SCENARIO

The application contains some user management functions that are powered by a GraphQL endpoint which forces the application to possess an access control vulnerability whereby we can induce the API to reveal user credential fields. We will try to exploit this vulnerability and get the access to administrator’s account.

**PROCEDURE**

1. Open the web application and study the request made to the server.
2. Navigate to the My Account page and try logging in with any fake credentials and study the request in BurpSuite’s Proxy tab.
3. Now copy that GraphQL URL and paste it into the bar in the InQL Scanner tab in BurpSuite.
4. Then, under query tab we can see that a query named **getUser.query** that returns a user's username and password.
5. Copy that query and send the fake login request with fake credentials to BurpSuite’s Repeater.
6. Then view the request into InQL mode and remove everything and insert the payload there in that Query field.
7. Then remove the **OperationName** field properly from the Pretty format.
8. Send the request and we see that the query’s response is the administrator’s username and password.

PAYLOAD

query {

getUser(id:1) {

password

id

username

}

}

REMEDIATION

1. Field-level Authorization: Implement authorization checks for every field, especially for sensitive data fields like passwords. The server-side logic should validate whether the requesting user has the permission to access the specified fields or not.
2. Hide Sensitive Fields: Remove sensitive fields from the GraphQL schema if they're not meant to be accessed by any client. If there is no business use-case for retrieving fields like passwords via the API, they shouldn't be exposed in the first place.
3. Scoped Tokens: Use scoped tokens to limit what actions a user can perform via the GraphQL API. For instance, a regular user's token should not have the ability to access administrative functions.
4. Rate Limiting: Rate limit requests to prevent brute force attacks or abuse.
5. Input Validation: Validate all inputs to protect against malicious queries. Ensure that there's a strong type system in place.