VILKA WEB APPLICATION SECURITY AUDIT REPORT

Performer

TLSGuard

Type of work

Black Box Web Application Security Assessment (Basic Audit)

Period of work

May 2026

Executive Summary

As part of the work, a security audit of the **vilka.lovable.app** web application was conducted using the **Black Box Web Application Security Assessment** method without providing access to the source code and internal infrastructure of the application.

During the testing, an analysis of the external attack surface, a study of authentication and access control mechanisms, an analysis of the Supabase API, verification of user input processing, and testing of the application's business logic were performed.

The audit revealed vulnerabilities of various levels of criticality. The most significant violation is the violation of object Level access control (**Broken Object Level Authorization / IDOR**), which allows an authorized user to access other users' data by modifying the object identifier in API requests. An architectural weakness related to the disclosure of object identifiers has also been identified, which significantly simplifies the exploitation of the main vulnerability.

The infrastructure analysis showed the use of modern technologies and the absence of signs of typical configuration errors that can lead to compromise of the application at the infrastructure level.

No critical vulnerabilities allowing Remote Code Execution, SQL injection, privilege escalation, or compromise of the server infrastructure were identified during the testing.

It is recommended to eliminate the identified deficiencies in accordance with their level of criticality, giving priority to the implementation of correct access control at the object level and

setting up **Row Level Security (RLS)** policies. After making the changes, it is recommended to re-test the security to confirm the effectiveness of the implemented protection measures.

---

1. The purpose of the audit

The purpose of the audit was to assess the current security level of the **vilka.lovable.app** web application, identify potential security vulnerabilities, and analyze the protection mechanisms implemented in the application.

During the work, special attention was paid to analyzing the external attack surface, authentication and authorization mechanisms, verifying access control to objects, examining APIs, analyzing application business logic, and identifying potential configuration errors that could violate the confidentiality, integrity, or availability of user data.

The audit was conducted using the **Black Box** method, without providing the source code, internal documentation, access to the infrastructure or administrative components of the application. All conclusions of the report are based solely on the results of an analysis of publicly available web application functions and manual testing of its operation mechanisms.

---

2. Testing Limitations

This security audit was conducted in the **Black Box Web Application Security Assessment** format, which provides for a study of exclusively publicly available web application components without access to the internal infrastructure.

As part of the work, **access was not provided** to the following components:

- the source code of the application;
- server infrastructure;
- administrative panels;
- web server configurations;
- the database;
- event logs (Logs);
- Continuous integration and delivery system (CI/CD);
- internal project documentation.

Due to these limitations, the audit results reflect the security status exclusively of the external surface of the application and the mechanisms available to authorized and unauthorized users.

It should be borne in mind that the absence of identified vulnerabilities in the framework of this testing does not guarantee the complete absence of security flaws in the internal components of the system, which were not accessed during the audit.

All conclusions and recommendations are based on the results of testing conducted during the specified period and reflect the status of the application at the time of completion.

---

3. Methodology

The security audit of the **vilka.lovable.app** web application was conducted using the **Black Box Web Application Security Assessment** method, in which testing was performed without access to the source code, internal infrastructure, and server configuration of the application.

The research used a combination of automated and manual testing aimed at identifying the most common web application vulnerabilities, business logic implementation errors, and deficiencies in access control mechanisms.

As part of the audit, the following research steps were performed:

- analysis of the external attack surface and available network services;
- definition of the technologies used and the architecture of the application;
- mapping of publicly available routes and APIs;
- analysis of authentication mechanisms and user session management;
- verification of authorization mechanisms and access control to application objects;
- testing the Supabase REST API business logic;
- analysis of user input processing;
- checking the security configuration of the web application;
- manual testing of detected potential vulnerabilities using specialized tools.

Special attention was paid to verifying the correctness of the implementation of **Row Level Security (RLS)**, mechanisms for delimiting access to user data, and API resilience to attacks of the **Broken Object Level Authorization (BOLA/IDOR)** class.

The results of the automated analysis were additionally confirmed by manual testing, which made it possible to exclude false positives and assess the actual possibility of exploiting the identified safety deficiencies.

---

4. Tools used

During the security audit, modern web application security analysis tools were used to study the network infrastructure, analyze HTTP traffic, test the REST API, and verify the business logic of the application.

The following tools were used to perform the work:

- **Burp Suite Professional** — analysis and modification of HTTP(S) traffic, API testing, manual vulnerability checking;
- **Burp Repeater** — playback and modification of HTTP requests;
- **Burp Intruder** — testing query parameters and verifying access control mechanisms;
- **Nmap** — network infrastructure analysis and discovery of available services;
- **Httpx** — checking the availability of web services and determining the technologies used;
- **WhatWeb** — identification of application components and technology stack;
- **Katana** — automatic mapping of the web application structure and search for entry points;
- **Dirsearch** — search for hidden directories and publicly available resources;
- **Nikto** — analysis of the web server configuration and verification of common security errors;
- **Nuclei** — automated verification of known vulnerability patterns.

The results of the automated analysis were additionally confirmed by manual testing, which made it possible to exclude false positive results and assess the possibility of practical exploitation of the identified vulnerabilities in the real-world operation of the application.

---

5. Infrastructure analysis results

During the audit, an analysis of the external infrastructure of the **vilka.lovable.app** web application was performed, aimed at identifying potential network-level vulnerabilities, configuration errors, and components that can increase the attack surface.

The study included identifying the technologies used, analyzing available services, mapping public entry points, and evaluating the overall configuration of the web application.

5.1 Discovered Services

The study found that the application is available exclusively over the secure HTTPS protocol and is located behind the reverse proxy server Cloudflare, which provides protection against some network attacks and hides the internal infrastructure of the application.

There were no open administrative interfaces, additional public services, or non-standard network components capable of significantly increasing the attack surface.

The results obtained indicate the correct organization of the external security perimeter and the absence of obvious infrastructural deficiencies accessible from the Internet.

5.2 Technologies used

During the analysis, the following technologies and components were identified:

- Cloudflare;
- Supabase;
- PostgreSQL;
- REST API;
- JavaScript;
- Single Page Application (SPA).

The technology stack used corresponds to the modern architecture of web applications and does not contain signs of using outdated or potentially unsafe components.

5.3 Identified entry points

During the mapping of the application, publicly available APIs and functional components were discovered that provide:

- user registration and authentication;
- user profile management;
- user search;
- messaging;
- interaction with group chats;
- processing requests to the Supabase REST API.

These components were used as the main objects of subsequent manual security testing.

5.4 Overall infrastructure assessment

According to the results of the analysis, no critical flaws were identified in the configuration of the public infrastructure. The use of Cloudflare as a protective perimeter, the use of modern development technologies and the lack of open administrative services have a positive impact on the overall security level of the application.

The main risk vector is not related to the infrastructure component, but to the implementation of authorization and access control mechanisms at the API level. The vulnerabilities identified during testing are due to the specifics of the implementation of the business logic of the application and are not the result of errors in the configuration of the server infrastructure.

---

6. Security Test Results

During the security audit of the **vilka.lovable.app** web application, the mechanisms of authentication, authorization, user data processing, the logic of interaction with the REST API, as well as the correctness of the implementation of access control mechanisms were investigated.

The main focus was on verifying the business logic of the application, implementing access control policies, and securing interaction with the Supabase platform.

According to the test results, two vulnerabilities of different criticality levels have been confirmed.

6.1 Broken Object Level Authorization (IDOR)

Risk level

High

Category

OWASP API1:2023 – Broken Object Level Authorization (BOLA)

Description

During the analysis of the REST API of the application, a violation of the mechanisms of access control at the object level (Broken Object Level Authorization, BOLA) was found.

The endpoint for obtaining user profile data searches for a record using the `id=eq.<UUID>` parameter, however, the server does not verify that the requested object belongs to the owner of the current user session. When processing the request, only the correctness of the JWT token is checked, while there is no authorization for access to a specific record.

As a result, any authorized user can get information about other users' profiles by substituting the object ID in the request.

The vulnerability is caused by the lack of well-configured policies **Row Level Security (RLS)** for the `profiles` table, which allows you to bypass the access control mechanism without exploiting additional security flaws.

Proof of Concept

During testing, the standard application request for obtaining its own profile data was intercepted.

After changing the parameter value

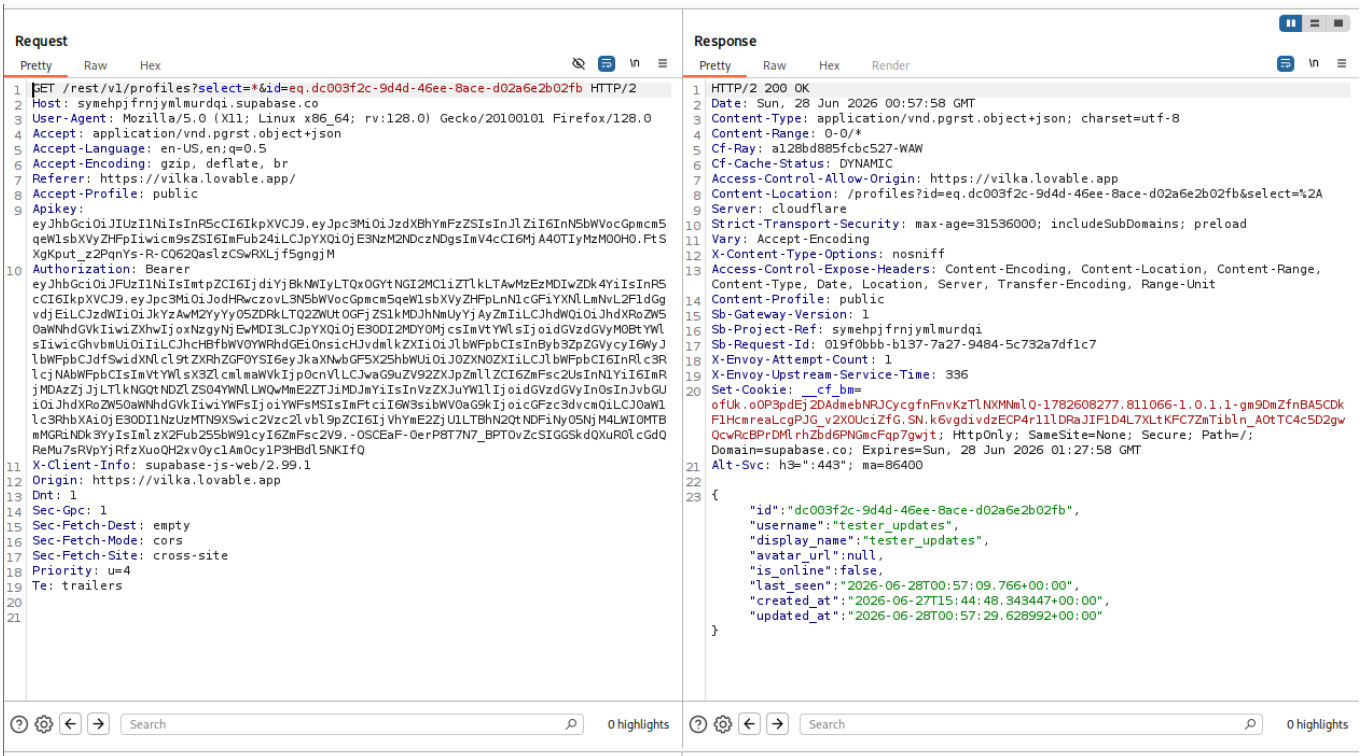
```
id=eq.<UUID>
```

for another user's ID, while saving its own JWT token, the server successfully processed the request and returned information about someone else's profile.

The server response contained the following data:

- user ID;
- user's name;
- display name;
- presence status;
- the time of the last activity (`last_seen');
- the date of account creation;
- the date of the last profile update.

The request ended with the response **HTTP/2 200 OK**, which confirms that there is no verification of access rights on the server side.



Influence

Privacy: High

Integrity: Missing

Availability: Not available

Risks

The exploitation of this vulnerability allows:

- get information about other users' profiles;
- to carry out mass collection of user data;
- disclose internal system metadata;
- violate the confidentiality of user information;
- use the data obtained to carry out subsequent social engineering attacks.

Recommendations

To eliminate the vulnerability, it is recommended:

- enable the **Row Level Security (RLS) mechanism** for the `profiles` table;
- implement server-side verification of the object's ownership to the current user by means of `auth.uid()`;
- prohibit the execution of operations for reading records belonging to other users;
- restrict the transfer of internal service fields ('last_seen', `created_at`, `updated_at`) via public APIs;
- perform repeated testing after making changes.

6.2 Exposure of Internal Object Identifiers (Security Through Obscurity)

Risk level

Medium

Category

CWE-656 – Reliance on Security Through Obscurity

Description

During the analysis of the application's business logic, it was found that user identifiers (UUIDs) used as access keys to objects are freely transferred to the client application through the standard functions of user search and group chats.

Using UUIDs as identifiers is not a security disadvantage in itself, however, the lack of additional authorization mechanisms makes them actual access keys to user data.

Thus, the application architecture is partly based on the principle of "security through concealment", suggesting that the unpredictability of UUIDs prevents the exploitation of vulnerabilities. In practice, identifiers are easily collected through legitimate application functions and can be used to automate the exploitation of the main vulnerability of Broken Object Level Authorization.

Proof of Concept

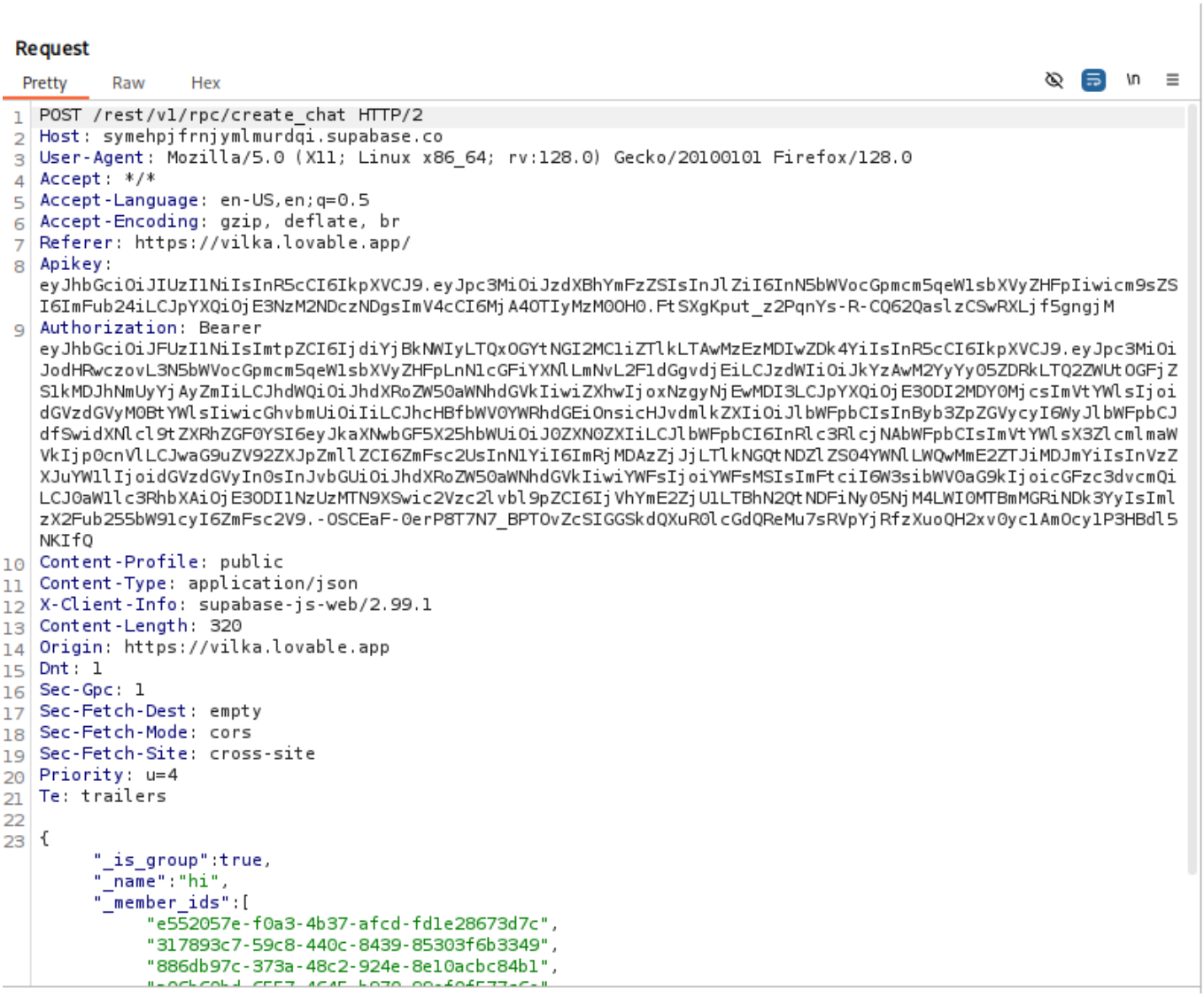
During the testing, the functions of searching for users and viewing information about participants in group chats were investigated.

When performing search queries, the server returns a JSON response containing the UUID of the found users.

The received identifiers can be used in endpoint requests.

```
/rest/v1/profiles
```

this allows you to automate obtaining data from a large number of users if there is a vulnerability described in the section **6.1** .



Influence

Privacy: Average

Integrity: Missing

Availability: Not available

Risks

Using this architecture greatly simplifies the exploitation of the underlying vulnerability and allows:

- automate the collection of user UUIDs;

- perform a massive search of objects;
- accelerate attacks on access control mechanisms;
- increase the scale of potential data leakage.

It should be noted that this problem does not compromise the application on its own, but it significantly increases the likelihood of successful exploitation of the Broken Object Level Authorization vulnerability.

Recommendations

To reduce the level of risk, it is recommended:

- do not consider UUID as an information protection mechanism;
- limit the transfer of internal identifiers and service metadata through public APIs;
- apply the principle of minimum necessary data (Least Privilege);
- implement strict access control to each object on the server side;
- use **Row Level Security (RLS) policies** for all data reading, modification, and deletion operations.

---

7. The final risk matrix

Vulnerability	Risk level
Broken Object Level Authorization (IDOR/BOLA)	**High**
Exposure of Internal Object Identifiers (Security Through Obscurity)	**Medium**

---

8. Conclusion

During the **vilka.lovable.app** security audit, the **Black Box Web Application Security Assessment** method assessed the external attack surface, authentication and authorization mechanisms, configuration of the public infrastructure, as well as the business logic of interaction with the API.

Based on the analysis results, it was found that the application uses a modern architecture based on the **Supabase** platform, is located behind the protective perimeter of **Cloudflare** and

does not contain signs of typical infrastructure vulnerabilities that can compromise the server side of the application. As part of the testing, vulnerabilities allowing Remote Code Execution, SQL injection, Cross-Site Scripting (XSS), privilege escalation, or circumvention of authentication mechanisms were not confirmed.

The most critical issue identified is the violation of object Level access control (**Broken Object Level Authorization / IDOR**), which allows an authorized user to access other users' profile data by changing the object identifier in API requests. Additionally, an architectural weakness has been identified related to the disclosure of internal user identifiers through legitimate application functions, which greatly simplifies the exploitation of the underlying vulnerability and increases the likelihood of automated user data collection.

The identified shortcomings are due to the specifics of the implementation of business logic and access control mechanisms, rather than errors in the configuration of the public infrastructure. The main reason for the critical vulnerability is the lack of well-configured policies **Row Level Security (RLS)** for the `profiles` table, which allows the server to process requests to other people's objects without checking whether the data belongs to the current user.

It is recommended to eliminate the identified deficiencies in accordance with their level of criticality, implement full-fledged access control at the object level, limit the transfer of redundant data through public APIs, and implement **Row Level Security** policies for all operations of reading and modifying user data. After the implementation of the proposed measures, it is recommended to conduct repeated security testing to confirm the effectiveness of the changes made and the absence of the possibility of re-exploiting the identified vulnerabilities.