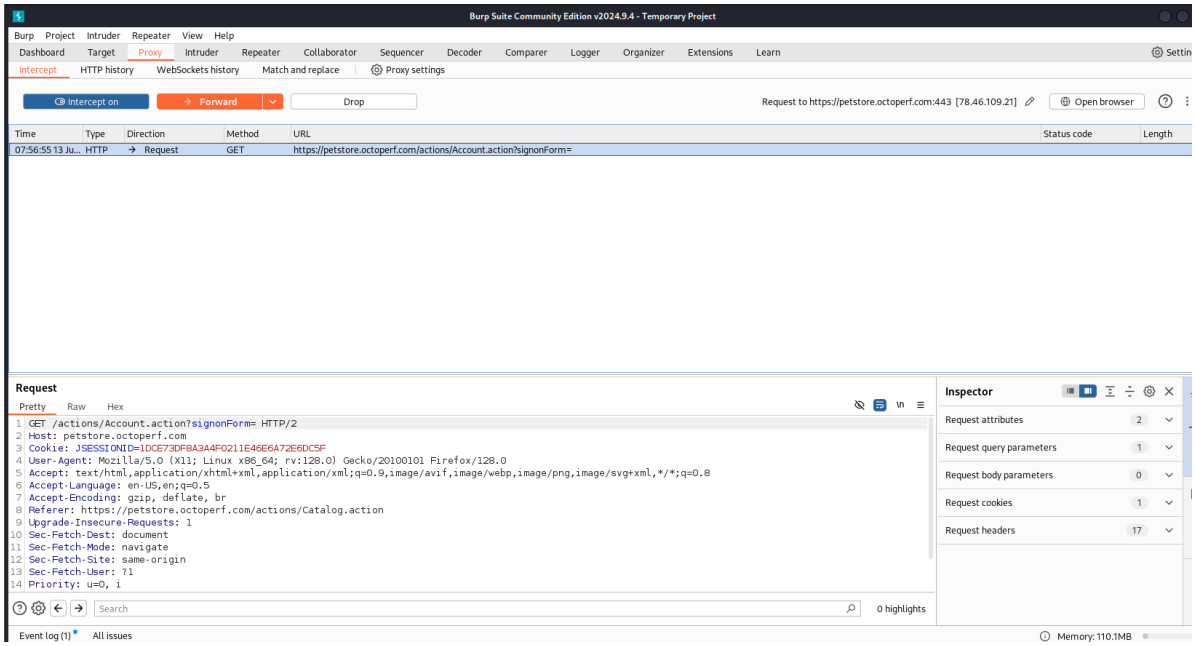
No regeneration took place — session ID remained the same before and after authentication.

🔥 Exploitable Scenario:

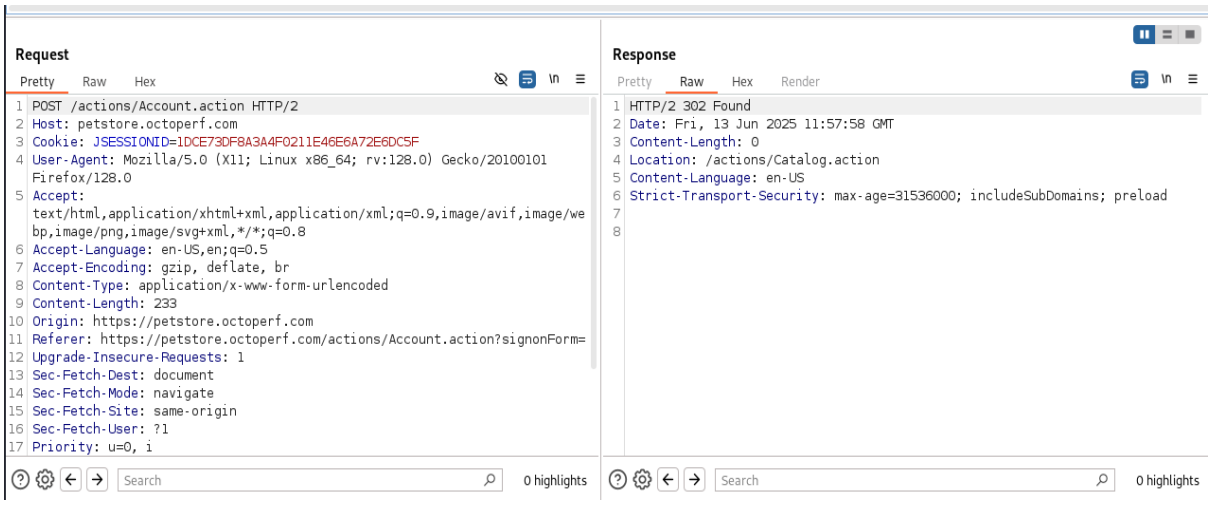
1. Attacker starts a session and notes the JSESSIONID.
2. Attacker sends a crafted link (with that session ID set via cookie or URL) to the victim.
3. Victim logs in → their session stays bound to the attacker's session ID.
4. Attacker reuses the same session ID and **gains full access to victim's authenticated session**.

🛠 Recommended Fix:

- Regenerate a fresh session ID **immediately after login** using secure session handling logic.
- Invalidate the old session (session.invalidate() in Java-based apps).
- Set security flags on cookies:
  - HttpOnly
  - Secure
  - SameSite=Strict
- Bind sessions to user-agent/IP heuristics for added protection.



NOTE : The Session ID before and after login is Same.



🛡 Vulnerability 6: Insecure Direct Object Reference (IDOR) Attempt on Order ID

Severity: Medium ⚠

⚠ Description:

The application uses predictable, incrementing Order IDs (e.g., 158747, 158748, etc.). When trying to access an order not belonging to the logged-in user, the server returns a **200 OK** HTTP response, along with a message:

"You may only view your own orders."

While this technically restricts access, it **still leaks metadata and confirms the existence of other users' orders**, which is **valuable information for attackers** performing IDOR enumeration or reconnaissance.

🔥 Exploitable Scenario:

An attacker can:

- Iterate through /Order.action?orderId=XXXX endpoints.
- Confirm which Order IDs are valid and exist (based on 200 OK vs 404 or other errors).
- Use this enumeration to build a map of user activities or brute force access if a future vulnerability is discovered.

📓 Example:

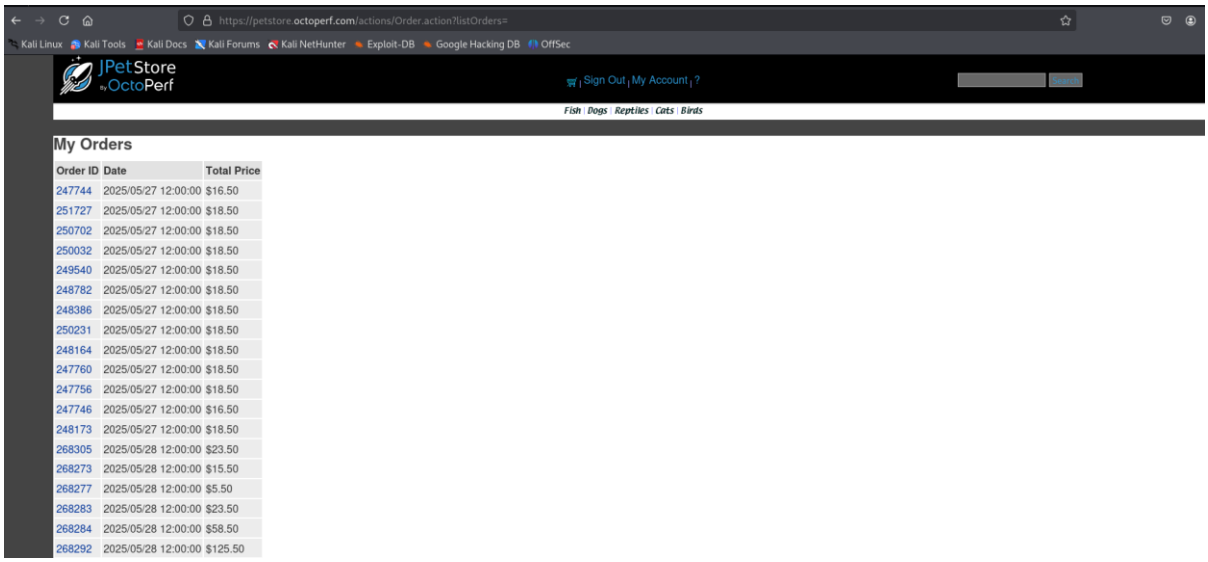
GET /actions/Order.action?orderId=158747 → shows valid order
GET /actions/Order.action?orderId=158748 → returns "You may only view your own orders." with HTTP 200
GET /actions/Order.action?orderId=999999 → 404 or blank page

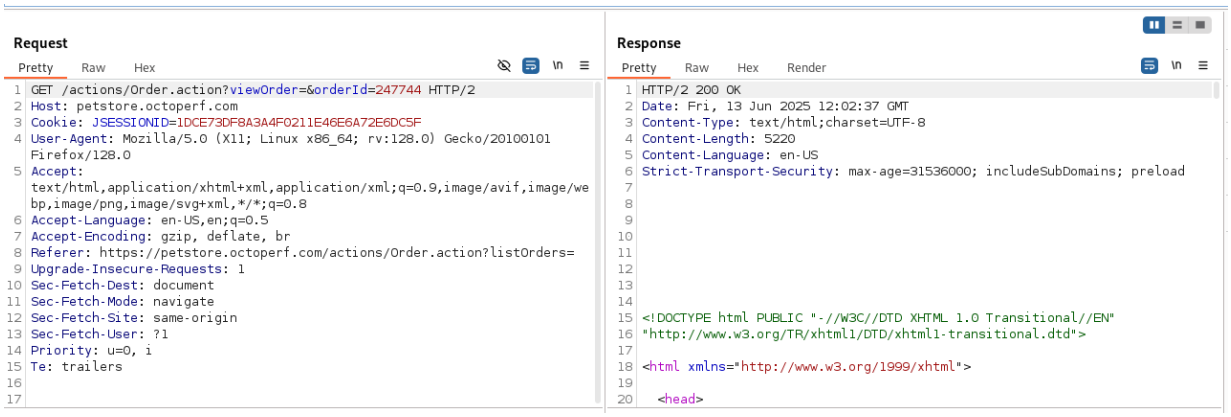
This proves the backend leaks **order existence** through response behavior.

🛠 Recommended Fix:

- Return **403 Forbidden** or **generic error messages** for unauthorized access.
- Avoid confirming the existence of resources the user doesn't own.
- Implement **access control checks** at the object level (user-role or session-based).
- Add logging and rate-limiting for excessive or sequential access patterns.
-

-





Request

Pretty Raw Hex

```
1  GET /actions/Order.action?viewOrder=&orderId=247744 HTTP/2
2  Host: petstore.octoperf.com
3  Cookie: JSESSIONID=1DCE73DF8A3A4F0211E46E6A72E6DC5F
4  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101
   Firefox/128.0
5  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
   bp,image/png,image/svg+xml,*/*;q=0.8
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate, br
8  Referer: https://petstore.octoperf.com/actions/Order.action?listOrders=
9  Upgrade-Insecure-Requests: 1
10 Sec-Fetch-Dest: document
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-User: ?1
14 Priority: u=0, i
15 Te: trailers
16
17
```

Response

Pretty Raw Hex Render

```
1  HTTP/2 200 OK
2  Date: Fri, 13 Jun 2025 12:02:37 GMT
3  Content-Type: text/html;charset=UTF-8
4  Content-Length: 5220
5  Content-Language: en-US
6  Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
7
8
9
10
11
12
13
14
15 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
16 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
17
18 <html xmlns="http://www.w3.org/1999/xhtml">
19
20   <head>
```

🐞 Vulnerability 7: Stack Trace Leak During Registration Error

Severity: Medium ⚠️

⚠️ Description:
When attempting to register with an already existing username or email, the application throws an **unhandled server-side exception** and displays a **full Java stack trace** in the browser. This output leaks sensitive backend implementation details, such as:

- Full exception names (SQLIntegrityConstraintViolationException, StripesServletException)
- Internal class structure (org.mybatis.jpetstore.service.AccountService)
- SQL constraints and table names (PK_ACCOUNT, ACCOUNT)
- Technologies in use: Spring, MyBatis, HSQLDB, Tomcat

This is considered **information disclosure**, and it increases the attack surface for further exploitation.

🔥 **Exploitable Scenario:**
An attacker can:

- Try registering with the same username/email repeatedly.
- Trigger a database constraint violation.
- Receive a complete backend stack trace.
- Extract technology stack, internal logic, and error handling flow from the exception.

This information can be used for:

- Planning targeted attacks (e.g., SQLi)
- Performing backend service enumeration
- Understanding database schema and application structure

📄 **Example:**

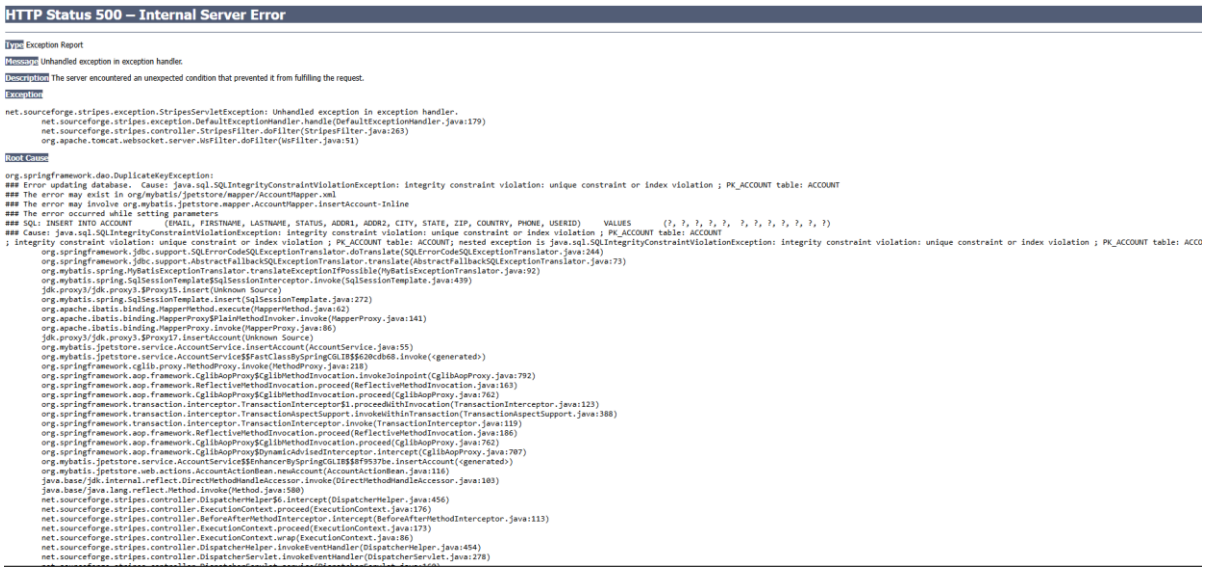
POST /actions/Account.action → Register with existing USERID
→ Response:
Type Exception Report
Message: Unhandled exception in exception handler

Exception: net.sourceforge.stripes.exception.StripesServletException
Caused by: java.sql.SQLIntegrityConstraintViolationException
Details: constraint violation; PK_ACCOUNT table: ACCOUNT

This proves that backend exception data is rendered directly to the client.

🛠 Recommended Fix:

- Do not return stack traces or backend error details to users.
- Implement generic, user-friendly error messages like "Username already taken."
- Log full stack traces **only** in server logs for debugging purposes.
- Validate input properly and handle exceptions securely.



HTTP Status 500 – Internal Server Error

Type Exception Report

Message Unhandled exception in exception handler.

Description The server encountered an unexpected condition that prevented it from fulfilling the request.

Exception

```
net.sourceforge.stripes.exception.StripesServletException: Unhandled exception in exception handler.
        net.sourceforge.stripes.exception.DefaultExceptionHandler.handle(DefaultExceptionHandler.java:179)
        net.sourceforge.stripes.controller.StripesFilter.doFilter(StripesFilter.java:263)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
```

Root Cause

```
org.springframework.dao.DuplicateKeyException:
### Error updating database.  Cause: java.sql.SQLIntegrityConstraintViolationException: integrity constraint violation: unique constraint or index violation ; PK_ACCOUNT table: ACCOUNT
### The error may exist in org/mybatis/jpetstore/mapper/AccountMapper.xml
### The error may involve org.mybatis.jpetstore.mapper.AccountMapper.insertAccount-Inline
### The error occurred while setting parameters
### SQL: INSERT INTO ACCOUNT         (EMAIL, FIRSTNAME, LASTNAME, STATUS, ADDR1, ADDR2, CITY, STATE, ZIP, COUNTRY, PHONE, USERID)     VALUES      (?, ?, ?, ?, ?,  ?, ?, ?, ?, ?, ?, ?)
### Cause: java.sql.SQLIntegrityConstraintViolationException: integrity constraint violation: unique constraint or index violation ; PK_ACCOUNT table: ACCOUNT
; integrity constraint violation: unique constraint or index violation ; PK_ACCOUNT table: ACCOUNT; nested exception is java.sql.SQLIntegrityConstraintViolationException: integrity constraint violation: unique constraint or index violation ; PK_ACCOUNT table: ACCO
        org.springframework.jdbc.support.SQLErrorCodeSQLExceptionTranslator.doTranslate(SQLErrorCodeSQLExceptionTranslator.java:244)
        org.springframework.jdbc.support.AbstractFallbackSQLExceptionTranslator.translate(AbstractFallbackSQLExceptionTranslator.java:73)
        org.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:92)
        org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:439)
        jdk.proxy3/jdk.proxy3.$Proxy15.insert(Unknown Source)
        org.mybatis.spring.SqlSessionTemplate.insert(SqlSessionTemplate.java:272)
        org.apache.ibatis.binding.MapperMethod.execute(MapperMethod.java:62)
        org.apache.ibatis.binding.MapperProxy$PlainMethodInvoker.invoke(MapperProxy.java:141)
        org.apache.ibatis.binding.MapperProxy.invoke(MapperProxy.java:86)
        jdk.proxy3/jdk.proxy3.$Proxy17.insertAccount(Unknown Source)
        org.mybatis.jpetstore.service.AccountService.insertAccount(AccountService.java:55)
        org.mybatis.jpetstore.service.AccountService$$FastClassBySpringCGLIB$$620cdb68.invoke(<generated>)
        org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
        org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.invokeJoinpoint(CglibAopProxy.java:792)
        org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
        org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:762)
        org.springframework.transaction.interceptor.TransactionInterceptor$1.proceedWithInvocation(TransactionInterceptor.java:123)
        org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:388)
        org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119)
        org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:186)
        org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:762)
        org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:707)
        org.mybatis.jpetstore.service.AccountService$$EnhancerBySpringCGLIB$$8f9537be.insertAccount(<generated>)
        org.mybatis.jpetstore.web.actions.AccountActionBean.newAccount(AccountActionBean.java:116)
        java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:103)
        java.base/java.lang.reflect.Method.invoke(Method.java:580)
        net.sourceforge.stripes.controller.DispatcherHelper$6.intercept(DispatcherHelper.java:456)
        net.sourceforge.stripes.controller.ExecutionContext.proceed(ExecutionContext.java:176)
        net.sourceforge.stripes.controller.BeforeAfterMethodInterceptor.intercept(BeforeAfterMethodInterceptor.java:113)
        net.sourceforge.stripes.controller.ExecutionContext.proceed(ExecutionContext.java:173)
        net.sourceforge.stripes.controller.ExecutionContext.wrap(ExecutionContext.java:86)
        net.sourceforge.stripes.controller.DispatcherHelper.invokeEventHandler(DispatcherHelper.java:454)
        net.sourceforge.stripes.controller.DispatcherServlet.invokeEventHandler(DispatcherServlet.java:278)
```

🛡 Vulnerability 8: Use of Vulnerable Components – Spring, MyBatis, jQuery

Severity: High ●

⍰ Description:
The application includes outdated and vulnerable third-party components:

Component	Version	Vulnerability
Spring Framework	4.1.x	CVE-2022-22965: Spring4Shell
MyBatis	3.x	CVE-2023-34055 – XXE risk
jQuery	1.4.2	Multiple XSS vulnerabilities

These components are unmaintained and known to contain critical vulnerabilities that can lead to remote code execution, cross-site scripting, or XML attacks.

🔥 **Exploitable Scenario:**
An attacker aware of Spring4Shell may attempt to exploit /actions/* endpoints with class loader injection.

🛠 **Recommended Fix:**

- Upgrade to supported and patched versions of all dependencies.
- Remove legacy jQuery and use a modern, minimal version.
- Use tools like OWASP Dependency-Check or Snyk to automate checks.

🛡 Vulnerability 9: Stored Cross-Site Scripting (XSS) in Account Details

Severity: High ●

□ Description:

The application allows a logged-in user to update their **account details** such as name, address, city, etc., without any input validation or sanitization. Malicious JavaScript payloads entered in these fields are stored and rendered back unsafely on user-facing pages, such as:

- Account information page
- Order confirmation pages
- Invoice or order history views

This leads to a **persistent XSS** vulnerability, where an attacker can execute arbitrary JavaScript in the context of other users who view this data.

🔥 **Exploitable Scenario:**

1. Attacker logs in and goes to **Edit Account Information**.
2. In the "First Name" field, inputs:

   <script>alert('XSS')</script>

3. Saves the form.
4. The application stores this value and renders it **without sanitization**.
5. When the user visits their account details page or order confirmation, the payload **executes in the browser**.

📓 **Example Payloads Used:**

- <script>alert('XSS')</script>
- <img src=x onerror=alert(1)>
- "><svg/onload=alert(1337)>

🛠 **Recommended Fix:**

- ✅ Implement **input validation and sanitization** on all user-supplied input.
- ✅ Encode output before rendering to HTML (e.g., use escapeHtml() in Java, or context-aware output encoding).
- ✅ Apply a **Content Security Policy (CSP)** to reduce XSS impact.
- ✅ Sanitize inputs at both client-side and server-side for defense in depth.

📄 Summary of Findings – Demoblaze

Total Vulnerabilities Reported: 9
Testing Approach: Manual black-box testing
Tools Used: Burp Suite, OWASP to 10, browser dev tools

🔎 Vulnerability Summary

Severity	Count
Critical	1
High	6
Medium	2
Low	0

❓ Key Findings

- **[Critical]** Unmasked and unvalidated credit card data accepted and displayed in plain text.
- **[High]** Login form vulnerable to brute-force (no rate-limiting, no CAPTCHA).
- **[High]** Weak default credentials (e.g., user:user) allow unauthorized access.
- **[High]** Session fixation due to non-regenerated session ID post-login.
- **[High]** Missing CSRF protection on order submission endpoint.
- **[High]** Stored XSS via unsanitized account detail fields.
- **[Medium]** IDOR enumeration leaks order metadata (200 OK with access denied message).
- **[Medium]** Full backend Java stack trace leaked on registration error.
- **[High]** Use of outdated libraries: Spring 4.1.x (CVE-2022-22965), jQuery 1.4.2 (XSS), MyBatis 3.x (XXE risk).

🛡️ Recommended Remediations

- Validate, mask, and tokenize payment inputs; remove card data storage.
- Enforce strong password policy and block weak/default logins.
- Add rate-limiting and CAPTCHA to login.
- Regenerate session ID after login; enable Secure/HttpOnly/SameSite on cookies.

- Implement CSRF protection via tokens and origin validation.
- Sanitize user input and apply output encoding to prevent XSS.
- Restrict object access at server-side; return proper HTTP status codes.
- Suppress detailed errors in client responses; log server-side only.
- Upgrade vulnerable dependencies to patched versions.