Final Project Report

Cybersecurity - ECE 455

Advisor: Daniel Gitzel

Yuri Hu, Faith Lin

Abstract

Investigating web security, this project delves into JWT-based authentication as a robust method for user identity validation. It meticulously examines potential vulnerabilities within the implemented paywall system and employs a Chrome extension to explore exploitable aspects. This dual approach is designed to comprehensively strengthen application security.

1 Problem Statement

This project aims to evaluate web server paywalls by analyzing user authentication mechanisms, JavaScript-based content blocking similar to paywalls, and JWT token-based session management. It seeks to uncover strengths and vulnerabilities within digital paywalls. Additionally, we explore existing bypass methods used in web-based paywalls to understand their real-world applications and efficacy in bypassing these barriers.

2 Related Work

https://github.com/iamadamdev/bypass-paywalls-chrome

The provided GitHub link is a Chrome extension designed to bypass various paywalls across different websites. It first checks if the current site matches a list of specific domains. If it doesn't match, it clears the browser's local storage to reset data associated with paywall tracking or user session limits.

Although the code varies by website, the main idea is that it generally focuses on detecting and removing elements that relate to paywalls. This includes identifying and eliminating elements linked to paywalls or removing specific elements and altering style attributes to unblock content. In some cases, it alters the website's URL to navigate around restrictions, especially where redirection is needed to access content. It also uses MutationObservers to monitor real-time changes in the DOM, targeting the addition of paywalls and ad elements, then removing them. This involves identifying the elements by their CSS selectors and then removing them through functions like removeDOMElement and removeClassesByPrefix. Additionally, JavaScript might control access to premium content by either checking the number of articles read or controlling access based on the subscription status stored in a cookie or server-side session. Therefore. disabling JavaScript in Chrome settings disables these mechanisms, bypassing the paywall.

3 Security analysis and/or threat model

Our project involves creating a pseudo-website with distinct user privileges, specifically a Subscriber and Nonsubscriber account and premium content. The system's security utilizes the generation of JSON Web Tokens (JWT) for session handling.

JWT tokens are generated server-side after successful user authentication and stored in the client's local storage. Although this method is efficient for stateless authentication, it is highly susceptible to Cross-Site Scripting (XSS) attacks. This vulnerability allows attackers to inject scripts and access sensitive information. In addition, if these tokens aren't properly secured (in terms of encryption and validation), and if they persist longer than necessary without adequate expiration policies, they become a target for attackers, allowing them to gain unauthorized access to user accounts and sensitive data.

Our user credentials are verified against a MongoDB database. But the security of this system largely depends on the security of the database and the strength of user passwords. This can be done through brute force attacks.

Our website is susceptible to Man-in-the-Middle (MITM) attacks if the communication between the client and server (including transmission of JWTs and user credentials) isn't secured or encrypted, compromising the integrity and confidentiality of data in transit.

4 technical analysis of solution/exploit

Servers typically refrain from retaining any data related to the JWTs they generate. Instead, each token stands alone, containing all necessary information internally. While this approach offers multiple benefits, it also introduces a core issue: servers lack insight into the original token contents or its initial signature. Consequently, inadequate signature verification could allow attackers to freely manipulate the token's remaining data. For instance, altering the username within the token could enable an attacker to mimic other authenticated users if the server identifies sessions using this data. Here, by modifying the hasSubscription field within the JWT payload, it provided an uncomplicated pathway for escalating privileges in terms of access control. From a non-subscribed user to a subscribed one.

To conduct a JWT token attack by bypassing the JWT authentication bypass via unverified signature, we created a Chrome extension to bypass our 'paywall' to elevate to privileges of a subscribed account. The code in the chrome extension begins by retrieving the JWT token named 'authToken' from the local storage of the browser. A standard JWT consists of three different parts – Header, Payload, and Signature. The base64url-encoded payload contains claims about the user. The JavaScript modifies the 'hasSubscription' field in the token's payload from 'false' to 'true'. Then the code reconstructs the token and encodes the modified payload, updating authToken in localStorage. This method relies on the client-side control over the JWT, as adversaries can easily use inspection tools to view JavaScript, CSS source code related to token handling. Furthermore, the exploit leverages the assumption that the server trusts the payload of the token without verifying its integrity or authenticity.

5 Methodology and Setup

Our project involved creating a pseudo website to explore the dynamics of digital paywalls and access privileges based on user subscriptions. We designed and developed a website that mimicked The New York Times and its subscription privileges. The site featured various (one) articles, which are behind a paywall, accessible only to subscribed users. This was created through a React app.

We then created two types of accounts – Subscriber and Nonsubscriber. Subscribers had full access to all content, while Nonsubscribers had limited access. Users could log into the website with their credentials verified through our server's authentication process.

Our server utilized MongoDB for database management, storing user credentials and subscription statuses. Upon successful login, the server generated a JWT for the user. This token was then stored in the browser's local storage. The JWT contained encoded data, including the username, which was used to manage session states and access privileges.

6 Conclusion and future work

In conclusion, our project examined the effectiveness and security of JWT-based authentication in digital paywalls, using a pseudo-website and a Chrome extension designed to test paywall bypass methods. We identified key vulnerabilities, particularly in token security and susceptibility to XSS attacks. In future works, our focus can shift towards having a verified JWT token. Perhaps including the development of more sophisticated token encryption methods, the implementation of two or multi-factor authentication for user verification, or the integration of defenses against common threats like XSS and MITM attacks.