Updated HackerOne Report: Broken Access Control via Endpoint Manipulation

This is an updated version of the HackerOne report, taking into account the clarification that the vulnerability is related to manipulating the API endpoint rather than a request body parameter.

Title: Broken Access Control leading to Cross-Tenant Data Leakage via API Endpoint Manipulation

Summary:
An authenticated user can bypass access controls by manipulating the API endpoint URL, allowing them to access, modify, or delete sensitive data belonging to another tenant (Store B) using a valid session token from their own tenant (Store A). This vulnerability is a critical Broken Access Control issue, as the API fails to validate that the authenticated user's session token is authorized to interact with the resources at the requested endpoint.
Vulnerability Description:
The application's API endpoints are vulnerable to a cross-tenant access control bypass. A valid session token, obtained from authenticating to a specific tenant (Store A), can be used to perform actions on another tenant (Store B) by simply changing the API endpoint URL. When a request is sent to https://storeA.example.com/api/v1/products, the API correctly authorizes the user to view Store A's products. However, if the same request, with the same authentication token, is sent to https://storeB.example.com/api/v1/products, the server processes the request and returns the products belonging to Store B.

This indicates a critical failure in the authorization logic, where the API endpoint itself is not being checked against the user's authentication context to ensure they have permission to access that specific tenant's data.

Steps to Reproduce:

1. **Attacker User (Store A):** Log in to Store A and obtain a valid authentication token.
2. **Victim User (Store B):** Ensure there is existing data in Store B (e.g., a list of products).
3. **Exploitation:** Using a tool like cURL or a browser extension, construct a request to the victim's endpoint, but use the attacker's authentication token.
   Example Request:
   GET https://storeB.example.com/api/v1/products
   Authorization: Bearer [ATTACKER_STORE_A_TOKEN]

4. **Observe:** The server will respond with a 200 OK status code and return the

product list from Store B, even though the authentication token belongs to a user from Store A.

Impact:

- **Confidentiality:** An attacker can view sensitive information from other tenants, such as product details, customer data, orders, and financial information.
- **Integrity:** An attacker can perform destructive actions like modifying or deleting data on other tenants.
- **Availability:** An attacker could potentially delete critical data, leading to a denial of service for the victim's tenant.
- **Reputation Damage:** The vulnerability could lead to a massive data breach, severely damaging the company's reputation and leading to regulatory fines.

Remediation:
The server must implement robust authorization checks on every API request. The authorization logic should perform the following actions:

1. **Authentication:** Verify that the provided authentication token is valid.
2. **Authorization:** After successful authentication, cross-reference the user's identity (extracted from the token) with the requested resource to ensure they are authorized to access that specific tenant's data.
3. **Strict Enforcement:** The API should strictly enforce that a user with a store_id of A can only access endpoints associated with storeA.example.com. Any attempt to access a different endpoint, such as storeB.example.com, should be denied.