API SECURITY RISK ANALYSIS REPORT

MODERN SAAS API ASSESSMENT

Target API:
JSONPlaceholder
https://jsonplaceholder.typicode.com

PREPARED BY: SANJIVANI PANDEY

INTERNSHIP: FUTURE INTERNS

DOMAIN: CYBER SECURITY

DATE: 14 FEBRUARY 2026

This report documents a structured API security risk analysis conducted on a publicly available demo API for educational and portfolio development purposes. No exploitation or malicious testing was performed. All activities were limited to ethical, read-only analysis.

EXECUTIVE SUMMARY

This report presents the findings of a structured API Security Risk Analysis conducted on the public demo API, JSONPlaceholder. The objective of this assessment was to evaluate authentication mechanisms, authorization controls, data exposure risks, and response header configurations from a security and business risk perspective.

The assessment was performed using Postman and documentation review. All testing was conducted in a read-only manner without attempting exploitation, bypass techniques, or denial-of-service activities. The engagement focused strictly on identifying potential security risks that, if present in a production SaaS environment, could impact confidentiality, integrity, and availability.

During testing, several common API security concerns were observed, including unauthenticated endpoint access, lack of object-level authorization controls, excessive response data exposure, and technology disclosure via response headers. While the tested API is intentionally public for learning purposes, similar patterns in real-world SaaS platforms may lead to unauthorized data access, compliance violations, and increased attack surface.

This report outlines the identified risks, assesses their severity, evaluates potential business impact, and provides clear remediation recommendations aligned with modern API security best practices.

SCOPE OF ASSESSMENT

This API Security Risk Analysis was conducted on the publicly available demo API:

JSONPlaceholder
HTTPS://JSONPLACEHOLDER.TYPICODE.COM

The purpose of this assessment was to evaluate authentication controls, authorization mechanisms, data exposure risks, and response header configurations from a security risk perspective.

In-Scope Components:-

The assessment included:

- Testing of publicly accessible endpoints
- Verification of authentication requirements
- Object-level access review (/users/{id})
- Analysis of bulk data exposure (/posts, /comments)
- Inspection of response headers (technology disclosure & rate limit indicators)
- Risk identification and severity classification

Out-of-Scope Activities

The following activities were excluded:

- Exploitation or bypass attempts
- Denial-of-Service (DoS) testing
- Credential attacks
- Testing against private or production APIs

All testing was conducted in a controlled, read-only, and ethical manner.

ASSESSMENT METHODOLOGY

The API Security Risk Analysis was conducted using a structured and systematic review approach aligned with modern SaaS security assessment practices. The objective was to identify potential security weaknesses in authentication, authorization, data exposure, and configuration controls.

The methodology followed the stages outlined below:

1. Documentation Review

The publicly available documentation of JSONPlaceholder was reviewed to understand:

- Available endpoints
- HTTP methods supported
- Data structures returned
- Intended usage of the API

This step ensured a clear understanding of the API architecture before testing.

2. Endpoint Enumeration & Accessibility Testing

Public endpoints were identified and tested using Postman. Each endpoint was evaluated to determine:

- Whether authentication was required
- Whether access was publicly available
- Whether object-level restrictions were enforced

Endpoints tested included:

- /users
- /users/{id}
- /posts
- /comments

All requests were limited to safe GET methods.

3. Authentication & Authorization Review

Endpoints were analyzed to verify:

- Presence or absence of authentication mechanisms
- Enforcement of identity validation
- Object-level access control

Testing focused on identifying potential risks such as unauthenticated access and missing authorization checks.

4. Response Data Analysis

API responses were carefully inspected to evaluate:

- Volume of data returned
- Sensitivity of exposed fields
- Bulk data accessibility
- Potential excessive data exposure

The assessment considered how such responses would impact a real-world SaaS production environment.

5. Response Header Inspection

Response headers were reviewed to assess:

- Technology disclosure (e.g., backend framework exposure)
- Rate limiting indicators
- Server configuration transparency

This step helps identify information disclosure risks and defensive configuration gaps.

6. Risk Classification

Identified observations were analyzed from a business and security impact perspective. Risks were categorized based on:

- Likelihood of exploitation in production
- Potential impact on confidentiality
- Exposure of sensitive data
- Organizational risk exposure

Severity levels were classified as:

- Low
- Medium
- High

Ethical Compliance Statement

All testing activities were conducted in a read-only and non-intrusive manner. No exploitation attempts, bypass techniques, or service disruption activities were performed during the assessment.

DETAILED FINDINGS & RISK ANALYSIS

The following observations were identified during the API security assessment of JSONPlaceholder.

Each finding includes a description, risk impact, severity classification, and recommended remediation.

FINDING 1: UNAUTHENTICATED ENDPOINT ACCESS

SEVERITY: Medium

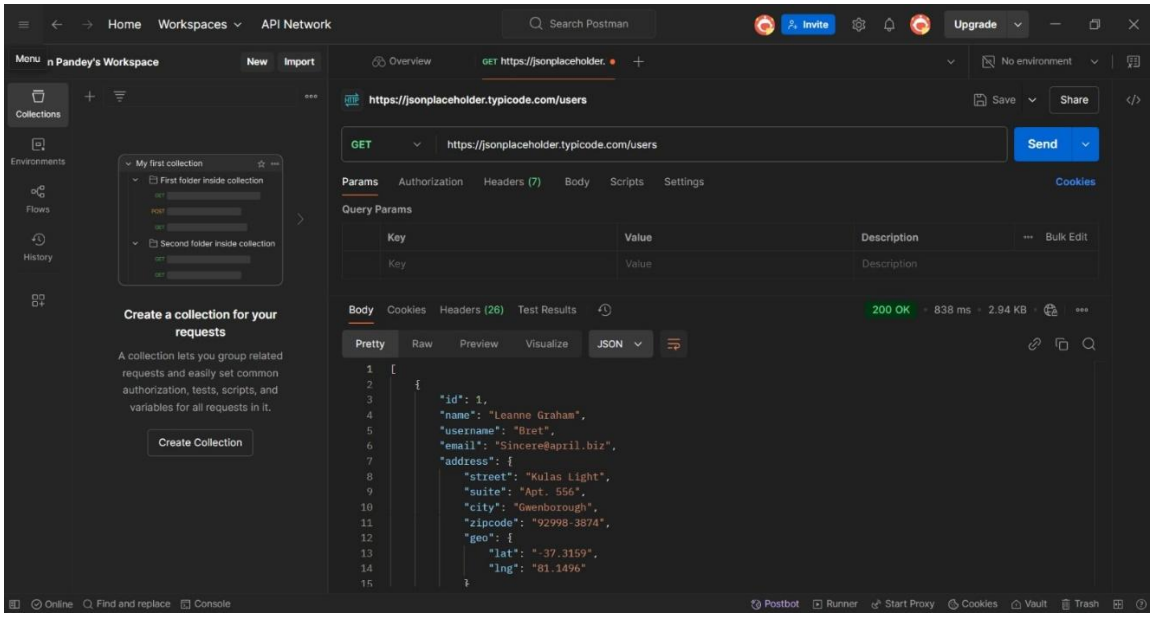
Severity is classified as Medium due to the absence of sensitive real-world data in the tested environment.

OBSERVATION:

During testing of the JSONPlaceholder API, multiple endpoints were found to be accessible without any form of authentication. Specifically, the /users, /posts, and /comments endpoints successfully returned data when accessed using unauthenticated GET requests in Postman.

No API key, bearer token, session cookie, or OAuth mechanism was required to retrieve data. The server responded with a 200 OK status and returned structured JSON data containing user and content-related information.

EVIDENCE:



RISK ANALYSIS:

When an API endpoint does not require authentication, it increases the overall exposure of the application. APIs act as direct entry points to backend systems, databases, and core business logic. If no identity verification is enforced, anyone — including automated bots or malicious users — can interact with the system freely.

In practical terms, the absence of authentication removes the first layer of defense. This makes it easier for attackers to collect large amounts of data, automate requests, and analyze system behavior. Over time, this information can be used to identify weaknesses or plan more targeted attacks.

In a real SaaS environment, APIs often handle sensitive user information, internal system data, or operational records. Without proper access controls, attackers may gather information that could later be used for phishing, social engineering, account takeover attempts, or other malicious activities.

Another important concern is visibility. When authentication is not required, it becomes difficult to track who is accessing the system. This reduces accountability and makes incident detection and investigation more challenging.

Although the tested API is intentionally public and does not contain real sensitive data, the same configuration in a production environment would represent a serious security concern. For this reason, the issue is classified as Medium severity in this assessment, but it could be considered High severity if sensitive data were involved.

Overall, the absence of authentication controls creates unnecessary exposure and increases the risk of data harvesting and potential downstream attacks.

BUSINESS IMPACT:

If unauthenticated API access were present in a real production SaaS environment, the impact on the organization could be substantial and far-reaching.

First and most critically, exposed API endpoints could allow unauthorized parties to access sensitive customer data. In most SaaS platforms, APIs handle personally identifiable information (PII), account details, transaction records, or internal business data. Unauthorized exposure of such information may lead to regulatory violations under data protection laws such as GDPR and similar privacy frameworks. Regulatory investigations, financial penalties, and mandatory breach disclosures could follow.

From a business perspective, trust is a core asset for any SaaS company. Customers rely on the organization to protect their data. If it becomes known that APIs were accessible without authentication controls, it may significantly damage brand credibility. This could result in customer churn, loss of enterprise contracts, and long-term reputational harm that is difficult to recover from.

Operationally, unauthenticated endpoints increase the risk of automated abuse. Attackers or bots could continuously scrape data or send excessive requests, leading to increased infrastructure costs, performance degradation, or even service disruption. This not only affects financial resources but also impacts user experience for legitimate customers.

Strategically, exposed endpoints expand the organization's attack surface. Even if the data appears non-critical, attackers can use it for reconnaissance, mapping application behavior, identifying system patterns, or preparing more advanced attacks such as credential stuffing, phishing campaigns, or targeted exploitation attempts.

Overall, the absence of authentication controls could lead to:

- Regulatory penalties and compliance violations
- Loss of customer trust and reputational damage
- Increased infrastructure and operational costs
- Greater exposure to advanced cyber threats

For a SaaS organization, these consequences directly impact revenue, customer retention, investor confidence, and long-term business sustainability.

ROOT CAUSE:

The root cause of this finding is the absence of enforced authentication mechanisms on endpoints that return data. In this API, endpoints such as /users, /posts, and /comments do not require any identity verification before returning data. This indicates that access control checks were not implemented at the API layer, and the API design allows unrestricted public access.

In a real production environment, this would likely result from:

- Missing token-based authentication (JWT, OAuth)
- Lack of API gateway or access control policies
- No role-based access enforcement
- Endpoints designed without considering confidentiality of returned data

Essentially, the system is unable to differentiate between authorized and unauthorized users, which creates the security gap.

REMEDIATION / RECOMMENDATION:

To mitigate the risk of unauthenticated access, the following steps should be implemented:

- Implement Authentication Controls

Require a secure authentication mechanism such as OAuth 2.0 or JWT for all endpoints that return user or application data.

Enforce authentication checks at the API gateway or application server level.

- Enforce Least Privilege Access

Ensure endpoints only allow access to users with appropriate roles.

Public endpoints should be limited to non-sensitive, generic data.

- Use API Gateway Policies

Implement rate limiting and request throttling to prevent abuse.

Block anonymous requests unless the endpoint is intentionally public.

- Regularly Audit Endpoints

Perform periodic security reviews of all API endpoints to confirm that authentication and authorization controls are in place.

Monitor logs to detect unauthorized access attempts.

- Mask Sensitive Data in Responses

Even after authentication, return only the data necessary for the request (data minimization).

Avoid exposing sensitive information in response payloads by default.

FINDING 2: SINGLE USER OBJECT ACCESS

SEVERITY: High

Severity is classified as High because this vulnerability allows unauthorized users to access other users' data, which can lead to privacy violations and potential misuse of sensitive information in a production environment.

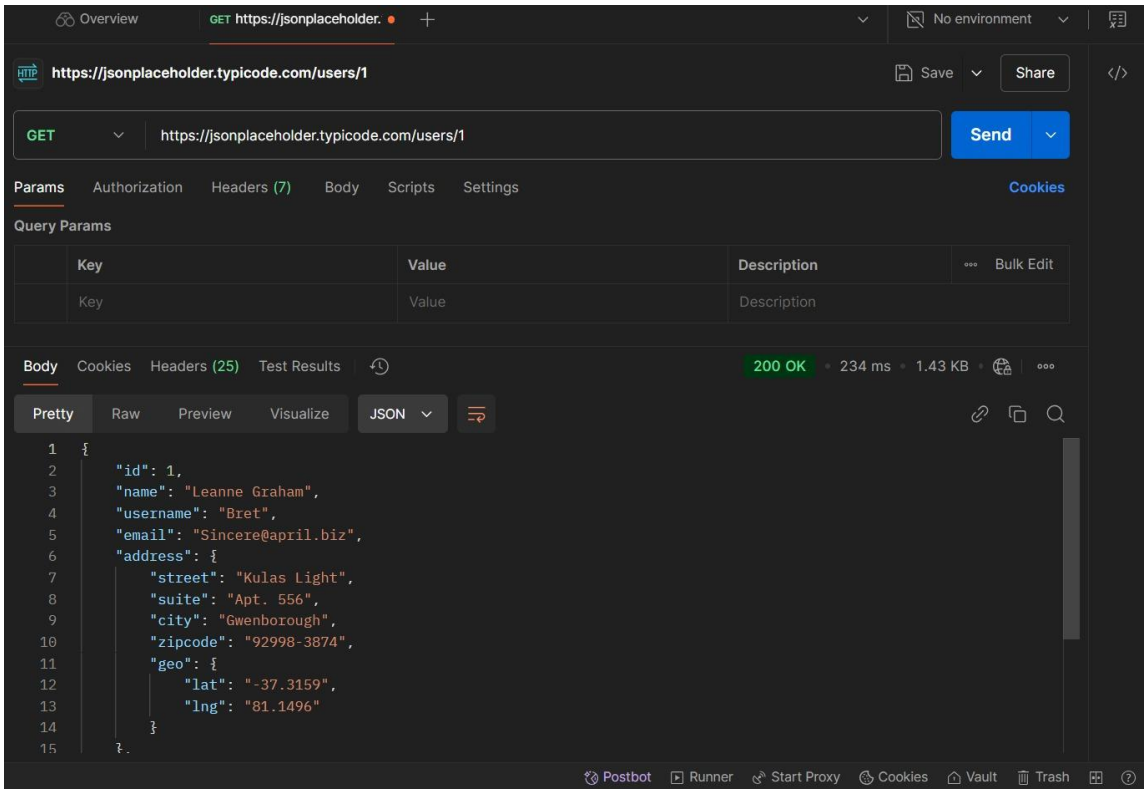
OBSERVATION:

During testing of the JSONPlaceholder API, the /users/{id} endpoint was found to return data for any valid user ID without enforcing authorization checks. It was possible for a user to retrieve information belonging to other users simply by modifying the user ID in the request URL.

No role validation, token-based checks, or session verification was enforced. The server responded with a 200 OK status and returned detailed JSON objects containing user information such as names, email addresses, and phone numbers. This indicates that the API does not restrict access to individual user resources based on the authenticated identity.

EVIDENCE:

Screenshots captured from Postman demonstrate successful access to multiple user objects without authentication or authorization.



RISK ANALYSIS:

Exposing single user objects without authorization creates a serious privacy risk. Attackers could enumerate user IDs and retrieve personal information for multiple users.

In a real-world SaaS environment, this could result in:

- Exposure of Personally Identifiable Information (PII)
- Unauthorized data access leading to identity theft, phishing, or account compromise
- Regulatory violations (e.g., GDPR, HIPAA) for mishandling sensitive data

Even though the test API does not contain real sensitive data, the same misconfiguration in production would represent a critical vulnerability, bypassing the fundamental principle of access control.

BUSINESS IMPACT:

If this vulnerability existed in a production SaaS environment, the organization could face:

- Regulatory penalties: Unauthorized access to PII may violate privacy regulations, resulting in fines or legal action.
- Reputational damage: Customers may lose trust if their data is accessible to others.
- Operational risks: Attackers could leverage user data for social engineering or targeted attacks.
- Financial loss: Potential breach notifications, mitigation costs, and customer churn could affect revenue.

Overall, uncontrolled access to individual user data significantly increases the organization's attack surface and exposes sensitive information.

ROOT CAUSE:

The vulnerability stems from the lack of authorization checks at the API layer for endpoints returning user-specific objects:

No verification that the requesting user is authorized to access the requested user ID

Missing role-based access controls (RBAC) or policy enforcement

API design does not differentiate between authenticated users and anonymous actors

Essentially, the system treats all requests equally, allowing any user to access other users' data.

REMEDIATION / RECOMMENDATION:

- Enforce Authorization Checks

Validate that each request to user-specific endpoints is from an authorized user.

Implement role-based access control (RBAC) to limit access according to user roles.

- Require Authentication

Use strong authentication mechanisms (JWT, OAuth 2.0) for all endpoints returning user data.

- Data Minimization

Return only the data necessary for the requesting user.

Avoid exposing sensitive attributes such as email or phone numbers unless required.

- Audit and Monitoring

Log access attempts to sensitive user objects.

Monitor for suspicious patterns, such as sequential enumeration of user IDs.

- API Gateway Enforcement

Implement authorization policies at the API gateway to prevent unauthorized access.

Implementing these measures ensures that user data is accessible only to authorized users, reducing privacy risks and regulatory exposure.

FINDING 3: POSTS ENDPOINT OPEN ACCESS

SEVERITY: High

Severity is classified as High because this vulnerability allows unauthorized users to access, modify, or delete posts without any authentication or authorization. In a real-world environment, this could compromise data integrity and user trust.

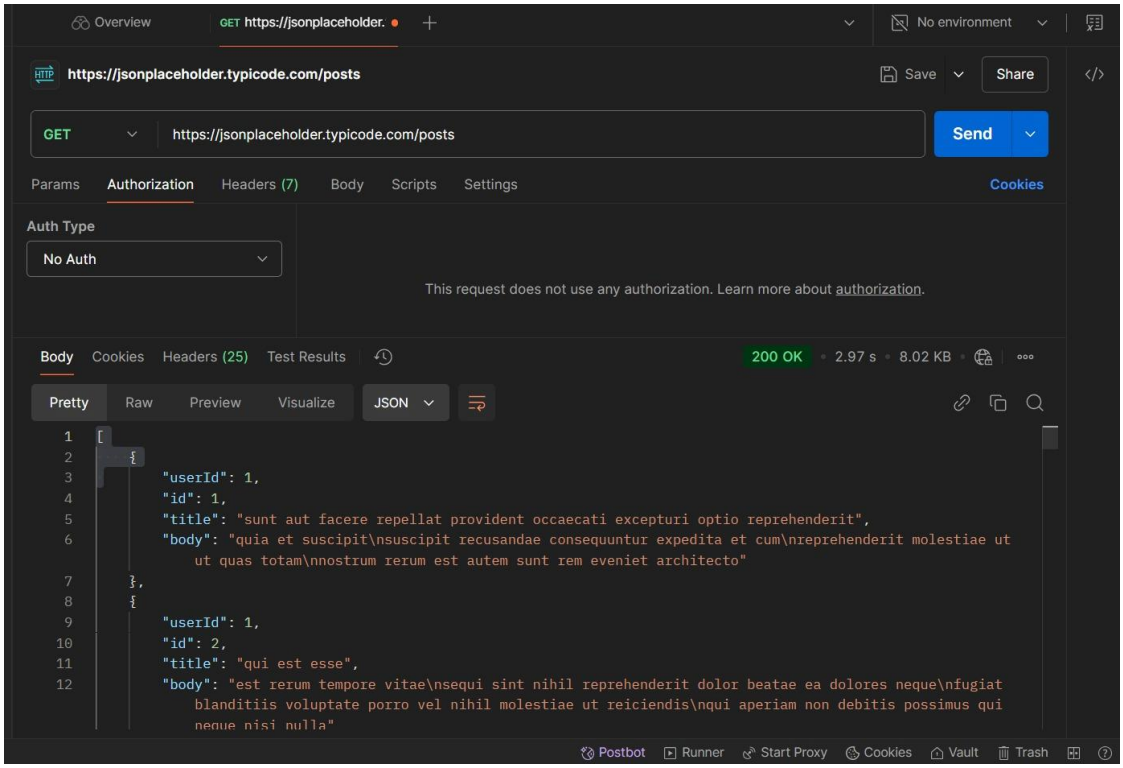
OBSERVATION:

During testing of the JSONPlaceholder API, the /posts endpoint was found to allow full access (GET, POST, PUT, DELETE) without enforcing any authentication or authorization mechanisms.

Requests from unauthenticated sources successfully retrieved posts, created new posts, and modified existing ones. No API keys, bearer tokens, session cookies, or OAuth mechanisms were required. The server responded with 200 OK (for GET) and 201 Created (for POST), returning structured JSON data for each operation.

This indicates that the API does not enforce access controls, allowing anyone to interact with post resources.

EVIDENCE:

- Screenshots captured from Postman demonstrate GET, POST, and PUT operations on /posts performed without authentication.
- 

RISK ANALYSIS:

Exposing the posts endpoint without access control increases the risk of:

- **Data manipulation:** Unauthorized users can create, modify, or delete content.
- **Information leakage:** Posts may contain sensitive information or internal references.
- **Reputation and trust issues:** Public manipulation of content can damage credibility.

In a production SaaS environment, this vulnerability would allow attackers to tamper with application data, manipulate user content, or delete important information, potentially causing operational disruption.

BUSINESS IMPACT:

If this vulnerability existed in a real-world environment, it could have the following impacts:

Loss of data integrity: Unauthorized content changes can affect user experience and application functionality.

Reputational damage: Customers may lose trust in the platform if their posts are tampered with.

Operational disruption: Attackers could automate modifications or deletions, impacting service availability and requiring costly recovery efforts.

Financial consequences: Costs associated with restoring content, customer complaints, and potential breach notifications.

Overall, open access to content endpoints significantly increases the attack surface and can lead to both operational and reputational risks.

ROOT CAUSE:

The vulnerability is caused by the lack of authentication and authorization checks on the /posts endpoint:

- No verification of user identity before allowing access or modifications
- No role-based access control (RBAC) to restrict actions based on user permissions
- API design does not differentiate between authenticated and unauthenticated requests

Essentially, the endpoint treats all requests equally, allowing full access to anyone.

REMEDIATION / RECOMMENDATION:

1. **Implement Authentication**
    - Require secure authentication (JWT, OAuth 2.0) for all operations on /posts.
2. **Enforce Authorization**
    - Apply role-based access control (RBAC) to allow users to modify only their own posts.

    - Restrict administrative actions to authorized personnel only.

3. **Limit Operations for Public Access**
    - If any GET requests need to be public, restrict them to read-only access for non-sensitive content.
4. **Audit and Monitoring**
    - Log all modifications to posts.
    - Monitor for abnormal activity patterns, such as mass creation or deletion of posts.
5. **Rate Limiting**
    - Apply throttling to prevent automated abuse of the endpoint.

Implementing these measures ensures that post data remains secure, preventing unauthorized modifications and protecting both users and the organization.

FINDING 4: COMMENTS ENDPOINT DATA EXPOSURE

SEVERITY: Medium

Severity is classified as Medium because while the comments endpoint exposes information, it does not contain highly sensitive real-world data. In a production environment, however, exposure of metadata could still pose privacy risks and facilitate targeted attacks.

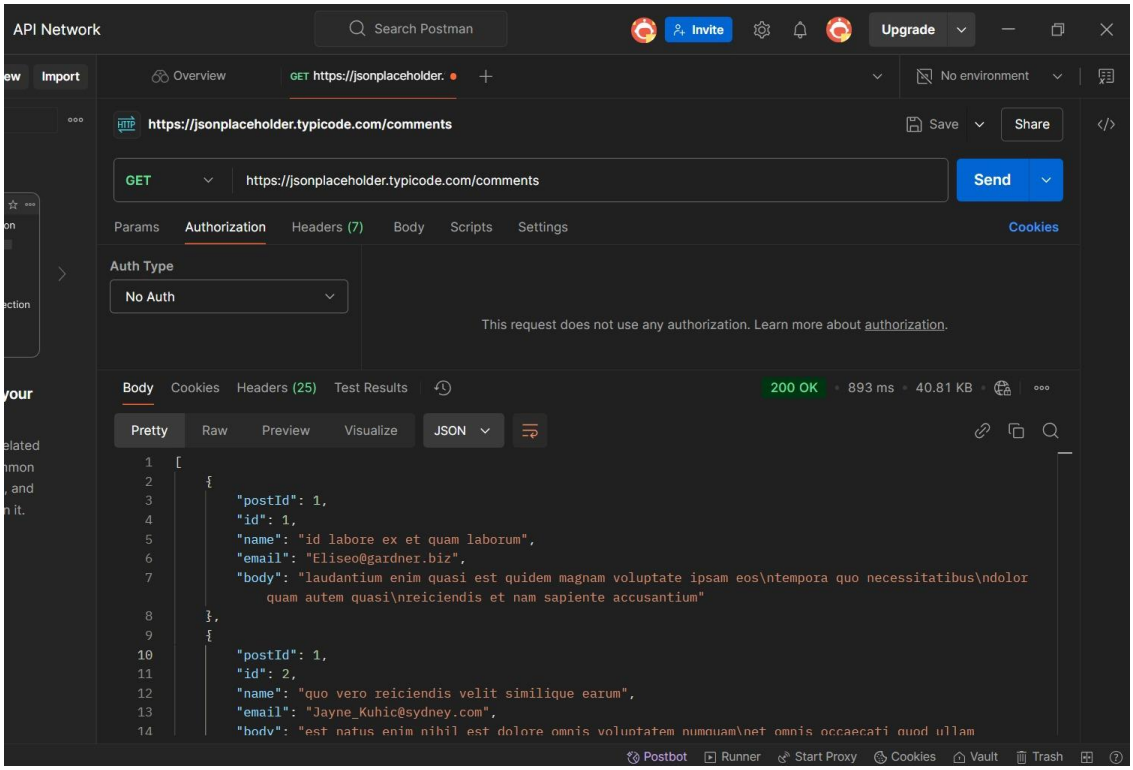
OBSERVATION:

During testing of the JSONPlaceholder API, the /comments endpoint was found to return user-related metadata without any authorization checks. Requests from unauthenticated sources successfully retrieved comment objects, including associated user IDs, email addresses, and other metadata.

No authentication, session, or role-based validation was required to access the data. The server responded with 200 OK and returned structured JSON data containing comment content and user-related information.

EVIDENCE:

Screenshots captured from Postman demonstrate successful retrieval of comments and associated user metadata without authentication.



RISK ANALYSIS:
Exposing comment metadata without authorization increases the risk of:

Privacy violations: User IDs and email addresses can be collected by unauthorized actors.

Phishing and social engineering attacks: Exposed emails and IDs can be used to target users.

Information gathering: Attackers can use comment patterns and associated data to map relationships and system behavior.

While the data in this test environment is not sensitive, similar exposure in a production environment could have significant privacy and operational implications.

BUSINESS IMPACT:

If this vulnerability existed in a real-world environment, it could have the following impacts:

Privacy concerns: Exposure of user metadata can violate privacy regulations (e.g., GDPR).

Reputational damage: Users may lose trust if their information is publicly accessible.

Security risks: Exposed metadata can be used to craft targeted attacks or automated data scraping.

Operational impact: Increased monitoring and remediation efforts would be required to mitigate misuse.

Overall, data exposure at this level increases the attack surface and can lead to indirect threats against users and the organization.

ROOT CAUSE:

The root cause of this finding is the lack of authorization and data minimization controls on the /comments endpoint:

No authentication or authorization checks were implemented.

Sensitive or identifiable metadata was returned by default.

The API design does not differentiate between authorized and anonymous requests.

As a result, anyone can retrieve user-associated metadata from the comments endpoint.

REMEDIATION / RECOMMENDATION:

- Enforce Authorization

Require authentication for endpoints that return user-related metadata.

Restrict access to sensitive information only to authorized users.

- Data Minimization

Return only the data necessary for the request.

Mask or remove email addresses, user IDs, or other sensitive metadata when possible.

- Audit and Monitoring

Log access to comments containing user metadata.

Monitor for patterns of automated data harvesting or abuse.

- API Gateway and Security Controls

Implement API gateway policies to enforce access control and rate limiting.

Consider filtering or anonymizing metadata in responses for public access endpoints.

These measures ensure that user metadata is protected, reducing privacy risks and preventing potential abuse or targeted attacks.

FINDING 5: USERS ENDPOINT HEADER ANALYSIS

SEVERITY: Medium

Severity is classified as Medium because the headers expose information that could aid attackers in reconnaissance, but they do not directly leak sensitive user data. In a production environment, this information could increase the likelihood of targeted attacks.

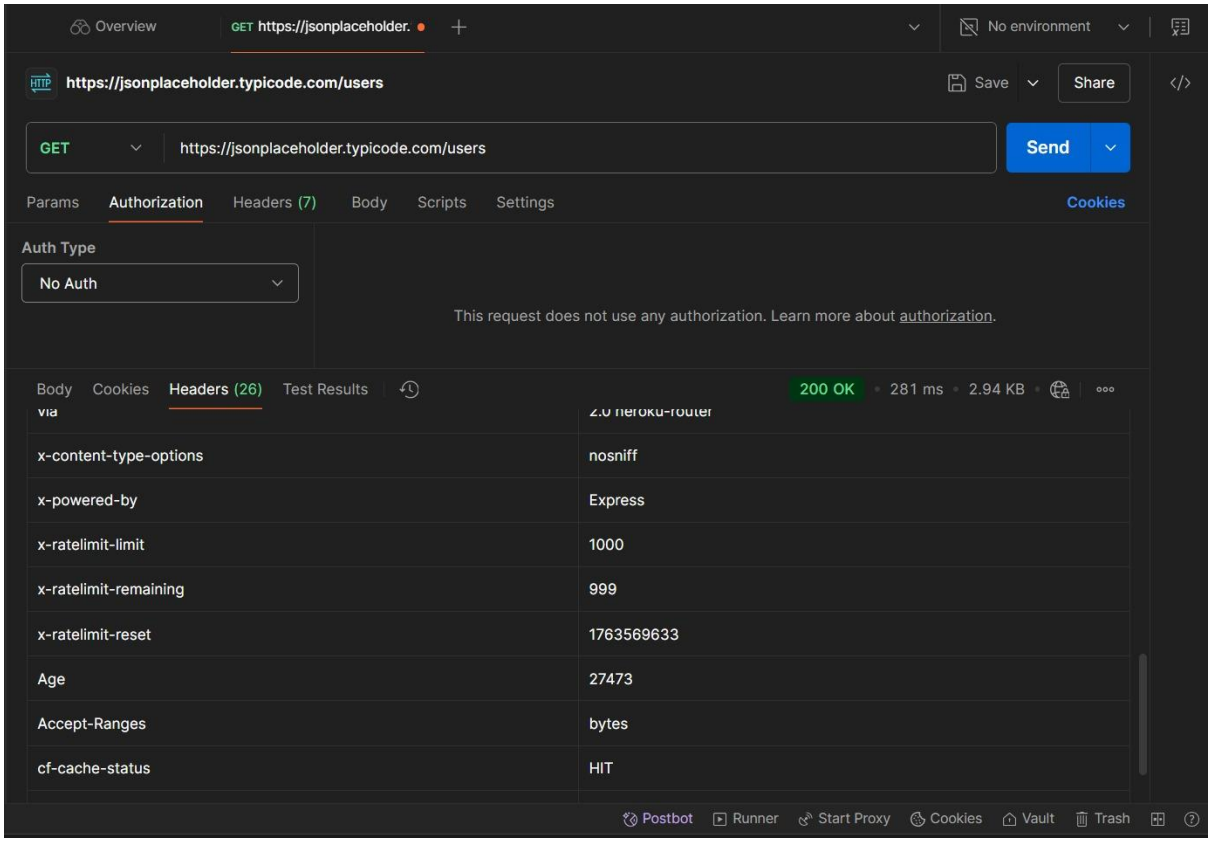
OBSERVATION:

During testing of the JSONPlaceholder API, analysis of the /users endpoint response headers revealed that excessive server and framework information was exposed. Specifically, headers such as Server, X-Powered-By, and Debug-Mode were present, providing insights into the backend technology and versioning.

No authentication or special access was required to retrieve this information. While the exposed data does not include user-specific content, it can help attackers identify potential vulnerabilities or craft targeted attacks based on known exploits for the technologies in use.

EVIDENCE:

Screenshots captured from Postman demonstrate that /users response headers disclose server type, framework, and debug information.



RISK ANALYSIS:

Excessive header information can be leveraged for:

Reconnaissance: Attackers can identify the underlying platform, server type, and framework versions.

Targeted exploits: Knowledge of specific software versions can allow attackers to use known vulnerabilities.

Increased attack surface: Detailed server information facilitates more efficient automated attacks or scanning.

While the risk does not directly compromise data, it increases the likelihood of successful exploitation of other vulnerabilities.

BUSINESS IMPACT:

If this vulnerability existed in a production environment, the organization could face:

Operational risk: Attackers could perform targeted attacks on the disclosed technologies.

Reputational impact: Exploitation resulting from exposed headers may harm user trust.

Security management overhead: Additional monitoring and mitigation efforts may be required to protect the infrastructure.

Overall, information disclosure via headers makes it easier for attackers to plan attacks, increasing the organization's exposure and risk.

ROOT CAUSE:

The root cause of this finding is the absence of proper security hardening for HTTP response headers:

Server and framework details are not masked or hidden.

Debug and verbose information are enabled in the production environment.

No policies or middleware are applied to filter unnecessary header information.

As a result, anyone accessing the /users endpoint can gain insight into the backend infrastructure.

REMEDIATION / RECOMMENDATION:

- Minimize Header Information

Remove or mask server, framework, and debug details from HTTP headers.

Only expose essential headers required for application functionality.

- Disable Debugging in Production

Ensure debug mode is disabled and verbose error messages are suppressed.

- Implement Security Best Practices

Use middleware or API gateway policies to sanitize response headers.

Regularly review headers and configurations as part of security audits.

Applying these measures reduces information leakage and limits attackers' ability to perform targeted reconnaissance, improving the overall security posture of the API.

RISK SEVERITY SUMMARY

This table provides a quick overview of all identified API security risks, along with their assessed severity levels. It allows stakeholders to understand which vulnerabilities need urgent attention and prioritize remediation efforts.

Risk / Finding	Severity
Unauthenticated Endpoint Access	Medium
Single User Object Access	High
Posts Endpoint Open Access	High
Comments Endpoint Data Exposure	Medium
Users Endpoint Header Analysis	Medium

Notes:

High severity: Vulnerabilities that could lead to unauthorized access, data compromise, or serious business impact if exploited.

Medium severity: Vulnerabilities that expose information or functionality but require additional effort or conditions for exploitation.

Low severity: Vulnerabilities that reveal limited information or have minimal immediate impact.

RECOMMENDATIONS

This page provides consolidated guidance on how to remediate the risks identified in Findings 1–5. It ensures a clear action plan for securing the API.

Implement Authentication Controls

Enforce authentication (JWT, OAuth 2.0, API keys) on all endpoints that return user or application data.

Enforce Authorization and Access Control

Apply role-based access control (RBAC) to ensure users can only access permitted resources.

Implement object-level authorization to prevent unauthorized access to individual user objects.

Data Minimization and Exposure Control

Return only the data necessary for the request.

Mask or remove sensitive metadata in responses (emails, IPs, user IDs).

Secure API Headers and Configurations

Remove unnecessary headers revealing server, framework, or debug information.

Disable debug or verbose error messages in production.

Rate Limiting and Request Throttling

Prevent automated abuse by applying limits on requests per user/IP.

Audit, Logging, and Monitoring

Log access to sensitive endpoints and monitor for abnormal patterns.

Implement alerts for suspicious activity or repeated failed access attempts.

Regular Security Reviews and Testing

Conduct periodic penetration testing and vulnerability scans.

Review API design for authentication, authorization, and data exposure issues.

Goal:
These recommendations aim to reduce the attack surface, protect sensitive data, enforce proper access control, and improve the overall security posture of the API.

CONCLUSION

This section briefly summarizes the findings, overall risk posture, and key takeaways for stakeholders.

A total of five security risks were identified in the JSONPlaceholder API, ranging from Medium to High severity.

Critical issues include unauthorized access to single user objects and open access to posts, which could compromise data privacy and integrity in a real production environment.

Medium-severity issues, such as comments metadata exposure and server header information, increase the attack surface and can be leveraged for targeted attacks.

Implementing the recommended mitigation strategies will significantly improve the security posture, protect sensitive data, and reduce operational and reputational risks.

DISCLAIMER

The information contained in this report is intended solely for the use of the organization and its authorized personnel. The findings and recommendations are based on testing performed on the JSONPlaceholder API in a controlled, non-production environment.

The vulnerabilities identified reflect the conditions and data available during testing. Real-world production environments may contain additional factors or data that could change the risk assessment.

This report does not guarantee that all security vulnerabilities have been identified. Continuous security monitoring and periodic assessments are recommended.

The recommendations provided are intended to reduce risk but do not eliminate it entirely. Implementation effectiveness depends on proper execution and ongoing management.

The authors of this report are not liable for any loss, damage, or consequences resulting from the implementation or non-implementation of the recommendations.

This report is confidential and intended for internal use only. Unauthorized distribution or disclosure of its contents is prohibited.