API3:2023 Broken Object Property Level Authorization

Threat agents/Attack vectors	Security Weakness	Impacts
API Specific : Exploitability **Easy**	Prevalence **Common** : Detectability **Easy**	Technical **Moderate** : Business Specific
APIs tend to expose endpoints that return all object's properties. This is particularly valid for REST APIs. For other protocols such as GraphQL, it may require crafted requests to specify which properties should be returned. Identifying these additional properties that can be manipulated requires more effort, but there are a few automated tools available to assist in this task.	Inspecting API responses is enough to identify sensitive information in returned objects' representations. Fuzzing is usually used to identify additional (hidden) properties. Whether they can be changed is a matter of crafting an API request and analyzing the response. Side-effect analysis may be required if the target property is not returned in the API response.	Unauthorized access to private/sensitive object properties may result in data disclosure, data loss, or data corruption. Under certain circumstances, unauthorized access to object properties can lead to privilege escalation or partial/full account takeover.

Is the API Vulnerable?

When allowing a user to access an object using an API endpoint, it is important to validate that the user has access to the specific object properties they are trying to access.

An API endpoint is vulnerable if:

- The API endpoint exposes properties of an object that are considered sensitive and should not be read by the user. (previously named: "Excessive Data Exposure")
- The API endpoint allows a user to change, add/or delete the value of a sensitive object's property which the user should not be able to access

(previously named: "Mass Assignment")

Example Attack Scenarios

Scenario #1

A dating app allows a user to report other users for inappropriate behavior. As part of this flow, the user clicks on a "report" button, and the following API call is triggered:

```
POST /graphql
{
  "operationName":"reportUser",
  "variables":{
    "userId": 313,
    "reason":["offensive behavior"]
  },
  "query":"mutation reportUser($userId: ID!, $reason: String!) {
    reportUser(userId: $userId, reason: $reason) {
      status
      message
      reportedUser {
        id
        fullName
        recentLocation
      }
    }
  }"
}
```

The API Endpoint is vulnerable since it allows the authenticated user to have access to sensitive (reported) user object properties, such as "fullName" and "recentLocation" that are not supposed to be accessed by other users.

Scenario #2

An online marketplace platform, that offers one type of users ("hosts") to rent out their apartment to another type of users ("guests"), requires the host to accept a booking made by a guest, before charging the guest for the stay.

As part of this flow, an API call is sent by the host to `POST /api/host/approve_booking` with the following legitimate payload:

```
{
  "approved": true,
  "comment": "Check-in is after 3pm"
}
```

The host replays the legitimate request, and adds the following malicious payload:

```
{
  "approved": true,
  "comment": "Check-in is after 3pm",
  "total_stay_price": "$1,000,000"
}
```

The API endpoint is vulnerable because there is no validation that the host should have access to the internal object property - `total_stay_price`, and the guest will be charged more than she was supposed to be.

Scenario #3

A social network that is based on short videos, enforces restrictive content filtering and censorship. Even if an uploaded video is blocked, the user can change the description of the video using the following API request:

```
PUT /api/video/update_video

{
  "description": "a funny video about cats"
}
```

A frustrated user can replay the legitimate request, and add the following malicious payload:

```
{
  "description": "a funny video about cats",
  "blocked": false
}
```

The API endpoint is vulnerable because there is no validation if the user should have access to the internal object property - `blocked`, and the user can change the value from `true` to `false` and unlock their own blocked content.

How To Prevent

- When exposing an object using an API endpoint, always make sure that the user should have access to the object's properties you expose.
- Avoid using generic methods such as `to_json()` and `to_string()`. Instead, cherry-pick specific object properties you specifically want to return.
- If possible, avoid using functions that automatically bind a client's input into code variables, internal objects, or object properties ("Mass Assignment").
- Allow changes only to the object's properties that should be updated by the client.
- Implement a schema-based response validation mechanism as an extra layer of security. As part of this mechanism, define and enforce data returned by all API methods.
- Keep returned data structures to the bare minimum, according to the business/functional requirements for the endpoint.