



Web App / API Penetration Test

Prepared for: Sample Inc

COMPLETE REPORT

April 16, 2026

Red Sentry

This engagement was performed in accordance with the Statement of Work, and the procedures were limited to those described in that agreement. The findings and recommendations in this report reflect the systems, scope, and timeframe defined in the Statement of Work.

While every effort was made to provide thorough coverage, no security assessment can identify every possible issue, and results should not be interpreted as a guarantee of complete security.

Contact Information

+1 (888) 337-0467

hello@redsentry.com



TABLE OF CONTENTS

Summary.....4

 Project Overview..... 4

 Goals..... 4

 Dates..... 4

 Findings..... 4

 Scope..... 5

Tests Performed.....6

Findings.....7

Appendices.....9

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, 2026.

Goals

Identify vulnerabilities in the company’s web application following the OWASP Top 10 methodology.

Dates

April 08, 2026	April 08, 2026 - April 24, 2026	April 16, 2026
KICKOFF	TESTING PERIOD	DELIVERY

Findings Overview

During the assessment, the testing team identified multiple security findings across the environment.

A finding represents a unique security issue with a distinct root cause.

Instances represent the number of instances where that issue appears within the assessed environment.

The following table summarizes the identified findings and the number of Instances observed during testing.

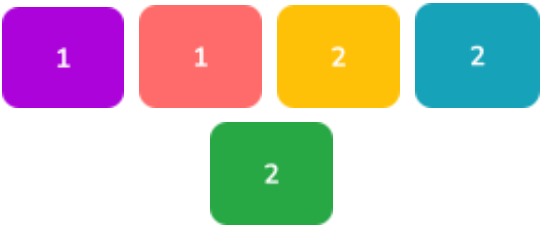


Severity	Finding ID	Instances
CRITICAL	T9PCFOR-F0001	1
HIGH	T9PCFOR-F0002	1
MEDIUM	T9PCFOR-F0003	1
	T9PCFOR-F0004	1
LOW	T9PCFOR-F0005	1
	T9PCFOR-F0006	1
INFO	T9PCFOR-F0007	1
	T9PCFOR-F0008	1

Overall Grade



GRADE



Scope

app.samplecompany.com

EXECUTIVE SUMMARY

The assessment team conducted a penetration test on Sample Company's web application. Overall, the application needs improvement as 8 issues were detected, including a critical one.

Regarding the severity of the findings, 1 critical vulnerability was detected. Some user input is directly embedded in database queries. This allows an attacker to interfere with the database queries and view data they are normally not able to retrieve.

Then, 1 high vulnerability was found as well. The web application authenticates users by sending them a 6 digit number to their email address. Since the application is not properly enforcing rate-limiting, an attacker could try brute forcing this code and take over the account.

Next, 2 instances of medium issues were identified, these are related to privilege escalation and stored cross-site scripting (XSS). These issues lead to the gain of additional privileges and a full user compromise, respectively.

Additionally, 2 types of low vulnerabilities were identified, relating to HTML injection in email and user enumeration. These issues allow an attacker to deface emails or supply phishing payloads and differentiate between valid and invalid users, respectively, which makes brute-force attacks more feasible.

Finally, 2 informational vulnerabilities were identified, relating to the exposure of the Google Maps API keys and the exposure of the application's API key. These issues lead to unauthorized usage and significant fees and unauthorized access to API endpoints.

In summary, to improve the organization's security posture, the assessment team recommends leveraging parameterized queries for all database queries, enforcing strict rate-limiting policies, avoid exposing sensitive configuration details, such as API keys or tokens in client-side code, implement proper access controls for all sensitive actions, use input validation and sanitization to prevent stored XSS attacks

and HTML injection attacks and returning the same message for valid and invalid email addresses when logging in.

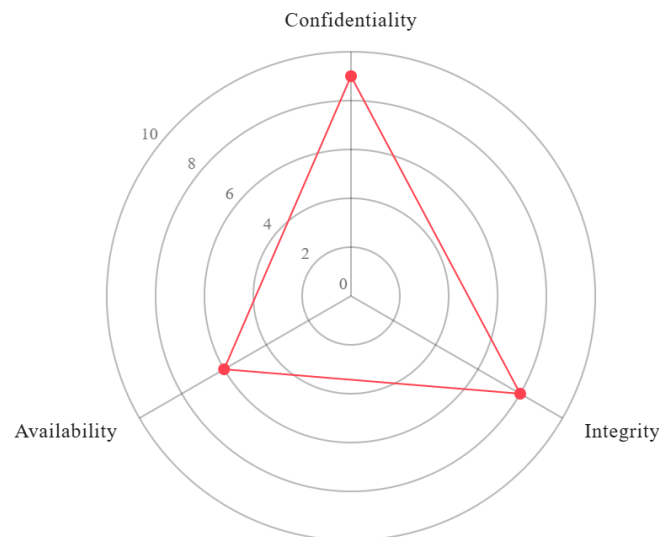


Figure 1: Security risk chart

As the image above shows, the overall risk value is high as all of the risk element scores are equal or above 4.

As a general rule, an overall risk below 4 shows low business impact, between 4 and 7 medium impact and above 7 high impact.

Test Performed

For this project, the assessment team followed both a black-box and a white-box approach to testing, looking for common vectors of attack according to the OWASP Top 10 methodology. From a black-box approach, the main goal is to discover issues from an unauthenticated perspective. This includes the following tests:

API Discovery

Using a proxy connection, intercept HTTP connections to discover underlying APIs that could expose resources for an unauthenticated user.

Authenticated Sensitive Data Exposure

Attempt to access information about resources like API keys, tokens, cloud credentials, or private information like Personally Identifiable Information (PII).

Authentication Bypass

We used a set of attacks to access information requiring session tokens. This involves active user enumeration attacks and password spraying from standard common lists.

Broken Access Controls

Look for ways to access resources from an unauthorized user.

Credential Stuffing

Attempt to make use of relevant leaked credentials gathered from the Dark Web or any other public or paid source.

Directory Discovery

Look for paths not available in tags present in the main HTML document. This was done using a brute-force approach based on standard lists of common path names.

Dorking:

Use sophisticated queries to uncover information revealed by search engines that may not normally be discoverable by conventional searches.

Injection

Use attacks like SQL/NoSQL injection, cross-site scripting (XSS), code injection, and command injection. Trying ways to inject code to cause unexpected behavior in the application.

Insecure Deserialization

Attempt to exploit weaknesses in the deserialization technology and/or implementation.



Known Vulnerabilities

Search for CVEs based on the technology stack being used.

Reconnaissance

Utilize publicly available tools to gather sensitive information, such as that provided with a Shodan search.

Security Misconfigurations

Search for misconfigurations related to implementations following bad practices from a software design and architecture perspective.

Sensitive Data Exposure

Analyzed all of the information available to an unauthenticated user that could be leveraged against the company's confidentiality.

Server-side request forgery (SSRF)

Use an attack that allows a malicious user to command the application server to make arbitrary requests, and that could lead to sensitive data exposure.

Session Management Vulnerabilities

Attempt to spot weaknesses in the way session cookies or tokens are implemented.

Web App / API Penetration Test Findings

SQL INJECTION

CRITICAL

Identifier	T9PCFOR-F0001
Severity	Critical
Instances	1

Description

SQL injection, also known as SQLI, is a common attack vector that uses malicious SQL code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.

Instances (1)

- `api.samplecompany/api/web/v1/call/outstanding`

Evidence

The carProviderNameSearch, invoiceNumberSearch, passengerNameSearch and voucherNum query parameters on /api/web/v1/rides/outstanding are vulnerable to SQL injection.



Figure 2: SQL injection validation

As shown in above Figure, appending a single quote (') results in a 500 internal server error.

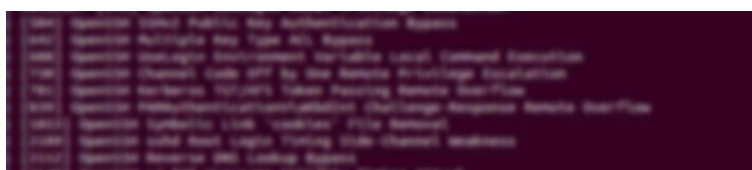


Figure 3: SQL injection closing query

If we append an additional single quote, we see the internal server error disappear, indicating that we closed the query correctly. We then tried extracting data by closing the query and appending a statement that always evaluates to true, so we extract data.

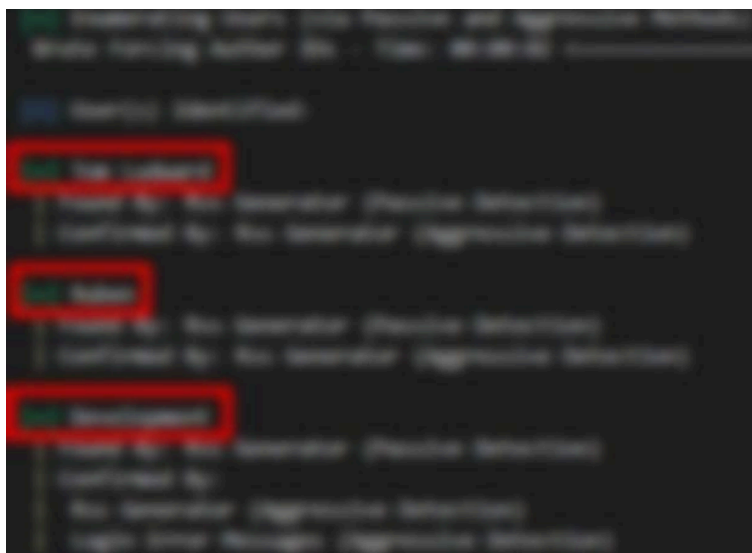


Figure 4: Extracting data using SQL injection

This vulnerability allows an attacker to gain unauthorized access to sensitive data such as user information, credit card information, and all other data stored in the database. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines.

Steps for Recreation

1. Navigate to the endpoint mentioned in the locations section.
2. Add a single quote (') and send the request. Note that the application throws an error.
3. Add one more single quote (') and observe that the error disappears.
4. Run sqlmap on the vulnerable endpoint and extract the tables, columns, etc.

Recommendations

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

- Be sure to use secure coding practices, independent of the programming language. All standard web development platforms (including PHP, ASP.NET, Java, and but also Python or Ruby) have mechanisms for avoiding SQL Injections, including Blind SQL Injections. Try to avoid dynamic SQL at all costs
- The best option is to use prepared queries, also known as parameterized statements. Also, you can use stored procedures that most SQL databases support (PostgreSQL, Oracle, MySQL, MS SQL Server). Additionally, escaping or filtering special characters (such as the single quote which is used for classic SQL Injections) for all user data inputs
- Do not rely solely on WAF/IDS/IPS mechanisms, always try to sanitize input from users

Resources

About SQL Injection

<https://portswigger.net/web-security/sql-injection>

ACCOUNT TAKEOVER

HIGH

Identifier	T9PCFOR-F0002
Severity	High
Instances	1

Description

Account Takeover is an attack whereby an attacker can take ownership of accounts.

Instances (1)

- api.samplecompany.com/api/web/v1/auth/login/complete

- Enforcing strong rate-limiting policies

Resources

About account takeover

<https://www.cloudflare.com/learning/access-management/account-takeover/>

STORED CROSS-SITE SCRIPTING

MEDIUM

Identifier	T9PCFOR-F0003
Severity	Medium
Instances	1

Description

Stored cross-site scripting (XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

Instances (1)

- api.samplecompany.com/api/web/v1/superadmin/documents/broadcastNotes

Upon updating the document on the notification board, we can append event handlers, as shown in the below Figure.



Figure 8: Javascript payload executed

Figure 9: XSS also pops on /new-booking

Steps for Recreation

1. Modify the document on the notification board with a JavaScript payload.
2. Hover over the notification functionality and notice that the inserted payload is executed successfully.

Recommendations

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

- Perform strict filtering when user input is received.
- Encode the output where user-controllable data is put into HTTP responses to prevent it from being interpreted as active content.
- As a last line of defense, you can use a Content Security Policy to reduce the severity of an XSS vulnerability.

Resources

About Cross-site scripting

<https://portswigger.net/web-security/cross-site-scripting>

PRIVILEGE ESCALATION

MEDIUM

Identifier	T9PCFOR-F0004
Severity	Medium
Instances	1

Description

This vulnerability allows an attacker to escalate privileges from one stage to another. This occurs when a user gets access to more resources or functionality that they are normally allowed, and such elevation or changes should have been prevented by the application.

Instances (1)

- api.samplecompany.com/api/web/v1/users//contactInfo

Evidence

The application allows an user with admin privileges to add their email address as a primary email address to the account of an user with super admin privileges. The authentication codes for that account will then be sent to the email address specified by the attacker, giving them the ability to login to the account and get super admin privileges.

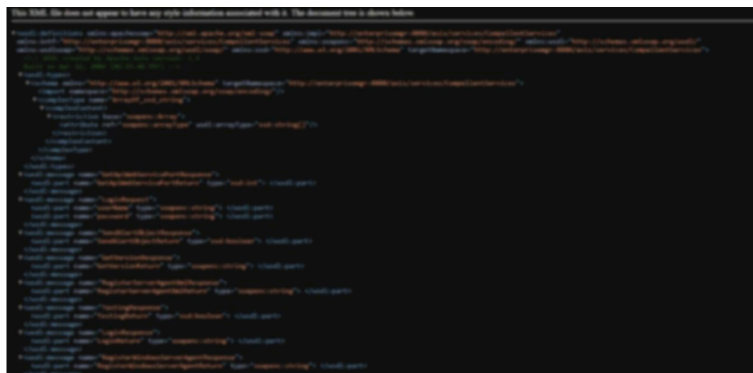


Figure 10: Manage user with super admin privileges

After adding our email address as the primary email address, the 6-digit OTP is sent to that email address, and we can log in as that user, as shown in Figure 11, and have super admin privileges.

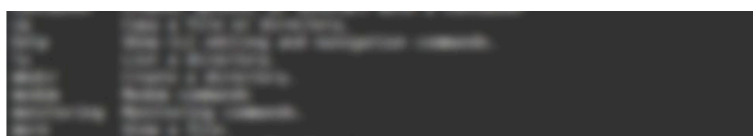


Figure 11: Super admin privileges

An attacker with access to an admin account can elevate their privileges to super admin, thereby increasing the impact the attacker can have on the application.

Steps for Recreation

1. Log in to the application as an admin and modify the super admin's primary email.
2. Click the Save button, then log in with the newly modified email address and note that the logged-in user is a super admin.

Recommendations

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

- Implement controls so that a user cannot edit the email addresses of users with higher privileges.

Resources

About privilege escalation

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/03-Testing_for_Privilege_Escalation

HTML INJECTION

LOW

Identifier	T9PCFOR-F0005
Severity	Low
Instances	1

Description

Hypertext Markup Language (HTML) injection is a technique used to take advantage of non-validated input to modify an email presented by the application to its users.

Instances (1)

- `api.samplecompany.com/api/web/v1/reservations/<reservation id=""></reservation>`

Evidence

Upon editing a booking, and putting HTML tags into a field, an email is sent to the users.

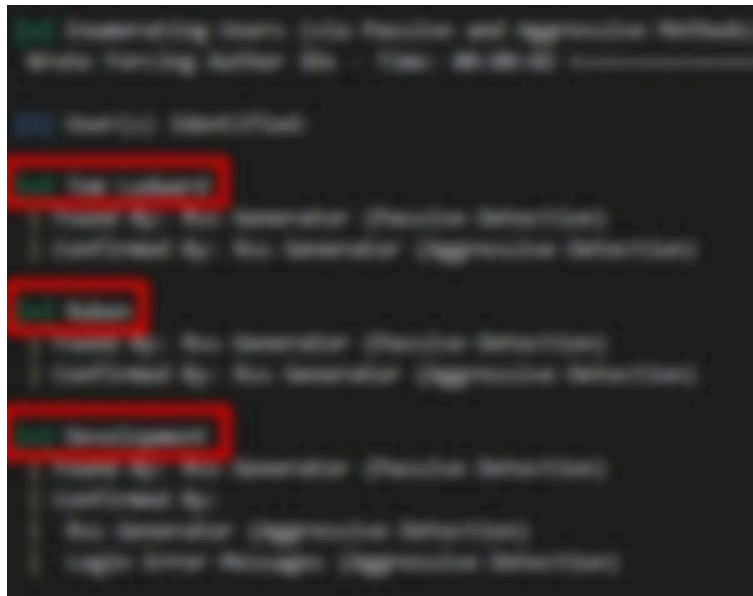


Figure 12: HTML injected into email

This results in the HTML being interpreted, as shown in below figure.



Figure 13: HTML injected into email

This allows an attacker to embed phishing content into the email that is more likely to be successful as it comes from a valid and trusted email address, namely `helpdesk@samplecompany.com`. It can also be used to supply NSFW images, etc to damage the company's reputation.

Steps for Recreation

1. Fill in HTML tags while creating a booking.
2. Note that the HTML payload is executed successfully when the email is opened.

Recommendations

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

- Remove any HTML-syntax substrings from user-input.

Resources

About HTML Injection

<https://www.imperva.com/learn/application-security/html-injection/>

USER ENUMERATION

LOW

Identifier	T9PCFOR-F0006
Severity	Low
Instances	1

Description

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

Instances (1)

- api.samplecompany.com/api/web/v1/auth/login/initiate

Evidence

When an email address does not exist within the application, the error message shown in below figure is returned.

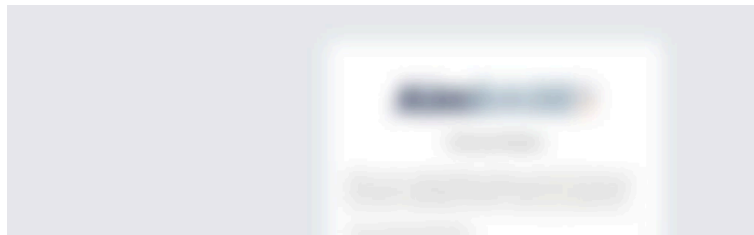


Figure 14: Error message when a user does not exist

If the email exists, the user is sent to a page that prompts them to enter a 6-digit OTP that's sent to the email address that they entered.

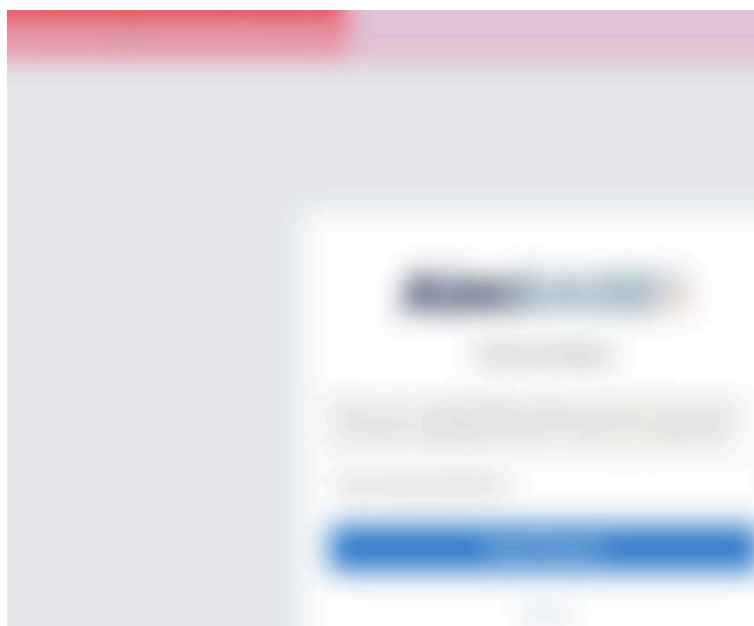


Figure 15: Page when user entered a valid email address

An attacker can differentiate between valid and invalid email addresses, meaning brute force attacks are more likely to succeed.

Steps for Recreation

1. Navigate to the login page.
2. Enter both existing and non-existing usernames with invalid passwords, and observe the application's response.

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 is incorrect, the attack surface decreases.

Resources

About User enumeration

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

GOOGLE MAPS API KEYS EXPOSED

INFO

Identifier	T9PCFOR-F0007
Severity	Info
Instances	1

Description

Google Maps API is a paid service that allows applications to embed & search from the Google Maps Database and use it on their own applications. While some of the services were free at the time of Early 2018, they changed their usage plan after that date. Since then, developers have needed to use an API key or client-id solution for all of the APIs they are using. These API keys have some security configurations for blocking unauthorized usage by malicious people, which does not come by default. Developers need to configure these security controls within their own decisions.

While these API keys are designed as public API keys and do not have any impact in terms of customer data confidentiality/integrity, these security configurations still need to be implemented to block unauthorized usage.

Instances (1)

- booking.samplecompany.com/app377a6dc.js

Evidence

The Google Maps API key is exposed at /app377a6dc.js:

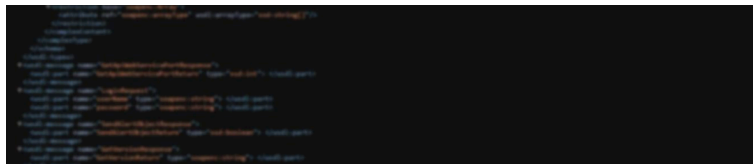


Figure 16: Google Maps API key exposed

Upon testing, we found that the key has the following rights:

- Directions API (\$5 per 1000 requests)
- Geocode API (\$5 per 1000 requests)
- Distance Matrix API (\$5 per 1000 elements)
- Find Place From Text API (\$17 per 1000 elements)
- Autocomplete API (\$2.83 per 1000 requests)
- Nearest Roads API (\$10 per 1000 requests)
- Route to Traveled API (\$10 per 1000 requests)
- Place Details API (\$17 per 1000 requests)
- Nearby Search-Places API (\$32 per 1000 requests)
- Text Search-Places API (\$32 per 1000 requests)
- Places Photo API (\$7 per 1000 requests)

Steps for Recreation

1. Navigate to the endpoint mentioned in the Location section and note that the application is disclosing the Google Maps API key in the response.

Recommendations

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

- It is recommended to refrain from disclosing this type of information either in the JavaScript file or client-side code.

Resources

About Google maps API keys usage

<https://ozguralp.medium.com/unauthorized-google-maps-api-key-usage-cases-and-why-you-need-to-care-1ccb28bf21e>

API KEY EXPOSED

INFO

Identifier	T9PCFOR-F0008
Severity	Info
Instances	1

Description

The application leverages an API key in the header when making requests to the API. If no valid ApiKey is specified in this header, the API will return a 401 Unauthorized. But since this value is exposed in app377a6dc.js it could be retrieved by attackers.

Instances (1)

- booking.samplecompany.com/app377a6dc.js

Evidence

The ApiKey value is exposed in /app377a6dc.js as shown in below figure:

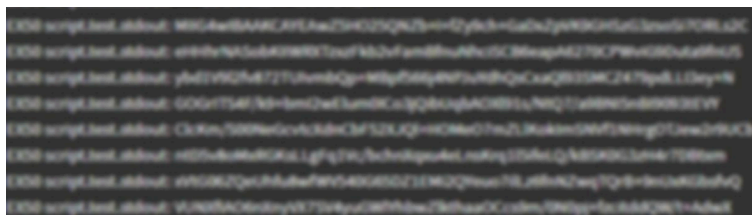


Figure 17: ApiKey exposed

We can then issue requests to the API using this header. If we supply a mock value, we retrieve the 401 Unauthorized as expected, as shown in figure.



Figure 18: 401 Unauthorized with invalid ApiKey value

If we supply the value we obtained from the JavaScript file, we get a 200 OK response, as shown in below figure.

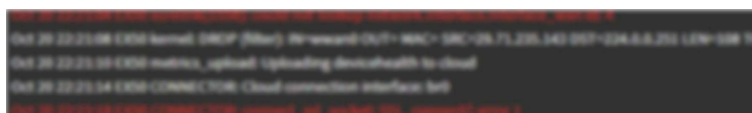


Figure 19: 200 OK with valid ApiKey

This may allow an attacker to gain unauthorized access and interact with API endpoints that do not require the JWT token.

Steps for Recreation

1. Navigate to the endpoint mentioned in the Location section and observe that the application discloses the API key in the response.

Recommendations

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

- It is recommended to refrain from disclosing this type of information either in the JavaScript file or client-side code.

Resources

About secure API Keys.

<https://dev.to/korconnect/secure-api-keys-on-the-frontend-vue-js-1l21>

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.

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

A purple rounded rectangular badge with the word "CRITICAL" in white, uppercase letters.

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

A red rounded rectangular badge with the word "HIGH" in white, uppercase letters.

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.