Web Application Penetration Testing Report Of Juice Shop For OWASP

Table of Contents

3 Project Summary
5 Vulnerability Details

Project Summary

EXECUTIVE SUMMARY

AnoF Demo conducted a comprehensive security assessment of OWASP in order to determine existing vulnerabilities and establish the current level of security risk associated with the environment and the technologies in use. This assessment harnessed penetration testing and social engineering techniques to provide OWASP management with an understanding of the risks and security posture of their corporate environment.

Project Details

This engagement has been conducted to assess the security posture of the high-value targets mentioned by our client OWASP. We have gone through the Juice Shop Web Application Penetration Testing as per OWASP Top 10 standards.

Scope

Scope	Scope Type	Start Date	End Date
http://10.10.14.68/	Web Application Penetration Testing	Oct. 26, 2022	Oct. 31, 2022

Description

The project is about Juice shop application security assessment. The project involves finding security vulnerabilities in the application

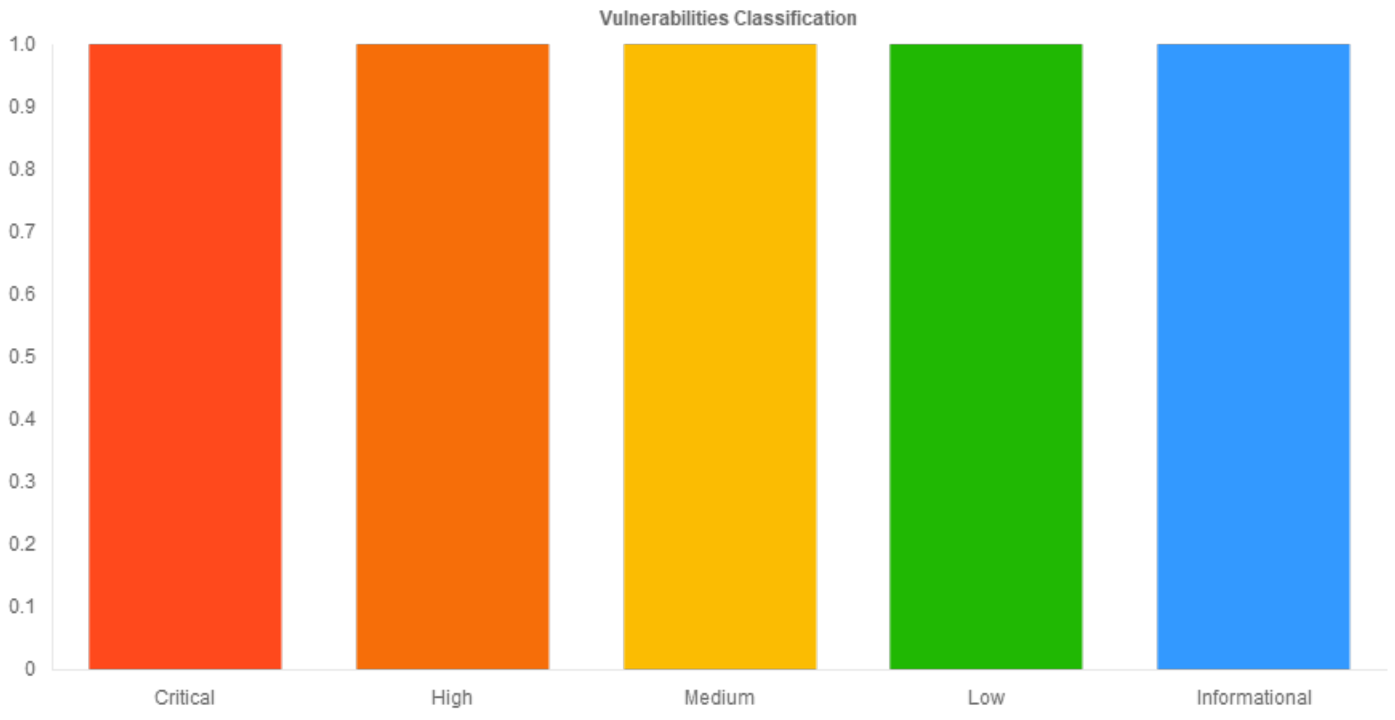
Project Involvement

Name	Email Address	Phone	Company
Test	test@owasp,org	0000000000	OWASP
sourav	admin@admin.com	1111111111	AnoF Demo

Vulnerability Details

Vulnerabilities Classification

Sr	Vulnerability Name	Severity	Status
1	SQL Injection	Critical	Vulnerable
2	Sensitive Data Exposure	High	Vulnerable
3	DOM Based Cross Site Scripting	Medium	Vulnerable
4	Application Error Message	Low	Vulnerable
5	Cross Origin Resource Sharing (CORS)	Informational	Confirm Fixed



SQL Injection

Vulnerable

Critical

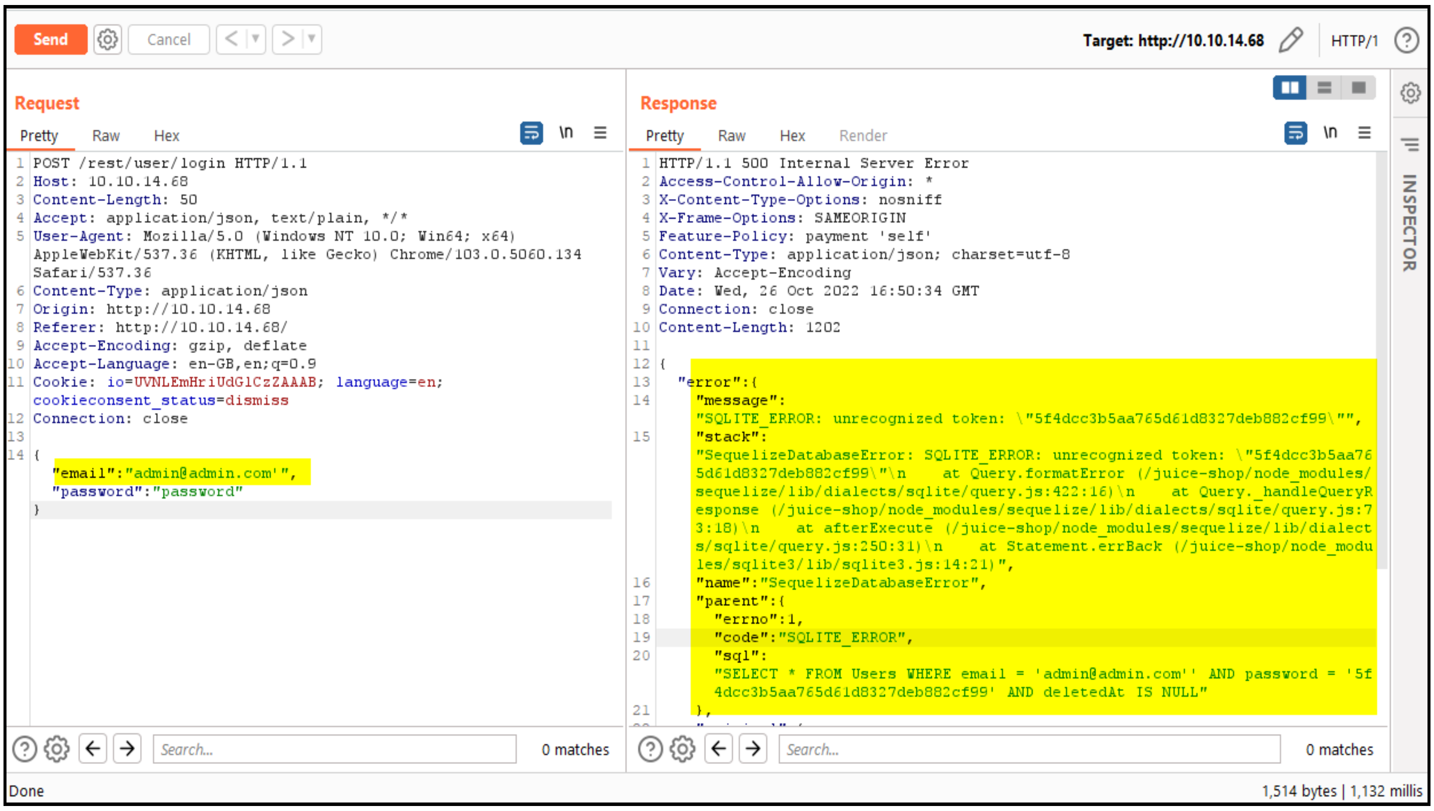
CVSS Score - 9.9
CVSS Vector - CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

Description

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application

Proof Of Concept

- The user intercept the login request and add single quote and get an SQL error.



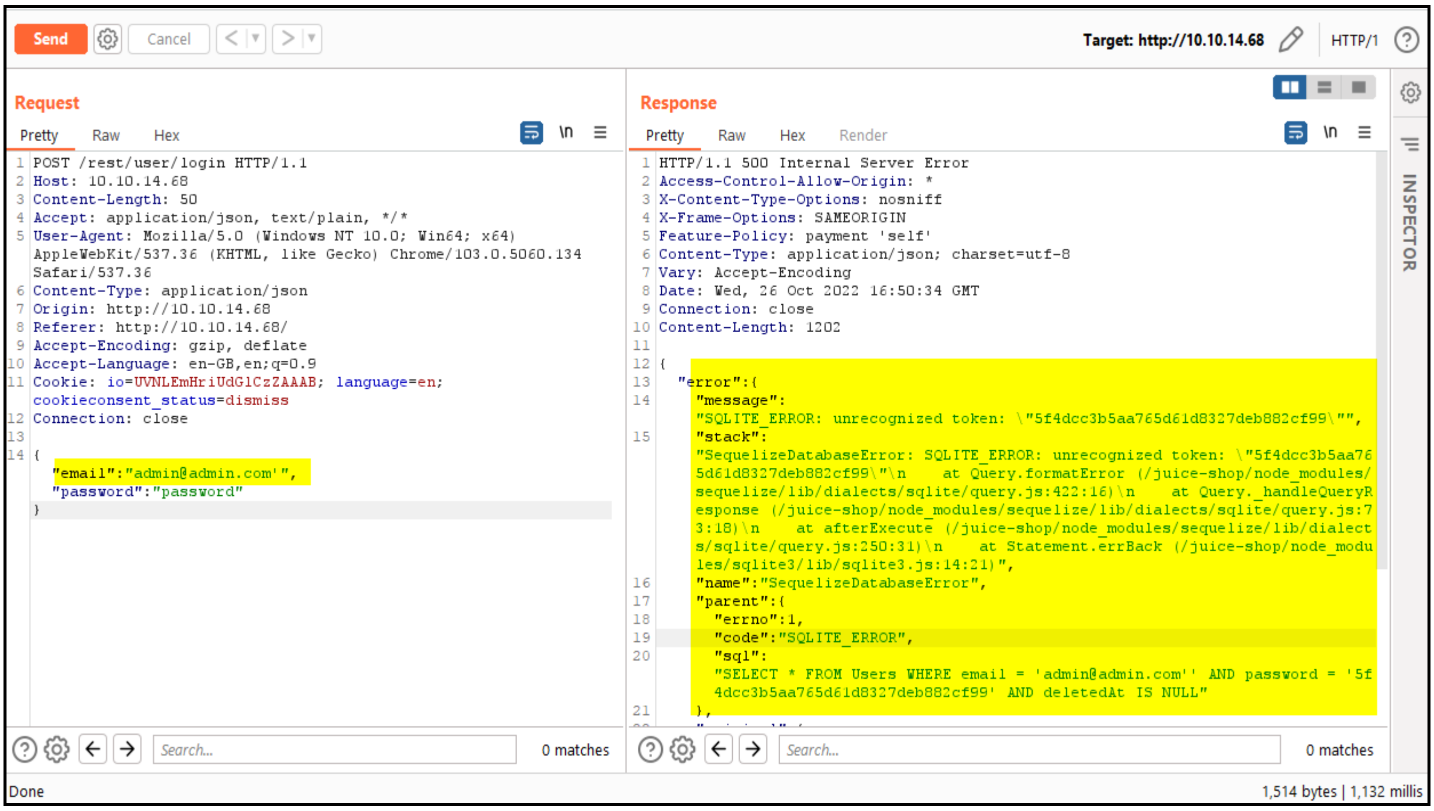
```
POST /rest/user/login HTTP/1.1
Host: 10.10.14.68
Content-Length: 50
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134
Safari/537.36
Content-Type: application/json
Origin: http://10.10.14.68
Referer: http://10.10.14.68/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en;q=0.9
Cookie: io=UVNLEmHriUdG1CzZAAAB; language=en;
cookieconsent_status=dismiss
Connection: close

{
  "email":"admin@admin.com'",
  "password":"password"
}
```

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Wed, 26 Oct 2022 16:50:34 GMT
Connection: close
Content-Length: 1202

{
  "error":{
    "message":
    "SQLITE_ERROR: unrecognized token: \"5f4dcc3b5aa765d61d8327deb882cf99\"",
    "stack":
    "SequelizeDatabaseError: SQLITE_ERROR: unrecognized token: \"5f4dcc3b5aa76
    5d61d8327deb882cf99\"\n    at Query.formatError (/juice-shop/node_modules/
    sequelize/lib/dialects/sqlite/query.js:422:16)\n    at Query._handleQueryR
    esponse (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:7
    3:18)\n    at afterExecute (/juice-shop/node_modules/sequelize/lib/dialect
    s/sqlite/query.js:250:31)\n    at Statement.errBack (/juice-shop/node_modu
    les/sqlite3/lib/sqlite3.js:14:21)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'admin@admin.com'' AND password = '5f
      4dcc3b5aa765d61d8327deb882cf99' AND deletedAt IS NULL"
    },
```

- The user craft the sql query and able to able the authentication



```
POST /rest/user/login HTTP/1.1
Host: 10.10.14.68
Content-Length: 50
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134
Safari/537.36
Content-Type: application/json
Origin: http://10.10.14.68
Referer: http://10.10.14.68/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en;q=0.9
Cookie: io=UVNLEmHriUdGlCzZAAAB; language=en;
cookieconsent_status=dismiss
Connection: close

{
  "email":"admin@admin.com'",
  "password":"password"
}
```

```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Wed, 26 Oct 2022 16:50:34 GMT
Connection: close
Content-Length: 1202

{
  "error":{
    "message":
    "SQLITE_ERROR: unrecognized token: \"5f4dcc3b5aa765d61d8327deb882cf99\"",
    "stack":
    "SequelizeDatabaseError: SQLITE_ERROR: unrecognized token: \"5f4dcc3b5aa76
    5d61d8327deb882cf99\"\n    at Query.formatError (/juice-shop/node_modules/
    sequelize/lib/dialects/sqlite/query.js:422:16)\n    at Query._handleQueryR
    esponse (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:7
    3:18)\n    at afterExecute (/juice-shop/node_modules/sequelize/lib/dialect
    s/sqlite/query.js:250:31)\n    at Statement.errBack (/juice-shop/node_modu
    les/sqlite3/lib/sqlite3.js:14:21)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'admin@admin.com'' AND password = '5f
      4dcc3b5aa765d61d8327deb882cf99' AND deletedAt IS NULL"
    },
```

Solution

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

Reference Link

- OWASP
- PortSwigger

Vulnerable Instances

URL	Paramter
http://10.10.14.68/rest/user/login	email

Sensitive Data Exposure

Vulnerable

High

CVSS Score - 8.2
CVSS Vector - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

Description

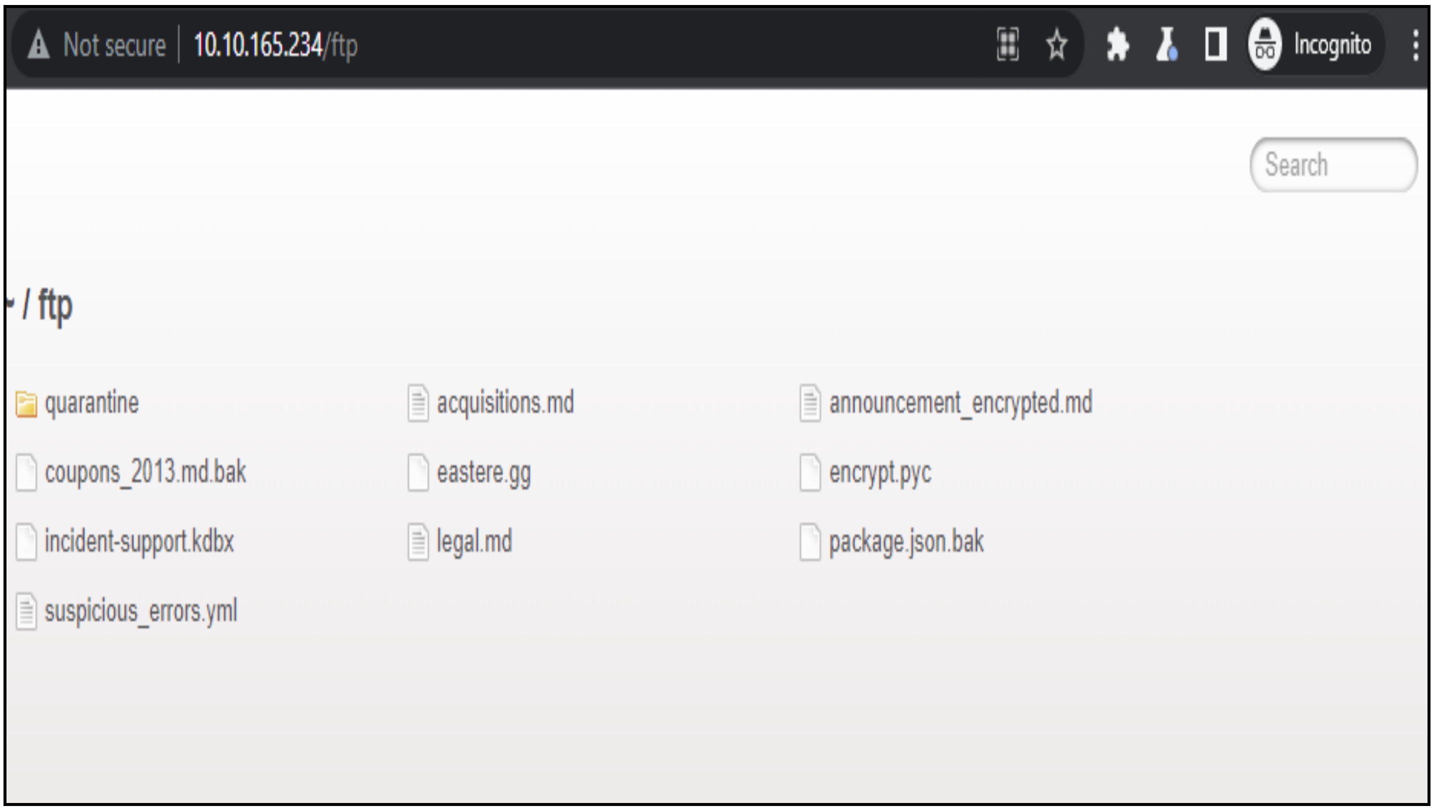
Information disclosure, also known as information leakage, is when a website unintentionally reveals sensitive information to its users. Depending on the context, websites may leak all kinds of information to a potential attacker, including:

- Data about other users, such as usernames or financial information
- Sensitive commercial or business data
- Technical details about the website and its infrastructure

The dangers of leaking sensitive user or business data are fairly obvious, but disclosing technical information can sometimes be just as serious. Although some of this information will be of limited use, it can potentially be a starting point for exposing an additional attack surface, which may contain other interesting vulnerabilities. The knowledge that you are able to gather could even provide the missing piece of the puzzle when trying to construct complex, high-severity attacks.

Proof Of Concept

- As can be seen, the application disclose the internal data.



~ / ftp

```
quarantine              acquisitions.md      announcement_encrypted.md
coupons_2013.md.bak     eastere.gg           encrypt.pyc
incident-support.kdbx   legal.md             package.json.bak
suspicious_errors.yml
```

Solution

Preventing information disclosure completely is tricky due to the huge variety of ways in which it can occur. However, there are some general best practices that you can follow to minimize the risk of these kinds of vulnerability creeping into your own websites.

- Make sure that everyone involved in producing the website is fully aware of what information is considered sensitive. Sometimes seemingly harmless information can be much more useful to an attacker than people realize. Highlighting these dangers can help make sure that sensitive information is handled more securely in general by your organization.
- Audit any code for potential information disclosure as part of your QA or build processes. It should be relatively easy to automate some of the associated tasks, such as stripping developer comments.
- Use generic error messages as much as possible. Don

Reference Link

- PortSwigger

Vulnerable Instances

URL	Paramter
http://10.10.165.234/ftp	

DOM Based Cross Site Scripting

Vulnerable

Medium

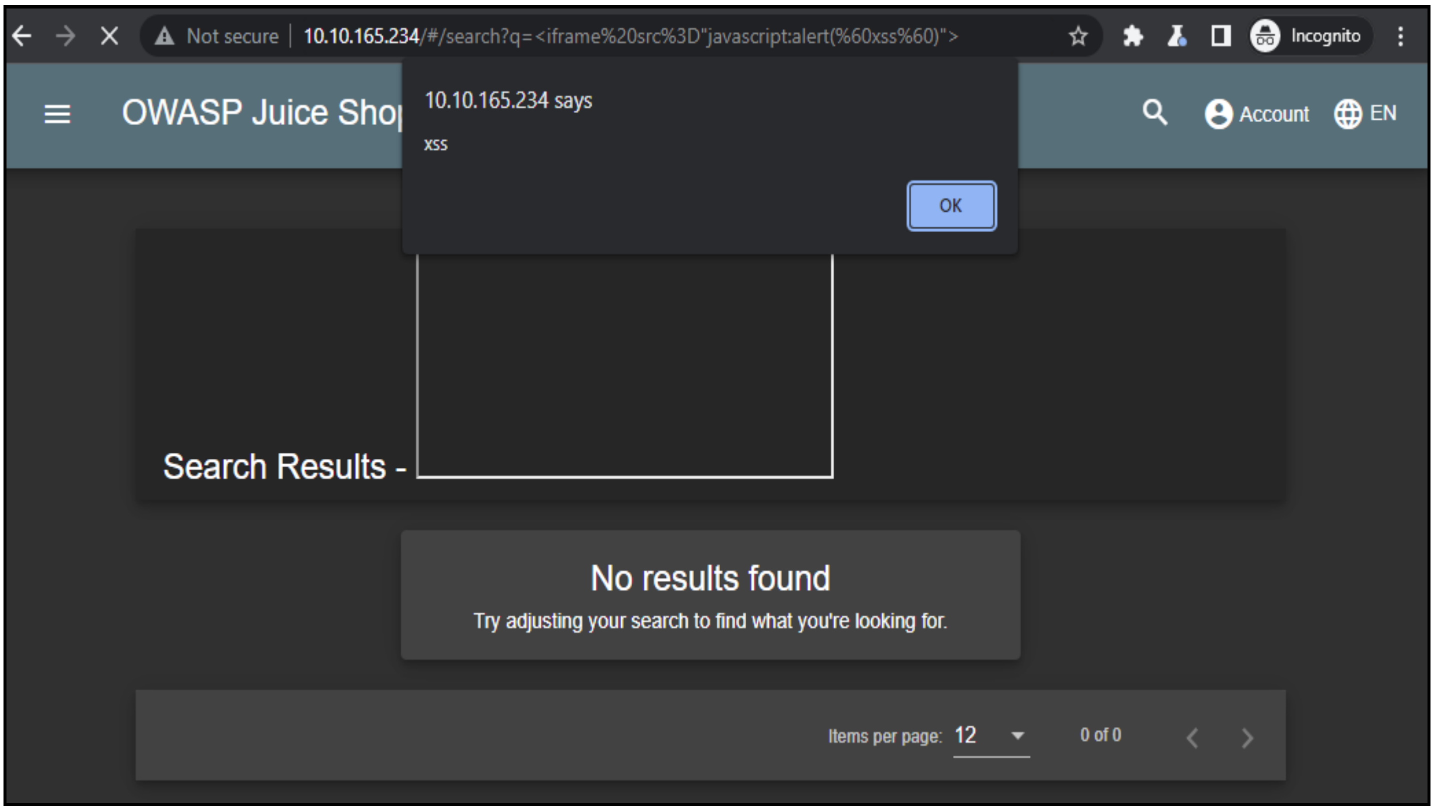
CVSS Score - 4.3
CVSS Vector - CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N

Description

DOM Based XSS (or as it is called in some texts,

Proof Of Concept

- As can be seen, the user is able to inject and execute the javascript code into the page.



Not secure | 10.10.165.234/#/search?q=<iframe%20src%3D"javascript:alert(%60xss%60)">

10.10.165.234 says

xss

Search Results -

No results found

Try adjusting your search to find what you're looking for.

Solution

The primary rule that you must follow to prevent DOM XSS is: sanitize all untrusted data, even if it is only used in client-side scripts. If you have to use user input on your page, always use it in the text context, never as HTML tags or any other potential code.

Avoid methods such as document.innerHTML and instead use safer functions, for example, document.innerText and document.textContent. If you can, entirely avoid using user input, especially if it affects DOM elements such as the document.url, the document.location, or the document.referrer.

Reference Link

- Acunetix

Vulnerable Instances

URL	Paramter
http://10.10.165.234/#/search	q

Application Error Message

Vulnerable

Low

CVSS Score - 3.7

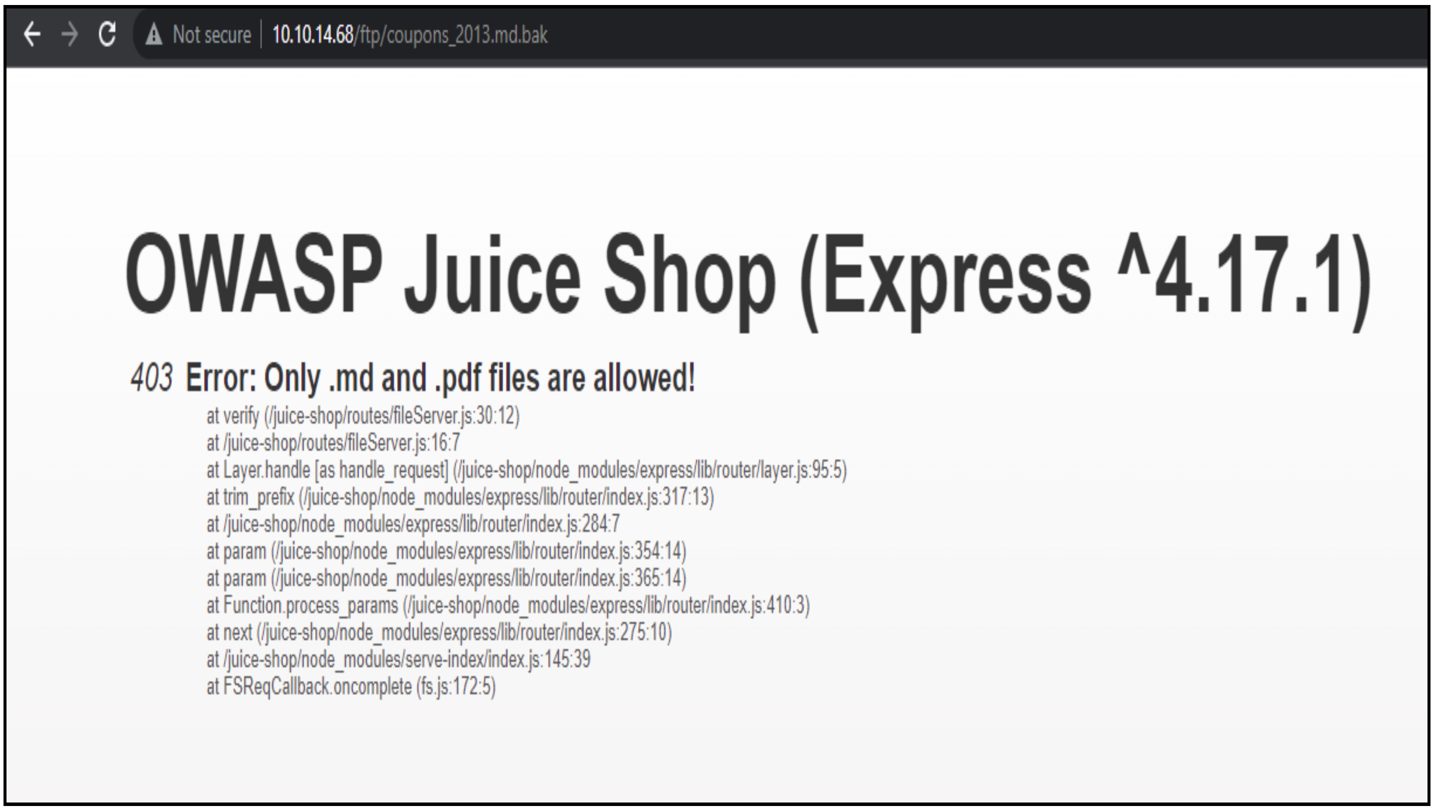
CVSS Vector - CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Description

Improper error handling leads to a variety of security problems. Common problems include when we expose our internal methods in Stack-traces, error codes, exceptions etc. and these are displayed to the hacker. An attacker is one who is waiting for such a thing to show up in order to

Proof Of Concept

- The user confirm that application disclose the error message.



Not secure | 10.10.14.68/ftp/coupons_2013.md.bak

```
OWASP Juice Shop (Express ^4.17.1)

403 Error: Only .md and .pdf files are allowed!
    at verify (/juice-shop/routes/fileServer.js:30:12)
    at /juice-shop/routes/fileServer.js:16:7
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:317:13)
    at /juice-shop/node_modules/express/lib/router/index.js:284:7
    at param (/juice-shop/node_modules/express/lib/router/index.js:354:14)
    at param (/juice-shop/node_modules/express/lib/router/index.js:365:14)
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:410:3)
    at next (/juice-shop/node_modules/express/lib/router/index.js:275:10)
    at /juice-shop/node_modules/serve-index/index.js:145:39
    at FSReqCallback.oncomplete (fs.js:172:5)
```

Solution

In the implementation, ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts. Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts. Note that the vast majority of web application attacks are never detected because so few sites have the capability to detect them. Therefore, the prevalence of web application security attacks is likely to be seriously underestimated.

Reference Link

- OWASP

Vulnerable Instances

URL	Paramter
http://10.10.14.68/ftp/coupons_2013.md.bak	

Cross Origin Resource Sharing (CORS)

Confirm Fixed

Informational

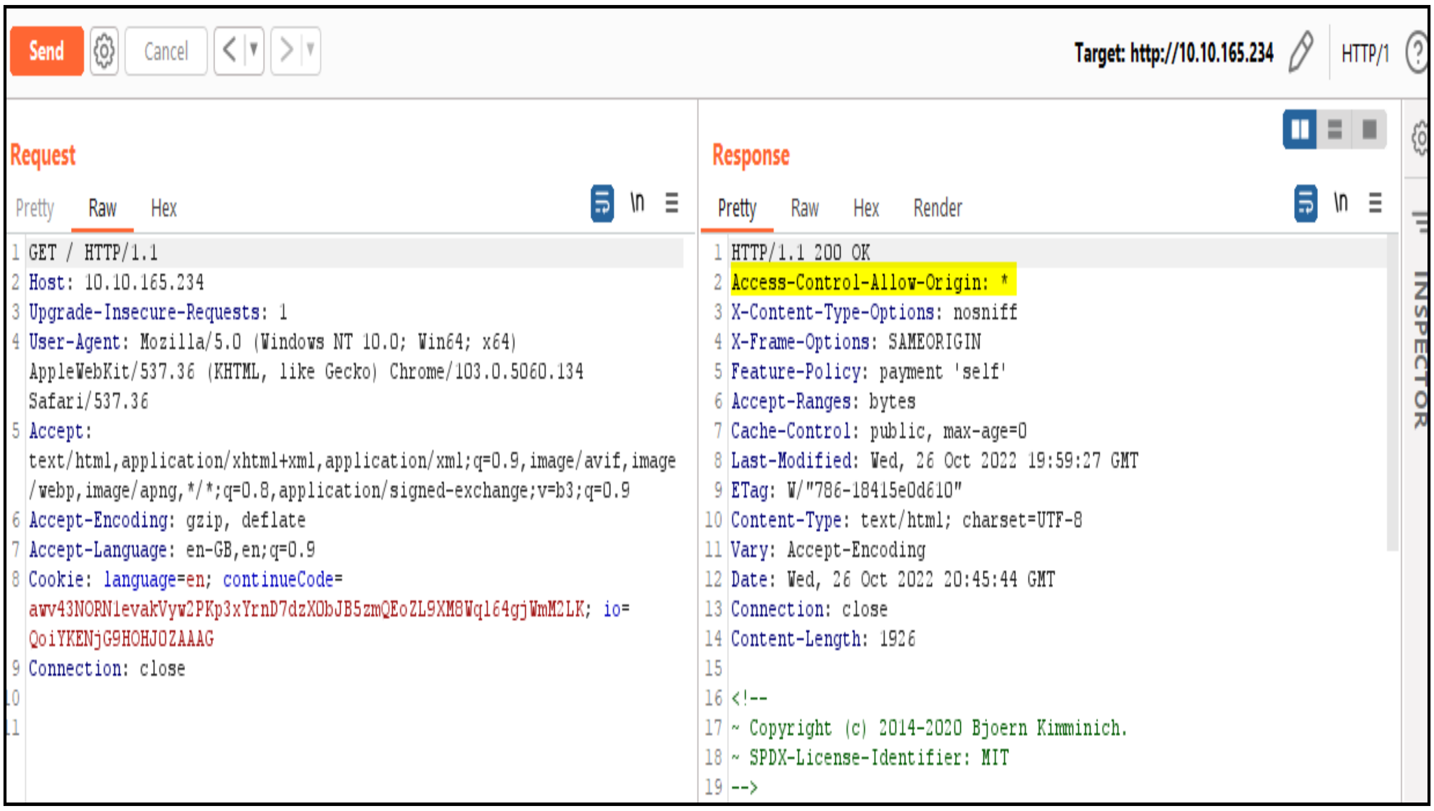
CVSS Score - 0.0
CVSS Vector - CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N

Description

Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy (SOP). However, it also provides potential for cross-domain attacks, if a website

Proof Of Concept

- The application use wild card for CORS



```
GET / HTTP/1.1
Host: 10.10.165.234
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.134
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en;q=0.9
Cookie: language=en; continueCode=
awv43NORN1evakVyw2PKp3xYrnD7dzXObJB5zmQEoZL9XM8Wq164gjWmM2LK; io=
QoiYKENjG9HOHJOZAAAG
Connection: close
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Wed, 26 Oct 2022 19:59:27 GMT
ETag: W/"786-18415e0d610"
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
Date: Wed, 26 Oct 2022 20:45:44 GMT
Connection: close
Content-Length: 1926

<!--
~ Copyright (c) 2014-2020 Bjoern Kimminich.
~ SPDX-License-Identifier: MIT
-->
```

Solution

- If a web resource contains sensitive information, the origin should be properly specified in the Access-Control-Allow-Origin header.
- It may seem obvious but origins specified in the Access-Control-Allow-Origin header should only be sites that are trusted. In particular, dynamically reflecting origins from cross-origin requests without validation is readily exploitable and should be avoided.
- Avoid using the header Access-Control-Allow-Origin: null. Cross-origin resource calls from internal documents and sandboxed requests can specify the null origin. CORS headers should be properly defined in respect of trusted origins for private and public servers.

Reference Link

- OWASP

Vulnerable Instances

URL	Paramter
http://10.10.165.234/	