



External Penetration Test

Prepared for: Sample Inc

COMPLETE REPORT

April 16, 2026

Red Sentry

This engagement was performed in accordance with the Statement of Work, and the procedures were limited to those described in that agreement. The findings and recommendations in this report reflect the systems, scope, and timeframe defined in the Statement of Work.

While every effort was made to provide thorough coverage, no security assessment can identify every possible issue, and results should not be interpreted as a guarantee of complete security.

Contact Information

+1 (888) 337-0467

hello@redsentry.com



TABLE OF CONTENTS

Summary.....	4
Project Overview.....	4
Goals.....	4
Dates.....	4
Findings.....	4
Scope.....	5
Tests Performed.....	6
Findings.....	7
Appendices.....	9

SUMMARY

Project Overview

Sample Company engaged Red Sentry to assess the security of their external environment. The following report details the findings identified during the course of the engagement, which started on January 1st, 2026.

Goals

Identify vulnerabilities in the company's external perimeter that could be leveraged by an attacker.

Dates

April 08, 2026

KICKOFF

April 08, 2026 - April 16, 2026

TESTING PERIOD

April 16, 2026

DELIVERY

Scope

- www.samplecompany.com



Findings Overview

During the assessment, the testing team identified multiple security findings across the environment.

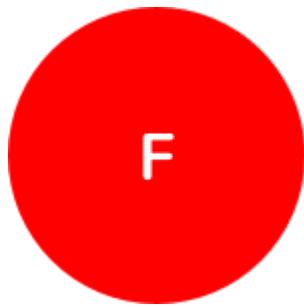
A finding represents a unique security issue with a distinct root cause.

Instances represent the number of instances where that issue appears within the assessed environment.

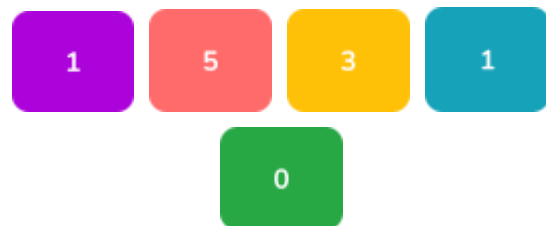
The following table summarizes the identified findings and the number of Instances observed during testing.

Severity	Finding ID	Instances
CRITICAL	T9H8YPG-F0001	2
	T9H8YPG-F0002	1
HIGH	T9H8YPG-F0003	1
	T9H8YPG-F0004	1
	T9H8YPG-F0005	1
	T9H8YPG-F0006	1
	T9H8YPG-F0007	1
MEDIUM	T9H8YPG-F0008	1
	T9H8YPG-F0009	1
	T9H8YPG-F0010	1
LOW		

Overall Grade



GRADE



EXECUTIVE SUMMARY

The assessment team conducted a thorough penetration test of Sample Company's external infrastructure, uncovering critical security issues.

Notably, two instances of a critical Remote Code Execution vulnerability were identified, posing a severe threat to the system.

Additionally, five high-severity findings were discovered, including Authentication Bypass leading to Stored XSS, Arbitrary Account Takeover, Insecure Direct Object Reference, Price Tampering, Sensitive Data Exposure, and Business Logic Vulnerability. These findings underscore significant risks to data confidentiality and operational integrity.

Furthermore, one low-severity finding was identified, related to a stack trace.

To address these issues, immediate remediation actions are recommended, including patching vulnerabilities, strengthening access controls, and implementing secure coding practices to enhance the project's security posture.

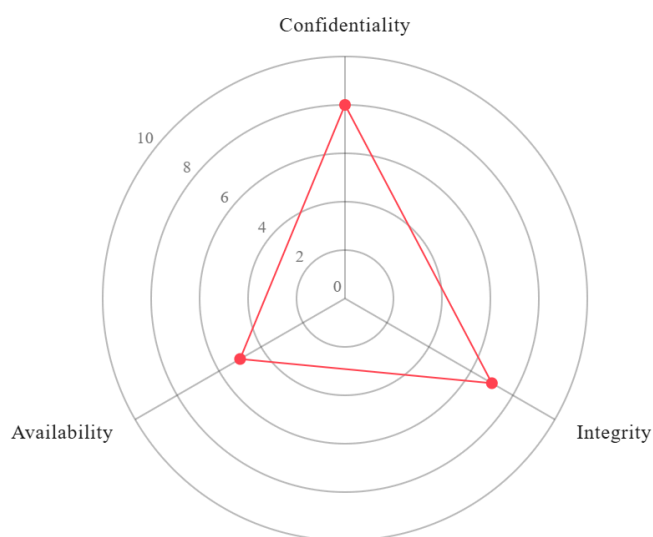


Figure 1: Security risk chart

As the image above shows, there's an overall high risk as most of the security elements risk scores are above 7. Therefore, the risk is in the high range.

As a general rule, an overall risk below 4 shows low business impact, between 4 and 7 medium impact and above 7 high impact.

Test Performed

For this project, the assessment team followed both a black-box approach to testing, looking for common vectors of attack according to the OWASP Top 10 methodology. From a black-box approach, the main goal is to discover issues from an unauthenticated perspective. This includes the following tests:

API Discovery

Using a proxy connection, intercept HTTP connections to discover underlying APIs that could expose resources for an unauthenticated user.

Authentication Bypass

We used a set of attacks to access information requiring session tokens. This involves active user enumeration attacks and password spraying from standard common lists.

Credential Stuffing

Attempt to make use of relevant leaked credentials gathered from the Dark Web or any other public or paid source.

Directory Discovery

Look for paths not available in tags present in the main HTML document. This was done using a brute-force approach based on standard lists of common path names.

Dorking

Use sophisticated queries to uncover information revealed by search engines that may not normally be discoverable by conventional searches.

Known Vulnerabilities

Search for CVEs based on the technology stack being used.

Reconnaissance

Utilize publicly available tools to gather sensitive information, such as that provided with a Shodan search.

Sensitive Data Exposure

Analyzed all of the information available to an unauthenticated user that could be leveraged against the company's confidentiality.

External Penetration Test Findings

REMOTE CODE EXECUTION

CRITICAL

Identifier	T9H8YPG-F0001
Severity	Critical
Instances	2

Description

This is one of the most dangerous types of computer vulnerabilities. It allows an attacker to remotely run malicious code within the target system on the local network or over the Internet. Physical access to the device is not required. An RCE vulnerability can lead to loss of control over the system or its individual components, as well as theft of sensitive data.

Instances (2)

- samplecompany.com
- app.samplecompany.com

Evidence

During the comprehensive penetration testing conducted by the assessment team, a web application was identified as a potential target. The team's attention was drawn to a specific endpoint, namely “_ignition/health-check.” Upon investigation, the team discovered a critical vulnerability marked by the presence of the configuration setting “can_execute_commands: true.” This alarming finding hinted at a potential remote code execution (RCE) vulnerability within the Web application framework.



Figure 2: Command Execution True

Subsequently, the assessment team leveraged their expertise to validate and exploit this vulnerability, ultimately confirming the presence of CVE-2021-3129. This particular vulnerability allows attackers to execute arbitrary code remotely, posing a significant threat to the security of the web application.

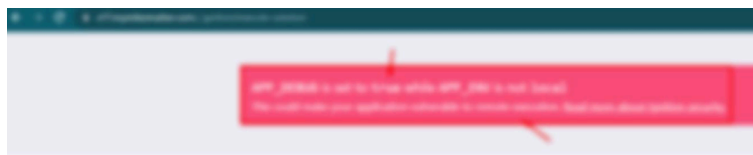


Figure 3: Debug mode Enabled

To carry out the exploitation, the team utilized a publicly available exploit for CVE-2021-3129, which is accessible at github.com/samplecompany. This exploit provides a streamlined method for attackers to take advantage of the identified vulnerability, emphasizing the critical nature of the security risk posed by the application's exposure to remote code execution.

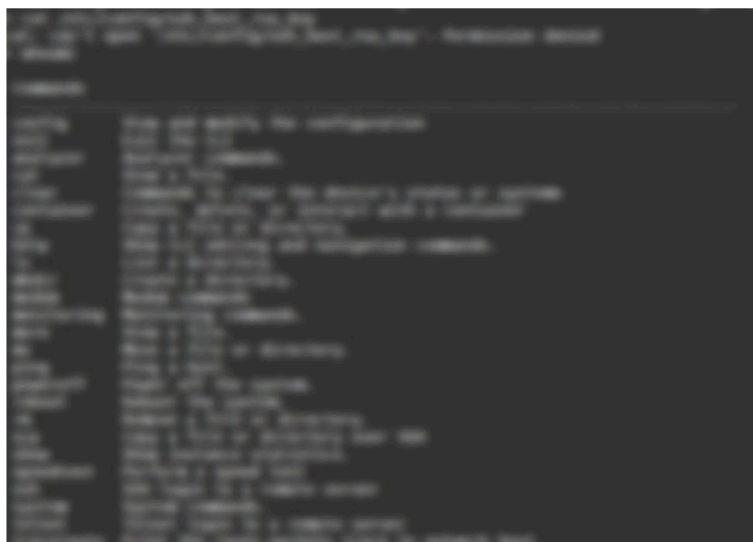


Figure 4: CVE Exploit

The CVE-2021-3129 vulnerability poses a severe security threat, enabling remote code execution. Exploitation allows malicious actors to execute arbitrary commands, compromising the integrity and confidentiality of the web application. This breach could result in unauthorized access, data manipulation, and potential damage to the overall system.

Steps for Recreation

1. Access the endpoint: `/_ignition/health-check`.
2. Observe **debug mode is enabled** and sensitive configuration is exposed.
3. Identify `can_execute_commands: true` in the response.
4. Confirm the application is vulnerable to **CVE-2021-3129**.
5. Use a publicly available exploit for CVE-2021-3129.
6. Send a crafted request to the vulnerable endpoint.
7. Observe successful **remote command execution (RCE)**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Immediately apply the latest security patches and updates to the framework to mitigate the CVE-2021-3129 vulnerability.
- Ensure that debug mode is disabled in production environments to reduce exposure to potential security risks.

Resources

CVE-2021-3129

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3129>

AUTHENTICATION BYPASS LEADS TO STORED XSS

HIGH

Identifier	T9H8YPG-F0002
Severity	High
Instances	1

Description

Authentication bypass leading to stored XSS represents a significant security risk for web applications. In this scenario, attackers exploit vulnerabilities in the application's authentication mechanisms to gain unauthorized access. By bypassing authentication controls, they can inject malicious scripts into the application's database or storage, where they remain stored (hence "stored XSS").

These injected scripts are designed to execute when accessed by other users, potentially compromising their sessions. This allows attackers to steal sensitive data such as credentials or session cookies, impersonate authenticated users, manipulate application functionality, and even conduct phishing attacks by presenting misleading content or forms.

Instances (1)

- sample.com

Evidence

During testing, it was observed that there is an 'xyz' functionality accessible to authenticated users. However, it was found that this functionality can also be accessed by unauthenticated users through manipulation of the response using Burp Suite. Furthermore, it can be exploited to reproduce stored Cross-Site Scripting (XSS).

When a user clicks on the 'xyz' functionality, the application sends an 'isAuthenticated' request to the server, as shown in the screenshot below. Since we are not authorized, the server responds with 'false'.

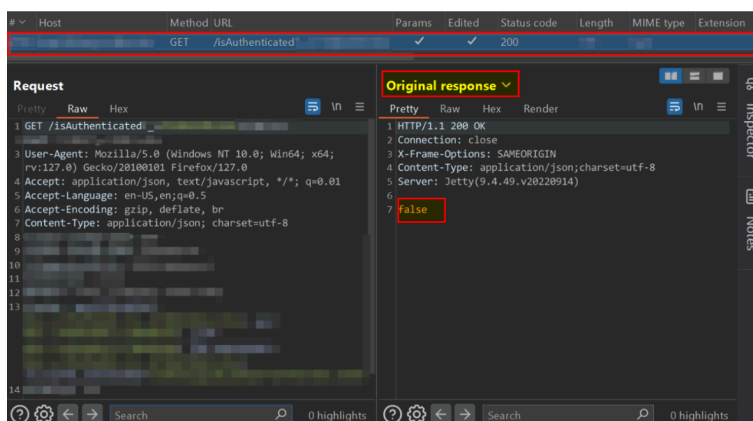


Figure 5: Burp screenshot showing the Original response

Now, in Burp Suite, Intercept the request, modify the response from 'false' to 'true,' and click the send button.

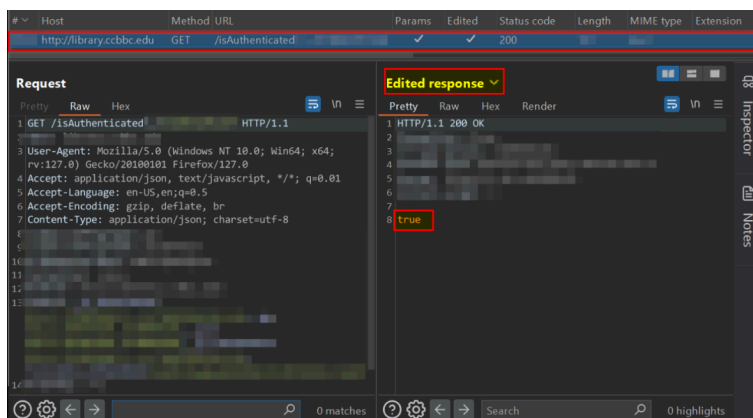


Figure 6: Burp screenshot showing the Edited response

Observe the response in the application: now, an unauthenticated user can add xyz, as depicted below. Enter the following payload into the 'xyz' functionality and click the 'Add' button.

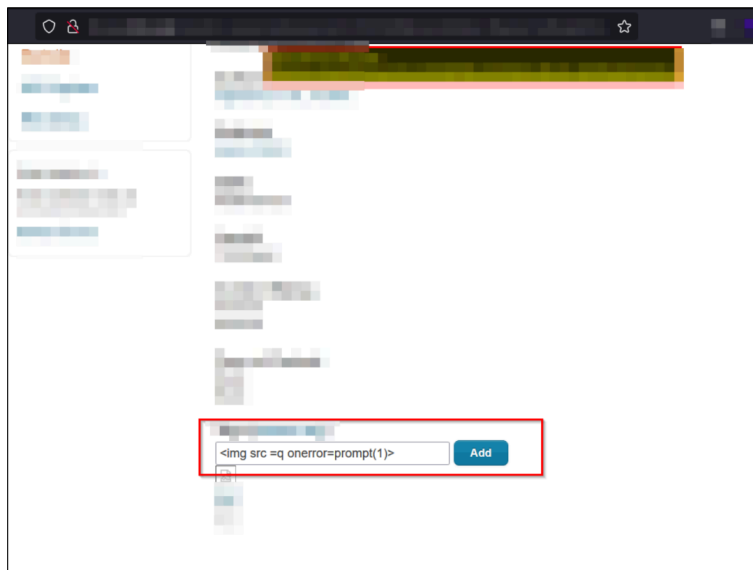


Figure 7: Burp screenshot showing unauthenticated users can access 'xyz' functionality. Observe that the payload has been executed successfully, indicating that the application is vulnerable to Stored cross-site scripting.

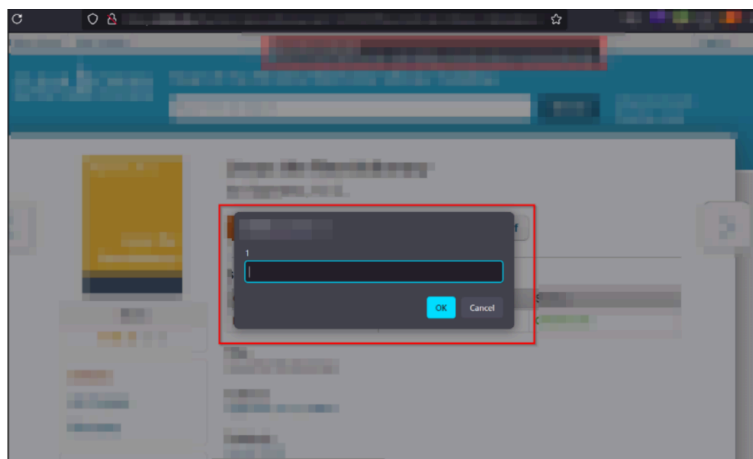


Figure 8: Burp screenshot showing that the XSS payload has been executed successfully. Attackers can inject and store harmful scripts within an application. These scripts can exploit vulnerabilities in user sessions, potentially compromising sensitive data such as credentials or session cookies. Attackers can impersonate authenticated users, manipulate data, conduct phishing attacks, and damage the application's reputation.

Steps for Recreation

1. Navigate to the 'xyz' functionality.
2. Intercept the request/response using Burp Suite.

3. Observe the server response: isAuthenticated: false.
4. Modify the response to isAuthenticated: true and forward it.
5. Gain access to the 'xyz' functionality as an unauthenticated user.
6. Enter a payload (e.g.,) and submit.
7. Observe the payload is **stored and executed**.
8. Confirm **Stored XSS via authentication bypass**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Implement Strong Authentication: Ensure robust authentication mechanisms and secure session management.
- Validate and Sanitize Input: Implement strict input validation to prevent the injection of malicious scripts. Use whitelisting where possible.
- Output Encoding: Encode output data to prevent script execution. Use proper escaping techniques suitable for the context (HTML, JavaScript, etc.).

Resources

More about Stored Cross-Site-Scripting and Authentication Bypass

<https://www.automox.com/blog/vulnerability-definition-authentication-bypass>

<https://portswigger.net/web-security/cross-site-scripting/stored>

ARBITRARY ACCOUNT TAKEOVER

HIGH

Identifier	T9H8YPG-F0003
Severity	High
Instances	1

Description

Arbitrary Account Takeover involves seizing control of user accounts without specific targeting. Attackers exploit vulnerabilities like weak authentication or flawed password reset mechanisms, compromising user privacy. Unauthorized access allows malicious activities, risking sensitive data exposure and unauthorized transactions.

Instances (1)

- www.samplecompany/password/reset

Evidence

The assessment team navigated to the password reset page at /password/reset. Initiating a password reset, they observed the system generating a POST request, including the hostname www.samplecompany.com. The team manipulated the hostname to an interactsh server, redirecting the password reset link to the controlled server.

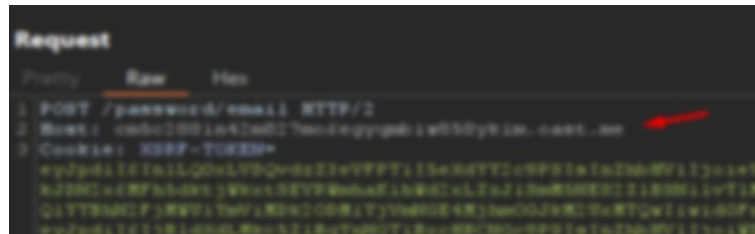


Figure 9: Host Header Injection

Below figure shows the **password reset link received**.

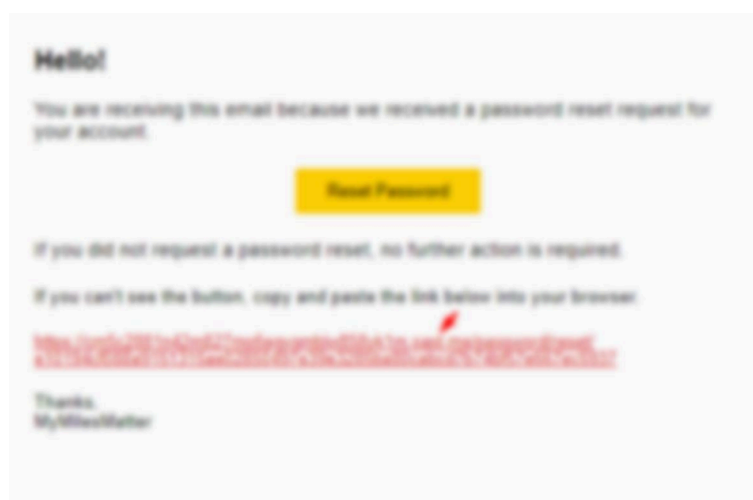


Figure 10: Password Reset Token

Leveraging this redirection, the assessment team intercepted and exfiltrated the password reset token using an interactsh client, demonstrating the vulnerability's exploitation potential.



Figure 11: Stolen Password Reset Token

Arbitrary Account Takeover poses a significant security threat by enabling unauthorized access to user accounts. Exploitation jeopardizes user privacy, allowing for unauthorized transactions and potential exposure of sensitive data. The attacker gains control without specific targeting, compromising the integrity and confidentiality of user information, with the potential for severe consequences such as financial losses and reputational damage.

Steps for Recreation

1. Navigate to `/password/reset` page.
2. Initiate a password reset request using a valid email address.
3. Intercept the POST request containing the **Host header** (e.g., `www.samplecompany.com`).
4. Modify the Host header to a **malicious/interactsh-controlled domain**.
5. Forward the request.
6. Observe the password reset link is generated using the attacker-controlled domain.
7. Capture the **password reset token via interactsh server**.
8. Use the stolen token to access the password reset functionality and reset the account password.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Implement additional verification steps, such as multi-factor authentication, to enhance the security of the password reset functionality and mitigate the risk of arbitrary account takeover.

- Strengthen hostname validation mechanisms to prevent manipulation, ensuring that password reset links are only generated for authorized and legitimate domains.

Resources

Password Reset Poisoning

<https://portswigger.net/web-security/host-header/exploiting/password-reset-poisoning>

INSECURE DIRECT OBJECT REFERENCE

HIGH

Identifier	T9H8YPG-F0004
Severity	High
Instances	1

Description

Insecure direct object references (IDOR) are a type of access control vulnerability that arise when an application uses user-supplied input to access objects directly. IDOR vulnerabilities are most commonly associated with horizontal privilege escalation, but they can also arise in relation to vertical privilege escalation.

Instances (1)

- sample.com

Evidence

During the assessment, the team discovered a potential Insecure Direct Object Reference (IDOR) vulnerability within the system. Initially, they identified the ability to send a POST request to delete an entry using the endpoint /REDACTED/entry/1054

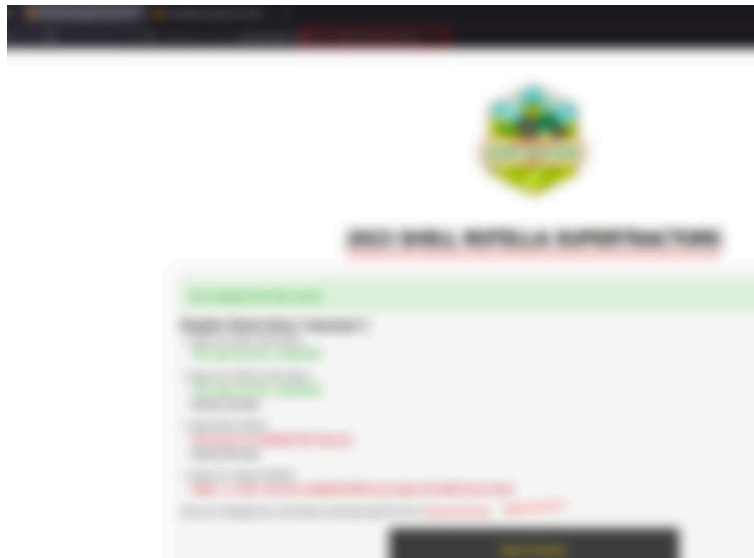


Figure 12: Entry Page

The below figure shows the entry delete request.

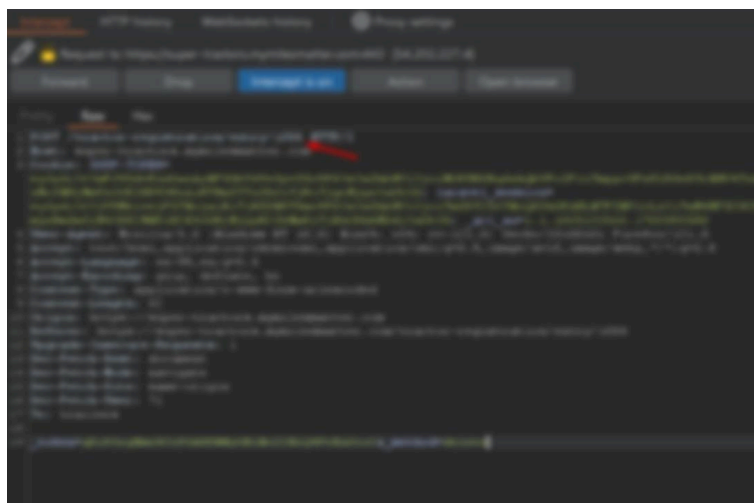


Figure 13: Entry Delete Request

Upon closer inspection, the team exploited the vulnerability by manipulating the ID parameter. By changing the ID number in the request, they successfully deleted entries belonging to other users. As a demonstration, the team targeted a demo entry with the ID 1055, showcasing the system's susceptibility to unauthorized deletion of arbitrary entries highlighting the critical IDOR security issue.



Figure 14: Deleted Entry

The IDOR vulnerability poses a severe security risk as it allows an attacker to manipulate entry IDs and delete arbitrary records. Exploiting this flaw could lead to unauthorized data deletion, impacting data integrity and confidentiality. This breach grants attackers undue control, enabling malicious manipulation and potential disruptions within the affected system.

Steps for Recreation

1. Navigate to the entry management feature.
2. Capture the **POST request** used to delete an entry (e.g., /REDACTED/entry/1054).
3. Modify the **entry ID** parameter in the request (e.g., change 1054 to 1055).
4. Forward the modified request.
5. Observe that the entry belonging to another user is **successfully deleted**.
6. Confirm presence of **IDOR vulnerability allowing unauthorized deletion of records**.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Implement access to certain functions limited to the user's role.
- Implement authorization checks with user policies and hierarchy.
- Do not rely on IDs that the client sends. Use IDs stored in the session object instead.
- Check authorization for each client request to access database.

- Use random IDs that cannot be guessed (UUIDs for instance)

Resources

Insecure direct object references (IDOR)

<https://portswigger.net/web-security/access-control/idor>

Insecure Direct Object Reference Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

PRICE TAMPERING

HIGH

Identifier	T9H8YPG-F0005
Severity	High
Instances	1

Description

The Price Tampering vulnerability in the application poses a security risk where unauthorized users can manipulate and tamper with the prices of products or services, leading to fraudulent transactions and financial losses, compromising the integrity of the e-commerce system. Attackers exploiting this vulnerability can maliciously alter the prices displayed to users, potentially leading to unauthorized discounts, overcharges, or other financial discrepancies.

Instances (1)

- sample.com

Evidence

The application permits unauthorized users to alter the fee amount, posing significant security and financial risks to the organization.

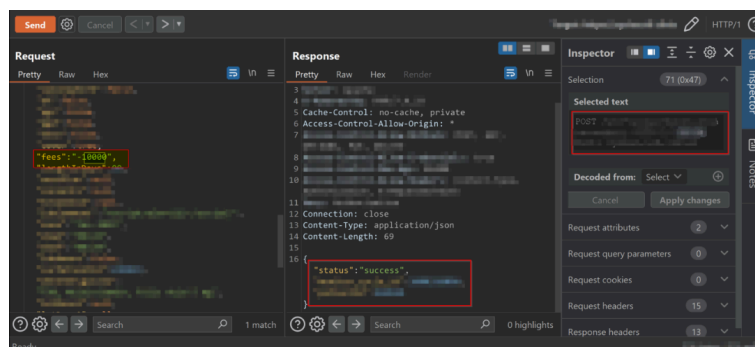


Figure 15: Burp screenshot showing the request with fees as “-10000”

In this scenario, attackers can input a negative value like -10,000 as fees due to inadequate request validation by the server. This oversight may result in substantial financial losses for the organization.

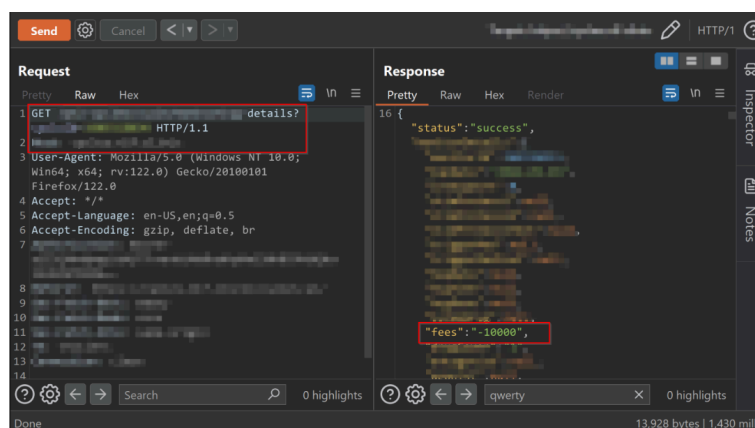


Figure 16: Burp screenshot showing the “GET” request with the updated “fees” values. The application's failure to validate requests results in the display of negative amounts, indicating that the organization owes money to the customer.

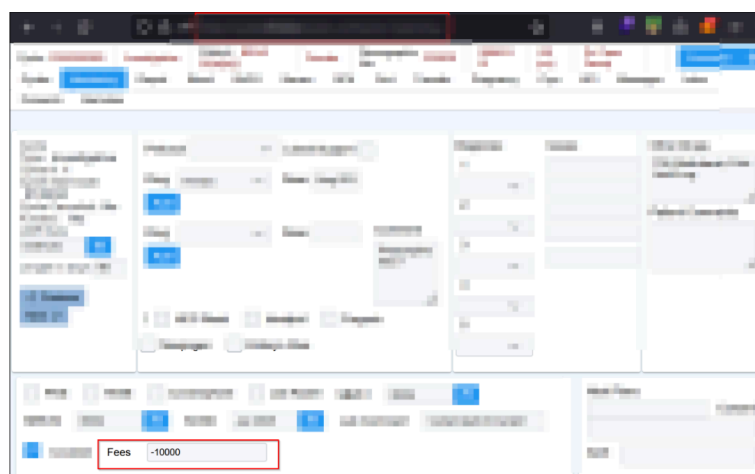


Figure 17: Browser screenshot showing that the fees amount has been successfully changed to negative

Attackers can fill negative prices, leading to financial losses, reputation damage, and potential legal repercussions. Customers may be deceived, impacting trust, and the company's competitiveness could suffer as competitors may exploit the situation. Additionally, regulatory compliance may be compromised, further exacerbating the consequences.

Steps for Recreation

1. Capture the request responsible for setting or updating the **fees amount**.
2. Modify the request parameter for fees (e.g., set value to -10000).
3. Forward the modified request to the server.
4. Observe that the server **accepts negative fee values without validation**.
5. Check the application response/UI and confirm the fees are updated to a negative amount.
6. Verify that the system reflects an incorrect balance (organization owes money to user).
7. Confirm lack of server-side input validation leading to financial manipulation vulnerability.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Ensure robust server-side validation of price-related parameters,
- Implement secure transmission protocols like HTTPS to encrypt communication.
- Employ client-side techniques such as immutable data structures to resist tampering attempts, conduct transaction integrity checks, enforce role-based access controls (RBAC) for price-related functionalities, and collaborate with payment processors to enhance security measures and fraud detection. These measures collectively fortify the e-commerce system against unauthorized alterations to prices, ensuring a secure and resilient defense posture.

Resources

More about Price Tampering:

https://owasp.org/www-community/attacks/Web_Parameter_Tampering

SENSITIVE DATA EXPOSURE

HIGH

Identifier	T9H8YPG-F0006
Severity	High
Instances	1

Description

Sensitive Data Exposure occurs when an organization unknowingly exposes sensitive data or when a security incident leads to the accidental or unlawful destruction, loss, alteration, or unauthorized disclosure of or access to sensitive data. Such exposure may occur due to inadequate protection of a database, misconfigurations when bringing up new instances of data stores, and inappropriate usage of data systems, among others.

Instances (1)

- sample.com

- Put authentication mechanisms in place in all your publicly facing applications, regardless of their environment (development, QA, etc.)

Resources

Sensitive data exposure

https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure

BUSINESS LOGIC VULNERABILITY: ALLOWING UNAUTHENTICATED USERS TO SUBMIT RATINGS

MEDIUM

Identifier	T9H8YPG-F0007
Severity	Medium
Instances	1

Description

This vulnerability arises when a system allows unauthenticated users to submit ratings without verifying their identity. This oversight can lead to inaccurate or malicious submissions that undermine the reliability of the rating system. It opens the door to manipulation, such as fake ratings skewing product perceptions or competitive sabotage. Proper authentication mechanisms are crucial to ensure that only authenticated users can contribute, maintaining the integrity and trustworthiness of the ratings provided.

Instances (1)

- sample.com/reviews

Evidence

The testing team observed that an unauthenticated user could give ratings for XYZ, and this task can be automated with the help of Burp Suite's Intruder tool, as shown in the figure below.

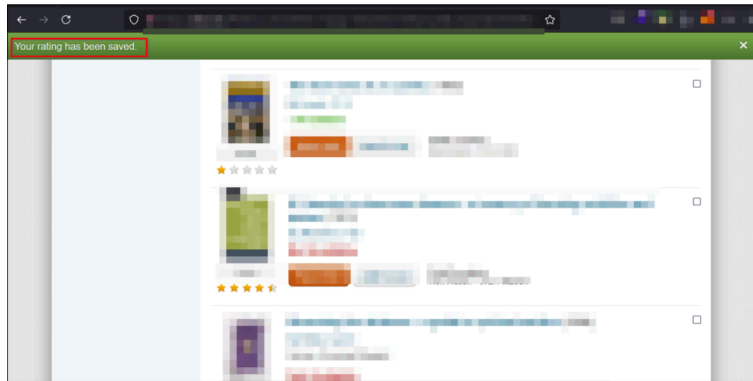


Figure 19: Burp screenshot showing that an unauthenticated user has submitted the ratings In the Burp suite, capture the 'xyz/reviews' request and send it to Burp Intruder. Configure an attack with 50 null payloads, then click on the “Start Attack.”

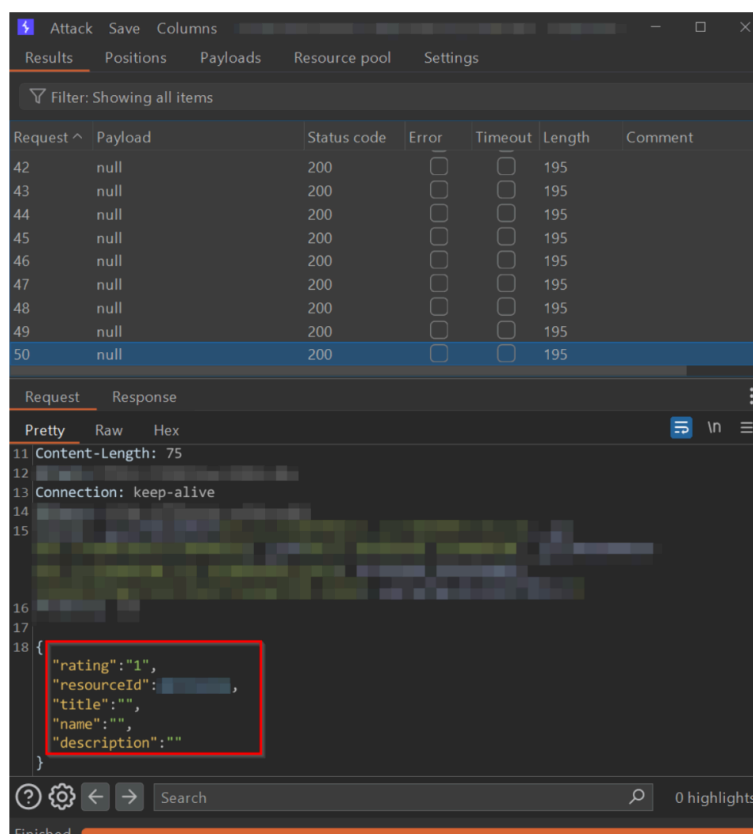


Figure 20: Burp screenshot showing intruder attack on 'reviews' request Observe that the attack is successful, stating that an unauthenticated user can submit ratings (Good or Poor) for XYZ using an intruder tool.

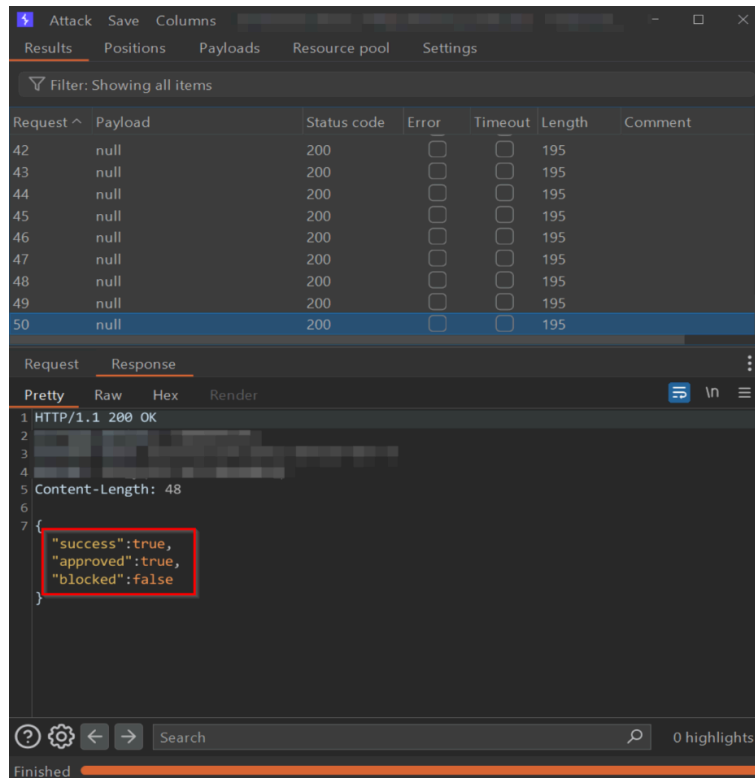


Figure 21: Burp screenshot showing a successful intruder attack.

By exploiting this vulnerability, malicious individuals can manipulate the reputation of XYZ products by posting biased or false ratings. This can deceive other users, impacting their decisions and potentially leading to reduced interest in certain items. Moreover, attackers can strategically damage the reputation of competitors' offerings by posting negative ratings, thereby gaining unfair advantages. Ultimately, this can harm the platform's reputation, decrease user engagement, and potentially affect revenue streams tied to product performance.

Steps for Recreation

1. Access the application without authentication and navigate to the **XYZ reviews/ratings** feature.
2. Capture the xyz/reviews request using Burp Suite and Send the request to **Burp Intruder**.
3. Configure payloads (e.g., 50 null or repeated values) and Start the Intruder attack.
4. Observe that ratings are successfully submitted without authentication.
5. Confirm that unauthenticated users can post **valid/invalid ratings repeatedly**.
6. Verify impact on product review integrity.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Implement Authentication: Require users to authenticate (e.g., create accounts, use OAuth) before allowing them to submit ratings.
- Enforce Authorization: Validate user permissions to ensure only authenticated users can submit ratings.
- Apply Rate Limiting: Restrict the number of ratings users can submit within a defined timeframe to prevent abuse.
- Enhance Input Validation: Validate and sanitize all rating submissions to prevent injection attacks and ensure data integrity.
- Enhance Input Validation: Validate and sanitize all rating submissions to prevent injection attacks and ensure data integrity.

Resources

More about Allowing Unauthenticated Users to Submit Ratings

<https://clouddestinations.com/blog/real-threat-of-unauthorized-access.html>

DIRECTORY LISTING ENABLED

MEDIUM

Identifier	T9H8YPG-F0008
Severity	Medium
Instances	1

Description

Directory listing is a web server feature that displays the directory contents when there is no index file in a website directory. It exposes sensitive files in the directory that aren't supposed to be accessed by users, like temporary files, crash dumps, or, in some cases, parts of the source code.

Instances (1)

- sample.com/wp-content/uploads/

Evidence

The application permits users to traverse server directories, evident from the depicted image, indicating the presence of enabled directory listing functionality.

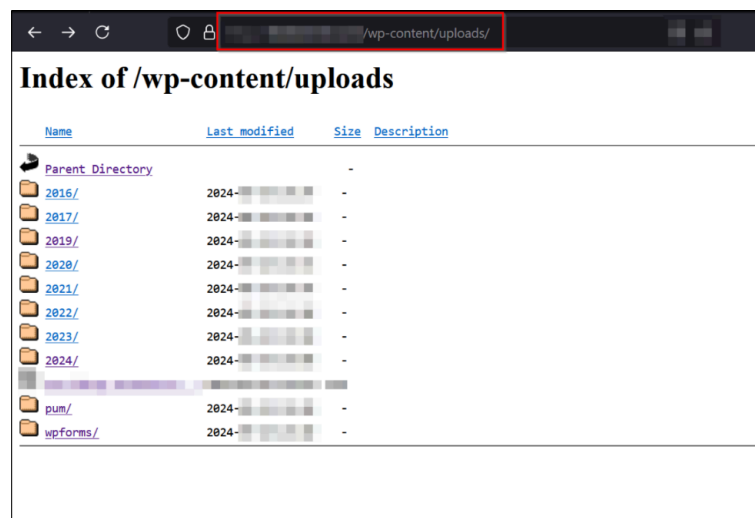


Figure 22: Browser screenshot showing that the directory listing functionality is enabled. Enabling directory listing exposes the server's directory structure, potentially revealing sensitive files or directories. Attackers can exploit this to gather information about the application's architecture and target specific files for unauthorized access or exploitation. This vulnerability can lead to data leakage, unauthorized access, and other security risks.

Steps for Recreation

1. Navigate to common directory paths (e.g., /uploads/, /files/, /images/).
2. Observe that a **directory listing page is displayed** instead of a web application response.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Prevent directory listings for all routes beneath the web root by configuring your web server.
- Put a default file (such as index.html) in each directory that the web server will display instead of returning a directory listing.

Resources

More about Directory listing:

https://portswigger.net/kb/issues/00600100_directory-listing

REFLECTED CROSS-SITE SCRIPTING(XSS)

MEDIUM

Identifier	T9H8YPG-F0009
Severity	Medium
Instances	1

Description

Reflected Cross-Site Scripting (XSS) is a prevalent web application security vulnerability that occurs when an attacker injects malicious scripts, typically JavaScript, into a web application. These malicious scripts are then executed by the victim's browser, often leading to unauthorized access to sensitive data, session hijacking, or defacement of web content. Reflected XSS attacks exploit the trust that a user has in a web application by manipulating input parameters and reflecting the injected script back to the user within the application's response, resulting in potentially severe consequences for both the user and the web application.

Instances (1)

- sample.com

Evidence

While parsing through the client's environment, the security team pointed to an uncovered “XYZ” directory where a search bar was discovered. The security team was able to confirm the presence of an XSS vulnerability immediately with a basic script by entering it into the search bar.

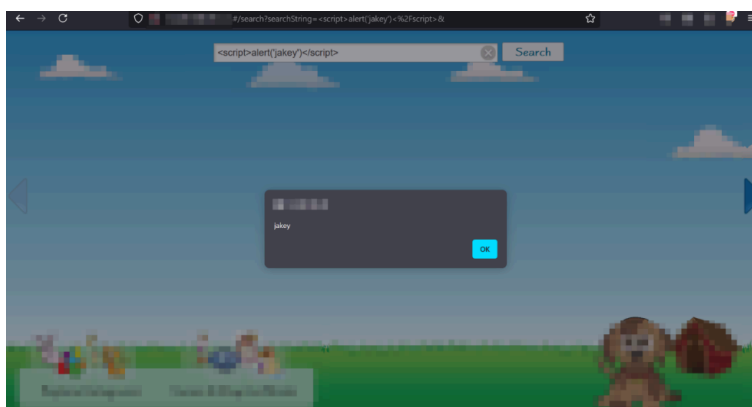


Figure 23: Screenshot showing the XSS payload executed successfully.

After it was confirmed that the site was vulnerable to XSS, the team headed back to the homepage of the in-scope URL and tried more sophisticated payloads that included event handlers. After using a more advanced payload on the home page's search bar, the security team was able to confirm that the home page was also vulnerable. However, the security team could not leverage this finding to create something higher.

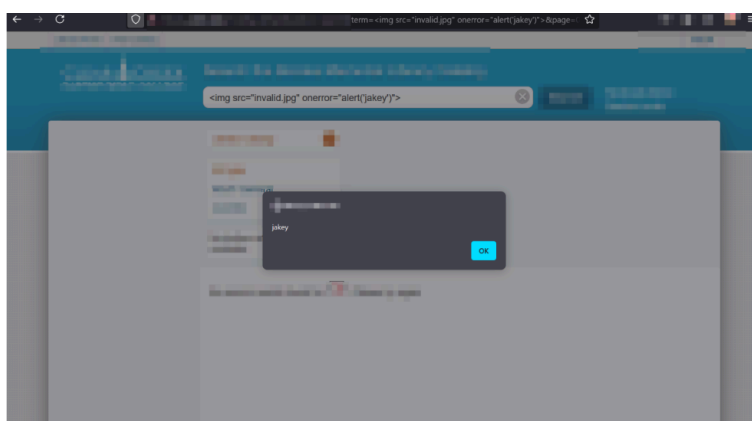


Figure 24: Screenshot showing the XSS payload executed successfully on the home page. By injecting malicious scripts into URLs or other user-input fields that are then echoed back by the web application, attackers can trick unsuspecting users into executing these scripts within their browsers. This allows attackers to steal session cookies, perform actions on behalf of the user, deface websites, redirect users to malicious sites, or even propagate worms across multiple users. Reflected XSS attacks are particularly advantageous as they can be used for targeted phishing,

spreading malware, or compromising sensitive data without direct server access, making them a potent tool in the arsenal of a malicious actor.

Steps for Recreation

1. Navigate to the **XYZ directory search bar**.
2. Enter a basic XSS payload (e.g., `<script>alert(1)</script>`) and Submit the search request.
3. Observe that the payload executes in the browser (XSS confirmed).
4. Navigate to the **home page search bar** of the application and Inject a more advanced XSS payload (e.g., event-handler based).
5. Submit the request and Observe that the payload is executed on the home page as well.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- Apply input validation and output encoding for user data.
- Use Content Security Policy (CSP) to restrict script sources.
- Utilize secure coding techniques like parameterized queries and prepared statements.

Resources

More about Reflected XSS

<https://portswigger.net/web-security/cross-site-scripting/reflected>

STACK TRACE

LOW

Identifier	T9H8YPG-F0010
Severity	Low
Instances	1

Description

A stack trace is a detailed report of the active stack frames at a specific point in time during the execution of a program. It typically lists the sequence of function calls leading to an error or exception, helping developers pinpoint the source of the issue within the code. Stack traces are crucial for debugging, providing insights into the call hierarchy and the state of the application when an error occurs.

Instances (1)

- sample.com/ava***y

Evidence

During the assessment, the team discovered a publicly accessible stack trace at `sample.com/ava***y`. This exposure revealed detailed information about the application's internal code structure and potential vulnerabilities, posing a significant security risk.

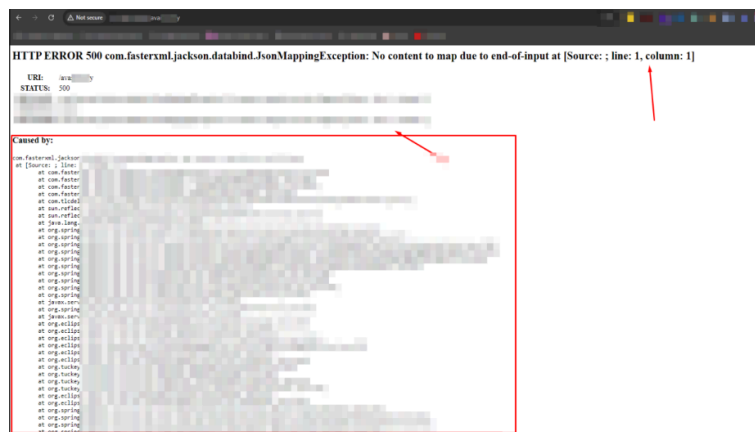


Figure 25: Screenshot showing the detailed stack trace.

The exposed stack trace provides attackers with detailed insights into the application's internal code and potential vulnerabilities, facilitating targeted attacks, exploiting weaknesses, and compromising sensitive data. Immediate measures are necessary to prevent information leakage and enhance application security.

Steps for Recreation

1. Open a browser and navigate to the URL `sample.com/ava***y` and observe that a **stack trace/debug error page** is displayed.

Recommendations

To mitigate the risk of this issue, the assessment team recommends the following steps:

- **Disable Stack Traces:** Configure the application to suppress stack traces in production environments to prevent information leakage.
- **Error Handling:** Implement comprehensive error handling to display user-friendly messages without revealing internal details.

Resources

More about Stack Trace

https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

APPENDIX (A) SEVERITY DESCRIPTION

The assessment team used the following criteria to rate the findings in this report. Red Sentry derived these risk ratings from the industry and organizations such as OWASP.

The severity of each finding in this report is independent. Finding severity ratings combine direct technical and business impact with the worst-case scenario in an attack chain. The more significant the impact, and the fewer vulnerabilities that must be exploited to achieve that impact, the higher the severity.

Critical

A purple rounded rectangular button with the word "CRITICAL" in white, uppercase letters.

Vulnerability is an otherwise high-severity issue with additional security implications that could lead to exceptional business impact. Examples: trivial exploit difficulty, business-critical data compromised, bypass of security controls, direct violation of communicated security objectives, and large-scale vulnerability exposure.

High

A red rounded rectangular button with the word "HIGH" in white, uppercase letters.

Vulnerability may result in direct exposure including, but not limited to: the loss of application control, execution of malicious code, or compromise of underlying host systems. The issue may also create a breach in the confidentiality or integrity of sensitive business data, customer information, and administrative and user accounts. In some instances, this exposure may extend farther in the infrastructure beyond the data and systems associated with the application.

Medium

MEDIUM

Vulnerability does not lead directly to the exposure of critical application functionality, sensitive business and customer data, or application credentials. However, it can be executed multiple times or leveraged in conjunction with another issue to cause direct exposure. Examples include brute-forcing and client-side input validation.

Low

LOW

Vulnerability may result in limited exposure of application control, sensitive business and customer data, or system information. This type of issue provides value only when combined with one or more issues of a higher risk classification. Examples include overly detailed error messages, the disclosure of system versioning information, and minor reliability issues.

Informational

INFO

Finding does not have a direct security impact but represents an opportunity for additional layers of security, is considered a best practice, or has the possibility of turning into an issue over time. Finding is a security-relevant observation that has no direct business impact or exploitability, but may lead to exploitable vulnerabilities. Examples include poor communication between organizations, documentation encouraging poor security practices, or lack of security training.