TOKEN-BASED AUTHENTICATION: NAVIGATING ACCESS SCENARIOS FOR SECURE USER VERIFICATION

MATEI-VASILE CĂPÎLNAȘ[1], TRAIAN SIMEDRU[2]

Abstract: The concepts of digitization and automation are becoming increasingly common nowadays. The development of web applications or the migration of different softwares to the web make the appearance of security breaches more and more frequent. This paper aims to provide a solution to secure the authentication process in order to reduce the risks of cyber-attacks.

Key words: web, cyberattacks, cybersecurity.

1. INTRODUCTION

The number of web applications is constantly growing. Although there is no exact statistic of the number of web applications, at the beginning of 2022 there were approximately 351.5 million domain names purchased. We can thus say that this is also the approximate number of web applications, regardless of their type (static or dynamic). This study's emphasis on web applications is driven by the rising prevalence of vulnerabilities and attacks within this domain, coupled with the limited number of research studies that provide visual insights into this issue. Fig.1 illustrates the evolving trends in the Open Web Application Security Project (OWASP) top ten vulnerabilities from 2017 to 2021.

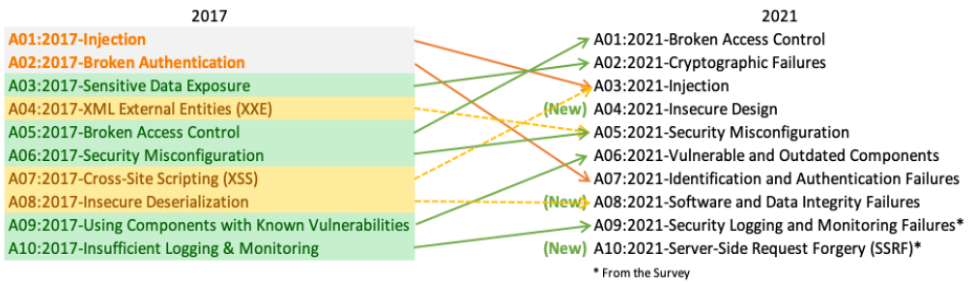

Fig.1. OWASP top ten vulnerabilities

[1]*Ph.D. Student, Assist. Prof. Eng., University of Alba Iulia, capilnas.matei@uab.ro*
[2]*Student, University of Alba Iulia, simedru.traian.infoen21@uab.ro*

Security vulnerabilities often creep into software during its creation. To catch and fix these issues within the software development cycle, organizations adopt various methods like security audits and static analysis. A wide array of commercial tools and numerous research endeavors are focused on aiding the discovery of these security flaws. Responsibility for application security commonly resides with a specialized team within organizations, known as the 'Software Security Group' or SSG [1]. This team consists of application security experts charged with performing both static and dynamic analyses to pinpoint security vulnerabilities lurking in application source code.

2. METHODOLOGY DESCRIPTION

TypeScript Remote Procedure Call (RPC) is a concept and approach that involves using TypeScript, a statically-typed superset of JavaScript, to facilitate communication between different parts of a software system or between different systems [2]. RPC is a mechanism that allows one piece of code to invoke functions or methods on a remote server or module as if they were local, abstracting away the complexities of network communication and data serialization [3].

Here's how TypeScript RPC typically works:

- Defining Remote Services: You define a set of services, along with their methods, that you want to expose for remote access. These services are typically written in TypeScript and represent the business logic or functionality you want to access remotely.
- Generating TypeScript Interfaces: You create TypeScript interfaces that define the structure of these remote services. These interfaces serve as a contract between the client and the server for the available methods and their parameters.
- Client-Server Communication: TypeScript RPC frameworks or libraries handle the communication between the client and server. When a client wants to invoke a method on a remote service, the framework takes care of serializing the data, sending it over a network (e.g., HTTP, WebSocket, or other transport protocols), and deserializing the response on the server side.
- Type Safety: One of the key advantages of using TypeScript for RPC is that it provides type safety. TypeScript checks the types of function parameters and return values, which can help catch errors at compile-time rather than runtime.
- Code Generation: Some TypeScript RPC frameworks or tools may also provide code generation capabilities. They can generate TypeScript code for client-side and server-side components, which ensures that the client and server both understand the service contracts.

Node.js is an open-source, cross-platform JavaScript runtime environment that allows you to execute JavaScript code on the server-side. It is built on the V8 JavaScript engine developed by Google for use in their Chrome web browser. Node.js extends the capabilities of JavaScript beyond just being a client-side scripting

language, enabling it to be used for server-side scripting and building network applications [4].

Key features and characteristics of Node.js include:

- Asynchronous and Non-blocking: Node.js is designed to be non-blocking and event-driven. This means it can handle many concurrent connections and I/O operations without getting blocked, making it highly efficient for building scalable and high-performance applications.
- Event Loop: Node.js uses an event-driven, single-threaded architecture, which allows it to efficiently manage asynchronous operations through an event loop.
- NPM (Node Package Manager): Node.js comes with a package manager called NPM that simplifies the installation and management of third-party libraries and modules, making it easy for developers to reuse code and share their own libraries.
- Server-Side Applications: Node.js is often used for developing server-side applications, such as web servers, API servers, and real-time applications like chat applications and online games.
- JavaScript: Node.js uses JavaScript as its primary programming language, which allows for code reuse between the client and server, making it easier for full-stack developers.
- Large Ecosystem: There is a vast ecosystem of open-source libraries and frameworks available through NPM, which can significantly speed up the development process and provide solutions for various use cases.

3. THE FUNCTIONALITIES OF THE APPLICATION

The primary aim of this application was centered around developing a robust authentication system that prioritized security. This was achieved by leveraging several key components such as Remote Procedure Calls (RPC), Cookies, Access Tokens, and Refresh Tokens. These elements were strategically employed to ensure a secure and reliable authentication process within the application.

When overseeing the authorization of a login attempt, the system considered three distinct scenarios or cases. In each of these cases, the system executed checks to verify the legitimacy and validity of two essential components: the Access Token and the Refresh Token.

The Access Token, which remained valid for a duration of one hour, was scrutinized by the system to ensure its current and legitimate status. Similarly, the Refresh Token, designed to remain valid for a span of one day, underwent examination by the system to confirm its authenticity and appropriateness for the login attempt.This dual verification process aimed to guarantee the security and integrity of the authentication procedure by evaluating the validity of these tokens within specified timeframes.

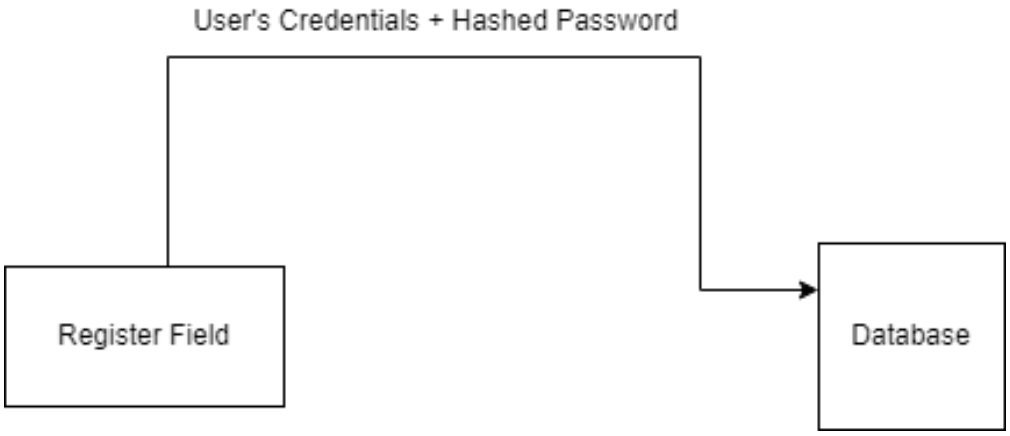

Fig.2. Registration flow

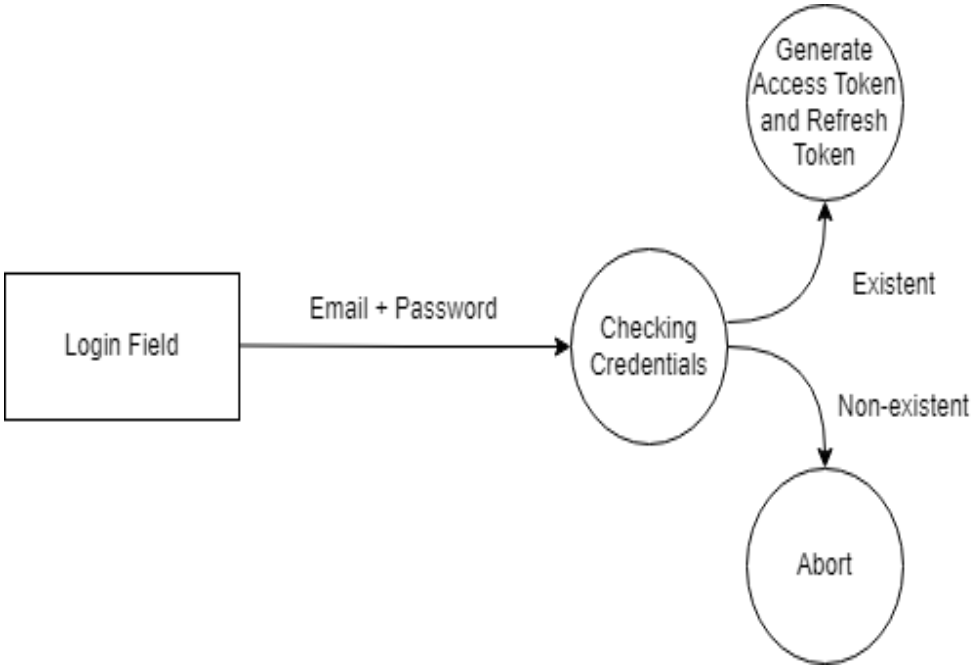

Fig.3. Login Flow

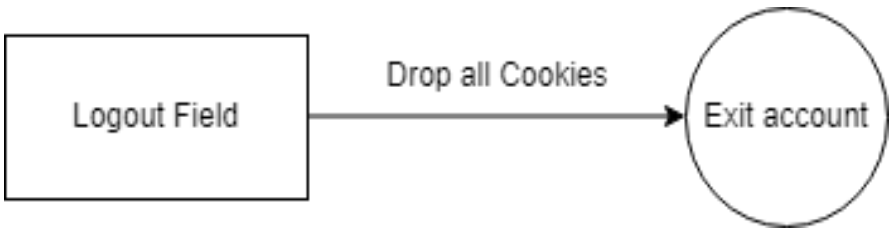

Fig.4. Logout Flow

Case 1

In this particular scenario where both the Access Token and the Refresh Token are confirmed as valid, a process called deserialization takes place specifically on the Access Token. Deserialization involves interpreting or converting the Access Token from its encoded or encrypted form into a readable format that contains information about the user and their access privileges.

Once the Access Token is deserialized, the user's provided credentials undergo validation. This step involves checking whether the user-provided credentials, such as a username and password, match the stored or expected credentials associated with the Access Token.

If the provided credentials are validated and deemed accurate, the user is granted access to the platform or system. Conversely, if the credentials fail to match or are found to be invalid during this validation process, access to the platform is denied, and the user's entry is rejected as a security measure. This method ensures that only users with authenticated and legitimate credentials are allowed access, while unauthorized or incorrect attempts are prevented from entering the platform.

Case 2

In this specific scenario, the Access Token has reached its expiration, rendering it invalid for further authentication purposes. However, it's important to note that while the Access Token has expired, the Refresh Token remains valid.

Given this situation, the system utilizes the Refresh Token as a mechanism to renew or regenerate the Access Token. This process involves using the Refresh Token to request a new Access Token from the authentication server without requiring the user to re-enter their credentials (such as username and password). The Refresh Token serves as a secure means to extend the user's authentication session without compromising security by sharing sensitive information repeatedly.

Once the system uses the Refresh Token to acquire a new Access Token, the authentication flow essentially mirrors the process from the previous scenario where both tokens were initially valid. The newly generated Access Token is then subjected to the same authentication flow: deserialization to extract user information and validation of credentials associated with the refreshed token.

Case 3

In this scenario, the Refresh Token has reached its expiration, making it invalid for generating new Access Tokens. However, despite the expiration of the Refresh Token, the Access Token remains valid and usable.

When the system encounters this situation, where the Refresh Token has expired but the Access Token is still valid, it continues the authentication flow without encountering any errors or disruptions. This is because the Access Token, which is the immediate credential used for authentication purposes, is still within its valid timeframe.

In essence, the authentication process replicates the flow observed in the first case. The system proceeds by deserializing the Access Token to extract user information and then validates the provided user credentials against this Access Token.

4. CONCLUSIONS

The authentication process within a system is a critical aspect ensuring security and user access. Understanding the various scenarios involving Access Tokens and Refresh Tokens provides a comprehensive view of how authentication flows can be managed in different situations.

Understanding these scenarios underscores the importance of a robust authentication system that leverages tokens effectively to ensure uninterrupted access for users while safeguarding against unauthorized entry. Implementing such practices enhances system security, user experience, and overall reliability

REFERENCES

[1]. Parth G., *Typescript microservices: build, deploy, and secure microservices using typescript combined with node*. Packt Publishing Ltd, 2018.

[2]. Thomas T.W., Tabassum M., Chu B., Lipford H., *Security during application development: an application security expert perspective*. Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). Association for Computing Machinery, New York, USA, pp.262, 1–12, 2018.

[3]. https://auth0.com/docs/secure/tokens/access-tokens

[4]. https://www.loginradius.com/blog/engineering/guest-post/what-are-refresh-tokens-and-when-to-use-them/