Analysis of Security Vulnerabilities in a Modern Stack Based on React and Vite, React Native, NestJS, and Prisma

STROUN Elie
Epitech Rennes
elie.stroun@epitech.eu

Contributors: Nathan Jeannot, Manech Dubreil, Aymeric Jouannet-Mimy, Pablo Jesus

***Abstract*—This paper presents a security study of a modern stack composed of React + Vite (frontend), React Native (mobile), NestJS (backend), Prisma (ORM), and PostgreSQL (database). The objective is to identify known vulnerabilities, analyze their potential impact, and propose mitigation recommendations. This study is based on a literature review (CVE, OWASP) as well as simple experimental tests.**

***Index Terms*—Security, React, React Native, NestJS, Prisma, PostgreSQL, OWASP, Vulnerabilities**

I. Introduction

Modern JavaScript/TypeScript frameworks such as React and NestJS have become standards in web and mobile development. However, their widespread adoption introduces new attack surfaces. This paper aims to examine the security flaws associated with this stack. We focus on the following technologies:

- **Frontend:** React + Vite
- **Mobile:** React Native
- **Backend:** NestJS
- **Database:** Prisma + PostgreSQL

We will analyze common vulnerabilities, their implications, and mitigation strategies.

II. Related Work

Briefly present the OWASP Top 10, studies on the security of JS frameworks, recent vulnerability reports (CVE).

The OWASP Top 10 is a regularly updated report outlining the most critical security risks to web applications. The 2021 edition highlights risks such as Injection, Broken Authentication, and Cross-Site Scripting (XSS),[5]. The National Vulnerability Database (NVD) provides a comprehensive list of known vulnerabilities, including those affecting JavaScript frameworks [7]. For instance, several CVEs have been reported for NestJS, including issues related to improper input validation and authentication bypass [8]. React, being one of the most popular frontend libraries, has also faced scrutiny. Common vulnerabilities include XSS attacks due to improper handling of user input and issues with component lifecycle methods that can lead to data leaks [9], [10]. Vite, a modern build tool for frontend development, has had vulnerabilities such as arbitrary file read issues [11]. For mobile applications, the OWASP Mobile Top 10 identifies risks like Insecure Data Storage and Insufficient Cryptography [6]. Numerous studies have examined the security of JavaScript frameworks. Many articles provide insights into common vulnerabilities and best practices for secure coding in these environments [14].

III. Methodology

- Selection of technologies.
- Literature review and research in vulnerability databases.

IV. Security Analysis

A. React + Vite (Frontend)

React applications are susceptible to several security vulnerabilities, primarily due to improper handling of user input and state management. Common issues include Cross-Site Scripting (XSS), where malicious scripts are injected into the application, and Component Injection, where attackers exploit component properties to manipulate the application behavior [9], [10]. Vite, as a build tool, can also introduce vulnerabilities if not configured properly. Issues such as arbitrary file reads and exposure of sensitive information through source maps have been reported [11].

B. React Native (Mobile)

React Native applications face unique security challenges, particularly in the areas of data storage and communication. The OWASP Mobile Top 10 highlights risks such as Insecure Data Storage, where sensitive information is not adequately protected on the device, and Insufficient Cryptography, which can lead to data exposure during transmission [6]. Additionally, improper handling of user input can result in vulnerabilities similar to those found in web applications, such as XSS and injection attacks [14].

C. NestJS (Backend)

NestJS, being a backend framework, is susceptible to different types of security vulnerabilities compared to frontend frameworks. Common issues include improper authentication and authorization mechanisms, which can lead to unauthorized

access to sensitive resources [3]. Additionally, vulnerabilities related to data validation and sanitization can expose applications to injection attacks and data breaches [3].

D. Prisma + PostgreSQL

Prisma, as an ORM, can help mitigate SQL injection risks by using parameterized queries. However, developers must still be cautious about how they construct queries and ensure that user input is never directly concatenated into SQL statements [12]. PostgreSQL itself is a robust database system, but it is not immune to security threats. Common vulnerabilities include misconfigurations, weak access controls, and unpatched software [12]. Implementing role-based access control (RBAC), encrypting sensitive data at rest and in transit, and regularly applying security patches are critical steps in securing a PostgreSQL database for a live application.

V. Discussion

The analysis reveals that while each component of the stack has its own set of vulnerabilities, many of these can be mitigated through best practices in secure coding, configuration, and regular updates. For instance, using libraries like DOMPurify can help prevent XSS attacks in React applications, while Implementing stong authentication and authorization mechanisms can secure NestJS backends [16]. Regularly updating dependencies and monitoring vulnerability databases like NVD for new CVEs is also crucial for maintaining securty across the stack. Ideally, a comprehensive security strategy should take into account the specific risks associated with each Technology and implement layered defenses to protect against a wide range of threats.

VI. Conclusion and Perspectives

This study highlights the importance of understanding and addressing security vulnerabilities in modern web and mobile development stacks with a growing number of users and threats. By adopting best practices and staying informed about emerging threats, developers can significantly reduce the risk of security breaches. Real threats exists and are most of the time the ones that we don't know yet. Future work could involve developing automated tools for vulnerability detection and mitigation specific to this stack.

References

[1] OWASP, *OWASP Top 10 Web and Mobile*, 2021.
[2] National Vulnerability Database, https://nvd.nist.gov/.
[3] NestJS Documentation, https://nestjs.com/.
[4] React Documentation, https://react.dev/.
[5] OWASP Foundation, "OWASP Top Ten Web Application Security Risks 2021 https://owasp.org/Top10/, 2021.
[6] OWASP Foundation, "OWASP Mobile Top Ten Security Risks 2016 https://owasp.org/www-project-mobile-top-10/, 2016.
[7] National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD) https://nvd.nist.gov/, 2025.
[8] MITRE, "Common Vulnerabilities and Exposures (CVE) for NestJS https://cve.mitre.org/, 2025.
[9] Z. Guo, M. Kang, V.N. Venkatakrishnan, R. Gjomemo and Y.Cao "ReactAppScan: Mining React Application Vulnerabilities via Component Graph https://dl.acm.org/doi/10.1145/3658644.3670331, 2024.
[10] Scofield O. Idehen, "Top Security Vulnerabilities in React Applications https://medium.com/@Scofield_Idehen/top-security-vulnerabilities-in-react-applications-b67f1c375494, 2023.
[11] Sangfor Technologies, "CVE-2025-31486: Vite Arbitrary File Read https://www.sangfor.com/farsight-labs-threat-intelligence/cybersecurity/cve-2025-31486-vite-arbitrary-file-read, 2025.
[12] Floris Van den Abeele, "Prisma and PostgreSQL vulnerable to NoSQL injection? A surprising security risk explained https://fr.aikido.dev/blog/prisma-and-postgresql-vulnerable-to-nosql-injection, 2025.
[13] Ddos, "Critical RCE Flaw (CVE-2025-54594) in React Native Bottom Tabs' GitHub Actions Exposed Secrets https://securityonline.info/critical-rce-flaw-cve-2025-54594-in-react-native-bottom-tab 2025.
[14] Ankit Dhawan on Medium, "Secure Your React Native App from Vulnerabilities https://medium.com/@ankit.ad.dhawan/secure-your-react-native-app-from-vulnerabilities-e23b5cb00b2a, 2024.
[15] Brittany Day on Linux Security, "Critical NestJS Vulnerability Exposes Developers to RCE Risk https://linuxsecurity.com/news/security-vulnerabilities/critical-nestjs-rce-vulnerability, 2025.
[16] Ankit on Medium, "How to Prevent Security Vulnerabilities in a NestJS API Before Deployment https://medium.com/@mohantaankit2002/how-to-prevent-security-vulnerabilities-in-a-nestjs-api-before-deployment-3c2eaa05e0 2025.