RedSentry

Modern Pentests to Fight Modern Hackers.

Sample Company

Web Application Penetration Test

COMPLETE REPORT

Jan 10, 2023

RedSentry

Contact Information
+1 (888) 337- 0467
andres@redsentry.com
jenny@redsentry.com

TABLE OF CONTENTS

Summary **4**

Project Overview 4

Goals 4

Dates 4

Findings 4

Scope 5

Executive Summary 5

Web Application Penetration Test Findings **6**

Insecure Direct Object Reference (IDOR) 6

GraphQL Introspection enabled 10

User Enumeration 13

Stack Trace 16

Appendix (A) Severity Description **18**

Severity Descriptions 18

Summary

Project Overview

Sample Company engaged Red Sentry to assess the security of their web application. The following report details the findings identified during the course of the engagement, which started on January 1st, 2022.

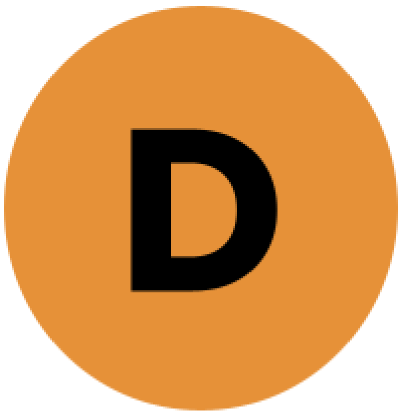
Goals

Identify vulnerabilities in the company's web application, both externally and internally, that could be leveraged by an attacker or malicious insider.

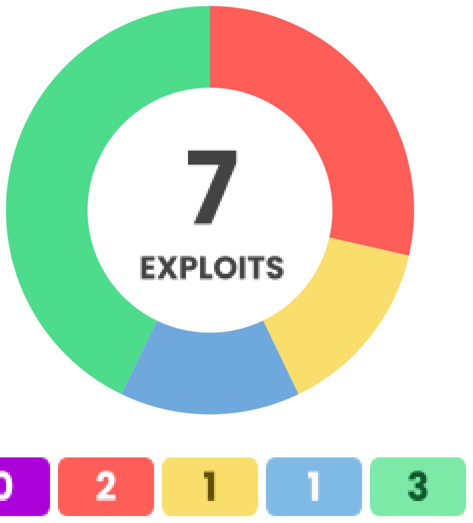
Dates

Jan 1, 2023	Jan 1-Jan 8, 2023	Jan 10, 2023
KICKOFF	**TESTING PERIOD**	**DELIVERY**

Findings



GRADE



Scope

- m.joinExample.com (staging environment)
- api.joinExample.com/graphql

Executive Summary

The assessment team conducted a penetration test on Sample Company's staging web application. Overall, the application presented 2 instances of high vulnerabilities among the total 7 discovered by the team. These vulnerabilities involved insecure object references, sensitive information exposure, and error messages improperly handled.

Ordered by type, two instances of a high vulnerability were found. An Insecure Direct Object Reference is an issue that allows unauthorized third parties to access information belonging to other entities like users, for example. In this way, User A could see the information related to User B. The application allowed this behavior on GraphQL queries that revealed data about users payments and login tokens.

A medium vulnerability was also identified. A GraphQL feature that allows listing all available queries was enabled. This finding can lead the way to discovering higher vulnerabilities like the one described above.

One low vulnerability, a user enumeration issue, was detected in the reset password query and, lastly, three instances of an informational issue were also found. Handling error messages in an improper manner can expose sensitive information about how the application works inside.

To improve the organization's security posture, the assessment team recommends implementing further input validations to avoid sensitive data exposure, restricting access to the application stack information by means of a properly defined role access schema and handling error messages in a proper manner.

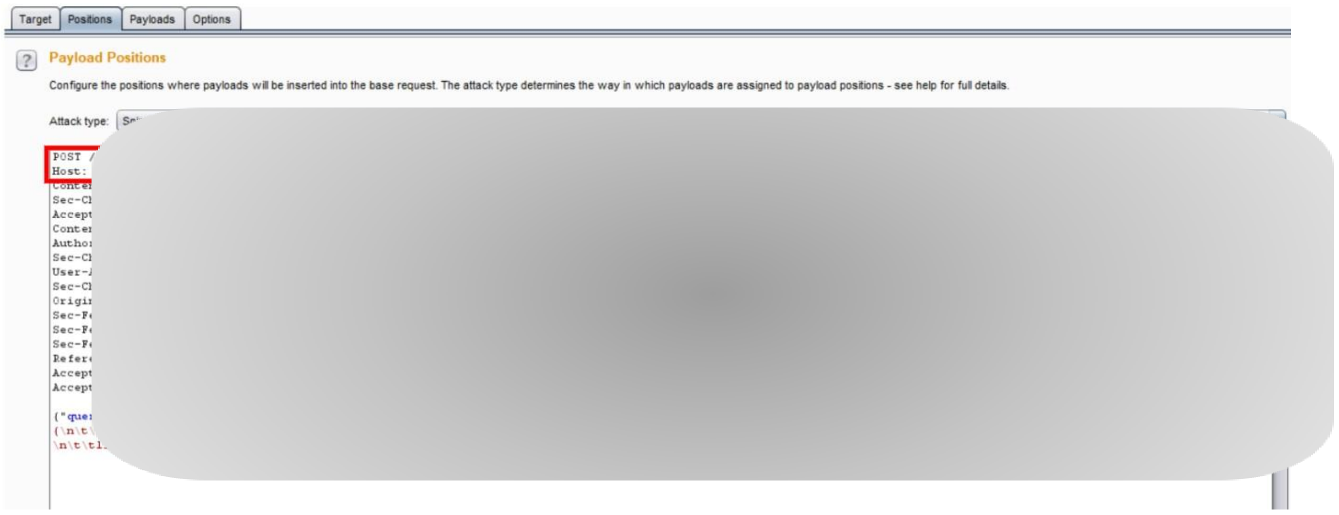
Insecure Direct Object Reference (IDOR)

HIGH

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.

Details

After extracting the list of allowed queries from the GraphQL API (see the GraphQL introspection enabled vulnerability for further information), the assessment team found two instances of queries using an insecure implementation of the "id" parameter to retrieve information about payments and user tokens.



Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

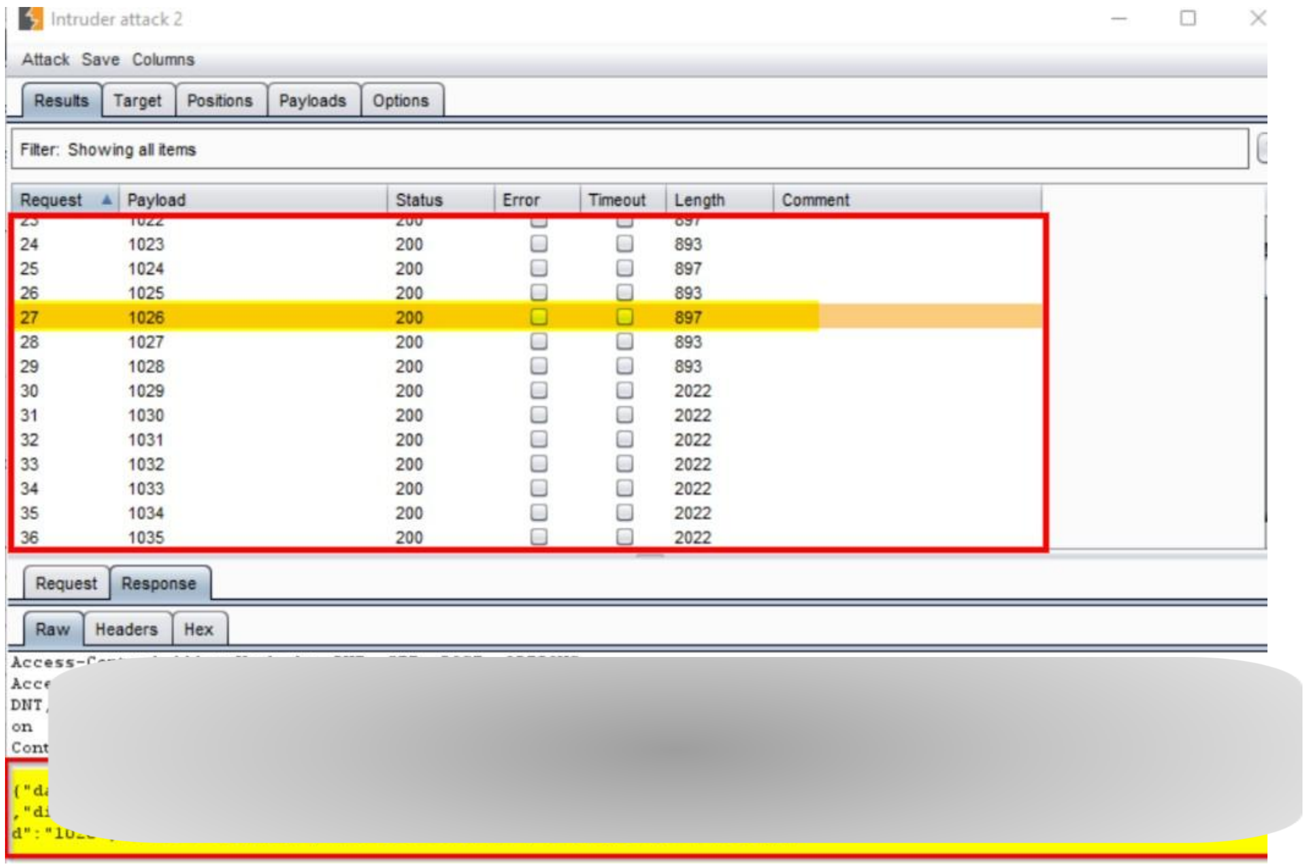
Attack type: [redacted]

```
POST /
Host:
Conte
Sec-Cl
Accept
Conte
Autho
Sec-Cl
User-/
Sec-Cl
Origi
Sec-F
Sec-F
Sec-F
Refer
Accept
Accept

{"que
{\n\t\
\n\t\tl
```

Figure 1: Payment data GraphQL query

From the image above, it can be seen that the query takes the “id” parameter as the only POST parameter and retrieves information about the credits, discounts, refunds and balances of the user matching that “id” parameter and since this parameter corresponds to a simple number which could either be increased or decreased, after fuzzing it the assessment team discovered that data from other users could be successfully extracted.

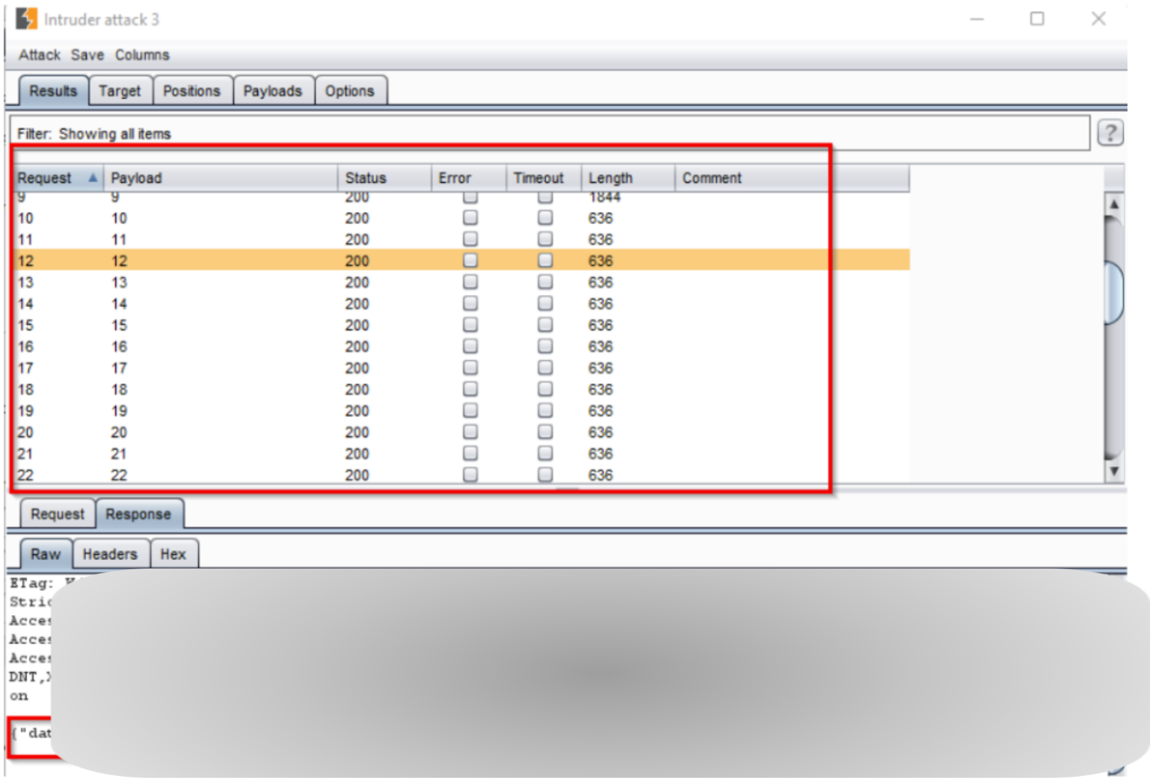


Request	Payload	Status	Error	Timeout	Length	Comment
23	1022	200	☐	☐	897	
24	1023	200	☐	☐	893	
25	1024	200	☐	☐	897	
26	1025	200	☐	☐	893	
27	1026	200	☐	☐	897	
28	1027	200	☐	☐	893	
29	1028	200	☐	☐	893	
30	1029	200	☐	☐	2022	
31	1030	200	☐	☐	2022	
32	1031	200	☐	☐	2022	
33	1032	200	☐	☐	2022	
34	1033	200	☐	☐	2022	
35	1034	200	☐	☐	2022	
36	1035	200	☐	☐	2022	

```
Access-Cont
Acce
DNT,
on
Cont
{"da
,"di
d":"10
```
[redacted]

Figure 2: Fuzzing id parameter from payment data GraphQL query

Relatedly, this same pattern applied to a query which retrieved information about users’ tokens. Tokens are alphanumeric combinations of characters which carry information that allows an application to validate if a login session corresponds to an authenticated user.



Request	Payload	Status	Error	Timeout	Length	Comment
9	9	200	☐	☐	1844	
10	10	200	☐	☐	636	
11	11	200	☐	☐	636	
12	12	200	☐	☐	636	
13	13	200	☐	☐	636	
14	14	200	☐	☐	636	
15	15	200	☐	☐	636	
16	16	200	☐	☐	636	
17	17	200	☐	☐	636	
18	18	200	☐	☐	636	
19	19	200	☐	☐	636	
20	20	200	☐	☐	636	
21	21	200	☐	☐	636	
22	22	200	☐	☐	636	

```
ETag: W
Stric
Acces
Acces
Acces
DNT,X
on

{"dat
```
[redacted]

Figure 3: Fuzzing id parameter from login token query

Once extracted, these tokens could be leveraged to make valid requests on behalf of other users and extract further information from their sessions.

IDORs are a common vulnerability found in applications and they're usually caused by design flaws such as using simple patterns as query parameters and the lack of enough validations that allows users to retrieve data from other users.

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)

Locations & Occurrences (2)

- api.Example.me (payment data query)
- api.Example.me (login token query)

Resources

GraphQL Cheat Sheet

cheatsheetseries.owasp.org/cheatsheets/GraphQL_Cheat_Sheet.html

Hacking GraphQL for fun and profit (Part 1)

www.secjuice.com/hacking-graphql-for-fun-and-profit-part-1-understanding-graphql-basics/

Hacking GraphQL for fun and profit (Part 2)

www.secjuice.com/hacking-graphql-for-fun-and-profit-part-2-methodology-and-examples/

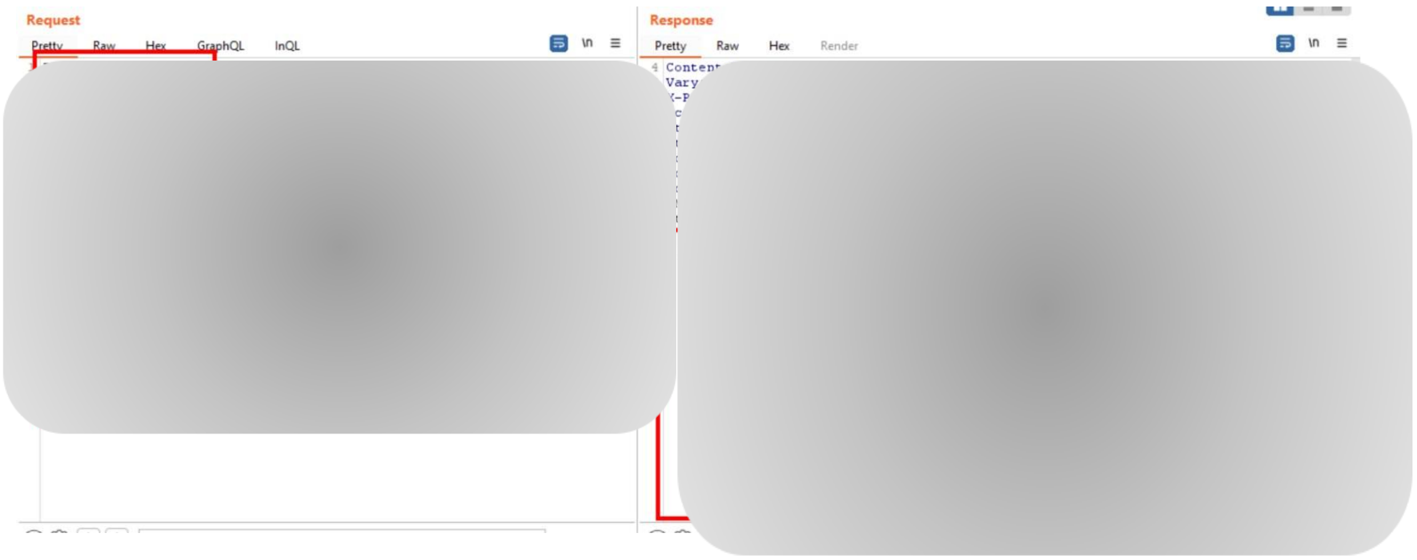
GraphQL Introspection enabled

MEDIUM

The GraphQL query language is strongly typed. Due to its strong type system, GraphQL gives you the ability to query and understand the underlying schema by using the Introspection feature. Introspection is the ability to query which resources are available in the current API schema, such as the queries, the types, the fields, and the directives it supports.

Details

The assessment team discovered that the api.Example.me GraphQL had the introspection feature enabled after making an Introspection query to retrieve information about the API schema.



[redacted]

[redacted]

Figure 4: GraphQL introspection query

This information serves as a map for a malicious user to know what queries are possible to make and what information is needed to make them work. This type of information increases the attack surface by revealing information that shouldn't be accessible to a basic user that could be leveraged to find critical vulnerabilities like SQL injections or IDORs.

Recommendations

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

- You can turn off introspection in production by setting the value of the introspection config key on your Apollo Server instance.
- Please note that it's possible for bad actors to learn how to write malicious queries by reverse engineering your GraphQL API through a lot of trial and error, disabling introspection is a form of security by obscurity. It's not the best form of security, but paired with other techniques like size, depth, amount limiting and operation whitelisting, it can make a substantial difference.

Locations & Occurrences (1)

- api.Example.me

Resources

Why you should disable GraphQL Introspection in production

www.apollographql.com/blog/graphql/security/why-you-should-disable-graphql-introspection-in-production/

HackerOne case report

hackerone.com/reports/1132803

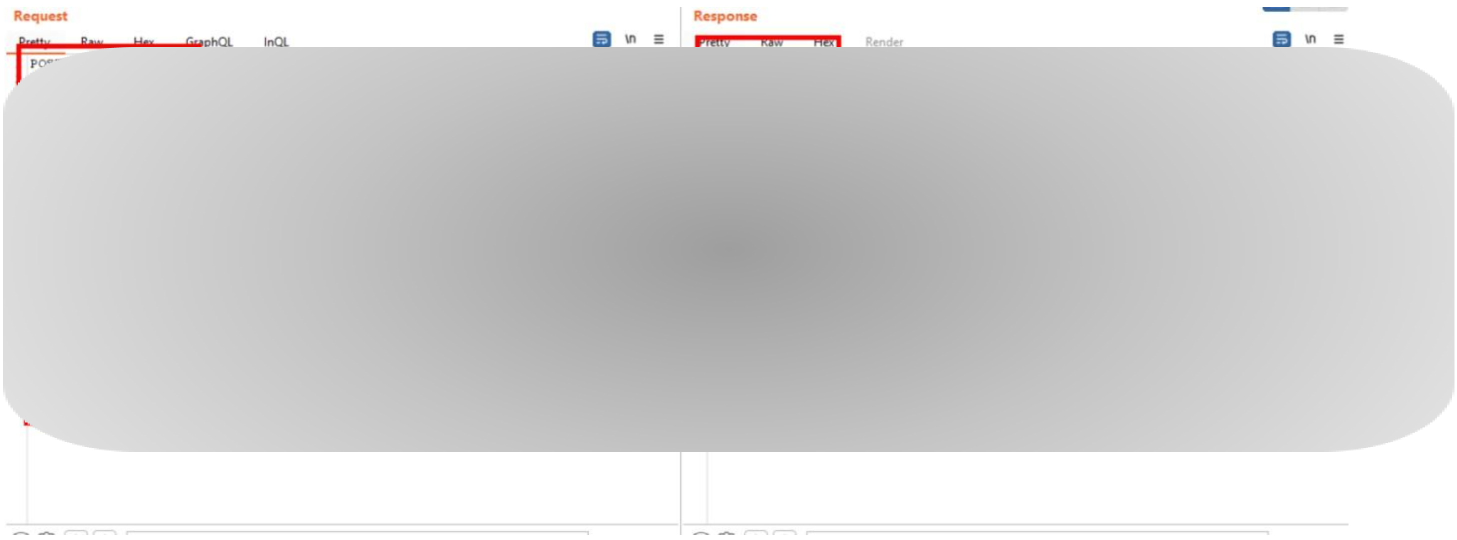
User Enumeration

LOW

User enumeration is when a malicious actor can use brute-force techniques to either guess or confirm valid users in a system. The web application reveals when a username exists on the system, either as a consequence of misconfiguration, or as a design decision.

Details

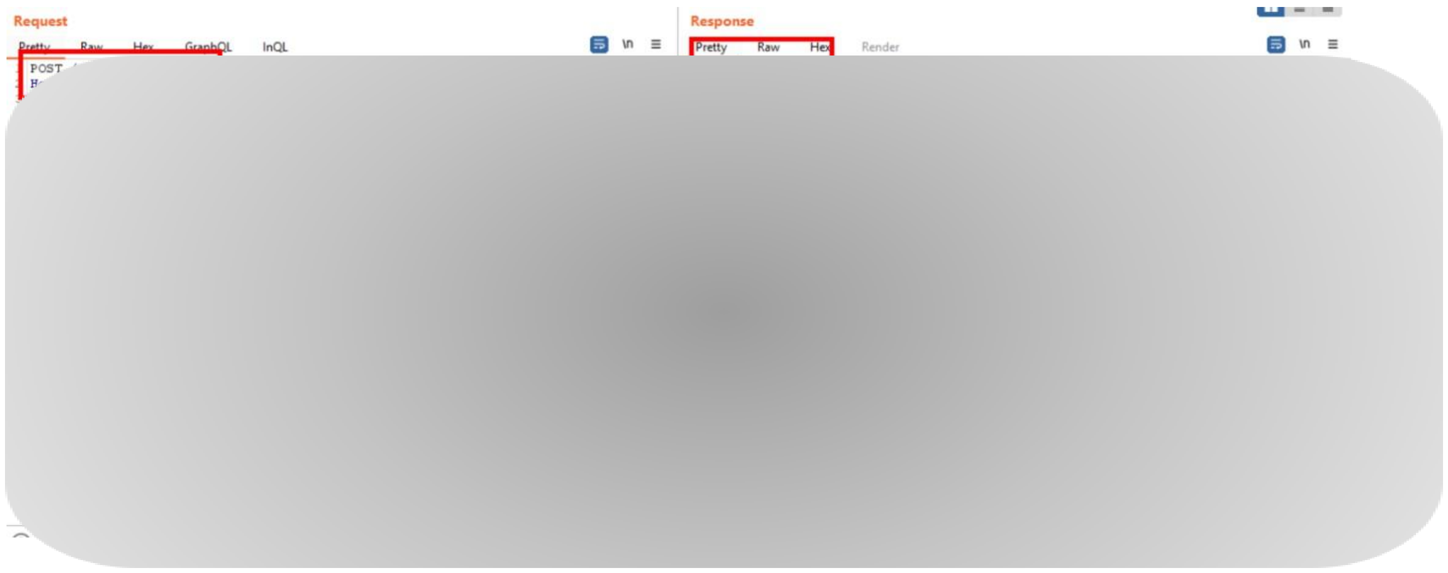
The assessment team discovered a reset password mutation located in the api.Example.me GraphQL API and identified a pattern that could allow a malicious user to enumerate valid users.



[redacted]

Figure 5: Reset password success case

Firstly, after using an email corresponding to a valid user, it displayed a success message and a 200 status response.



POST

[redacted]

Figure 6: Reset password failure case

Secondly, after making the same request but using a randomly made email, it displayed an error message clearly stating that the user couldn't be found with the given email.

These response patterns can be leveraged to extract information about the list of users and then attempt to extract the login password by means of brute forcing or phishing techniques.

Recommendations

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

- An effective remediation would be to have the server respond with a generic message that does not indicate which field is incorrect. If the response does not indicate whether the username or the password are incorrect, the attack surface decreases

Locations & Occurrences (1)

- api.Example.me/graphql (Reset password query)

Resources

User enumeration

https://www.rapid7.com/blog/post/2017/06/15/about-user-enumeration/

Testing for Account Enumeration and Guessable User Account

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account

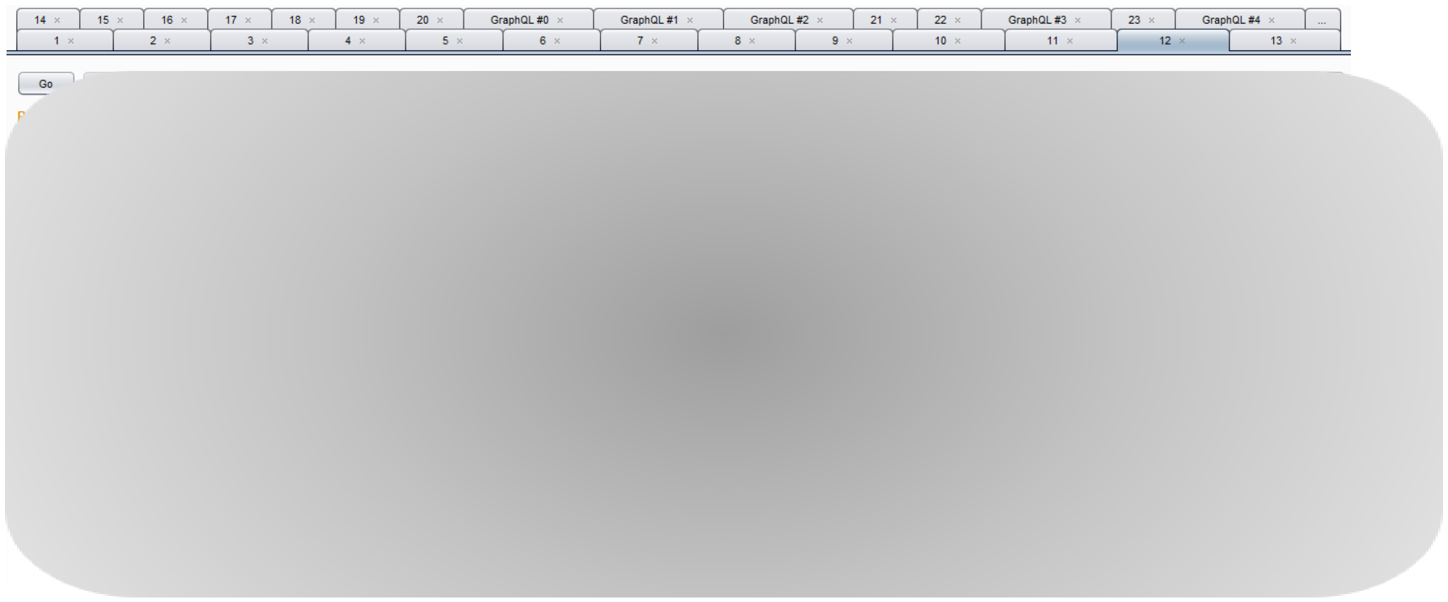
Stack Trace

INFO

Improperly-handled verbose error messages reveal debug Information which gives insights that can be exploited to retrieve sensitive data or to discover further vulnerabilities inside the underlying application.

Details

The assessment team found 3 instances of improperly-handled error after performing HTTP requests to the assets listed below.



[redacted]

Figure 7: Verbose error message

With this information, an attacker could gain insight on how the application works or its technology stack and leverage this to craft a payload to retrieve further information.

Recommendations

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

- Always do input validations
- Avoid displaying debugging messages

Locations & Occurrences (3)

- api.joinExample.com/graphql?query={__schema}
- api.Example.me/graphql (Server name disclosure)
- api.Example.me/graphql (Invalid characters inputted)

Resources

Improper Error Handling

owasp.org/www-community/Improper_Error_Handling

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.

Severity Descriptions

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

CRITICAL

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

HIGH

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.