MEASURING IMPROPER TOKEN INVALIDATION IN REAL-WORLD WEB LOGINS

KAZI FARHAT LAMISA

A THESIS

IN

THE DEPARTMENT OF

CONCORDIA INSTITUTE FOR INFORMATION SYSTEMS ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

INFORMATION SYSTEMS SECURITY

AT CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

NOVEMBER 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Kazi Farhat Lamisa**

Entitled: **Measuring Improper Token Invalidation in Real-World Web Logins**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

Dr. Walter Lucia _____ Chair

Dr. Mohammad Mannan _____ Supervisor

Dr. Amr Youssef _____ Supervisor

Dr. Walter Lucia _____ Examiner

Dr. Chadi Assi _____ Examiner

Approved by _____

Dr. Chun Wang, Director
Concordia Institute for Information Systems Engineering

_____ 2024

_____

Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Measuring Improper Token Invalidation in Real-World Web Logins

Kazi Farhat Lamisa

In this thesis, we examine token invalidation flaws in real-world web applications. While previous research has focused on cookie invalidation upon user logout, we investigate the issue by examining token-based authorization in various contexts (e.g., user logout, user verification, password recovery). Specifically, we focus on JSON Web Tokens (JWTs) to look for invalidation flaws, as JWTs are stateless and require web servers to implement a separate invalidation mechanism for instant invalidation. To audit invalidation flaws, we conduct a large-scale study on the top 1 million websites from Google's Chrome User Experience Report (CrUX). We develop an automated tool, *LoginPlus* that handles tasks such as user registration, login and password recovery. Utilizing *LoginPlus*, first we identify JWTs used to fetch resources from a web server while the user is authenticated and determine whether these tokens are invalidated upon user logout. *LoginPlus* also handles all emails sent during account creation and password recovery, identifying whether tokens included in URLs sent to users for email verification or password recovery are invalidated after a single use, checking if these URLs remain reusable over time. Finally, we evaluate whether the websites invalidate all previously active sessions after a user resets their password through the password recovery mechanism (e.g., 'forgot password').

Our analysis provides a comprehensive overview of the current state of invalidation issues in token-based authorization schemes across real-world websites and reveals several significant findings regarding token invalidation mechanisms in web authorization. Firstly,

85% of the websites using JWTs for authorization do not implement an explicit invalidation mechanism upon user logout. Additionally, 2.67% of websites log users in into the system via verification URLs directly, and 48% of these sites do not invalidate the tokens in the verification URLs even after 24 hours. Furthermore, 13.8% of websites that allow password recovery through email URLs do not invalidate the tokens after they have been used once. Lastly, 54% of the websites where we successfully reset a password do not invalidate previously active sessions.

Acknowledgments

I would like to begin by expressing my heartfelt thanks to God for providing me with the strength and perseverance to complete this journey. I am incredibly grateful to my supervisors, Dr. Mohammad Mannan and Dr. Amr Youssef, for their invaluable guidance, support, and patience throughout the course of my research. Their thoughtful feedback and encouragement were instrumental in shaping this work, and their financial support made it possible. I also want to extend my appreciation to my colleagues in the Madiba Security Research Group for their support and insightful discussions, which greatly enhanced my academic experience. Lastly, I owe a special thanks to my family for their unwavering emotional and financial support, which has been the foundation of my success.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Cookies have long been a fundamental component of web authentication and authorization, enabling websites to store user data and manage sessions. In many early works, researchers have explored privacy and security implication of cookies in various respects such as user tracking and user consent [16, 37, 40, 14]. On the other hand, token-based authentication and authorization is a relatively recent concept that offers several improvements over traditional cookie-based methods in terms of scalability, security, and flexibility. Unlike cookies, tokens are stateless and do not require server-side sessions, which significantly enhances scalability by reducing server load and simplifying horizontal scaling. In terms of security, tokens such as JSON Web Token (JWT), can contain user permissions and roles as claims and can be encrypted and signed to maintain their integrity and confidentiality [41]. This enhances security compared to cookies, which are susceptible to various attacks such as Cross-Site Request Forgery (CSRF) [15, 42], and often lack such robust security measures. Despite these advantages, token-based authentication needs to be further investigated in order to learn its implication in real world websites. The adoption of tokens in web authentication and authorization represents a major evolution in web security

practices, necessitating in-depth analysis and exploration to address potential vulnerabilities and ensure robust protection for users and applications.

1.2 Motivation

One of the most crucial tasks for web developers is invalidating user sessions when necessary, such as when a user logs out. This involves invalidating all cookies or tokens associated with an authenticated session. Previous studies, such as those by Calzavara et al. [12] and Mundada et al. [36] have focused on session cookies and explored whether web developers properly invalidate them upon logout. However, no studies have focused on invalidating tokens when necessary. Additionally, there are other scenarios where web servers might need to invalidate tokens. For example, after resetting a password through an out-of-band verifier such as email, where a URL containing a token was sent, the token included in the URL should be invalidated to ensure that no one else can reuse the URL to reset the password again. Throughout this thesis we refer to this issue as improper invalidation problem.

Although previous studies (e.g., [12, 36]) have investigated cookie invalidation after user logout by checking the browser state, our study examines different token-based authorization flows, both on browser sessions and individual HTTP requests. For example, our experiment on JWT invalidation after user logout focuses on individual HTTP requests that use JWTs to authorize various resources, rather than user sessions. In contrast, other authorization flows in our study, such as user verification through email, password recovery, and session invalidation after a password reset, involve assessing whether the browser still indicates that the user is logged in after these actions.

1.3 Problem Statement

In our thesis, we devise an automated analysis tool, *LoginPlus* to measure improper invalidation problems in multiple token-based authorization flows e.g., JWT invalidation after user logout, URL token invalidation after user verification and password recovery etc. We follow the OWASP Application Security Verification Standards (ASVS) [3] to determine different authorization flows and design test cases accordingly. Among the various recommendations by ASVS on secure session management and credential recovery, we focused on those mentioned in Appendix A.1 for our study and transform these recommendations into improper invalidation problems as described below:

The first invalidation problem we explore is the invalidation of JWTs after user logout. JWTs are stateless and thus not required to be stored on the server. There is a trade-off between server-side management and invalidating JWTs upon logout. While using JWTs simplifies server-side session management, developers must either need to depend on automatic expiration of JWTs or implement explicit methods to invalidate JWTs upon user logout. We detail JWT invalidation in Chapter 2. In our first experiment, we seek to understand whether web developers depend on automatic expiration of JWT or adopt any explicit method for instant invalidation. Our tool, *LoginPlus* captures HTTP requests containing JWTs when a user is logged in and performs invalidation test after user is logged out in order to determine whether there is any explicit method at server side for invalidating JWTs.

ASVS advises that out-of-band verifiers such as emails should expire authentication codes or tokens within 10 minutes to limit the window of vulnerability. These tokens should be usable only once to prevent reuse in case of interception. Additionally, during password credential recovery, the current password should never be revealed in any form to maintain its confidentiality and the communication between the out-of-band authenticator and verifier should always occur over a secure channel to protect the integrity and

confidentiality of the authentication process. Our tool, *LoginPlus*, verifies user accounts through URLs sent in emails and checks whether these URLs remain usable after being used once. Additionally, it determines whether these URLs can grant access directly to user accounts. *LoginPlus* also performs password recovery using URLs sent in emails and assesses their validity. In both cases, we perform the validity check once instantly, then after 24 hours and 7 days. Furthermore, we check if TLS was used when sending these emails and verify whether the password appears in plaintext within the emails. Finally, we take into consideration of the session termination after finishing the password recovery process. These comprehensive approaches ensure that the integrity and confidentiality of the user verification and password recovery process is protected.

1.4 Contributions

Summary of our contributions and notable findings.

(1) We design and develop a completely automated black-box tool *LoginPlus* to audit flaws in token-based authorization flows in websites in the wild. *LoginPlus* consists of multiple modules that provide support such as registration, login and logout, monitoring HTTP traffic and password recovery through email. It is also capable of performing differential analysis to capture improper invalidation problems in token-based authorization flows.

(2) We ran our experiment on Google's Chrome User Experience Report (CrUX) [17] top 1M websites. We are able to submit a registration form for 44,012 websites. We successfully login to 26,551 websites and logout from 5,642 websites. We automatically verify 6,392 websites through user account verification URLs and successfully reset password using password recovery method for 1,248 websites.

(3) We show evidence of improper JWT invalidation by completely automated analysis indicating that developers do not tend to implement explicit methods to invalidate JWTs upon user logout. According to our automated test results, 446 out of 520 websites (85%) tested for JWT invalidation do not invalidate JWTs included in requests after user logout.

(4) We find that some websites give access to user accounts through the verification URLs directly. We detect 171/6,392 such websites and for 83 of them the verification URLs stay valid even after 24 hours meaning anyone having the URLs can gain full or partial access to the user account. Beside verification URLs, we also evaluate the validity of password recovery URLs and find that for 173/1,248 websites the password recovery URLs stay usable even after resetting password using them before.

(5) Our study reveals that 674/1,248 (54%) of the websites do not terminate all the existing sessions after a user resets password through a password recovery method. This case poses threat specially when a user loses device or is logged in a device without access to it.

(6) We explore various ways the tokens included in verification URLs and password reset URLs can be leaked. We found that 57 websites do not use TLS while sending a verification or password recovery email. 146/6,392 (2.2%) and 89/1,248 (7.1%) websites serve URLs over HTTP instead of HTTPS for verification and password recovery respectively. Moreover, 339/6,392 (5.3%) and 42/1,248 (3.3%) websites store the unique token included in the verification and password reset URLs in local storage correspondingly.

1.5 Thesis Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 presents the foundational information on token-based authorization, reviews relevant literature, and develops the research questions derived from this background knowledge. Chapters 3 and 4 provide a comprehensive overview of our system, detailing the design and implementation in the former, and presenting the results in the latter. Finally, in Chapter 5, we summarize the key findings, address the limitations of the study, offer practical recommendations, and outline potential directions for future research.

Chapter 2

Background

2.1 JSON Web Token

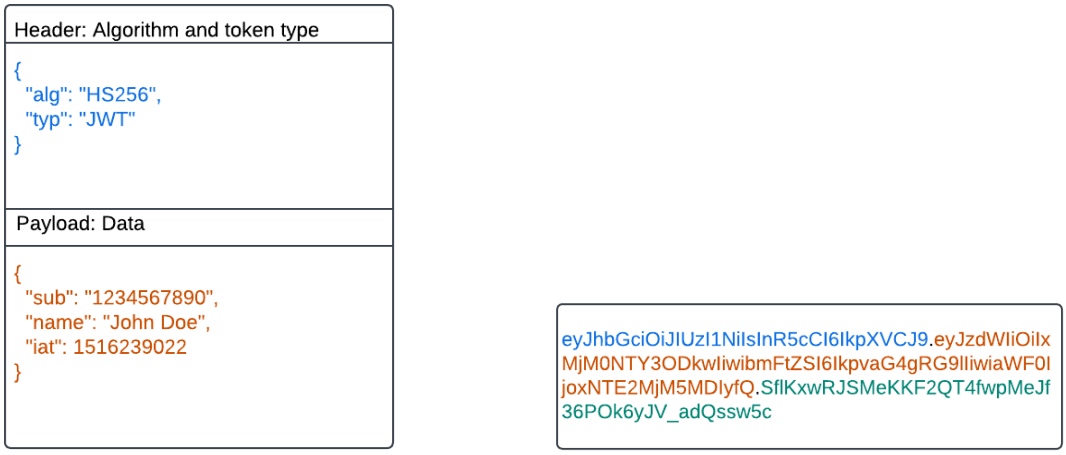
JSON Web Token is an internet standard optionally containing some data and signature that is supposed to be exchanged between a server and a client. According to RFC 7519 [41], it has three parts: *header, payload* and *signature*.

(1) Header. Header contains the name of the cryptographic algorithm that has been used to sign the token using an attribute called 'alg'. Typically it is HMAC or any public key algorithm. Along with the 'alg' attribute, there can also be 'kid' attribute which refers to a key. It is not the secret key rather it points to a database entry or any URL end-point where the receiver of the token can look for the secret. The header is base64 encoded.

(2) Payload. Payload is the part containing relevant data like email or username, issuer name, expiration time, time of issuing etc. These are usually called 'claims' and developers can use them to store necessary information [41]. The payload is also base64 encoded.

(3) Signature. It is the last part of the token. The base64 encoded header and payload

are appended together. It is then signed digitally or made integrity protected with a Message Authentication Code (MAC) and/or encrypted. All of the three parts are then added using dots(.). This part is verified by the server to validate the integrity of the information appended in the payload.



(a) Decoded form **(b)** Encoded form

Figure 2.1: Structure of a JSON Web Token (JWT)

Among all the claims that can be included inside the payload, we are interested in 'exp' which defines the expiration date on or after which the token must not be accepted for any processing. In general, a token should not work on or after its expiration time. To invalidate a JWT before its expiry time, developers need to implement an explicit mechanism at server-side. Below we describe a few of the existing invalidation mechanisms.

Short-Lived Tokens. The easiest solution to the invalidation problem is to make the expiry date shorter so that it does not remain valid after that short period of time. However, if a user leaves the system while the token is still valid, even if the server stops issuing new tokens, the old token remains valid for a short period of time. Although there is no universally fixed expiration time for JWTs, the term 'short-lived' varies based on factors such as the token renewal mechanism, server resources, the sensitivity of the information

being accessed, and the usability requirements of the application.

Blacklisting. In the cases where automatic expiration does not apply, meaning server needs to invalidate the token instantaneously, methods like blacklisting can be adopted. The server maintains a blacklist using in-memory cache, database or any other mechanism depending on the scale of the application. The list gets updated whenever a user logs out or terminates an active session. Developers can trim the list by removing older tokens that have naturally expired as those tokens will never get accepted. However, maintaining a blacklist undermines the benefit of using JWT. As JWT is stateless, the benefit of using it is that the server do not need to store it to verify any user activity. It is digitally signed and so when a token arrives, the server is just supposed to verify the signature and the expiry time to accept it. Because of this trade-off, many developers might not consider blacklisting mechanism for invalidating the token.

Changing JWT Secret. For achieving immediate token invalidation, server can change the signing secret of the JWT. In that case, it will not accept any token with the invalidated secret. However, in this mechanism whenever a server invalidates the token by changing the secret, all the JWTs will be invalidated at the same time across the system. This is because server usually has a single secret for issuing JWTs. On the other hand, if the server wants to just invalidate only tokens for a particular user who is leaving the system, it will have to maintain a list of secrets separately for each users. In that case, it undermines the purpose of using the JWT and adds an overhead to resource management.

2.2 Research Questions

2.2.1 RQ1: JWT Invalidation After User Logout

The principal gain of using a JWT is that the web servers using JWT for authorization purposes do not need to store the tokens. Whenever a request is made to the server using a

JWT, the server verifies the digital signature to ensure its authenticity and integrity. A JWT is expected to be invalid when its corresponding expiration time has passed. The expiration time should be checked by a web server every time a token arrives. Since the token is digitally signed or integrity protected, the expiration time cannot be forged by anyone.

The problem occurs when the server needs to invalidate a JWT upon user leaving the system before the token hits expiry time. In general, with the user leaving the system, e.g., user logout, the web servers need to invalidate all the cookies and tokens. Nevertheless, it is possible that all the session cookies have been invalidated except the JWT. For example, consider a request where a JWT is being used to fetch some personal information and the JWT is included in the Authorization header of the request. In this case, although the session cookies might be invalidated and the website shows a logged-out behavior, this request containing a JWT can still return a valid response if the web server does not invalidate the JWT too.

We seek to provide an answer to the question of how not invalidating JWT affects the system, regardless of whether session cookies are invalidated. Due to the statelessness of JWT, developers need to implement a mechanism to invalidate the JWT explicitly. Whenever there is no such mechanism implemented, the server is basically depending on the expiration time of the token. As a result, a JWT that is not associated with session management but used to fetch some kind of resource within a request might still be valid after user logout. If someone other than the user gets the token, they will still be able to fetch some resources from the server without having access to the user sessions.

2.2.2 RQ2: URL Invalidation After User Verification and Password Recovery

We explore whether the tokens sent to users through account verification and password reset URLs are properly invalidated after a single use. This addresses an OWASP ASVS

standard that should be maintained in a web application (see Appendix A.1). We consider any kind of token in this case, whether it's a JWT or any random string chosen by the developer, to account for scenarios where the token is not a JWT.

Usually, these tokens included in the verification and password recovery URLs should not serve their purpose once they have been used. Although some websites may set an expiry time for the URLs, allowing users to use the URLs more than once within the expiry time, it should be as short as possible. This is because if those tokens reach any malicious actors, they can reset the password using the URLs without the user being aware of it. The verification URLs are especially vulnerable when websites log in users directly through the URLs. In this case, anyone having the URL or the token included in the URL will have access to the user account by just clicking on the URL or by issuing a request containing the token to the endpoint responsible for account verification. Note that in the second scenario, the verification URLs might not always give access to a user account entirely; rather, they might show some personal information of the user.

In this study, we aim to determine the prevalence of the insecure practice of allowing URLs to remain reusable in real-world applications.

2.2.3 RQ3: Session Invalidation After Password Recovery

According to OWASP ASVS, after completing the password recovery process, there should be an option for the users to terminate all the active sessions. That is through this user action, all the session cookies and tokens should be invalidated and the user should be logged out of all the sessions. However, designing such an experiment is not trivial. Thus, instead of checking whether a website is providing options to the users, we check whether all the previously active sessions are terminated by the web server without users taking any action.

We argue that users only reset the password using some password recovery mechanism

(i.e., ‘forgot password’) when they are not in possession of any authenticated session. For example, when a user loses the device she was logged into and try to recover the account by resetting her password, the user would expect that the active session in her lost device gets terminated. If the session is not terminated, the malicious actor holding the device will be able to access the user account without the knowledge of the user.

Thus, our objective is to determine how well web applications adhere to the OWASP ASVS recommendation regarding the termination of all previously active sessions after completing the password recovery process, particularly when users initiate password reset without an authenticated session.

2.2.4 RQ4: Cookie Invalidation After User Logout

The authentication and session cookies are the key elements that keep a user logged in into a website and allow the server to identify a user whenever a request is issued from the browser to the web server. The expected behaviour upon user logout is that all the cookies associated with identifying and authenticating a user should be instantly invalidated regardless their expiry time. However, previous studies [12, 36] show that sometimes the session cookies are not properly invalidated at server side upon user logout. In other words, the web server still accepts the cookies of a logged out session. In our study, we re-explore the occurrence of improper session cookie invalidation where we check if the session cookies are invalidated properly after logout by visiting the login URLs of the websites with the collected cookies from previously valid sessions.

2.3 Related Work

2.3.1 JSON Web Token

In terms of session invalidation after user logout, we focus on JWT invalidation instead of session cookies. There have been a few academic studies on the invalidation methods for JWT. Janoky et al. [28] discussed limitations of several common revocation methods for JWT such as short-lived tokens, blacklisting, changing secret etc. They suggested a new method for invalidating JWT after user logout by arranging users as groups (e.g., based on hashing usernames); by changing the JWT secret only for the group instead of all the users. In a follow up study, Janoky et al. [29] provided evidence of the effectiveness of their previously suggested method based on several metrics such as architectural complexity, invalidation latency, scalability etc. In other studies, researchers evaluated the performance of algorithms that are used for signing a JWT [39, 8, 51]. The most recent study by Xu et al. [51] found that 65.92% of analyzed JWT applications contain cryptographic vulnerabilities due to poor key management practices highlighting significant security risks stemming from incorrect cryptographic implementations in JWT usage. Furthermore, many studies proposed authentication schemes using JWT for different platforms (e.g., IoT, Web) [7, 25, 6, 24, 19]. In contrast to these studies, we conduct a measurement on the overall usage of JWT and the magnitude of improper invalidation of JWT in websites.

2.3.2 Large-Scale Security and Privacy Analysis for Web

Most of the large scale studies have been done on either the landing page or the authentication pages (e.g., login and registration page) since accessing the authentication-only area is subjected to the complex task of user account creation. The studies cover various privacy and security aspects such as credential leakage, browser fingerprinting and the overall security of the pages [49, 46, 45, 13, 30]. A few of the recent studies have taken the

initiative of automating the process of either registration or login or both in order to measure security vulnerabilities on websites in the wild [21, 43, 18, 12, 32]. The privacy and security aspects have a broad range; including analyzing implementation decisions made throughout the login process, security of authentication cookies, session invalidation upon logout, security compliance of SSO providers to name a few.

2.3.3 Single Sign-On (SSO) with Automation

Ghasemisharif et al. [22] conducted an empirical analysis on SSO systems identifying common security flaws in popular SSO providers and emphasized on improved security measures. In a later work, Ghasemisharif et al. [21] developed an approach to fully automating black-box auditing framework for detecting violations and non-compliance of secure practices in Facebook's Relying Parties (RP). Ardi et al. [9] explored the prevalence of SSO on websites and examined whether it is feasible to use a few accounts to sign into a large number of websites. Their study shows that more than 50% of the top 10k websites are accessible through third-party SSO providers. While significant research has focused on the security and privacy implications of public Single Sign-On (SSO) services and protocols, with several frameworks developed to assess these concerns by automating the SSO login process [34, 52, 50, 26, 38], fewer studies have taken the approach of analyzing security and privacy issues in traditional registration and login mechanisms. This gap highlights the need for more research on vulnerabilities in conventional login processes, where users manually register and authenticate without the use of SSO.

2.3.4 Third-party Cookies and Scripts

Drakonakis et al. [18] detect authentication and authorization flaws revolving around the handling of cookies and stemming from the incorrect, incomplete, or non-existent deployment of appropriate security mechanisms. Chen et al. [14] explore the growing practice

of using first-party cookies set by third-party JavaScript for tracking, even when third-party cookies are blocked. The study found that 97.72% of the Alexa top 10K websites use first-party cookies for tracking, with 57.66% leaking unique user identifiers to third parties. Roomi et al. [43] measured the presence of third-party scripts in login page of websites. Sprecher et al. [47] reveals that third-party JavaScript often accesses user-supplied PII without adequate control, leading to privacy risks. After analyzing 100,000 websites, the researchers highlight the complexity of third-party script interactions with PII and propose new privacy solutions to address these gaps in the web execution model. Another large scale study was done in [48] in the context of web3 based decentralized applications and wallets for blockchain. They identified third-party scripts running in the context of web pages, implicitly or explicitly ping for wallet objects. They also identify third-party websites that collect wallet addresses through decentralized apps and wallet extensions.

2.3.5 Security of Email Communication

In prior studies, email headers have been used to detect spam [33, 11, 31]. We analyze the headers of the received emails in order to determine whether proper TLS connection was used or not. In other words, we identify which emails show the alert that the sender did not encrypt the message.

2.3.6 Comparison to Closely Related Works

The automated studies mentioned previously ([49, 46, 45, 13]) exhibit some differences in the approaches that were taken for account creation and login. Ghasemisharif et al. [21] developed a login approach based on Facebook SSO. They were able to login to 1.9k/100k websites using Facebook SSO. The studies done by Roomi et al. [43] and Kubicek et al. [32] used traditional registration process with email, username and password to complete registration while Drakonakis et al. [18] used both traditional process

and Facebook SSO. However, Kubicek et al. [32] did not go through the login process as they only analyze received emails and GDPR compliance of user consent during registration. On the other hand, Drakonakis et al. [18] and Roomi et al. [43] successfully registered to 25,242/1.6M (2,066 using Facebook SSO) and 45k/1M websites respectively. Calzavara et al. [12] used a crowd-sourced database called BugMeNot [4] and automated only the login process. They successfully logged into 6,766/53.6k websites. In our study, we use the dataset containing only registration and login URLs obtained from Roomi et al. [43]. However, in terms of determining successful registration and login we take a different approach than theirs. In their study they first filled up the registration form and submit. To verify successful registration they used decision tree classifiers operating on page features upon form submission. By following this approach they reported 45k successful registration. On the contrary, we do not evaluate the page after form submission to identify successful registration; rather we report successful account creation only if we can actually login by using the credentials that we used during form submission. Thus we report form submission for 44,012 websites and successful login for 26,551 websites. Our approach is particularly well-suited to our study because all our test cases require successful account creation.

With respect to security analyses, the intersection between these studies and our study is shown in Table 2.1. Firstly, the overlap between our study and Calzavara et al. [12] is that we re-evaluate the state of session cookie invalidation after user logout, identify cookies missing security attributes ('Secure' and 'HttpOnly') in general and measure PII leakage through vulnerable cookies and local storage. Drakonakis et al. [18] centered their study around identifying vulnerable authentication cookies and show how the vulnerable cookies can leak user private data. Since our study focuses on tokens instead of cookies, we do not go through the computationally expensive process of identifying cookies that are associated with authenticated sessions. Roomi et al. [43] identified vulnerable implementation decisions (e.g., credentials transmission through insecure channel, accepting typo-tolerant

password etc.) made by developers through out the login process. Our study overlaps with the part of their study where they submit a password reset request using 'forgot password' in order to see if a website sends a plaintext password in email. However, they do not attempt to reset the password as we do. Lastly, although all the studies that create user account conduct user account verification by email ([18, 32, 43]), none of them verify whether a user is directly logged in through verification URLs.

Table 2.1: A summary of the test cases conducted in our study and their overlap with closely related works. A '✓' indicates that the test is also covered in the respective study, while a '-' denotes that the test is not covered in the study.

Category	[18]	[12]	[43]	**Our work**
Publication year	2020	2021	2023	
Registration Stats				
- registration	automated	collected [1]	automated	automated
- # of sites approached	1.6M	53.6k	1M	1M
- top list used	Alexa	Tranco	CrUX	CrUX
- CAPTCHA	-	-	✓	✓
Login and Logout Stats				
- # of successful login	25k	6k	45k	26,551
- logout	-	✓	-	✓
- # of sites logged out	-	3.4K	-	5,642
JWT invalidation after user logout				
- #of websites JWT found	-	-	-	6% (1,616/26,551)
- #of websites found with invalidation issue	-	-	-	85% (446/520)
Password recovery				
- request password reset	-	-	✓	✓
- #of websites send a plaintext password in email	-	-	570/44,703	174/44,012
- reset password	-	-	-	✓
- successful password reset	-	-	-	1,248
- # of websites URL token invalidation issue found	-	-	-	173/1,248
Email verification				
- verify user through email	✓	-	✓	✓
- check access given to user account through verification URL	-	-	-	✓
- # of websites give access to user account through verification URL	-	-	-	171/6,392
- # of websites for which verification URLs remain valid after 24 hours	-	-	-	83/171
Session invalidation after password reset				
- # of websites not terminating active sessions after password recovery	-	-	-	54% (674/1,248)
Cookies				
- invalidation after user logout	-	✓	-	✓
- # of websites found with invalidation issue	-	18% (604/3.1k)	-	9% (513/5.6k)
- cookie attributes ('Secure' and 'HttpOnly')	✓	✓	-	✓
- identify authentication cookies	✓	✓	-	-
Privacy leakage through				
- vulnerable cookies (missing 'Secure' and/or 'HttpOnly' attributes)	✓	✓	-	✓
- lack of TLS in email communication	-	-	-	✓
- plaintext password in emails	-	-	✓	✓

In summary, there have been large-scale automated studies that evaluate user session management, cookies, single-sign-on ecosystem. There are also works that use JWT to

[1] In the study by Calzavara et al. [12], BugMeNot [4], a crowd-sourced database of shared credentials for websites, was utilized instead of manually or automatically creating user accounts.

implement robust authentication mechanisms. Moreover, some studies have pointed out the improper invalidation problem with JWT for authentication and authorization and suggested novel methods for invalidating JWT. However, none of the studies measure the prevalence of JWT in the wild. More importantly, how widespread is the practice of improper invalidation of JWT remains unexplored. In this work, we explore the prevalence of JWT in the wild and measure the improper invalidation problem. We not only conduct token invalidation test after logout but also explore other possible flows such as user account verification and password recovery where tokens included in the URLs need to be invalidated after using once. Through our measurement study, we assess privacy leakage through improperly invalidated JWTs, insecure emails, vulnerable cookies

2.4 Threat Model

2.4.1 On-Path Network Attacker

This type of attacker is capable of only eavesdropping e.g., inspect unencrypted data by intercepting on-path traffic. They do not modify anything during eavesdropping. Whenever a connection is over HTTP instead of HTTPS, such an attacker can read everything on the network. In our case, if a JWT is transported through a cookie from the web server to the client through HTTP network without the 'Secure' flag, an eavesdropper can get the token. Note that although the JWT tokens might be transported through a cookie, the token is still evaluated as a JWT. Since the data exchanged between a web server and a client over HTTP is not encrypted, JWTs transported through the response body is also visible to the attacker. Regardless user logout, the attacker can then use that token to fetch data. The attacker becomes more powerful if the token is not properly invalidated even after user logs out of the account and has a longer lifetime. In the context of email verification and resetting password, if the websites do not use HTTPS, any on-path network attacker can

obtain the verification and password reset URLs whenever the users perform the respective actions.

2.4.2 Active Web Attacker

This kind of attacker is capable of actively launching an attack by executing some JavaScript code. For example, if a web application has a XSS vulnerability, an attacker can execute a malicious script that might steal cookies or session tokens within client's browser. In the case where a JWT is transported through a cookie without the 'HttpOnly' flag, it becomes accessible from the browser of a user. Due to the improper invalidation problem for JWT and longer expiry time, users are more vulnerable if a JWT gets stolen by an active attacker. Moreover, the JWT can be stored in local storage or session storage. In both cases, the tokens are accessible through JavaScript. Any JavaScript controlled by a malicious party running in the application's context can acquire the token. Likewise, during email verification and password reset a unique token is issued to the user. This token can be stored in local storage or session storage which makes them accessible through JavaScript.

2.4.3 Physical Access Attacker

This type of attacker has physical access to the user's device. For example, if a user loses their device or uses a shared public device and forgets to log out and clear all cookies, the attacker can exploit this access. In such cases, when the legitimate user realizes the need to change their password to log out of the session on the stolen or shared device, they may face trouble if the website lacks a mechanism to automatically log out the user after a password reset.

The scope of our study is to verify the validity of the cookies and tokens in multiple scenarios described in this chapter. Although we assume attackers who can get hold of the cookies and tokens, our goal is not to identify which cookies and tokens are vulnerable to

being stolen, rather we focus on discovering problems stemming from server-side improper invalidation of the cookies and tokens.

2.5 Ethical Consideration

Since our work is a large scale study on websites requiring user account creation, we have considered several ethical issues. First, we create no more than one account per website as per the requirement of our study. We use SAAT [21] during account creation to submit registration forms and verify login. The tool simulates user behaviour during navigation through the websites. To solve CAPTCHA we use an AI driven service AZCaptcha [10] that was also used in a prior study [43]. For login and navigating through the user account we use the same waiting time of 30 seconds between navigations as SAAT. During the analyses we performed, we only send a particular request to the server three times (instantly after logout, 24 hours after logout and 7 days after logout). Prior works [43] have triggered password recovery method but do not reset the password. In our work we submit a new password and verify the attempt by logging in using the new password. We do this experiment using our user accounts and Gmail accounts that we have access to. In Chapter 4.2.1, we present some illustrative examples on JWT invalidation where we conduct several experiments requiring to change user's personal information, fetching PII etc. We note that we conduct this experiments using our own test accounts.

Chapter 3

System Design and Implementation

3.1 Overview

In this chapter, we explore the system design and implementation of our automated registration, login, and post-login analysis tool, *LoginPlus*. *LoginPlus* comprises four modules:

(1) Registration. This part of our framework is used to create user accounts in websites. We use SAAT [21] to submit a registration form. Note that we do not use the CAPTCHA solving mechanism of SAAT. Instead, we incorporate an AI driven CAPTCHA service into our registration pipeline described in Section 3.2.2.

(2) EmailHandler. The EmailHandler is responsible for verifying received URLs during account creation and managing reset URLs during password recovery.

(3) Login and Logout. We use this module to log users into websites and to log them out. Subsequently, we perform experiments to test for improper token invalidation. For user login verification, we utilize the login verification mechanism of SAAT.

(4) Password Recovery. In this module, we explore password reset pages and submit

requests to reset passwords. The module then communicates with the EmailHandler in order to get the URLs that have been received through emails and attempts to reset passwords. It also performs session invalidation experiment after resetting passwords.

Figure 3.1 provides a high-level overview of the steps involved in the LoginPlus methodology, illustrating the system design and interaction between different modules.

3.2 URL Discovery, Registration, Login and Logout

Before analyzing improper invalidation problems, we first need to register and log in to confirm account creation. We then attempt to log out, as some of our analysis require the logout action. We describe the methodology step-by-step in the following sections.

3.2.1 Detecting Registration and Login Forms

We crawl the websites which have a missing entry or if any of the URLs in the dataset are no longer valid. We do the crawling in a Breadth First Search (BFS) manner with maximum depth = 2 by following the data collection method described in [18]. First we take the landing page of a website (depth = zero) and search for HTML forms for authentication (login or registration). We save all the URLs found on the landing page that point to a possible authentication page (depth = 1). We then evaluate each of the candidate pages and save the possible URLs found on them. Afterwards, we evaluate the saved URLs (depth = 2) and stop our search there. To avoid excessive crawling of one website, we keep the number of saved links limited to 15 in each of the steps. In the worst case, we send 30 requests to a website. In the case where we can not find any authentication page for a website, we search on the internet for the registration and login URLs using several search engines (startpage, bing, duckduckgo). Note that as we used an already filtered dataset, we

only crawled 384,847 websites out of 1M that allow registration and login.

3.2.2 Registration

After discovering registration and login pages, we can now proceed to test account creation. We use the registration module of SAAT [21] to submit a registration form with several modifications. We describe them in the following sections.

Filling Out Forms. We identify and fill out the fields for username, email, password, date of birth, country, phone number on the registration page from HTML input elements using SAAT [21]. In this tool, 'input' fields for personal information are identified based on keywords. Then the forms are filled out with dummy data to trigger dynamically loaded elements. After everything is loaded, filling out with actual data (e.g., the actual email, username, password etc.) is started. A page navigation threshold of 4 and invalid trial of 2 is maintained throughout the process. We do not make any major change to this method except for adding new keywords from our manual inspection.

Form Submission. We follow the logic implemented in SAAT for submitting forms with a slight modification in the logic. In SAAT, keyword based search is used for the 'input' and 'button' elements with type = 'submit' for finding the submit button. If no match is found, it is considered that there is no way to submit the form. However, we try to submit the form even if no match was found and there is only one 'input' or one 'button' with type = 'submit' in the form. This method is useful when the keywords are in different language other than English and the keyword based search can not identify the element for submission. We do not attempt to login right away after submitting the form to verify if an account was created. We separate registration and login for three main reasons: *1.* If account creation requires email verification, we need to wait for the email to be verified, and the arrival time of the verification email is unpredictable. *2.* We prefer to log in and conduct post-login analysis from a different IP address to avoid detection during crawling.

3. Capturing all HTTP traffic after a user logs in adds some performance overhead to our pipeline. Therefore, we first submit registration forms and leave the verification of successful account creation by logging in for the next stage. However, sometimes after a registration form submission, the user might be logged in automatically into the account. In such scenarios, we verify the user's logged-in status after submitting a form and proceed to log them out if necessary.

CAPTCHAs. We have found CAPTCHAs in 37% registration pages while attempting form submission. We decided to solve CAPTCHAs using an AI driven service called AZ-Captcha [10]. The tool SAAT [21] only considers solving Google's ReCaptcha. We consider image based captcha and Hcaptcha along with ReCaptcha. To trigger the CAPTCHA settings of a website, we first fill out the fields with dummy data before attempting actual submission. We then extract relevant information for solving CAPTCHA and send it to AZCaptcha [10]. We also save the response fields for the CAPTCHAs and add the answer returned by AZCaptcha with form submission.

Handling Account Verification with Gmail API. For creating test accounts we use Gmail accounts as our email.We control accessing these accounts through the Gmail API. Unlike SAAT, we keep the verification process running separately instead of incorporating email verification into the account creation pipeline. For each successful registration form submission that contains email, the registration module stores relevant information, typically the website name. The email verification module fetches the stored information and checks whether any verification email was sent from the website. It accesses a particular Gmail account every 10 minutes and checks for unread verification emails against a list of websites that it has fetched from the registration module. Once verification is done for a website, it is removed from the list of websites stored to get verified.

In SAAT [21], the registration process of an account waits for 20 seconds to get verified each time it fills out a form regardless the outcome of submission attempt. We observe that

most of the time the verification email is not sent within 20 seconds. We also observe that typically the expiry time of a verification URL is more than 10 minutes. Therefore, we opted to access the Gmail accounts within a 10-minute interval. Our design removes the overhead of waiting for verification email while registering on a particular website, optimizes the usage of Gmail API and solves the problem of verifying accounts in the case where an email arrives after 20 seconds.

3.2.3 Login and Logout

After successful form submissions, we start our login procedure. This module also handles logout and performs automated analysis for JWT and cookie invalidation.

Login. For all the websites for which a registration form is submitted, we attempt login by entering username/email and password. Sometimes CAPTCHA is present in a login page. We follow the same process as registration for solving CAPTCHAs. We first fill the login form with dummy data to make CAPTCHA visible. Note that we do not submit the form with dummy data. We then collect the CAPTCHA information and send it to AZCaptcha. We fill out forms with CAPTCHA only when AZCaptcha returns an answer. In the case where captcha is not present, we fill out the form with the credentials directly used to register and verify whether the login attempt is successful or not.

Verify Login Attempt. It is crucial for our study to successfully differentiate between a successful and a failed login attempt. We follow the state change and detection mechanism of the tool SAAT [21]. After we submit a login form with correct credentials we visit the login page in a separate tab which has shared storage with the current page and check unique identifiers to identify the user as described in [21]. If we can identify a login form then definitely the user is not logged in. If there is no login form then we have landed either to the profile of the user or the home page of the website. For that reason, we consider a user is logged in only when we can find any of the user specific identifiers (e.g., username,

email, first name, last name) or a logout button. Thus, we determine whether the user is logged in or not. We also rely on this method whenever we need to check the state of a page we are in.

Logout. We look for logout elements in the page we landed after a successful login. We search through HTML elements based on keywords specific to logout. In some cases, logout elements are dynamically loaded upon user interaction. We do not consider the case where user needs to interact with the menu, where user profile, settings etc., are listed along with logout button i.e., the logout element is dynamically loaded. After successful click on the logout element, we check the state of the logged out page and consider logout action successful only when the state changing mechanism indicates that the user is not logged in.

3.3 Automated Analysis

We run our automated analysis on the entire dataset. Later we randomly choose a small subset of the dataset and manually verify our heuristics in order to evaluate the performance of *LoginPlus*. In the following sections we describe the methodologies for automated analysis.

3.3.1 JWT Invalidation

Detecting JWTs. We capture the HTTP traffic in order to detect JWTs that are being used by the web server. We look for JWT in two request headers, 'Authorization' and 'Cookies'. We use regex that matches a well-formed JWT. We do not detect incomplete tokens that do not follow the standard structure of JWT i.e., tokens that do not have all of the three parts. We decode the JWT and analyze the header and payload. We look for any secret e.g, password, API keys, cryptographic keys etc., in the payload other than name and email using this database PatternDB [5] for secret patterns. We then calculate the expiration

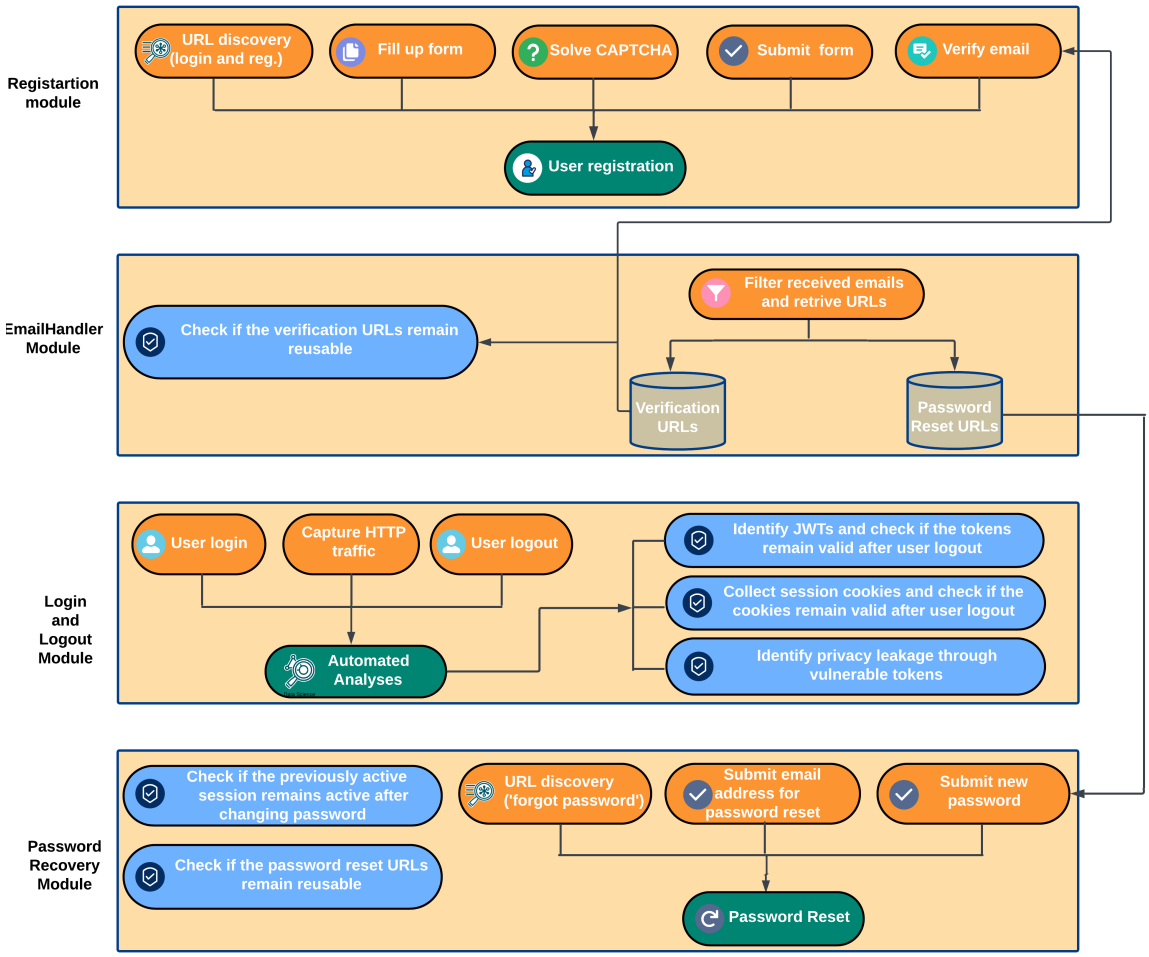


Figure 3.1: Overview of the primary modules and their respective components within the *LoginPlus* workflow. Various experiments are conducted across these modules. For instance, the EmailHandler module identifies verification and password reset URLs, and it also conducts invalidation tests for verification URLs. The Password Recovery module communicates with EmailHandler to retrieve password reset URLs and performs URL invalidation for these links. Additionally, our Login and Logout module conducts multiple analyses, including JWT and cookie invalidation after user logout.

time from the claim 'exp'.

Invalidation Test. If we find a JWT in the 'Cookie' header, that is, if a JWT is being used as a cookie, we try to determine whether it is being verified by the server. To do that we intercept the request that contains JWT as cookie, remove that cookie and send the request to the server. If we get a response that indicates that the request was invalid, we consider that the JWT is required and thus would be verified by the server. JWTs that

are used as bearer token do not go through this step as they are always required. Thus we make sure that the JWTs we find are functional to the server and save all the requests and responses containing these functional JWTs for further analysis. We observed that although some websites may use JWT as a cookie, the JWT itself is treated as it is supposed to be when it reaches the web server.

After collecting all the necessary data from HTTP traffic, we proceed to analyzing whether the JWTs are being invalidated properly by the web server upon logout. We save all the request-response containing JWTs in previous step. We then log out from our test account. Ideally a user should be logged out from the server the moment a logged out page is shown to the user. We wait for any logged out page is shown to the user and then wait for 60 seconds so that the server gets time to complete the log out process. After that we launch the unauthorized requests with JWTs. In the scenario where the JWTs are being used as a cookie, we remove all other cookies except the one containing a JWT. After receiving response from the web server, we compare this response and the one we saved when user was logged in, and identify if the unauthorized request still returns an authorized response.

Invalid Response Detection. Comparing responses from authenticated and unauthenticated requests is not trivial for several reasons. We simply can not compare the status code returned or the response body directly. In many cases, we observed that the two responses return same status code but either the response body or the headers of the response of the unauthenticated request indicates that the request was invalid. In that case, the response body and response headers might contain key-words pointing to an error. We first check two trivial cases i.e., we compare the status code and response body first. If the status code is changed, we consider that the tokens in the unauthenticated request was invalidated. For the second scenario, if the response body is same, then the tokens in the unauthenticated request are considered valid. When none of the above cases is true, we search for key-words that point to an error. If we still can not find any key-word from listed in Appendix A.2

in the response body indicating an error, we consider that the tokens were not invalidated and the response body contains data similar to the response of the authenticated request e.g., a token that is randomized with each request. The reason we check the trivial cases first before doing a key-word based search is that in some cases the response body might contain data that is ambiguous e.g., unable to parse or the data is too large.

Detecting Privacy Leakage Through Invalid Response. In order to detect privacy leakage, we make requests to the URL end-points that return personally identifiable data. After logging in, usually a homepage is shown to the user. The resources loaded in that page might not be customized for the user, meaning the captured responses after clicking the submit button of a login form might not contain sensitive data. For this reason, we follow the URLs in the homepage of a user that points to settings and user profile. This way we explore the authenticated area of the user account. Sometimes navigating to the pages containing settings and user information might need user interaction. We avoid user interaction as it might destroy execution context of the page and we will not be able to find the logout element.

At this point, whether the response from the unauthorized request is valid or not, a web server should not return any sensitive information. We consider email, password, date of birth, phone number etc., that we provided during registration as sensitive Personally Identifiable Information (PII). We look for any PII in the response from unauthorized request. We also look for any trace of API secrets that should not be revealed. We use PatternDB [5] database containing patterns for API secrets to identify them in response body. We only consider the patterns that are marked with high confidence in the database.

3.3.2 URL Invalidation After Email Verification

We verify emails using Gmail API following the procedures described in Section 3.2.2. We only consider conducting validation for the websites that send a verification URL rather

than a verification code. Since our email verification module is separated from the registration pipeline and runs in parallel, we do not wait for verification email to arrive during registration. Consequently, we do not input the verification code back into the form. Once we are done with registration process and the email is verified for a website, we visit the verification URL again using a fresh browser instance. First we visit the verification URL immediately and save the name of the affected website. We then determine if the user is logged in following the method described in Section 3.2.2. Finally, we visit the verification URLs again after 24 hours for the website where a user was logged in during the previous visit. We then check the state of the page visited to determine whether a user is logged in or not using the same method. We repeat the same experiment again after 7 days on the websites that were previously reported in the experiment after 24 hours.

3.3.3 URL Invalidation After Password Recovery

In total, we attempt to reset passwords four times. This is because in order to determine the validity of the URLs we first need to change the password. In the subsequent three attempts we determine whether the URLs remain valid instantly, after 24 hours and after 7 days. We describe the method in the below paragraphs.

In our first attempt, we look for password reset URLs on the login and registration pages and save the URLs. We do a keyword (e.g., 'forget password', 'forgot password', 'recover password', 'reset password' etc.) based search for finding those URLs. We then visit the URLs for resetting password. We look for 'form' element which takes email as input. We fill out the email field and submit the form. We maintain a separate thread similar to account verification process that accesses the Gmail accounts every 10 minutes and look for new emails for resetting passwords. As soon as we receive a new email for resetting password, we visit the URL in a fresh browser instant. We then look for 'form' elements containing two password fields, one for the new password and another for confirming password. We

submit the form after filling it out. We then close the browser and visit the same URL using a fresh browser instant.

In the fresh browser instant, we similarly look for 'new password' and 'confirm password' fields again. At this point, we consider three possibilities i.e., we can find the two password fields or the page can show an error message or it can be redirected to login page. In the two later cases, we consider that the password reset URL is not reusable i.e., the website is not allowing us to reuse the URL to reset password. In the first case, we fill out the fields and submit the form. After submitting the form, we look for alerts and error messages which indicate that the URL is invalid in order to rule out non-reusable URLs. We look for the keywords listed in Appendix A.2 for error message. Sometimes whether password was reset or not websites do not show any message, instead they redirect the user to the login page. For that reason, where we can not determine from any message whether the password reset attempt was successful, we try to login to the website using newly submitted password. If we can login then we consider that we have successfully reused the password reset URL. We save the name of the affected websites and password reset URLs for those websites for further testing.

We run the same experiment described in the previous paragraph after 24 hours visiting the same URLs for the websites for which the URLs were reusable in previous experiment. We rule out non-reusable URLs by observing error messages and verify password reset attempt by logging in again into the website. Similarly, we attempt to reset password again after 7 days using the same URLs for the websites that were flagged in previous experiment after 7 days.

3.3.4 Session Invalidation After Password Recovery

We conduct this experiment in parallel to the first attempt of resetting passwords. Before resetting the password of a website for the first time, we log in the user using our login

module with a fresh browser instance. We navigate to the login page, find username, email and password elements to fill out and submit the form. We then wait for the successful password reset signal which is running in a separate browser instance in parallel. This time we do not follow authenticated URLs of the user account. We also do not logout as it is not relevant to this test case. Note that logout is only required for JWT and cookie invalidation tests. Since we do the password reset test after JWT invalidation test was finished for all the websites, it becomes irrelevant whether we logout or not. After we submit the new password and identify that the submission was successful, we reload the logged in page and determine whether the user is still logged in. Note that if our login attempt verification method indicates that user could not be logged in, we do not reload and make any decision, rather simply close the browser. However, the password reset goes on as it is for the password reset link test described in Section 3.3.3.

3.3.5 URL Token Leakage

We measure the extent to which the tokens included in the verification and password reset URLs can be captured by an attacker. For that, firstly we extract the tokens in the URL. Then we look for the presence of the token in the local storage of the browser while visiting those URLs. The implication is that the local storage is accessible through any JavaScript running the context of the page. Any malicious script injected by the attacker will be able to obtain the token. Then we measure whether an on-path network attacker is able to obtain the token. Whenever a verification or password reset email is sent to the user without TLS, an attacker on the network will be able to capture the URLs. To measure that, we analyze the headers of the received verification emails. Note that these headers are different from the HTTP header received during communication. These headers are part of payload of the email object. An email object contains HTTP headers, data and payload. Payload stores some headers that contain information about senders and receivers along

the relay path. An email originating from a mail server might be relayed through a few other servers and finally reach the receiver's client. The information about the connection between the servers are stored in the payload. For our study, we focus on the 'Received' headers on the relay path as it contains information about sender and receiver servers. We look for any trace of TLS usage in the information provided in the 'Received' headers. We do a keyword based search to look for the usage of any version of TLS. If we can not find any trace of TLS usage in any of the connections, we mark the email as vulnerable.

3.3.6 Cookie Invalidation After User Logout

For this experiment, we first log into the website using username, email and password. After verifying that the user is logged in, we extract all the cookies for next stage. We then log out from the system. Upon successful log out, we wait for 30 seconds before issuing the next request to the web server. After 30 seconds, we launch a fresh browser instance and visit the login URL again. This time we inject the saved cookies from the previous valid sessions to the page and visit the login URL. We then check whether our state changing mechanism indicates that the user is logged in or not. If our method specify that the user is logged in, we conclude that the authentication and session cookies were not invalidated instantly upon user logout. We conduct the experiment again for a website after 24 hours and after 7 days. We inject the cookies captured first time the user was logged in into a new browser instance and visit the login URL. We check the state of the page and determine whether the user has been logged in with the previous cookies.

3.3.7 Vulnerable Cookies

We retrieve and save all the cookies handled by the browser using Puppeteer's 'page.cookies()' method. We save these cookies present on a page when a user is logged in. We then analyze these saved cookies to check for the 'Secure' and 'HttpOnly' attributes. A cookie is

considered vulnerable if either or both attributes are set to false. Identifying authentication cookies is beyond the scope of our study. Note that we only check these flags for first-party cookies.

Privacy Leakage Through Vulnerable Cookies. Cookies set by first-parties are likely to contain sensitive information. For example, they might contain username, email, password etc., to identify a user. The cookies containing sensitive information should receive absolute protection. They should have both the 'Secure' and 'HttpOnly' flag set to true. Otherwise, if any cookie has 'Secure' flag missing, the browser might send it through any unencrypted connection. On the other hand, if any cookie has 'HttpOnly' flag missing, it will be accessible through HTML document. We look for the cookies that contain sensitive data but are missing either one of the attributes or both.

3.4 Web Crawling and Browser Traffic Monitoring

We carry out web crawling through our organization's IP gateways. Since we avoid sending a high volume of requests that could lead to our IP addresses being blocked, using just two different IP addresses (one each for registration and post-login analysis) suffices for our study. To perform web crawling and browser automation, we utilized Puppeteer [23] with headless Chrome instances. For each page, we allowed a 30-second wait time as SAAT to ensure all resources were fully loaded. Additionally, we imposed a 30-second interval between each navigation to maintain a systematic and controlled browsing process. To reduce the load on web servers and prevent triggering anti-bot mechanisms, we limited the frequency of crawling during URL discovery on each website. This approach helps to ensure that our study did not disrupt the normal operation of the sites being analyzed or cause undue strain on their resources.

We use the Chrome DevTools Protocol (CDP) to communicate with our running browsers.

Through Puppeteer's CDPSession class, we capture HTTP traffic. To capture requests, we subscribe to the 'Network.requestWillBeSent' event, and for capturing responses, we subscribe to the 'Network.responseReceived' event. To obtain the Set-Cookie attribute from the response headers, we subscribe to the 'Network.responseReceivedExtraInfo' event, which is triggered when additional information like Set-Cookie is included in the response header. We use the 'Network.getResponseBody' method to retrieve the response body. We initiate the CDPSession just before submitting a login form and detach the session just before logout. This ensures that we capture only the requests and responses occurring while the user is logged in to an account.

We utilize SAAT's state detection method to verify successful login attempts, which has a 3% false positive rate for identifying such attempts (see [21]). During our manual inspection of test cases that required logging in first, we observed a false positive rate of 7%. It is important to note that our experiments were conducted on a different dataset than that used in [21]. However, the false positives do not affect some of our experiments directly. For example, the JWT and cookie invalidation experiments require a user to logout. Logout is not possible unless the accounts were actually created and *LoginPlus* was able to log in the user correctly. Unless the logout process itself generates a false positive, the results described in Sections 4.5 and 4.2 have lower chances of having false positives. Another issue identified through our manual verification is that JWTs can be discovered as we delve deeper into user accounts, potentially leading to more improper invalidation issues. Since we cannot explore user accounts in the same manner as actual users through automation, we may have overlooked several instances of improper token invalidation.

For email verification, a verification email including a URL will not be sent unless registration form was submitted successfully. Likewise, receiving a password recovery URL is only possible if an account is already created. Beside the registration and login process, other steps in our study might produce false positives. For that, we randomly

choose domains and verify the results manually.

3.5 Challenges

Our automated analysis is faced with several challenges. For instance, because of the non-deterministic behaviour of the websites, automating the login, logout and password recovery process becomes complicated. Moreover, browsing through the authenticated area of the user account and determining logout functionality is often thrown into disarray because of loading dynamic contents and multiple redirections. Furthermore, registration and login pages often contain CAPTCHA challenges. Although we tackle this challenge by using an AI driven CAPTCHA service AZCaptcha [10], we observed that in many cases the service can not solve CAPTCHA or a timeout occurs before receiving a solution from the service. Because of these challenges and the fact that two of our test scenarios depend on the logout functionality, our results merely define a lower bound in terms of figures reported. Note that we notified the operators of the vulnerable websites mentioned in this work through email communications.

Chapter 4

Results

In this Chapter, we describe the dataset we collected and the results of the automated analysis conducted using *LoginPlus*. Additionally, we discuss some interesting examples identified during our manual verification.

4.1 Dataset Collected

We conducted our large-scale measurement study from December 15, 2023 to January 31, 2024 on Google's CrUX [17] top 1M websites. Websites in CrUX are ranked by the most seen websites by Chrome browser. As per the analysis from the study by Ruth et al. [44], Google's CrUX [17] is the most accurate and reliable top list. We use the November, 2023 snapshot for top 1M list and initially use the dataset from Roomi et al. [43]. The dataset comprises 384,847 websites that support registration and login. It includes login URLs for 359,133/384,847 websites and registration URLs for 258,152/384,847 websites. However, 25,714 websites lack a login URL, and 126,695 websites do not have a registration URL. We newly discovered login URLs for an additional 6,289 websites and registration URLs for another 9,534 websites (see Section 3.2.1). Consequently, the dataset now contains 365,422/384,847 websites with login URLs and 267,686/384,847 websites with

registration URLs. For websites where only one type of URL is available, we use that URL for both registration and login attempts.

We initiated our registration process by using SAAT to submit forms for the 384,847 websites. We successfully submitted a registration form for 44,012/384,847 websites. However, a successful form submission alone may not lead to a successful registration. We assume that a registration is successful if we can log into the user account. Out of the 44,012 websites, there might be cases where the registration was not successful in spite of successful form submission indicating a false positive. Likewise, there might be cases where an account was created but we could not log in due to some reasons indicating false negatives. We discuss the possible reasons below.

We consider a login attempt successful only if the state change result described in Section 3.2.3 indicates that a user is logged in. Out of 44,012 successful registration form submission, we could log into 26,551 websites. We note that our registration and login attempts might fail for a variety of reasons. First, although a form was submitted successfully, it might not be the registration form meaning the form was misidentified as a registration form. Secondly, the email verification might not be successful. Although the chance of a verification URL to be expired is low as we access the emails every 10 minutes, the verification URL might redirect to another page where a user has to submit more information. We do not consider this multi-stage registration. Furthermore, after submitting a form, when we visit the website for the second time to log in, some websites might detect our automated tool and block login attempt. Finally, there might be CAPTCHA errors, incorrect evaluation of HTML elements or other non-deterministic behaviour such as browser closure behind failed registration and login.

We received a first email from 12,028 websites, of which 6,392 required email verification via a verification URL (see also Table 4.1). Among these emails, 1,128 out of 12,028 contained a verification code, and 4,508 out of 12,028 were general welcome/confirmation

emails. Our data indicates that websites mostly use verification URLs for email verification.

Table 4.1: Number of websites for which a first email was received.

Type	#websites
Verification URL	6,392
Verification code	1,128
Confirmation email	4,508
Total	12,028

Table 4.2 provides an overview of our registration, login and logout attempts by website ranking. We can see that form submission rates are consistent across the four ranking ranges. The highest login rate is observed among the top 1k websites. Our manual verification revealed that the top websites tend to name their HTML elements more consistently than lower-ranked websites. Since we use keyword-based searches to identify registration and login forms, this results in higher accuracy in identifying those forms on top-ranked websites compared to those with lower rankings. The logout rate is comparatively lower in top ranked websites. This observation suggests that the logout mechanism might be complex for the top websites. For instance, we observed that in most top websites the logout button is either dynamically loaded or requires multiple levels of navigation.

Table 4.2: Overview of total analyzed websites: form Submission, login, and logout across rankings

Ranking	Total analyzed	Form submission	Login	Logout
1K	554	81 (14%)	74 (91%)	11 (15%)
10K	5,002	667 (13%)	354 (53%)	98 (27%)
100K	49,660	6,209 (12%)	2,956 (47%)	961 (32%)
1M	384,847	44,012 (11%)	26,551 (60%)	5,642 (21%)

4.2 JWT Invalidation After User Logout

We detected the use of JWT after successful login in 1,616 websites. Out of this 1,616 websites, logout was possible for 520 websites. Thus, these 520 websites are the candidate websites for our JWT invalidation experiment. We found that 446/520 websites do not invalidate JWT after user logout. The experiment answers RQ1 about JWTs. Firstly, from our analysis above, we can see that 1,616/26,551 websites, that is 6% of the websites use JWT for authorization purposes. Note that this analysis is based on the condition that we could create an account for the websites as well as able to log into the account. We did not navigate deeper into user accounts during automated analysis, as automating such navigation to mimic real user behavior is not trivial. Therefore there may be additional websites that were not fully explored. From our manual observation, we found that in some websites a JWT can be identified if we navigate through the different parts of the user account.

Secondly, we found that around 85% (446/520) of our candidate websites do not invalidate the JWT for at least one of the requests after the user is logged out. Note that we consider only the requests that are sent to the first-party domain. Thus, it is likely that there was not an explicit mechanism to invalidate the JWT after user logout rather websites were relying on the automated expiration of the tokens. Table 4.4 shows breakdown of number of websites by ranking. Interestingly, the percentage remains near uniform but high across the rankings indicating that developers are widely depending on the automatic expiration of the tokens rather than invalidating them instantly.

Overall, we identified 1,665 unique JWTs across 1,616 websites. We conducted an analysis focusing on the ‘alg’ and ‘exp’ attributes in these tokens. Table 4.3 presents our findings related to the ‘alg’ attribute, notably showing that none of the JWTs used the value ‘none’ for ‘alg’ attribute. However, we identified 9 (1%) JWTs that ambiguously use the name ‘RSA’ for the signing algorithm. This might create confusion during validating

JWTs since most libraries check for standard names (e.g., 'RS512', 'RS384') defined in RFC 7518 [27].

Additionally, we analyze the expiry times of the JWTs that do not get invalidated upon user logout. From our observation, we found that the minimum expiry time is 5 minutes and the maximum time is 20 years. We have extracted an expiry time for 968/1,665 JWTs. For the rest of the tokens we could not identify an expiry time. Upon manual inspection, we found that those tokens do not contain the 'exp' attribute in the payload. The data in Figure 4.1 shows a varied distribution across different expiry times, indicating that there is no universal standard among websites. Each website may choose an expiry time based on its specific security requirements, user behavior patterns, and operational needs. The wide range of expiry times (from minutes to months) suggests that there is no consensus universally adopted despite best practices recommending that JWT expiry times should be as short as possible, ideally around 15 minutes or less [35]. In practice, only 22.2% JWTs had expiry times less than 30 minutes. This variability highlights the importance for developers and security practitioners to carefully consider their application's context and security requirements when choosing JWT expiry times.

Table 4.3: Distribution of algorithms used for JWT signing.

Algorithm	% JWTs
RS256	24%
HS256	68%
HS512	3%
ES256A	3%
RSA	1%
RS512	1%

Manual Verification. We do our manual verification for JWT invalidation after *Login-Plus* is able to detect JWT automatically and analyze them. We are interested in JWTs that are used when a user is logged in. To identify which websites use JWT for post-login

Table 4.4: Number of websites by ranking for which JWT was not invalidated after user logout

Ranking	Not Invalidated	Total Analyzed
1K	8 (88%)	9
10K	37 (94%)	39
100K	225 (91%)	246
1M	446 (85%)	520

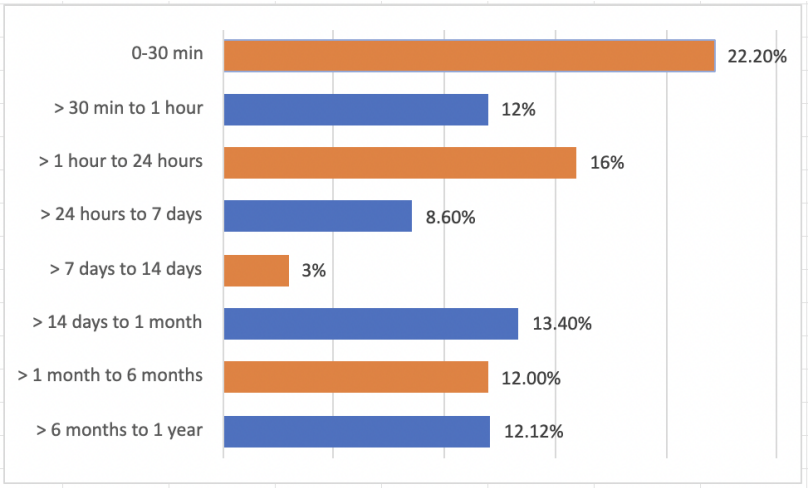


Figure 4.1: Frequency distribution of observed JWT expiry times.

activities from a large number of websites it is necessary to finish the automated analysis first. To assist the manual verification of our system we also save all the requests and responses along with all the JWTs in decoded form for the websites that use JWT for authorization. Then we sample 50 websites randomly that our system marked as affected and analyze them manually. We manually login to the websites and capture traffic using Burp Suite [1]. We identify all the requests from Burp Suite history by cross matching with the saved requests by *LoginPlus*. We then verify whether the JWTs in the requests are actually being used by the server. After that we logout of the system. We then remove all cookies except the JWTs and launch the requests again using Burp Suite Repeater. By comparing the responses we verify whether the JWTs are invalidated or not.

In addition, we randomly choose 50 websites from the set of websites where we did not find the presence of JWT using *LoginPlus*. We then browse the user accounts for these

50 websites simulating real user behavior, and found JWTs present in 5 out of 50 websites (10%), compared to the 6% detected by automatic analysis. This observation suggests that the numbers provided by our automated analysis represent only a conservative estimate, potentially underestimating the prevalence of JWT usage across all the websites.

4.2.1 Notable Examples

Fitbit.com. The fitness website issues two JWTs when a user logs in: one for accessing personal information (e.g., name, age, birth date, phone number) and another for tracking fitness progress (e.g., weight loss, food plan). Both tokens should stop working when the user logs out.

When the user logs out, the first token correctly stops working and the user's personal information is no longer accessible. However, the second token, used for accessing fitness progress, continues to work even after logout. Surprisingly, this second token can also be used to access the user's personal information after logout.

This issue arises because the website only has a system to invalidate the first token, but not the second one. Since the tokens are stateless, the server accepts the second token as valid even after logout, leading to a security risk in case the tokens get stolen.

Telegraph.co.uk. The news sharing platform issues an access token along with a refresh token after login. The access token is a JWT with expiry time 1 month which is longer than recommended [35]. After logout both the access token and the refresh token remain valid. The access token is not only used for accessing user's personal data (e.g., full name, address, phone number etc.), it is also used to update user's information on profile. Since the token does not get invalidated after logout, it is possible to update user's information even after logout. Interestingly, in the request for updating user's information an Id token is also included. However, the server does not evaluate the Id token. We replaced the Id token with a random string and the response did not change. It is possible that the server

only verifies the access token which they do not invalidate after user logout.

Medielogin.dk. This platform is an easy-to-use personal login website that can be integrated with other websites and apps, allowing the same Medielogin credentials to be used across different related sites. Although the JWTs used on the website have a shorter expiry time of 1 hour, they are not invalidated after the user logs out. Consequently, if any of these tokens are exposed, unauthorized parties could access and update the user's personal details.

4.3 URL Invalidation After Email Verification and Password Recovery

4.3.1 Email Verification.

We have verified emails for 6,392 websites by visiting verification URLs. Out of this 6,392 websites i.e., 6,392 verification URLs, for 171 verification URLs a user was logged in directly right after verification. Note that these 171 verification URLs were received from 171 different websites. We visit this 171 verification URLs again after 24 hours and identify whether the user gets logged in. Out of these 171 verification URLs, 83 remained valid even after 24 hours, allowing users to log in directly by clicking the URLs. We visit the 83 verification URLs after 7 days and only 34 of them remained valid and the user was logged into the system. Note that we have had already visited the URLs once during the email verification process.

From our analysis above we can see that, around 2.6% of the websites log in the user into the system directly through verification URLs. Usually, the practice of logging in the user directly might not be threatening for the users. However, the URLs are meant to be used only once. In other words, if the URLs remain valid for a prolonged time period, it

increases the attack surface in case the URLs or the tokens included in the URLs get leaked.

Manual Verification. We randomly sampled 50 verification emails marked by our analysis and corresponding URLs to manually check if the URLs are still valid. For this, we copy and paste the URLs in a separate browser in a separate machine. Among the 50 websites we checked manually for vulnerable email verification URLs, we have found that 70% of them are adult websites. We have also observed that the websites give different level of access upon visiting the URLs. For example, two websites give full access to the user account. We could browse the different parts (e.g., profile, settings) of the user accounts. Other websites show a static page with user information that were filled while creating the accounts.

4.3.2 Password Recovery

We identified password reset URLs on the login pages of 27,701 out of 44,012 websites. We attempted to reset the passwords for these 27,701 websites through password recovery. For 1,438 of these sites, we received an email regarding the password reset request. As detailed in Table 4.5, we automatically retrieved a URL for resetting the password for 1,248 out of these 1,438 websites using *LoginPlus*. During our manual inspection later for the rest 190/1,438 emails, we found that 163 were incorrectly identified as emails regarding password reset and thus, they do not contain a reset URL. Additionally, 8 emails were identified as emails associated with password reset but contained a reset URL that our tool could not locate, and 19 emails included a temporary password sent upon requesting the reset.

We proceeded to submit a new password for these 1,248 websites using the provided URLs. Out of these, 219 websites did not display any error message when attempting to reset the password again immediately after the first reset. Upon verifying by logging in with the newly changed password, we found that we could successfully log in to 173 websites,

indicating that these password reset URLs were not invalidated instantly.

We then attempted to reset the password again using those 173 URLs after a 24-hour interval. This time, 118 URLs remained valid even after 24 hours. We infer that the URLs that were valid within 24 hours but became invalidated afterwards likely had an expiration time of 24 hours or less. After seven days, only 3 of these URLs remained reusable for resetting passwords.

Table 4.5: Number of websites for which a password recovery attempt was successful in multiple stage of the experiment

Password recovery	#websites
1st attempt	1,248
2nd attempt (30 seconds)	173
3rd attempt (after 24 hours)	118
4th attempt (after 7 days)	3

Manual Verification. We randomly selected 50 password recovery emails from the outcome of our analysis, along with their corresponding URLs, to manually verify if the URLs were still valid. We also check those three password reset URLs that remained valid after 7 days. We identified one site ranked within the top 50K, *escambia.k12.fl.us*. During our manual inspection in January 2024, we discovered that clicking the password reset URL sent via email generated and displayed a new password on the page. When we checked this URL again after 48 hours, it was still valid. If an attacker had gained access to this URL, they could have repeatedly logged into the user's account without the user's knowledge. When checked again in June 2024, this issue was found to be resolved, and the URL was no longer valid.

4.4 Session Invalidation After Password Recovery

We conduct this experiment to assess session invalidation after password recovery on websites where we successfully reset the password for the first time. We were able to

reset passwords for 1,248 websites for the first time using password recovery URLs (see Table 4.5). Our analysis revealed that 674/1,248 websites do not terminate active sessions after a password reset. This means for 54% of the websites the previously active sessions remain active regardless of whether the legitimate user is logged in or out. Consequently, the cookies and tokens associated with authentication for those sessions remain valid and acceptable by the web servers. Although users may believe they have successfully reset their password and secured their account, the reality might be otherwise. If those active sessions are controlled by a malicious actor and the websites do not notify legitimate users about them or provide options to logout of them, the password recovery system essentially fails. Since all sessions are not terminated, the malicious actor retains control of the account without the real user's knowledge.

Table 4.6 presents data on the prevalence of session invalidation following password recovery across different website rankings. It shows that none of the top 1K websites failed to invalidate previous sessions, indicating a high level of security. However, this percentage rises to 32% within the top 10K websites, 45% within the top 100K websites, and 51% within the top 1M websites. This trend indicates a significant increase in the lack of session invalidation as website rankings decrease.

The data suggests that higher-ranked websites tend to implement more compliant security measures during the password recovery process, ensuring that any previous sessions are invalidated to protect user accounts from unauthorized access. In contrast, lower-ranked websites are more likely to overlook this critical security step, making them more vulnerable to potential security breaches.

Manual Verification. For this test we choose 50 websites randomly from the resulting websites marked affected during our analysis. We first login into the accounts in one browser. We then visit the login page using a separate browser with different user-agent and request a password reset for the account. We reset the password using the password

Table 4.6: Number of websites by ranking for which previous active sessions were not invalidated after password recovery.

Ranking	Not Invalidated	Total Analyzed
1K	0	20
10K	33 (32%)	102
100K	215 (45%)	475
1M	674 (54%)	1,248

reset URL sent in the email. We then reload the page where the user was logged in. In addition to reloading the page, in this manual verification step, we browse the user account. We found that 39/50 websites did not terminate sessions after a user reset password. For the rest 11/50 websites, we assume that there could be several reasons for the false positive result by the automated analysis such as login attempt verification produced false positive, the password was not actually reset etc.

We also manually examined several well-known websites to determine whether they terminate sessions after password recovery or offer an option to log out of previously active sessions. Our observations revealed a variety of behaviors: For instance, *RedHat.com* does not immediately terminate active sessions and does not provide an option for users to log out of them. However, it automatically terminates these sessions after five minutes. On the other hand, *AliExpress.com* instantly invalidates the session upon password recovery. We also found some websites, like *Rakuten.ca*, do not terminate active sessions after password recovery, which could leave accounts vulnerable to unauthorized access. We remained active and periodically browsed the user profile within 1 hour of changing the password on the previously logged in device, noticing that the session was not terminated. We then stopped browsing leading to a period of inactivity and, consequently, the session was terminated. This inconsistency in session management practices highlights the varying levels of security measures implemented across different websites, with some offering more immediate and robust protections than others.

4.5 Cookie Invalidation After User Logout

We revisited the experiment originally conducted by Calzavara et al. [12]. Since testing for cookie invalidation on a session level necessitates logging out of the system, we executed this experiment on the subset of 5,642 websites from which we could successfully log out. Among these, we found that for 513 out of 5,642 (9%) websites, the session cookies were not promptly invalidated upon logout. To further investigate, we repeated the experiment with the same set of cookies for each websites after a 24-hour interval on these 513 websites. Remarkably, even after 24 hours, 325 websites still failed to invalidate the cookies. Subsequently, after seven days, 203 websites continued to exhibit the same issue of not invalidating the session cookies. In comparison, Calzavara et al. reported a higher rate of 18% for websites that did not invalidate cookies post-user logout. Our experiment, however, indicates a lower figure, which may suggest some improvements over time.

4.6 Privacy Leakage

Table 4.7 illustrates an overall picture of the findings from our analysis for privacy leakage.

Privacy leakage through JWTs. Based on our analysis, we found that out of 520 websites examined, 446 websites, constituting a majority, initiated at least one request where the JWT remained valid even after user logout. In total, we documented 1,249 instances across these 446 websites, wherein the requests returned sensitive information such as personal data, API keys, or other confidential details. Furthermore, we specifically identified 18 websites where such requests yielded information such as usernames, email addresses, or passwords. Out of the 18 websites, 6 websites were found where passwords appeared in the responses corresponding to requests containing JWTs, and these JWTs were not invalidated after the users logged out.

URL token leakage. We found that 89 out of 1,248 password reset URLs and 146 out of 6,392 verification URLs are served over HTTP instead of HTTPS. This lack of secure protocol increases the risk of interception. Additionally, 42 out of 1,248 websites store the unique token from the password reset URL in the browser's local storage. For verification URLs, 339 out of 6,392 websites store the unique token in local storage as well. Storing tokens in local storage is risky because an attacker could potentially capture these tokens by injecting malicious JavaScript into the browser [20].

We also received 174 verification and password reset emails that included the user password in plaintext. Out of these, 15/174 verification emails mentioned the current user password, and 159/174 password recovery emails revealed the current user password instead of providing options such as a URL or code to reset the password. This practice not only suggests that user passwords might have been stored in plaintext on the server side [43], but also violates the OWASP recommendation that password recovery processes should not expose current passwords in any way (see Appendix A.1).

Furthermore, we analyzed the headers of the verification and password reset emails we received. Our analysis revealed that 57 different websites do not use TLS when sending these emails. Since verification and password recovery emails contain sensitive information, particularly the password recovery emails, sending them without TLS exposes users to significant risks. If an attacker intercepts these emails, they could gain access to the URLs and potentially reset user account passwords. This vulnerability highlights the importance of using secure transmission methods to protect user data during such critical operations.

Privacy leakage through cookies. Overall, 16,971 websites have atleast one cookie that is missing either the 'Secure' attribute or the 'HttpOnly' attribute. 16,197/16,971 websites have atleast one cookie for which the 'Secure' attribute is missing and 16,626/16,971 websites for which at least one cookie has the 'HttpOnly' attribute missing. These are the cookies saved by the browser after a user is logged in. Note that here we consider only

Table 4.7: Privacy leakage detected by our analysis

Type	# of websites
PII leakage:	
- through vulnerable cookies	580
- due to improperly invalidated JWTs	18
- username exposure	10
- password exposure	6
- email exposure	12
- plaintext passwords in emails	174
Token leakage:	
- through password recovery URLs:	
- served over HTTP	89
- tokens stored in local storage	42
- through verification URLs:	
- served over HTTP	146
- tokens stored in local storage	339
- lack of TLS in email communication	57

cookies set by first-party websites. We also do not identify among the saved cookies that are directly associated with authentication.

We also identify whether any first-party cookie that either has 'Secure' attribute missing or 'HttpOnly' attribute missing contains any sensitive information. In this case, we consider email, username and password in any of the formats: plaintext, SHA1, SHA256, MD5 and base64 encoded as sensitive information. We have found 580 such websites in total that have atleast one vulnerable cookie i.e., either missing 'Secure' or 'HttpOnly' attribute containing any of the mentioned sensitive information. Among the 580 websites, 141 websites have atleast one cookie containing sensitive information but missing 'Secure' attribute and 555 websites have atleast one cookie containing sensitive information but missing 'HttpOnly' attribute.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Our research focused on the critical task of invalidating user sessions, particularly after sensitive operations like password recovery. Unlike previous studies that primarily examined cookie invalidation, we explored the lesser-studied area of token invalidation across various web authentication flows. This included investigating the improper invalidation of JWTs post-user logout, and the persistence of verification and password recovery URLs, which could potentially grant unauthorized access. Using our automated tool, *LoginPlus*, we analyzed top websites and found that many websites failed to invalidate JWTs upon user logout, indicating websites do not implement any explicit method to invalidate JWTs before expiry time. Moreover, several websites allowed continued access through verification URLs well beyond their intended use and did not invalidate password reset URLs, underscoring vulnerabilities in token management. Our findings also highlighted security lapses such as the use of non-secure HTTP protocols for sending sensitive URLs and storing tokens insecurely in local storage. In conclusion, our study contributes empirical insights into the real-world implementation of token invalidation mechanisms and calls for enhanced security measures to protect user credentials and ensure robust session management in web

applications.

5.2 Limitations

Since our entire framework is based on browser automation, it occasionally exhibits unexpected behaviors such as browser closures, redirections, and page crashes. These issues have also been noted in prior works [12, 18, 21, 43]. In our study, all of the experiments have strict preconditions to meet such as registration and login. Some of the experiments were also dependent on user logout. For example, although we were able to login to 26,551 websites and found the usage of JWT in 1,616 websites, we ran the test for JWT invalidation on 520 websites as in these 520 websites we could logout automatically. Therefore, there might have been websites in those 1,616 websites where JWT was found but we missed as we could not automatically logout. Another crucial example is the password recovery process. It requires us to have an account on a website, find the correct HTML element to make a password reset request, filter the emails and find the correct email that contains recovery URL and finally submit form with new password. In between these steps, if we face any browser issue or network issue we practically miss that website from being evaluated.

5.3 Recommendations for Developers

- **Use shorter expiry time:** Reduce the token lifespan to minimize the window of vulnerability if a token is compromised. Shorter expiration times limit the duration of access in case of attacks.

- **Implement an explicit mechanism JWT invalidation:** Relying solely on token expiration is risky. Developers should explicitly invalidate JWTs for events such as

user logout, ensuring the token is rendered unusable immediately.

- **Use libraries for time-based claims:** Choose libraries that properly handle time-based claims like exp (expiration) to avoid misconfigurations, ensuring tokens expire as intended.

- **Ensure prompt token invalidation for recovery URLs:** Password recovery and verification URLs should expire quickly and be valid for one-time use, preventing unauthorized reuse in case of interception.

- **Use TLS for token communications:** Ensure that sensitive tokens are transmitted securely by using TLS for all communications, preventing exposure to man-in-the-middle attacks.

5.4 Future Works

- Extend *LoginPlus* to cover additional token-based authorization flows and test other common vulnerabilities, such as replay attacks.

- Investigate other forms of token management such as SAML based authentication and session handling across more diverse web applications, including mobile platforms.

- Explore improvements to standards like OWASP ASVS to address emerging token-based security issues more comprehensively.

- Analyze open-source libraries that implement JWT-based authorization schemes to identify vulnerabilities, specifically focusing on whether they correctly handle token expiration and token validation.

Bibliography

[1] Burp suite. Available at https://portswigger.net/burp.

[2] Easylist. https://easylist.to/index.html.

[3] Owasp application security verification standard (asvs). https://owasp.org/www-project-application-security-verification-standard/.

[4] Bugmenot. https://bugmenot.com, 2014.

[5] M. Ahmed. Secrets patterns database. https://github.com/mazen160/secrets-patterns-db/tree/master.

[6] S. Ahmed and Q. Mahmood. An authentication based scheme for applications using json web token. In *2019 22nd International Multitopic Conference (INMIC)*, pages 1–6, 2019.

[7] Akanksha and A. Chaturvedi. Comparison of different authentication techniques and steps to implement robust jwt authentication. In *2022 7th International Conference on Communication and Electronics Systems (ICCES)*, pages 772–779, 2022.

[8] A. Alkhulaifi and E.-S. M. El-Alfy. Exploring lattice-based post-quantum signature for jwt authentication: Review and case study. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5, 2020.

[9] C. Ardi and M. Calder. The prevalence of single sign-on on the web: Towards the next generation of web content measurement. In *Proceedings of the 2023 ACM on Internet Measurement Conference*, IMC '23, page 124–130, New York, NY, USA, 2023. Association for Computing Machinery.

[10] AZCaptcha. Auto captcha solver service and cheap captcha bypass service provider-azcaptchas. `https://azcaptcha.com/`.

[11] C. Beaman and H. Isah. Anomaly detection in emails using machine learning and header information, 2022.

[12] S. Calzavara, H. L. Jonker, B. Krumnow, and A. Rabitti. Measuring web session security at scale. *Comput. Secur.*, 111:102472, 2021.

[13] M. Chatzimpyrros, K. Solomos, and S. Ioannidis. *You Shall Not Register! Detecting Privacy Leaks Across Registration Forms*, pages 91–104. 02 2020.

[14] Q. Chen, P. Ilia, M. Polychronakis, and A. Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *Proceedings of the Web Conference 2021*, WWW '21, page 2117–2129, New York, NY, USA, 2021. Association for Computing Machinery.

[15] L. Compagna, H. Jonker, J. Krochewski, B. Krumnow, and M. Sahin. A preliminary study on the adoption and effectiveness of samesite cookies as a csrf defence. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 49–59, 2021.

[16] A. Dabrowski, G. Merzdovnik, J. Ullrich, G. Sendera, and E. Weippl. Measuring cookies and web privacy in a post-gdpr world. In D. Choffnes and M. Barcellos, editors, *Passive and Active Measurement*, pages 258–270, Cham, 2019. Springer International Publishing.

[17] C. Developers. Chrome ux report. https://developer.chrome.com/docs/crux/.

[18] K. Drakonakis, S. Ioannidis, and J. Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1953–1970, New York, NY, USA, 2020. Association for Computing Machinery.

[19] O. Ethelbert, F. F. Moghaddam, P. Wieder, and R. Yahyapour. A json token-based authentication and access management schema for cloud saas applications. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 47–53, 2017.

[20] O. Foundation. Html5 security cheat sheet. https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html#local-storage.

[21] M. Ghasemisharif, C. Kanich, and J. Polakis. Towards automated auditing for account and session management flaws in single sign-on deployments. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1524–1524. IEEE Computer Society, 2022.

[22] M. Ghasemisharif, A. Ramesh, S. Checkoway, C. Kanich, and J. Polakis. O single Sign-Off, where art thou? an empirical analysis of single Sign-On account hijacking and session management on the web. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1475–1492, Baltimore, MD, Aug. 2018. USENIX Association.

[23] Google. Puppeteer. https://github.com/puppeteer/puppeteer, 2021.

[24] M. Haekal and Eliyani. Token-based authentication using json web token on sikasir restful web service. pages 175–179, 01 2016.

[25] N. Hong, M. Kim, M.-S. Jun, and J. Kang. A study on a jwt-based user authentication and api assessment scheme using imei in a smart home environment. *Sustainability*, 9(7), 2017.

[26] L. Jannett, V. Mladenov, C. Mainka, and J. Schwenk. Distinct: Identity theft using in-browser communications in dual-window single sign-on. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1553–1567, New York, NY, USA, 2022. Association for Computing Machinery.

[27] M. B. Jones. JSON Web Algorithms (JWA). RFC 7518, May 2015.

[28] L. Jánoky, P. Ekler, and J. Levendovszky. Evaluating the performance of novel jwt revocation strategy. *Acta Cybernetica*, 25, 08 2021.

[29] L. Jánoky, J. Levendovszky, and P. Ekler. An analysis on the revoking mechanisms for json web tokens. *International Journal of Distributed Sensor Networks*, 14:155014771880153, 09 2018.

[30] D. Kats, D. Silva, and J. Roturier. Who knows i like jelly beans? an investigation into search privacy. *Proceedings on Privacy Enhancing Technologies*, 2022:426–446, 04 2022.

[31] S. A. Khamis, C. F. M. Foozy, M. F. A. Aziz, and N. Rahim. Header based email spam detection framework using support vector machine (svm) technique. In R. Ghazali, N. M. Nawi, M. M. Deris, and J. H. Abawajy, editors, *Recent Advances on Soft Computing and Data Mining*, pages 57–65, Cham, 2020. Springer International Publishing.

[32] K. Kubicek, J. Merane, A. Bouhoula, and D. Basin. Automating website registration for studying gdpr compliance. pages 1295–1306, 05 2024.

[33] P. Kulkarni, J. R. Saini, and H. Acharya. Effect of header-based features on accuracy of classifiers for spam email classification. *International Journal of Advanced Computer Science and Applications*, 11(3), 2020.

[34] S. G. Morkonda, S. Chiasson, and P. C. van Oorschot. Empirical analysis and privacy implications in oauth-based single sign-on systems. In *Proceedings of the 20th Workshop on Workshop on Privacy in the Electronic Society*, WPES '21, page 195–208, New York, NY, USA, 2021. Association for Computing Machinery.

[35] P. Mouriya. Long-lived jwt - abuse and mitigation. `https://blog.rootrwx.com/long-lived-jwt/`.

[36] Y. Mundada, N. Feamster, and B. Krishnamurthy. Half-baked cookies: Hardening cookie-based authentication for the modern web. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIACCS '16, page 675–685, New York, NY, USA, 2016. Association for Computing Machinery.

[37] O. Pantelić, K. Jovic, and S. Krstović. Cookies implementation analysis and the impact on user privacy regarding gdpr and ccpa regulations. *Sustainability*, 2022.

[38] T.-H. Pham, Q.-H. Vo, H. Dao, and K. Fukuda. Ssologin: A framework for automated web privacy measurement with sso logins. In *Proceedings of the 18th Asian Internet Engineering Conference*, AINTEC '23, page 69–77, New York, NY, USA, 2023. Association for Computing Machinery.

[39] A. Rahmatulloh, R. Gunawan, and F. M. S. Nursuwars. Performance comparison of signed algorithms on json web token. *IOP Conference Series: Materials Science and Engineering*, 550, 2019.

[40] A. Rasaii, S. Singh, D. Gosain, and O. Gasser. Exploring the cookieverse: A multi-perspective analysis of web cookies, 2023.

[41] RFC Editor. Json web token (jwt). Technical Report 7519, 2015.

[42] G. E. Rodríguez, J. G. Torres, P. Flores, and D. E. Benavides. Cross-site scripting (xss) attacks and mitigation: A survey. *Computer Networks*, 166:106960, 2020.

[43] S. A. Roomi and F. Li. A Large-Scale measurement of website login policies. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2061–2078, Anaheim, CA, Aug. 2023. USENIX Association.

[44] K. Ruth, D. Kumar, B. Wang, L. Valenta, and Z. Durumeric. Toppling top lists: evaluating the accuracy of popular website lists. In *IMC '22: ACM Internet Measurement Conference*, page 374–387, New York, United States, October 2022. Association for Computing Machinery.

[45] A. Senol, G. Acar, M. Humbert, and F. Z. Borgesius. Leaky forms: A study of email and password exfiltration before form submission. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1813–1830, Boston, MA, Aug. 2022. USENIX Association.

[46] A. Senol, A. Ukani, D. Cutler, and I. Bilogrevic. The double edged sword: Identifying authentication pages and their fingerprinting behavior. In *Proceedings of the ACM on Web Conference 2024*, WWW '24, page 1690–1701, New York, NY, USA, 2024. Association for Computing Machinery.

[47] S. Sprecher, C. Kerschbaumer, and E. Kirda. Sok: All or nothing - a postmortem of solutions to the third-party script inclusion permission model and a path forward. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 206–222, 2022.

[48] C. F. Torres, F. Willi, and S. Shinde. Is your wallet snitching on you? an analysis on the privacy implications of web3. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 769–786, Anaheim, CA, Aug. 2023. USENIX Association.

[49] S. Van Acker, D. Hausknecht, and A. Sabelfeld. Measuring login webpage security. In *Proceedings of the Symposium on Applied Computing*, SAC '17, page 1753–1760, New York, NY, USA, 2017. Association for Computing Machinery.

[50] M. Westers, T. Wich, L. Jannett, V. Mladenov, C. Mainka, and A. Mayer. Sso-monitor: Fully-automatic large-scale landscape, security, and privacy analyses of single sign-on in the wild. *ArXiv*, abs/2302.01024, 2023.

[51] B. Xu, S. Jia, J. Lin, F. Zheng, Y. Ma, L. Liu, X. Gu, and L. Song. Jwtkey: Automatic cryptographic vulnerability detection in jwt applications. In G. Tsudik, M. Conti, K. Liang, and G. Smaragdakis, editors, *Computer Security – ESORICS 2023*, pages 263–282, Cham, 2024. Springer Nature Switzerland.

[52] Y. Zhou and D. Evans. SSOScan: Automated testing of web applications for single Sign-On vulnerabilities. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 495–510, San Diego, CA, Aug. 2014. USENIX Association.

Appendix A

A.1 Application Security Verification Standards (ASVS) Recommendations

The list of OWASP ASVS recommendations [3] that we follow in our study:

(1) Verify that logout invalidates the session tokens.

(2) Verify that stateless session tokens use digital signatures, encryption and other security measures.

(3) Verify that out of band verifier expires out of band authentication codes or tokens after 10 minutes.

(4) Verify that out of band verifier authentication tokens are only usable once.

(5) Verify password credential recovery does not reveal the current password in any way.

(6) Verify that the out of band authenticator and verifier communicates over a secure channel.

(7) Verify that the application gives the option to terminate all other active sessions after a successful password change (including change via password reset/recovery).

A.2 Keywords for Identifying Errors

'invalid', 'incorrect', 'error', 'could not', 'expire', 'bad token', 'bad request', 'timed out', 'not found', 'unable', 'problem', 'sorry', 'wrong', 'failed', 'does not exist', 'no longer valid', 'not valid', 'not correct', 're-submit', 'resubmit', 'unauthorized', 'does not match'

A.3 Third-Party Cookies and Scripts

We collect third-party cookies found in a page when a user is logged in. We categorize the cookies as *advertisement*, *trackers* and *unknown* based on EasyPrivacy and EasyList [2]. We have found 1,677 websites containing at least one third-party cookie saved by browser when a user is in logged in state. In total 320 websites contain either one third-party cookie that is associated with either tracking or advertising. 63/320 websites have at least one tracking cookie and 318/320 websites contain at least one cookie associated with advertising. We have found 26 unique tracking and advertising third-party websites.

We list all the third-party scripts in login page. After a login page is loaded we search all the 'script' elements and determine from the source if the script belongs to third-party. If the source of the script do not contain the website name we mark it as a third-party script. Our method might mark a first-party script as third-party in the case where the first-party do not use their original website name for the source of the script. Similarly when a login attempt is successful we collect scripts from every page as we navigate until we log out of the account. We then identify whether any new third-party script is loaded when the user was logged in. We also categsorize the third-party scripts as tracking scripts, advertisement and unknown using EasyList.

We found that 89% of websites that has a login page contains at least one third-party script. On average there are 18 third-party scripts in a login page. We have found third-party scripts in 87% of websites after a user is logged in. On average 19 third-party scripts

are found after logging in. We also observed that 24% websites add new third-party scripts after logging in. 73% of websites have tracking scripts, 56% websites have advertisement scripts and 38% websites have unknown third-party scripts after login.

Overall, we identified 1,277 unique third-party domains that execute scripts after logging in. Among them 291 are categorized as tracking domains, 189 are as advertising and the rest remain unknown. Among the third-party domains, *googletamanager.com* has the highest number of occurrence. Other domains that are frequently seen are *google-analytics.com*, *facebook.net* and *doubleclick.net*. Table A.1 contains the statistics of the presence of third-party scripts.

Table A.1: Top 20 Third-Party Domains

website	**type**	**No. of Websites**
googletamanager	tracker	482
google-analytics	tracker	332
doubleclick	advertising	250
facebook	tracker	209
gstatic	tracker	180
googlesyndication	advertising	176
googleadservices	tracker	143
googletaservices	advertising	142
cloudflare	tracker	136
jsdelivr	advertising	95
ajax	tracker	89
criteo	tracker	87
cloudflareinsights	tracker	86
bing	tracker	65
ampprojects	unknown	60
usync	unknown	58
tiktok	tracker	53
pubmatic	advertising	49
hotjar	tracker	48
clarity	tracker	47