Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.079484

ARTICLE

Unveiling Authentication Forgery in OpenID Connect under Web Frameworks: A Formal Analysis of CSRF-Based Attack Paths

Xingyun Hu[1,2], Siqi Lu[1,2,*], Liujia Cai[1,2], Ye Feng[1,2], Shuhao Gu[1,2], Tao Hu[1] and Yongjuan Wang[1,2,*]

[1]Information Engineering University, Zhengzhou, China
[2]Henan Key Laboratory of Network Cryptography Technology, Zhengzhou, China
*Corresponding Authors: Siqi Lu. Email: 080lusiqi@sina.com; Yongjuan Wang. Email: pinkywyj@163.com

Received: 22 January 2026; Accepted: 23 February 2026; Published: 08 May 2026

ABSTRACT: With the widespread adoption of web applications and cloud services, the OAuth 2.0-based OpenID Connect (OIDC) Single Sign-on (SSO) protocol has become the core of modern digital identity authentication. Although the OIDC protocol itself has strict security specifications, its implementation in real-world web frameworks can introduce critical vulnerabilities, particularly the improper omission of the *state* parameter, which leads to severe authentication forgery risks. Existing research often overlooks these implementation-level flaws, especially from a formal analysis perspective. This paper addresses this gap by formally analyzing the authentication forgery attack resulting from the missing *state* parameter. We construct a high-fidelity web framework model and, using the Tamarin formal analysis tool, systematically analyze the flawed OIDC implementation. Specifically, we demonstrate an attack path where cross-site request forgery is leveraged as a vector to deceive the relying party, ultimately achieving identity binding forgery—linking the attacker's identity to the victim's session. In response to this forgery vulnerability, this article proposes and formally verifies corresponding patches to successfully defend against such attacks. Finally, this paper provides concrete guidance for developers. This research, through formal methods, characterizes a replicable authentication forgery pattern within modern web architectures, providing a robust theoretical and practical foundation for hardening SSO systems against such advanced forgery threats.

KEYWORDS: SSO; authentication forgery; formal analysis; web security vulnerabilities

1 Introduction

Single Sign-on (SSO) technology, as the core solution for modern digital identity management, simplifies the authentication process across multiple application scenarios. Among various SSO protocols, OpenID Connect (OIDC), built on the OAuth 2.0 framework [1], has become the industry's de facto standard. OIDC is widely used in critical scenarios such as social media login and enterprise identity governance [2,3]. However, the trust placed in OIDC can be subverted by sophisticated authentication forgery techniques that exploit subtle implementation flaws.

The complexity of the OIDC protocol makes its practical deployment a fertile ground for such vulnerabilities. These vulnerabilities often do not stem from the protocol specification itself, but from implementation flaws—defects where developers deviate from or improperly implement security-critical components. While the OIDC standard is secure in principle, the security of its real-world deployments is not guaranteed, often undermined by critical implementation errors. Although the specification defines parameters like *state* to prevent threats like Cross-Site Request Forgery (CSRF) [2], its security is highly

dependent on correct implementation [4,5]. In practice, developers often omit "recommended" security parameters, with the absence of the *state* parameter being a particularly common and high-risk oversight [6–9]. This parameter is designed to bind an authentication request to a user's session. Its absence, a critical implementation flaw, does not just weaken session integrity; it creates a direct forgery vector, introducing a severe CSRF vulnerability. This allowing an attacker to manipulate the authentication flow and forge the identity binding, ultimately leading to account takeover [10].

Despite extensive research on OIDC security [4,11,12], a significant gap remains in analyzing these implementation-level flaws within realistic web contexts. Much of the existing work relies on theoretical analysis [1,13], case studies [5,14], or black-box testing [15,16], which often fail to provide a rigorous, verifiable explanation of the attack mechanics. Furthermore, existing formal verification studies [17–20] tend to oversimplify the underlying web framework (e.g., cookie management, redirect behavior) [19,21,22], making them unsuitable for discovering subtle forgery vectors that arise from the complex interactions between the protocol and the web environment [23,24]. This highlights an urgent need for a formal model that can accurately represent a non-standard OIDC implementation within a realistic web framework to systematically analyze the resulting forgery vulnerabilities [25,26].

To fill this research gap, this paper adopts a formal approach to analyze a critical authentication forgery technique that exploits the absence of the *state* parameter. Based on our previous work on formal web application modeling [21], we construct a high-fidelity model of a typical web environment (as shown in Fig. 1), including the browser, Relying Party (RP), and Identity Provider (IdP). We then model the OIDC authorization code (authcode) flow within this framework, specifically under the condition that the *state* parameter is missing. Using the Tamarin prover [17,27] for automated analysis, we successfully derived an attack path that demonstrates this forgery mechanism and formally verified the effectiveness of the standard mitigation strategy [28]. The main contributions of this paper are:

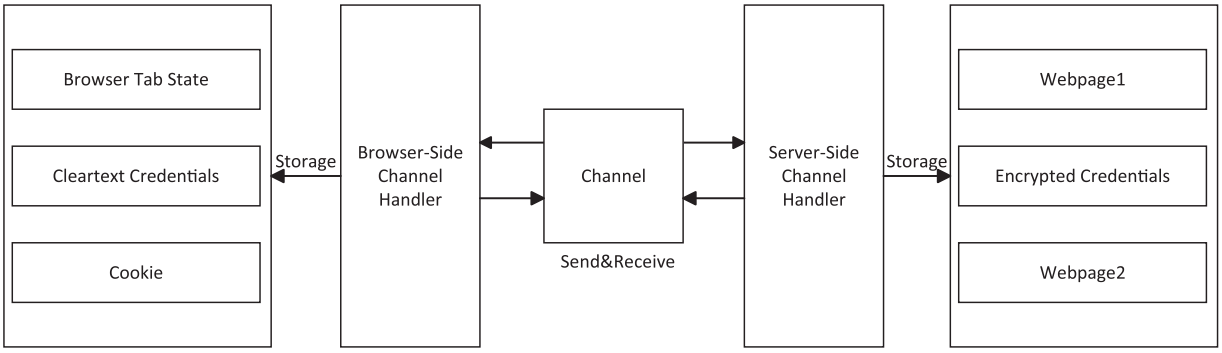

Figure 1: Formal model overview of the web framework.

- We construct a high-fidelity formal model of the web framework, formalizing intricate components to provide a realistic foundation for uncovering subtle implementation-level vulnerabilities.
- We formally discover and characterize a critical authentication forgery attack, demonstrating that an attacker can exploit the missing state parameter to illicitly bind their identity to a victim's session via CSRF.
- We propose a sound countermeasure to neutralize this forgery vulnerability and provided security recommendations as a defense against this class of attack.

The remainder of this paper is organized as follows: Section 2 reviews related work, Section 3 details the formal modeling process, Section 4 presents the security analysis and verification results, Section 5 discusses limitations and future work, and Section 6 provides a summary.

2 Related Work

The research in this paper is situated at the intersection of two closely related fields: the security analysis of the OIDC protocol, particularly studies on implementation flaws that enable authentication forgery; and the technological advancements in analyzing web security protocols using formal methods, especially the Tamarin prover. This review aims to position our work by highlighting the gaps in existing research concerning the formal analysis of authentication forgery techniques.

In the analysis of OIDC security, existing studies have revealed significant risks in its implementation that can be exploited for identity forgery [4,5,11]. This issue is not unique to OIDC; identifying weaknesses in authentication protocol implementations is a recurring challenge in security research, with analogous vulnerabilities discovered in architectures such as multi-server authentication systems [29] and two-factor schemes for wireless sensor networks [30]. Although the protocol specification defines parameters like *state* to counter attacks such as CSRF [2], empirical studies show that their neglect or incorrect handling frequently leads to security incidents [6,7,9,31,32]. These studies effectively demonstrate the "existence" of vulnerabilities through large-scale testing or case analysis [5,12,14]. However, they often rely on black-box or gray-box testing [15,16] and fall short in providing a mathematically rigorous explanation of the underlying forgery mechanism, the precise attack paths within a complete system context, and a strict proof of the effectiveness of countermeasures against such forgery technologies [23,33]. They show that a forgery is possible, but not precisely how it is constructed.

Formal methods offer a powerful approach to address these deficiencies [26,34]. The Tamarin prover, an advanced symbolic protocol verification tool [17], has been successfully applied to analyze complex protocols like TLS 1.3 [35]. In the context of SSO, studies have used Tamarin to verify the security of OAuth 2.0 [1] or the standard OIDC specification [13,18]. However, these works share a common limitation: their models often oversimplify the underlying web framework (e.g., cookie management, same-origin policy) [19,21,22] and primarily focus on correct protocol implementations, assuming all parties adhere strictly to the specification [36,37]. Consequently, they are ill-suited to analyze forgery vectors that arise specifically from implementation deviations.

Furthermore, recent research has begun to extend OIDC's application scope beyond traditional web SSO. For instance, some studies focus on adapting OIDC for end-to-end user authentication in messaging and video conferencing [38], while others explore its role in federated authentication across heterogeneous environments like cloud, edge, and fog computing [39]. Innovations also include integrating OIDC with blockchain for secure token delivery in Zero Trust networks [40] or enhancing privacy and load balancing in multi-cloud SSO systems [41]. Even in emerging fields like Unmanned Aerial Vehicle (UAV) systems, OIDC is being leveraged for secure device and user authentication [42]. These advancements, while expanding the protocol's utility, also introduce new interaction complexities and potential security considerations that underscore the importance of rigorous, context-aware formal analysis, which remains the focus of our work.

To explicitly contrast our approach with these simplifications, Table 1 summarizes the key differences in modeling critical web components and highlights their security implications.

Table 1: Comparison of web framework modeling approaches.

Feature Component	Simplified Models (Typical Approach in Prior Work)	Our High-Fidelity Model (This Paper)	Security Implication of High-Fidelity Modeling
Cookie Management	Abstracted as a simple session identifier or ignored. Session state is treated as an idealized primitive.	Modeled as a stateful "cookie jar" with security attributes (Domain, Path, Secure, HttpOnly) and automatic attachment logic.	**Crucial for discovering CSRF**, as the attack vector relies on the browser's automatic and unconscious attachment of session cookies to requests induced by an attacker.
HTTP Redirects	Modeled as an atomic message-forwarding event between protocol participants, losing the browser's context.	Modeled as a browser-driven, automatic state transition. The browser initiates a new request, inheriting the security context (e.g., cookies).	**Enables tracing the complete attack chain**, revealing how an illegitimate request causally leads to a valid, but malicious, authentication response.
Browser Concurrency (Tabs)	The user agent is modeled as a single, monolithic client, making it difficult to represent concurrent activities.	Explicitly models individual browser tabs, each with its own navigational state, while sharing a common cookie storage.	**Accurately represents the attack scenario** where a victim's legitimate session (in one tab) is exploited by a malicious page loaded in another tab within the same browser instance.
Same-Origin Policy (SOP)	Abstracted as a simple, all-or-nothing check, or its nuanced effects on "write" operations are ignored.	Modeled by its effects on browser behavior, such as constraining cookie attachment but permitting cross-origin form submissions ("write" operations).	**Precisely explains why SOP does not prevent CSRF**. Our model captures that an attacker can induce a cross-origin "write" request, which the browser executes with valid credentials, even if the attacker cannot "read" the response.

In reality, the most insidious security threats, including many authentication forgery techniques, often stem from such deviations from the specification. Research into the refined formal analysis of non-standard OIDC implementations—such as those lacking the *state* parameter—remains a significant gap. Although prior work has attempted to build more realistic web formal models [22,43,44], the systematic application of these models to dissect specific forgery mechanisms in OIDC and formally verify their mitigations has not been undertaken [8,45,46].

This paper directly addresses this research gap. We do not re-verify the standard OIDC protocol. Instead, building upon a high-fidelity web framework model [21], we construct and analyze a flawed OIDC implementation that lacks the *state* parameter. Our goal is to formally dissect the authentication forgery technology that this specific flaw enables, revealing its step-by-step execution and proving the efficacy of its countermeasure [47].

3 Preliminary

This section provides the necessary background for the formal analysis. To build a comprehensive security model, we first establish the ideal behavior of the system under analysis. Therefore, we begin by outlining the normative process of the OIDC authorization code flow (Section 3.1). We then define the capabilities of the malicious entity that seeks to subvert this process in our Threat Model (Section 3.2). Finally, to formally analyze the interaction between the protocol and the adversary, we introduce the Tamarin prover, the symbolic verification tool employed in our study (Section 3.3).

3.1 OIDC Authorization Code Flow

3.1.1 Core Participants

This section models the core participants in the OIDC web SSO architecture. The model establishes the foundational entities and their cryptographic relationships that constitute the OIDC protocol ecosystem. Each participant is characterized by their specific cryptographic capabilities and functional roles within the authentication flow. The participants are defined as follows, with their roles and formal representations summarized in Table 2.

Table 2: Core participants in the OIDC web SSO models.

Participant	Role	Security Parameters	Key Functions
User	Initiator	–	Triggers authentication by accessing RP services.
UA	Protocol Executor	Same-Origin policy	Manages HTTP requests, cookies, redirects; renders content.
RP	Service Provider	*state*, *nonce*	Initiates OIDC flow, validates tokens, grants access.
IdP	Authenticator	Signing keys, Token expiration	Verifies user identity, issues tokens, manages credentials.
Adversary	Malicious Entity	Constrained by crypto assumptions	Controls malicious sites, induces CSRF requests, exploits browser behaviors.

The OIDC protocol involves five core participants that interact through well-defined cryptographic operations:

- **User (U):** The initiator of the authentication process, represented as an entity that seeks access to protected resources. The user's identity is bound to their credentials at the Identity Provider.

- **User Agent/Browser (UA):** The cryptographic mediator that executes protocol operations on behalf of the user. The browser maintains critical security states, which can be modeled by its ability to handle key material and cookies.
- **Relying Party (RP):** The service provider that relies on the IdP for authentication. Its cryptographic identity is defined by its OIDC client credentials. A critical function of the RP is to generate and validate the state parameter for CSRF protection:

$$state = H(nonce \,||\, timestamp \,||\, session_{context}) \tag{1}$$

- **Identity Provider (IdP):** The trusted authentication authority that issues cryptographic tokens. The IdP's core cryptographic capability is token signing. The ID Token is generated as:

$$IDToken = Sign_{K_{priv}^{IdP}}(sub, aud, exp, nonce, ...) \tag{2}$$

where $Sign_{K_{priv}^{IdP}}$ represents the digital signature using the IdP's private key.

- **Adversary (A):** The malicious entity within the Dolev-Yao model enhanced with web-specific capabilities, subject to the constraints of perfect cryptography.

3.1.2 Communication Process

This section establishes a rigorous cryptographic foundation for analyzing OIDC vulnerabilities by formalizing the standard authorization code flow (as shown in Fig. 2) through mathematical representations. Rather than relying on procedural descriptions, we employ cryptographic primitives and formal notations to precisely capture the security-critical transformations occurring at each protocol step. The model systematically defines the generation, transmission, and verification mechanisms of core security parameters—state, nonce, and cryptographic tokens—using mathematical formulations that expose the underlying security guarantees. This formal approach enables precise vulnerability analysis in subsequent sections by providing an unambiguous specification of the intended protocol behavior under ideal conditions.

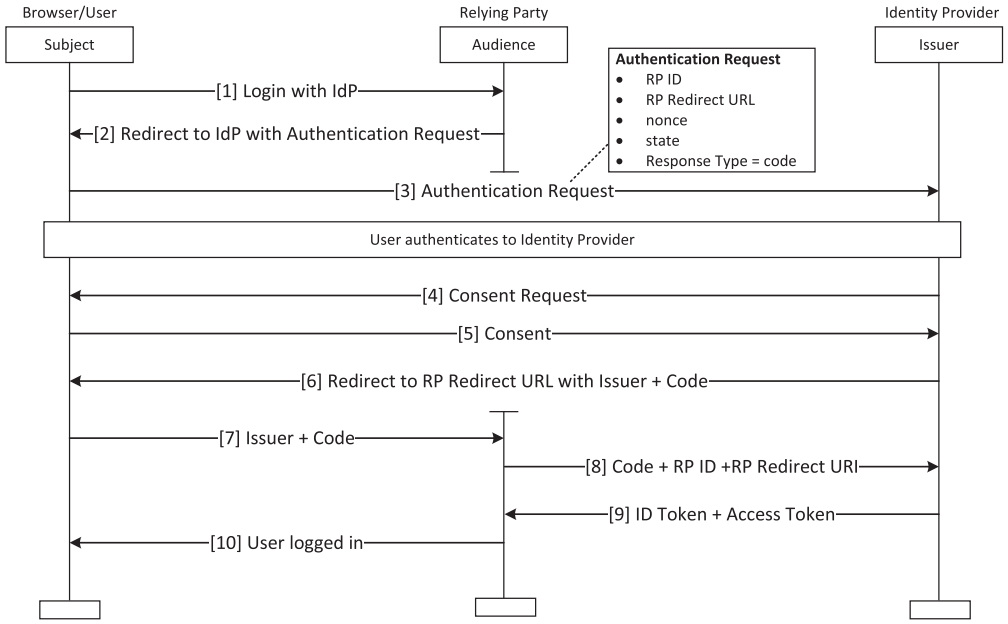

Figure 2: Ideal flow chart of the OIDC protocol.

a. **Initiation of Authentication Request** (Steps 1–2)

The authentication process begins when a user attempts to access a protected resource hosted by the RP. This action triggers the protocol initiation phase, where the RP generates critical security parameters to ensure the integrity and security of the authentication flow. The generation of these parameters is a foundational step in mitigating common web-based attacks, such as CSRF and replay attacks. Specifically, the RP computes the stateparameter as a cryptographic hash to bind the authentication request to the user's session context. The formulation is as follows:

$$state = H(nonce_{\mathrm{RP}} \oplus timestamp \oplus session_{context}) \tag{3}$$

where $H : \{0,1\}^* \to \{0,1\}^n$ represents a cryptographic hash function, $\oplus$ denotes bitwise XOR operation, and $nonce_{\mathrm{RP}}$ is a fresh random value. This combination guarantees that the stateparameter is unique, unpredictable, and tied to the specific authentication attempt. Simultaneously, the RP generates the *nonce* parameter using a cryptographically secure pseudorandom number generator (CSPRNG) to ensure randomness and prevent predictability:

$$nonce = G(\omega) \tag{4}$$

where G is a cryptographically secure pseudorandom generator and ω is a random seed. The *nonce* serves as a unique identifier for the authentication request, ensuring that each session is distinct and resistant to replay attacks. These parameters are cryptographically bound to the session context.

b. **Initiation of Authentication Request** (Steps 3–6)

The authentication request initiated by the user and forwarded by the RP now reaches the IdP for processing. This phase constitutes the core of the identity verification process, where the IdP authenticates the user's credentials and generates the authorization grant. As depicted in Fig. 2, this stage involves critical interactions between the user, the browser, and the IdP, ensuring that only legitimate users can proceed to the token exchange phase.

The IdP processes the authentication request by verifying the user credentials through the verification function. Upon successful authentication, the IdP generates an authorization code that cryptographically binds the session parameters:

$$authcode = Sign_{K_{priv}^{IdP}}(client_{id} \,||\, state \,||\, scope \,||\, timestamp) \tag{5}$$

where $||$ denotes concatenation. This cryptographic binding guarantees that any alteration to the parameters during transmission will invalidate the signature, thereby protecting against manipulation by adversaries.

The generated authorization code is then transmitted back to the RP via the user's browser. This step relies on the secure redirection mechanism modeled in Section 4.1.4, where the browser automatically handles the redirect while preserving the security context. The authorization code serves as a short-lived, one-time-use token that the RP can later exchange for an access token and ID token, completing the authentication process.

c. **Token Exchange and Verification** (Steps 7–9)

In this phase, the RP validates the integrity and authenticity of the authorization response before establishing a user session. This phase involves multiple cryptographic verification steps that collectively ensure the protocol's resistance to various attacks, including CSRF and replay attacks. The entire process is governed by rigorous security checks that form the cornerstone of the protocol's trust model.

The core security verification occurs when the RP processes the authorization response. The state parameter validation is performed as:

$$\text{Verify}_{state} = \begin{cases} 1 & \text{if } state_{received} = state_{stored} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Following successful state validation, the RP initiates the token request through a secure back-end channel to the Identity Provider's token endpoint. This request involves the computation:

$$token_{request} = Enc_{K_{pub}^{IdP}}(client_{id} \,||\, client_{secret} \,||\, authcode) \tag{7}$$

where $Enc_{K_{pub}^{IdP}}$ represents encryption using the IdP's public key. The ID Token signature verification constitutes a critical security check:

$$\text{Verify}_{Sig} = \text{Verify}_{K_{pub}^{IdP}}(IDToken, Sign_{K_{priv}^{IdP}}(IDToken)) \tag{8}$$

This check guarantees that the ID Token corresponds to the current authentication session rather than a previously issued token. The comprehensive verification process, encompassing state validation, secure token exchange, and cryptographic signature verification, establishes a robust security foundation for the subsequent session establishment phase.

d. **Session Establishment** (Step 10)

The final phase of the OIDC authorization code flow is the establishment of a secure, authenticated session between the user and the RP. This step marks the successful culmination of the protocol, where the RP transitions from validating transient credentials to creating a persistent, stateful session for the user.

Upon successful validation of the ID Token's signature and the *nonce* parameter—confirming both the token's authenticity and its freshness—the RP derives a session key to secure subsequent interactions. This session key will subsequently be used to protect communications between the user agent and the RP, for instance, by encrypting or integrity-protecting sensitive data exchanges.

A final, crucial verification is performed to ensure the token's freshness and to prevent replay attacks. The RP checks that the *nonce* value contained within the received ID Token matches the value stored during the initial authentication request.

This check guarantees that the ID Token was issued specifically for the current protocol instance and not replayed from a previous session. Only after this verification succeeds does the RP officially establish the user session, typically by issuing a session cookie to the browser and storing the session_key associated with the user's identity on the server side. The user is then granted access to the requested protected resource.

Having detailed the ideal operation of the OIDC protocol, we now define the adversary who seeks to exploit deviations from this standard flow.

3.2 Threat Model

This section establishes the adversary model and foundational assumptions for our formal analysis. Our model refines the classical Dolev-Yao model by incorporating web-specific adversarial capabilities and constraints, focusing on the attack vectors arising from the interplay between the OIDC protocol and the web platform.

3.2.1 Adversary Capabilities

The adversary (A) in our model is a powerful entity that controls the network but is constrained by the security mechanisms of the web platform and cryptographic assumptions. Its capabilities are formally defined as a set of operations $A_{cap} = A_{net} \cup A_{web}$, where A_{net} represents network-level capabilities and A_{web} represents web-specific capabilities.

Network-Level Capabilities (A_{net}): The adversary has full control over the public communication channels (e.g., browser-to-server HTTP/HTTPS traffic). This allows A to eavesdrop on, intercept, modify, drop, or inject messages on these channels. Formally, this is modeled by the adversary's ability to handle the following set of operations on public channels:

$$A_{net} = \{\text{Eavesdrop}(m), \text{Inject}(m, c), \text{Modify}(m, m'), \text{Drop}(m)\} \tag{9}$$

where c represents the communication channel identifier, and m & m' represents messages transmitted over channels.

However, these capabilities are constrained for protected channels. We assume that critical back-end communications, such as token requests between the RP and the IdP, are protected by secure TLS channels. Therefore, the adversary will be unable to decrypt or tamper the messages in the channel. This constraint is formalized as: for any message m_{TLS} on a TLS-secured channel, $\text{Decrypt}(m_{TLS}) \notin A_{cap}$ and $\text{Tamper}(m_{TLS}) \notin A_{cap}$.

Web-Specific Capabilities (A_{web}): This is the most critical aspect of our model, enabling the analysis of web-specific attacks like CSRF. The adversary can operate malicious web infrastructure and craft content that exploits the automated, context-unaware behavior of the user's browser to initiate unauthorized requests. Formally, this is modeled by the adversary's ability to perform the following set of operations:

$$A_{web} = \{\text{Control}(dmalicious), \text{LureUser}(U, d_{malicious}), \text{InduceRequest}(U, url_{target}, P)\} \tag{10}$$

where $d_{malicious}$ is a domain under adversary control, U is a victim user, url_{target} is the target URL for the induced request, and P represents the request payload (e.g., hidden form data).

These capabilities are constrained by browser security policies like the Same-Origin Policy (SOP). A key limitation is that the adversary cannot directly access or exfiltrate session credentials protected by the HttpOnly flag. Instead of direct access, the adversary's InduceRequest capability leverages the browser's inherent auto-attach mechanism for cookies.

Based on the formal definition and constraints of the adversary's capabilities mentioned above, Fig. 3 (a schematic diagram of the web adversary model) intuitively presents the relationships among the various participants in our threat model and the core attack paths. As shown in the figure, the model clearly distinguishes the trusted area from the adversary area. This figure visually reinforces the core of our model: the adversary's power does not lie in cracking cryptography or completely controlling the network, but in acting as a malicious Web participant, precisely exploiting the inherent automation mechanisms of the browser and Web platform to undermine the protocol logic.

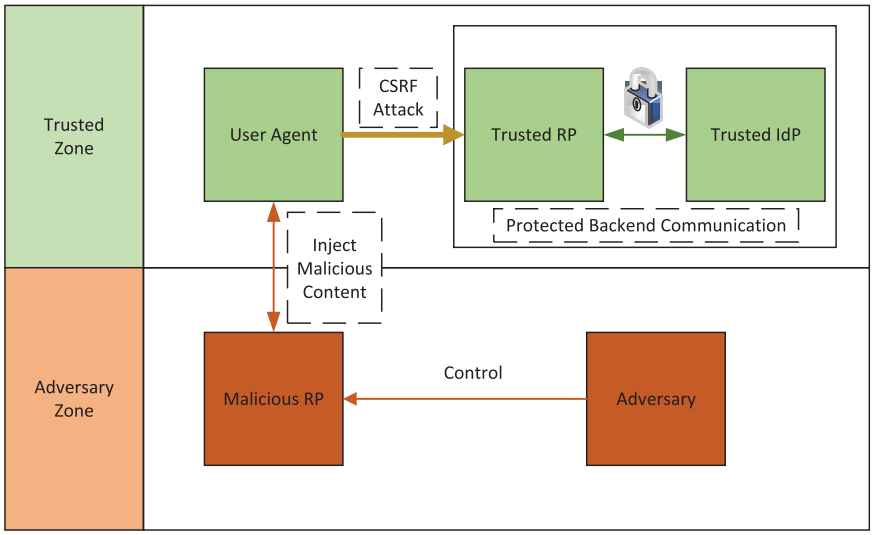

Figure 3: Diagram of web adversary model.

3.2.2 Security Assumptions

To construct a tractable yet meaningful formal analysis, our study operates under a series of foundational modeling assumptions. These assumptions are designed to isolate the analysis to the logic of the protocol and its interaction with the web framework.

Perfect Cryptography: We posit that all cryptographic primitives—including encryption ($Enc_k(m)$), digital signatures ($Sign_k(m)$), and hash functions ($H(m)$)—are computationally secure and function as ideal black boxes. Consequently, an adversary's power is limited to exploiting logical flaws rather than breaking the underlying mathematics. Formally, for any probabilistic polynomial-time adversary A and any security parameter λ, the advantage of A in winning a security game against a primitive is negligible.

Trusted Public Key Infrastructure (PKI): We assume a trusted PKI that guarantees the authentic binding between participant identities (e.g., the RP and IdP) and their public keys. For an entity E, its public key pk_E is reliably bound via a certificate $Cert_E$.

Web Platform Integrity: We assume the integrity of the user's client software; the browser is treated as a trusted component that correctly enforces its own security policies, such as the SOP and cookie handling rules. Our model, therefore, deliberately excludes attacks targeting browser implementation bugs or the underlying operating system. The adherence to the SOP is modeled as a constraint: a script from origin O_1 cannot access resources from origin O_2 if $O_1 \neq O_2$.

With the protocol's intended behavior and the adversary's capabilities formally defined, we now introduce the tool that enables us to model their interaction and automatically verify security properties.

3.3 Tamarin Prover

This section provides a concise introduction to the Tamarin prover, the symbolic verification tool employed in our formal analysis of the OIDC protocol. Tamarin operates within the symbolic model of cryptography, where cryptographic primitives are idealized as algebraic operations following the Dolev-Yao adversary model. Unlike computational approaches that reason about probabilistic polynomial-time adversaries, Tamarin's symbolic method enables fully automated verification of security protocols against an unbounded number of sessions.

The Tamarin framework builds upon three fundamental concepts that form the basis of our protocol specification:

- **Term Algebra for Messages:** Messages are represented as terms in a sorted algebra. Cryptographic operations are modeled through function symbols that obey algebraic equations. For instance:
  (1) Pairing: $pair(m_1, m_2)$ represents the concatenation of messages m_1 and m_2.
  (2) Symmetric encryption: $senc(m, k)$ denotes the encryption of message m with key k.
  (3) Asymmetric encryption: $aenc(m, pk(sk))$ represents public-key encryption.
  (4) The decryption is governed by the equation: $adec(aenc(m, pk(sk)), sk) = m$
- **Fact-Based System State:** The protocol state is maintained through a multiset of facts, which can be either linear (consumable) or persistent (always available). Key fact types include:
  (1) State facts: $St(session, role, params)$ representing protocol state.
  (2) Communication facts: $In(m)$ and $Out(m)$ for message input/output.
  (3) Freshness facts: $Fr(n)$ for generating fresh nonces.
  (4) Action facts: $AuthAttempt(user, server)$ for recording security-relevant events.
- **Multiset Rewriting Rules:** Protocol behaviors are specified through rules of the form: $\frac{\text{Precondition Facts [Action Facts]}}{\text{Conclusion Facts}}$

Each rule defines a state transition where the premise facts are consumed and conclusion facts are produced. For example, a simplified server authentication rule can be expressed as:

$$\frac{In(aenc((user, n), pk(sk_S))), Fr(k), !Ltk(S, sk_S)}{Out(senc((n, k), pk(user))), AuthCommit(S, user, n)} \tag{11}$$

4 Formal Models

This section constructs the formal models that enable a rigorous analysis of OIDC protocol security within web frameworks. We elaborate a refined model of the web framework, capturing critical elements. Then we define the security attributes required for formal verification and configures the Tamarin prover setup for automated analysis.

4.1 Web Framework

This section constructs a refined formal model of the web framework to serve as the foundation for analyzing the OIDC protocol's security vulnerabilities, with a specific focus on the flawed implementation resulting from the omission of the stateparameter. Unlike prior studies that oversimplify web components, our model captures the intricate interactions between the protocol logic and the underlying web platform by formalizing five core elements: web page characterization, cookie generation and storage, same-origin policy enforcement, redirect behaviors, and secure channel management. These elements collectively define the environment in which authentication forgery attacks, such as CSRF, can be systematically exploited and verified. By integrating high-fidelity representations of browser automation (e.g., cookie handling and redirect chains) and security constraints (e.g., SOP and TLS-protected channels), this model enables a precise analysis of how the missing stateparameter breaks the binding between authentication requests and responses, ultimately leading to exploitable vulnerabilities. The following subsections detail each component's formalization.

4.1.1 Web page

To accurately model the attack surface presented by web browsers in SSO protocols, we introduce a fine-grained formalization of the web page and browser interaction mechanism. Moving beyond the prevalent

simplification in prior formal analyses [2,23] which often model the user agent as a monolithic, stateless client, our model explicitly distinguishes between the browser instance and individual browser tabs. This distinction is critical for capturing concurrent protocol executions and the state isolation (or lack thereof) that is fundamental to web-based attacks like CSRF. The core of this modeling is the *Create_Tab* rule, whose formal specification is provided in Fig. 4.

```
rule Create_Tab:
let loaded_webpage = <'NULL', 'NULL', 'NULL', 'NULL'>
    pending_request = 'NULL'
in
    [ Fr(~tab_id)
    , !Browser(browser_id) ]
  --[ Create_Tab(~tab_id, browser_id) ]->
     [ Tab_State(~tab_id, browser_id, loaded_webpage, pending_request) ]
```

Figure 4: Modeling of browser tabs.

Modeling Concurrency and State via Browser Tabs. The rule *Create_Tab* models the initiation of a new browsing context. It generates a fresh, unique identifier $\sim tab_{id}$ for the new tab, scoped within a specific browser instance $browser_{id}$. The state of the tab is then captured by the linear fact $Tab\_State(\sim tab_{id}, browser_{id}, loaded\_webpage, pending\_request)$. The *loaded_webpage* parameter, initialized as ⟨'NULL', 'NULL', 'NULL', 'NULL'⟩, is a 4-tuple representing the currently loaded page's URL components (protocol, domain, path, file). The *pending_request* field, initialized to 'NULL', tracks any in-flight HTTP request initiated from this tab. This two-tiered state representation—a persistent browser identity coupled with dynamic, per-tab linear state—enables our model to analyze scenarios where multiple OIDC flows (e.g., one initiated by the user and another induced by an attacker) occur in parallel within the same browser, sharing the underlying cookie storage but maintaining separate navigation and request contexts. This precision is a key advancement over models that treat the browser as a single sequential process.

Formalizing the Web Page Lifecycle. The *loaded_webpage* state is not static; it evolves through rules that process HTTP responses (e.g., *Response_Handler*). When a response is received, the *Tab_State* fact is updated to reflect the new *loaded_webpage*. This formally captures the browser's navigation behavior, which is essential for modeling the OIDC flow: the initial request loads the RP's page, a redirect loads the IdP's login page, and the final callback loads the RP's endpoint again. By tracking the exact page loaded in each tab, our model can enforce security-relevant constraints, such as ensuring that a form submission originates from a page of a specific origin, thereby providing a formal ground for reasoning about navigation-based attacks.

Enabling Precise Attack Analysis. The explicit modeling of tabs and their state is fundamental to our discovery of the CSRF vulnerability. It allows us to formalize the attack scenario where an attacker-controlled page (loaded in one tab/malicious site, as shown in Fig. 5) induces a request that is processed in the context of the user's session with the RP (associated with the RP's origin in the browser's cookie store). The *pending_request* field further allows tracing the causality of request-response cycles across redirects, which is crucial for verifying whether an OIDC callback can be correlated back to a user-initiated request—the very property violated when the *state* parameter is missing. Therefore, this web page characterization is not merely a syntactic construct but the foundational layer upon which the subtle interaction flaws between the OIDC protocol and the web platform are exposed and rigorously analyzed.

```
rule CSRF_Attacker_Creates_Malicious_Link:
let malicious_url = <'HTTPS', $RP, $RP_Path, $RP_File>
    idp_url = <'HTTPS', $IdP, $IdP_Path, $IdP_File>
    malicious_webpage = <'MaliciousPage', malicious_url, idp_url, <'AuthReq', rp_id, rp_redirect_url,
'ResponseType_is_Code'>>
in
    [ !CSRF_Attacker(attacker_id)
    , !Discovered(idp_url, rp_url, rp_redirect_url, rp_id, client_id) ]
  --[ CSRF_Attacker_Creates_Link(attacker_id, $RP, $IdP)
    , CSRF_Attack_Initiated() ]->
    [ !Malicious_Webpage(attacker_id, malicious_webpage) ]
```

Figure 5: Modeling of malicious webpage.

4.1.2 Cookie Generation and Storage

In web-based SSO systems, cookies serve as the primary mechanism for maintaining session state and authentication context between the user's browser and the RP. However, the automatic handling of cookies by browsers—while essential for usability—also introduces critical security risks, particularly as the enabling factor for CSRF attacks. Traditional formal models of security protocols often oversimplify or entirely abstract away cookie management, treating session state as an idealized primitive [23,25]. This abstraction fails to capture the nuanced behaviors of real-world browsers, where cookies are automatically generated, stored, and attached to outbound requests based on complex rules involving domains, paths, and security flags. Our work addresses this gap by introducing a refined formal model that precisely captures the entire lifecycle of cookies—from generation and secure storage to automatic attachment—within the Tamarin prover framework. The core of this model is visualized in Fig. 6.

```
rule Create_Cookie_Storage:
let cookie = 'NULL'
in
    [ !Browser(browser_id) ]
  --[ Create_Cookie_Storage(browser_id, $Server) ]->
    [ Cookie_Storage(browser_id, $Server, cookie) ]
```

Figure 6: Modeling of cookie management mechanism.

Stateful Cookie Jar with Security Attributes: We model the browser's cookie storage not as a simple set but as a stateful structure that persists security-critical attributes. Each cookie is represented as a tuple incorporating its name, value, domain, path, and—crucially—the *Secure* and *HttpOnly* flags. The generation of a new cookie is triggered by a Set-Cookie header in an HTTP response, formalized by a rule (e.g., *RP_Sets_Cookie*) that creates a persistent fact:

$$!Cookie\_Storage(browser\_id,\ RP, \langle name, value, domain, path, Secure, HttpOnly\rangle)$$

This persistence accurately reflects the browser's behavior of maintaining cookies across sessions and navigation events. The inclusion of security flags allows our model to enforce real-world constraints; for instance, a cookie marked *Secure* = 'true' can only be transmitted over HTTPS channels, a constraint we formalize as a Tamarin restriction rule.

Precise Automatic Attachment Mechanism: The most significant innovation lies in formally encoding the browser's automatic cookie-sending logic. This is defined by rules (e.g., *Request_Handler_With_Cookie*) that govern HTTP request generation. The rule's premise checks for the existence of a persistent $!Cookie\_Storage$ fact whose *domain* and *path* attributes match the target URL of the outgoing request. If a match is found and any security flags (like *Secure*) are satisfied by the request context, the rule fires, and the cookie is automatically added to the request headers, represented by an action fact like $Cookie\_Attached(browser\_id, RP, cookie\_value)$. This mechanistic modeling is vital—it captures the exact

vulnerability exploited in CSRF attacks: the browser automatically and unconsciously adds authentication cookies to requests, even those induced by an adversary.

Integration with the Adversary Model: Our cookie model directly interacts with the web-specific adversary capabilities defined in Section 3.2.1. While the adversary cannot directly read cookies marked *HttpOnly* (modeled as a constraint $ExtractHttpOnlyCookie(c) \notin \mathcal{A}_{cap}$), they can exploit the automatic attachment mechanism. By luring a user to a malicious page, the adversary can induce the browser to make a request to the target RP. Our model formally shows that this induced request will satisfy the premises of the cookie attachment rule, causing the browser to automatically include the victim's session cookie. This precise interplay between the adversary's induction capability and the browser's automated cookie handling is what allows us to derive the concrete CSRF attack path in the flawed OIDC model.

This fine-grained formalization of cookie generation and storage provides the necessary foundation for analyzing implementation flaws like the missing *state* parameter. By accurately representing how cookies maintain state and how that state is automatically transmitted, our model can demonstrate precisely how the broken link between the authentication request and response (due to the missing *state*) allows an adversary to hijack the automatic cookie mechanism to forge an authentication session. The corresponding Tamarin rules, illustrated in Fig. 6, thus serve as the critical component that bridges the web platform's mechanics with the protocol-level logic, enabling a rigorous proof of the CSRF vulnerability.

4.1.3 Same-Origin Policy

The Same-Origin Policy (SOP) is a cornerstone of the web security architecture, preventing data leakage and cross-site attacks by restricting script access between different origins. However, in existing formal models, the SOP is often oversimplified into an abstract "same-origin check" black box, which fails to capture its nuanced interactions with browser mechanisms (such as cookie handling and redirects), thus making it difficult to analyze the root cause of vulnerabilities like CSRF. The innovation of this work lies in abandoning the isolated abstraction of the SOP and instead precisely modeling it by formalizing its derivative security effects—namely, the constraints it imposes on the browser's automatic behaviors. Specifically, the security guarantees of the SOP are embedded within response-handling rules, implementing access control through origin-matching conditions, the core logic of which is illustrated in Fig. 7.

```
rule Response_Handler_Cookie:
let cookie = <cookiename, cookievalue, <'HTTPonly', httponly>, <'Secure', secure>>
    …
    webpage = <webpage_type, webpage_url, webpage_target_url, webpage_action>
    response = <webpage, cookie>
in
    [ Browser_Receive(old_pending_request, url, response)
    , Tab_State(tab_id, browser_id, old_webpage, old_pending_request)
    , Cookie_Storage(browser_id, $Webpage_Server, old_cookie) ]
  --[ Response_Handler_Cookie(old_pending_request)
     …]->
    [ Tab_State(tab_id, browser_id, webpage, 'NULL')
    , Cookie_Storage(browser_id, $Webpage_Server, cookie) ]
```

Figure 7: Modeling of response handling mechanism.

Our model does not directly define the SOP as an independent policy but rather reflects its effects by refining the browser's mechanism for verifying request origins. As described in Section 4.1.2, the automatic attachment of cookies is constrained by the SOP: the browser only appends credentials when the request target matches the cookie's domain. This constraint is formalized as a precondition for request rules.

This verification ensures that cross-origin requests do not automatically carry sensitive credentials, thereby deriving the core protective effect of the SOP. However, the SOP only restricts "read" operations (e.g., script access to cross-origin resources) but not "write" operations (e.g., form submissions). This critical limitation is exploited by attackers: as detailed in Section 4.1.1, a malicious webpage can induce the user to initiate a cross-site request (such as with a hidden form), triggering the browser to send a request with cookies to a same-origin target (like the RP). Although the SOP prevents the malicious script from reading the response, the request itself is still successfully sent.

The novelty of this modeling approach is that it reveals the fundamental reason why the SOP cannot prevent CSRF: the attacker does not need to violate the SOP's "read" restrictions but instead abuses its permitted "write" mechanisms. By decomposing the effects of the SOP into concrete rules (such as origin verification), our model can precisely analyze vulnerabilities arising from the interaction between the protocol and the platform, rather than relying on idealized assumptions. The same-origin check rules, working in concert with the cookie management rules, collectively constrain the browser's behavior, providing a high-fidelity foundation for the formal verification in Section 5.

4.1.4 Redirection Handler

Redirects constitute the primary execution engine of the OIDC authorization code flow, orchestrating the user agent's navigation between the RP, IdP, and back. Prior formal analyses of web protocols frequently model redirects as simple, atomic message-forwarding events, abstracting away the browser's automated, stateful processing of HTTP redirection responses. This abstraction misses a critical attack vector: the browser automatically initiates new requests based on redirect instructions, seamlessly inheriting and propagating the security context (e.g., session cookies) of the prior request. Our work introduces a novel, mechanistic formalization of redirect handling that precisely captures this automation, a key enabler for the CSRF attack in the flawed OIDC implementation. The core of this modeling is the *Redirect_Handler_Cookie* rule, whose formal specification is provided in Fig. 8.

```
rule Redirect_Handler_Cookie:
let received_cookie = <cookiename, cookievalue, <'HTTPonly', httponly>, <'Secure', secure>>
    webpage = <'Redirect', webpage_url, webpage_target_url, webpage_parameters>
    …
in
    […]
  --[…]->
    [ Browser_Send(new_pending_request, browser_id, webpage_target_url, request)
    …]
```

Figure 8: Modeling of redirect response.

Mechanistic Modeling of the Request-Response Chain. We eschew the common abstraction of a "protocol participant sending a message to another" for redirects. Instead, we model the browser as an automaton that processes HTTP responses. The *Redirect_Handler_Cookie* rule is triggered upon receiving a *webpage* fact of type 'Redirect'. The rule deconstructs this fact to extract the *webpage_target_url*. Crucially, the rule's conclusion involves the action *Browser_Send*(*new_pending_request*, *browser_id*, *webpage_target_url*, *request*). This formally captures the unsolicited, automatic generation of a new HTTP request by the browser, directed by the server's response. This automation is the driving force behind the OIDC flow: the RP's authentication request triggers a redirect to the IdP, and the IdP's authorization response triggers a redirect back to the RP's callback endpoint. By modeling this as an automatic state transition, we can trace the complete request chain and analyze its security properties, which is impossible in models where redirects are mere message sends.

Context Inheritance and Credential Propagation. A redirect is not a new, isolated session; it carries forward the security context of the initiating page. Our model integrates redirect handling with the fine-grained state management defined in Sections 4.1.1 and 4.1.2. The *Redirect_Handler_Cookie* rule's premise includes a *received_cookie* fact. The rule's name and logic imply that it handles the case where a relevant cookie exists for the *webpage_target_url*'s domain. Therefore, when the rule fires and triggers *Browser_Send*, the generated request automatically includes this cookie. This formalizes the security-critical behavior that an attacker can exploit: a redirect initiated from a malicious site (or induced by an attacker) to a trusted RP will still cause the browser to automatically attach the user's valid session cookie for that RP to the request. The rule also respects security attributes like 'Secure', ensuring cookies are only sent over HTTPS, thereby integrating channel security constraints from Section 4.1.5.

This precise modeling of redirects is indispensable for our vulnerability analysis. In the flawed OIDC flow, the attacker lures the victim to a malicious page, which triggers a request to the RP. The RP, missing the *state* parameter, issues a standard redirect to the IdP. The victim's browser, as per our *Redirect_Handler* rules, automatically follows this redirect, authenticates with the IdP, and is then redirected by the IdP back to the RP. This entire chain—spanning malicious site, RP, and IdP—is executed automatically by the browser's redirect-handling logic. Because the model tracks the propagation of the request context and the automatic cookie attachment at each step, Tamarin can formally derive that the final callback request to the RP, bearing a valid authorization code, is causally disconnected from the user's original authentication intent, yet carries the user's credentials. This conclusively proves the CSRF vulnerability arises from the broken logical link (*state*), not from a failure of the redirect or cookie mechanisms themselves, which are operating as designed. Our rule in Fig. 8 is thus the formal linchpin connecting the web platform's automation to the protocol-level flaw.

4.1.5 Security Channel

The accurate modeling of communication channels is pivotal for analyzing web-based protocols like OIDC, as channel security directly impacts the confidentiality and integrity of authentication flows. While prior formal models often treat channels as idealized abstractions or overlook the distinction between front-end and back-end communications, our work innovates by introducing a fine-grained, behaviorally accurate formalization of channel security within the Tamarin prover. This enables precise reasoning about how adversarial capabilities are constrained or enabled based on channel types, which is crucial for vulnerabilities like CSRF that exploit browser-server interactions. The core of our channel modeling is encapsulated in the formal rules depicted in Fig. 9.

```
rule Browser_To_Server_Https:
let url = <'HTTPS', $Server, $Path, $File>
in
    [ Browser_Send(request_id, browser_id, url, message)
    , !Channel($Server, browser_id, channel_id) ]
  -->
    [ Server_Receive(request_id, channel_id, url, message) ]
```

Figure 9: Modeling of the secure channel.

Insecure Front-End Channel: This channel models the browser-mediated HTTP/HTTPS communication between the user agent and servers (RP, IdP, or adversary). It is explicitly defined as adversarial-controlled, meaning all messages transmitted on it are subject to eavesdropping, injection, and modification. Formally, this is achieved by routing all front-end messages through Tamarin's built-in $Out(m)$ and $In(m)$ facts, which represent the public network accessible to the adversary. A critical refinement in our model is that we do not model the TLS handshake for HTTPS on this channel; instead, we capture its security effect through constraints. For instance, we enforce that cookies marked *Secure* = 'true' can only be sent over this channel if the request uses the HTTPS protocol, a constraint formalized as a Tamarin restriction:

restriction *secure* :

$$\forall prot, ser, path, file, \mathsf{cookiename}, cookievalue, httponly, i.$$
$$\mathsf{Cookie\_Is\_Sent}(\langle prot, ser, path, file\rangle, \langle cookiename, cookievalue, \langle \text{'HTTPonly'}, httponly\rangle,$$
$$\langle \text{'Secure'}, \text{'true'}\rangle\rangle)\ @i \Longrightarrow (prot = \text{'HTTPS'}) \quad (12)$$

This approach allows us to focus on the protocol logic without being encumbered by cryptographic details of TLS, while still accurately reflecting the web's security semantics.

Secure Back-End Channel: This channel models the direct, TLS-protected server-to-server communication (e.g., between RP and IdP for token exchange). Its security is abstracted as an ideal, tamper-proof conduit. The innovation here is the use of dedicated, private facts for communication, such as $RP\_To\_IdP(m)$ and $IdP\_To\_RP(m)$, which are not accessible via the public $In(m)/Out(m)$ interface. This formally enforces that the adversary cannot intercept or modify messages on this channel ($Decrypt(m_{BackEnd}) \notin \mathcal{A}_{cap}$ and $Tamper(m_{BackEnd}) \notin \mathcal{A}_{cap}$). The rules for token request and response (e.g., *RP_Sends_AuthCode_For_Token* and *IdP_Receives_Token_Request*) utilize these facts, ensuring that sensitive data like the *client_secret* and authorization code are transmitted confidentially and integrally.

Adversary-Controlled Channel: This is not a distinct channel but rather the adversary's ability to act as a server principal on the Front-End Channel. The adversary $\mathcal{A}$ controls its own domain and server $S_{\mathcal{A}}$, granting it full control over messages directed to its origin. This serves as the launchpad for CSRF attacks, where $\mathcal{A}$ crafts malicious pages to induce the user's browser into making unintended requests.

This holistic model, which explicitly defines the roles, their interactions, and the distinct security characteristics of their communication channels (summarized in Fig. 10), provides the necessary structure for a rigorous and precise formalization of the OIDC protocol.

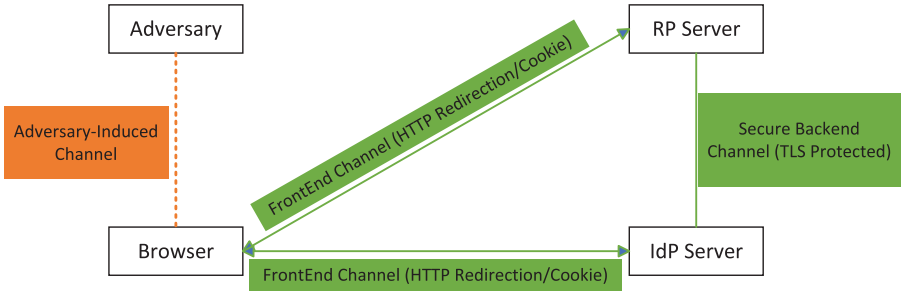

Figure 10: Diagram of communication channel modeling.

4.2 Security Attributes

Let $\mathcal{P}_{\text{OIDC}}$ denote our formal model of the OIDC authorization code flow protocol, specified in the Tamarin prover framework. The model captures the interactions among three principal roles: the RP, the IdP, and the UA.

Definition 1 (Execution Trace): *An execution trace of $\mathcal{P}_{OIDC}$ is a finite sequence: $\tau = \langle (F_1, t_1), (F_2, t_2), \ldots, (F_n, t_n)\rangle$ where each $F_i \in \mathcal{F}$ is an action fact occurring at timepoint $t_i \in \mathbb{T}$, with $\mathbb{T}$ being a totally ordered set of timepoints satisfying $t_1 < t_2 < \cdots < t_n$. We denote by $\mathsf{Traces}(\mathcal{P}_{OIDC})$ the set of all valid execution traces. $\mathcal{F}$ is the set of action facts (events) that label transitions.*

To provide a rigorous foundation for our analysis, we define two core security attributes that capture the essential security promises of the OIDC protocol. These attributes are then formalized as logical properties within the Tamarin prover's framework, transforming abstract security goals into concrete, verifiable claims.

Definition 2 (Honest OIDC Request Origin): *Let Φ_{unique} denote the conjunction of uniqueness constraints for all protocol events. The protocol $\mathcal{P}_{OIDC}$ satisfies* Honest OIDC Request Origin *if for all traces τ:*

$$\Phi_{\text{unique}} \Longrightarrow \begin{array}{l} \forall\, rp, idp, \#i.\ \mathsf{RP\_Sends\_AuthReq}(rp, idp)@i \\ \qquad \Longrightarrow \exists\, bid, \#j.\ \mathsf{User\_Starts}(rp, idp, bid)@j \wedge j < i \end{array} \tag{13}$$

This property asserts that every authorization request to the RP must originate from a legitimate user action. A counterexample to this property demonstrates the existence of a CSRF attack vector.

The primary security promise of any authentication protocol is authentication agreement, which guarantees that a resulting session is a faithful outcome of the user's intent. In the context of OIDC, this means a session established at the RP must have been preceded by a genuine, user-initiated request. We formalize this fundamental property using a temporal logic lemma (as shown in Fig. 11), which asserts a causal and chronological dependency: for every event where an RP claims to complete an authentication ($RP\_Sends\_AuthReq$), there must have existed a prior event where the user explicitly initiated that process ($User\_Starts\_OIDC$). This formalization directly targets the core threat of CSRF. A violation of this lemma is not merely a procedural error; it is a formal proof that the system allows an attacker to forge a user's identity and establish a session without their consent.

```
lemma honest_oidc_request_origin:
"All rp idp #i. RP_Sends_AuthReq(rp, idp) @i
 ==> (Ex browser_id #j. User_Starts_OIDC2(rp, idp, browser_id) @j & #j < #i)"
```

Figure 11: Formal analysis program of authentication agreement.

Definition 3 (Access Token Exchange Completeness): *The protocol $\mathcal{P}_{OIDC}$ satisfies* Access Token Exchange Completeness *if there exists a valid trace τ such that:*

$$\exists\, rp, idp, code, token, \#i, \#j, \#k.\ \begin{array}{l} \mathsf{RP\_Sends\_Code}(rp, idp, code)@i \wedge \\ \mathsf{IdP\_Recv\_Token\_Req}(idp, rp, code)@j \wedge \\ \mathsf{RP\_Recv\_Token}(rp, idp, token)@k \ \wedge\ i < j < k \end{array} \tag{14}$$

This property ensures the complete token exchange workflow executes in the correct temporal order.

Beyond ensuring correct intent, a secure protocol must also guarantee correct process. This is captured by our second attribute execution integrity. This property asserts that the protocol's security does not rely on wishful thinking but on the verifiable completion of all critical security-relevant steps. Instead of a single overarching claim, we define this through a series of executability lemmas that serve as checkpoints throughout the protocol flow. These lemmas demand proof that complete paths exist for essential sub-protocols, such as $rp\_registration\_complete$ and the secure token exchange ($access\_token\_exchange\_complete$, as shown in Fig. 12). Furthermore, they enforce the integrity of cryptographic objects through claims like $id\_token\_content\_integrity$ and $digital\_signature\_authenticity$. Together, these lemmas ensure that a successful protocol run is not just one that reaches the end but one that has passed through all the necessary security gates.

```
lemma access_token_exchange_complete:
exists-trace
"Ex rp idp authcode access_token #i #j #k.
  (RP_Sends_AuthCode_For_Token(rp, idp, authcode) @ i &
   IdP_Receives_Token_Request(idp, rp, authcode) @ j &
   RP_Receives_AccessToken(rp, idp, access_token) @ k &
   #i < #j & #j < #k)"
```

Figure 12: Formal analysis program of access token exchanging.

By formalizing these properties, we establish precise and unambiguous criteria for success and failure. The Tamarin prover's task is then to act as an exhaustive adversary, systematically exploring every possible execution trace allowed by our model. If it finds a path that violates our defined properties—for instance, an authentication agreement without user initiation—it has discovered a concrete attack. If all possible paths uphold these properties, we gain strong formal assurance of the model's security.

5 Security Evaluation

5.1 Experimental Setup

To ensure the reproducibility and transparency of our formal analysis, this section details the experimental configuration used for the Tamarin prover. The verification was conducted on a server equipped with an Intel Xeon Gold 6248R CPU (24 cores, 48 threads @ 3.00 GHz) and 128 GB of RAM, running Ubuntu 20.04.4 LTS.

We used Tamarin Prover version 1.8.0 for the analysis. Our formal model of the flawed OIDC implementation consists of approximately 50 multiset rewriting rules, defining the protocol participants' behavior, web framework interactions, and the adversary's capabilities. The primary security property, formalized as the honest_oidc_request_origin lemma (as shown in Fig. 11), was verified using Tamarin's standard automated proof search command: tamarin-prover –prove=honest_oidc_request_origin FlawedOIDC.spthy. The analysis was executed in parallel on 16 threads and completed in approximately 3.5 h, during which Tamarin successfully found the counterexample detailed in the following section.

5.2 Results

5.2.1 Automated Verification and Attack Trajectory

After completing the definition of security attributes and configuring Tamarin analysis, we conducted an automated formal verification of the flawed OIDC model for the missing *state* parameter validation as constructed in Section 4.1. The verification was carried out based on the consistency lemma of authentication defined in Section 4.2. The Tamarin prover was set to search for any counterexamples that violated this lemma in all possible protocol execution paths. The analysis results (as shown in Fig. 13) indicated that the consistency lemma did not hold in this flawed model. Tamarin successfully found a counterexample, namely a specific attack trajectory, proving that the adversary could prompt the RP to establish a session without satisfying the "user authentication intention first" condition.

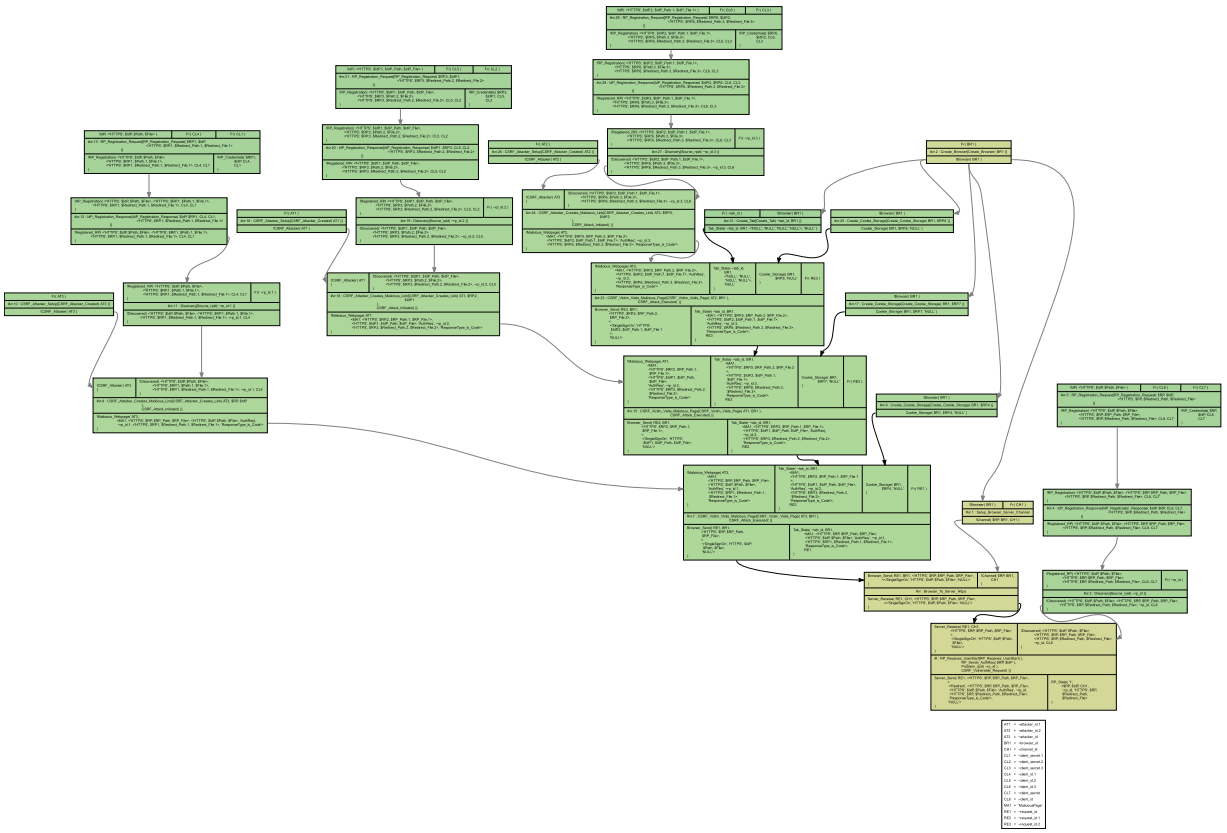

Figure 13: Part of formal analysis results of authentication agreement.

The existence of this attack trajectory conclusively confirmed from a formal logical perspective that there is a exploitable CSRF attack vulnerability in the OIDC implementation with missing *state* parameter validation. The counterexample trajectory output by Tamarin precisely reproduced a complete attack process. This trajectory shows that the adversary induces authenticated users to initiate cross-site requests through a malicious page, and the user's browser completes the legal authentication under the adversary's inducement and sends an effective authorization code to the RP's redirect endpoint. Due to the missing *state* parameter validation in the RP, the malicious redirect request constructed by the adversary was wrongly accepted. The RP established a user session based on the adversary-controlled request, resulting in session hijacking.

The discovery of this attack trajectory conclusively confirmed the existence of the CSRF vulnerability from the computational logic perspective. It is particularly noteworthy that the attack path was derived completely under the symbolic model of Tamarin, without relying on any probabilistic attacks or additional security assumptions. This proves the inevitability of the vulnerability: as long as the *state* parameter validation is missing in the RP implementation, in any web environment that conforms to the model assumptions, the adversary can achieve the attack through the aforementioned path.

5.2.2 Attack Path and Principles

Before dissecting the formal attack trace, it is helpful to first walk through the attack scenario in natural language to build intuition. The core idea of the attack is not to steal the victim's credentials, but to trick the victim's browser into performing a legitimate OIDC authentication flow on the attacker's behalf. The missing

state parameter is the critical enabler, as it prevents the RP from distinguishing a user-initiated request from an attacker-induced one. The attack unfolds as follows:

1. **The Bait:** An attacker convinces a victim, who is already logged into a legitimate service (the RP), to visit a malicious website controlled by the attacker.
2. **The Hidden Request:** On this malicious site, a hidden script automatically triggers an OIDC authentication request to the legitimate RP. From the RP's perspective, this request appears to come from the victim's browser.
3. **The Browser's Compliance:** The victim's browser, following standard behavior, automatically attaches the victim's valid session cookie to this outgoing request to the RP.
4. **The Protocol's Flaw:** The RP receives the request with a valid cookie but, crucially, without a *state* parameter to verify its origin. The RP mistakenly assumes the request is legitimate and initiates the standard OIDC redirect flow to the IdP.
5. **The Hijacked Authentication:** The browser, now at the IdP, authenticates seamlessly (as the victim is already logged in there too) and receives a valid authorization code, which is sent back to the RP's redirect endpoint. The attacker, having initiated this flow, intercepts this authorization code.
6. **The Takeover:** The attacker now uses this valid authorization code—which is tied to the victim's identity—to complete the authentication process, successfully linking their own session to the victim's account on the RP. This results in a complete session hijack.

Based on the counterexample trace discovered by the Tamarin prover and in combination with the attack flowchart shown in Fig. 14, this subsection conducts a deep analysis of the CSRF attack path caused by the missing *state* parameter. This attack path clearly reveals how an adversary can exploit the aforementioned protocol flaws to bypass security mechanisms, and the process can be divided into four key stages.

a. **Attack Preparation** (Steps 1–2)

The adversary first constructs a link containing malicious OIDC requests (Step 1), and induces users to click on it through social engineering tactics (Step 2). The request points to the legitimate RP authentication endpoint, but includes parameters controlled by the adversary. Due to the complete absence of the *state* parameter in the RP implementation, it is impossible to establish a verifiable session context for this request, laying the foundation for the subsequent attack.

b. **Protocol Interaction Hijackin**g (Steps 3–11)

When the user clicks on the malicious link, their browser will initiate an authentication request to the RP (Step 3). Since the user may maintain the RP's login status in the browser, the browser will automatically carry valid session cookies. When the RP processes this request, due to the lack of *state* parameter verification, it cannot distinguish whether this is an active request from the user or an induced request by the adversary, continuing the normal process and forwarding the authentication request to the IdP (Steps 4–6).

After the IdP receives the request, since the user may have maintained the IdP's login status (Steps 14–15) in the browser, it skips the authentication step and directly returns the authorization code (Steps 7–8). The key point of this stage is that the adversary indirectly triggers the complete OIDC process by inducing the user to click the link, but the actual control of the entire process is in the hands of the adversary.

c. **Authorization Code Hijacking and Confusion** (Steps 12–18)

The core of the attack lies in Steps 12–18. The adversary initiates another authorization request using the same browser environment, but with a different channel identifier (attacker channel). Due to the lack of *state* parameter binding, the RP cannot identify this as a new session unrelated to the previous request. When the IdP processes this request, based on the existing authentication session, it directly

returns the authorization code (Steps 16–18), but this authorization code is transmitted through the channel controlled by the adversary.

d. **Attack Completion** (Steps 19–20)

After the RP receives the authorization code in Step 19, since it cannot verify through the *state* parameter whether the authorization code corresponds to the original request that is legal, it mistakenly associates it with the user's initial session (Step 19). Eventually, the RP completes the token exchange based on the authorization code provided by the adversary and establishes a session for the adversary (Step 20). This action constitutes the final authentication forgery, where the victim's authenticated session is hijacked by the attacker.

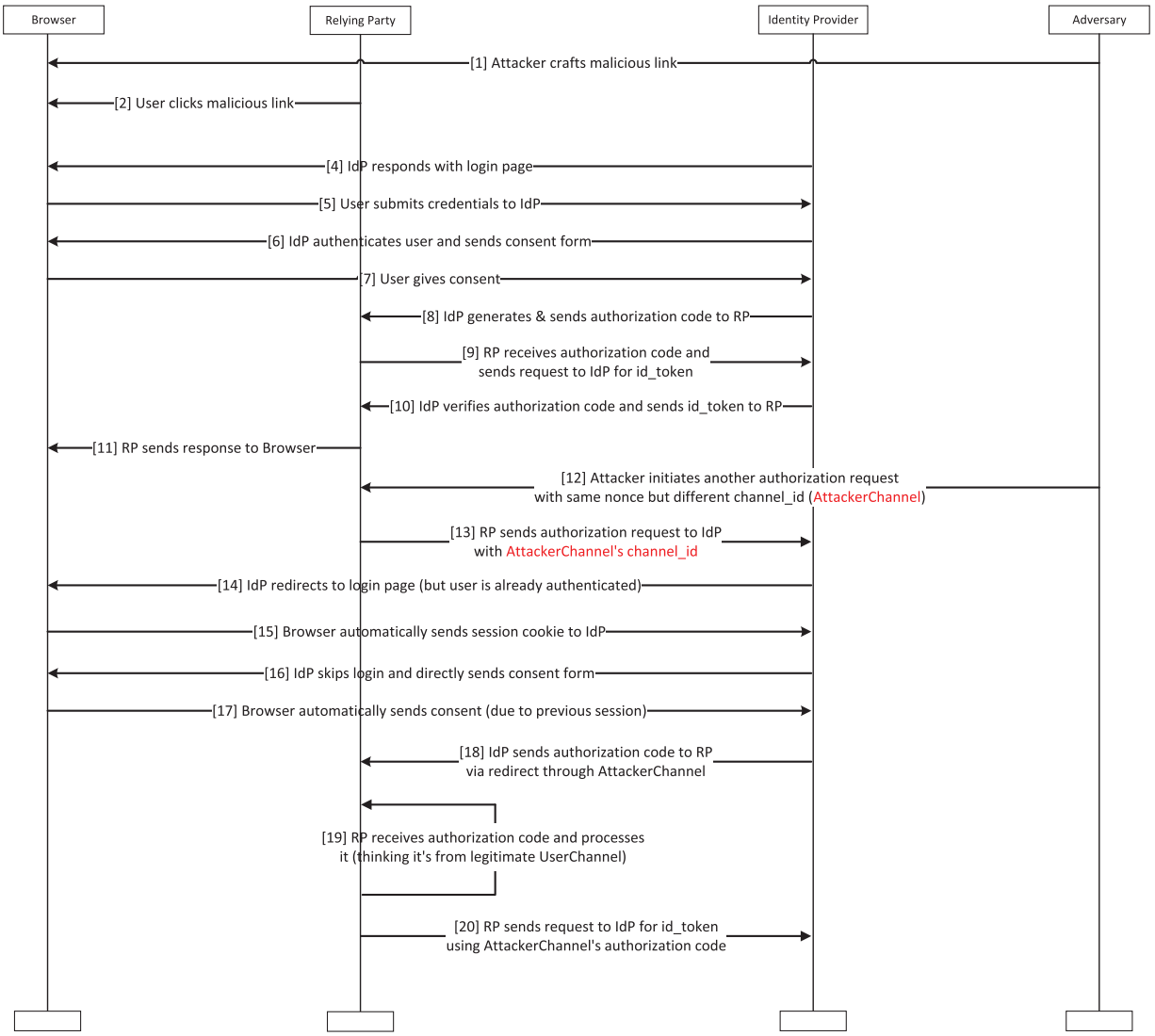

Figure 14: Flow chart of CSRF attack [1,2,4–20].

The fundamental reason for the success of this attack lies in the fact that the binding relationship between the authentication request and the authorization response is entirely maintained by the *state* parameter. Without this parameter, the RP loses the following key capabilities:

- **Source verification:** Unable to distinguish between requests initiated by users and those induced by adversaries.
- **Session association:** Unable to correctly associate authorization responses with specific authentication requests.
- **Intention confirmation:** Unable to confirm whether the authorization process originated from the user's explicit authentication intention.

The essence of an attack lies in the adversary's abuse of the browser's automatic authentication mechanism (automatically carried by session cookies) and the logical flaws of the protocol, using the user's legitimate identity credentials to hijack them and utilize them in the protocol process controlled by the attacker. This type of attack is carried out entirely within the scope permitted by web standards, without relying on any network layer attacks or cryptographic cracking, thus having extremely high feasibility and threat potential in real-world scenarios.

5.3 Proposed Fixes

In response to the CSRF vulnerability identified by our formal analysis, this section proposes a remedial solution. The vulnerability's root cause is the broken link between an authentication request and its corresponding authorization response. Therefore, the remedy lies in correctly implementing the *state* parameter verification mechanism as specified by the OIDC standard, thereby re-establishing this critical binding. The core principle is to use the *state* parameter as a unique, unforgeable token that binds the user's session to a specific authentication flow. The effectiveness of this patched approach is not just a theoretical claim; we have formally verified it, as detailed in the following section.

5.4 Formal Verification of the Patched Model

To rigorously validate our proposed fix, we create a patched version of our formal model and subjected it to the same verification process using the Tamarin prover. This section details the model modifications, the verification process, and the results. The patched model incorporates the *state* parameter's lifecycle as follows:

1. **State Generation and Storage:** We modify the *RP_Initiates_Auth* rule. When an authentication request begins, the RP now generates a fresh, unpredictable value for the *state* parameter (using Tamarin's *Fr* fact for freshness). This value is stored in a new fact, *RP_SessionState(session_id, user, state_value)*, effectively binding it to the user's session context.

2. **State Transmission:** The generated *state_value* is included in the authentication request message sent to the IdP and subsequently returned by the IdP in the authorization response.

3. **State Verification:** We introduce a critical premise in the *RP_Receives_AuthCode* rule. For the rule to fire (i.e., for the RP to accept the authorization code), the state parameter receive in the redirect must exactly match the *state_value* stored in the *RP_SessionState* fact associated with the current session. If no matching state value is found, or if it doesn't match, the protocol execution for that trace halts, modeling a failed verification.

After an exhaustive search of all possible execution traces (completed in approximately 4.1 hours on the same hardware), Tamarin concluded with the result: *"All traces proved"*. This confirms that no counterexample exists for the security property in the patched model. The strict verification of the *state* parameter successfully breaks the attack path found in the flawed model. Any CSRF-induced request from an attacker, lacking the correct, session-bound *state* value, is now rejected by the RP, thus preventing the authentication forgery. This formal proof provides strong assurance that the correct implementation of the

state parameter is not only a recommended practice but a sufficient countermeasure against this class of CSRF attacks in the OIDC flow.

6 Discussion and Outlook

6.1 Security Recommendations

Based on our formal analysis, we offer key recommendations for translating these security findings into robust engineering practices for OIDC implementations. These guidelines are designed to proactively mitigate risks like CSRF.

The primary and most critical recommendation is the unwavering enforcement of *state* parameter verification. Our research confirms that this parameter is the central defense mechanism against CSRF in the OIDC flow. Implementations must treat its handling as non-negotiable. This involves two mandatory server-side actions: first, for every authentication request, the RP must generate a cryptographically strong, unpredictable *state* value and bind it to the user's session. Second, upon receiving the authorization redirect, the RP must strictly validate that the returned *state* value matches the one stored in the session. This verification must be an integral part of the core code path that cannot be bypassed, and any failure should result in the immediate termination of the flow and a logged security event.

Beyond this core protocol requirement, we advocate for a defense-in-depth strategy to enhance overall security. While the *state* parameter is essential, a resilient system should employ multiple layers of protection. This includes strengthening session management by setting the $SameSite = Strict$ or $SameSite = Lax$ attribute on session cookies, which provides a powerful browser-level mitigation against many cross-site attacks. Furthermore, implementing strong client authentication mechanisms, such as using $client\_secret$ or private key $JWTs$, is crucial for verifying the identity of the RP itself, preventing attackers from impersonating legitimate client applications. By combining strict protocol adherence with a layered defense approach, developers can significantly elevate the security posture of their OIDC-based SSO systems.

6.2 Research Limitations and Future Work

This study reveals the CSRF vulnerability caused by the absence of the *state* parameter in the implementation of the OIDC protocol through formal methods, and verifies the effectiveness of the patching solution. However, it still has certain limitations. Firstly, the formal model constructed in this study simplifies the real environment unnecessarily. Although it models the key mechanisms of the web platform (such as cookie management and same-origin policy) in detail, it assumes that the core security mechanisms of the browser are correctly implemented and not bypassed. It does not consider the impact of browser vulnerabilities or unexpected user behaviors. Additionally, the model mainly focuses on the classic web browser-server interaction scenario and does not fully cover the more complex communication patterns (such as authorization code transmission through web messaging or inter-application communication) in emerging architectures like Single-page Applications (SPA) and native mobile applications. The security assumptions and threat models in these environments may differ from those in the classical web scenarios.

Secondly, the analytical focus of this study is on the CSRF attack caused by the specific implementation defect of ignoring the *state* parameter. It has not systematically covered other types of security risks that may exist in real deployments, such as vulnerabilities related to token management (such as token leakage, token replay), open redirect attacks related to lax redirect URI verification, and other logical flaws that may occur in the OAuth 2.0 authorization code flow. These additional attack vectors are worthy of further exploration in future research.

Finally, the formalization method itself has certain limitations. Although the Tamarin prover is powerful, its analysis ability is constrained by the problem of state space explosion. Although this study controls the state space through careful model abstraction, for extremely complex or parameterized systems, fully automated verification still faces scalability challenges. Additionally, the results of formal verification are highly dependent on the accuracy of the model definition. If the model fails to fully capture certain subtle semantics or platform characteristics of the protocol, the analysis conclusions may be biased.

Based on these limitations, future research work can be carried out in the following directions: extend the formal model to emerging application scenarios such as mobile devices and SPA, and conduct detailed modeling and analysis of the specific processes defined by OAuth 2.0 and OIDC for these client types; define and verify more comprehensive security attributes, such as enhanced privacy attributes, revocability and confidentiality of tokens, to conduct a more comprehensive assessment of protocol security; explore paths to more closely integrate formal analysis tools with development practices, such as developing prototype tools that can automatically extract abstract models from common implementation code or perform lightweight verification, to enhance the practical value of formal methods in the software development life cycle. By addressing these limitations and challenges, formal methods are expected to play a more extensive and in-depth guiding role in ensuring the security of web SSO systems.

7 Conclusion

This research focused on the security issues of the OIDC SSO protocol under the web architecture, particularly the CSRF vulnerability that arises from the implementation flaw of omitting the *state* parameter. A series of systematic formal modeling and analysis work was carried out. The research first constructed a refined formal model of the web architecture, which accurately depicted the interaction behaviors of the browser, the server, and the key web security mechanisms. On this basis, the formal specification of the OIDC implementation flaw—the omission of the *state* parameter—was emphasized. Through the automated analysis of the Tamarin prover, the existence of the CSRF vulnerability under this flawed model was successfully discovered and strictly verified. From the formal logical perspective, the inevitability of the vulnerability was revealed. In response to this vulnerability, a strict patch solution following the OIDC specification was proposed, and the patched model was reconstructed and verified. The results confirmed that the patch solution can effectively eliminate the attack path and restore the security of the protocol.

The main contributions of this research lie in: Firstly, it confirmed the effectiveness of refined Web modeling in discovering deep vulnerabilities arising from the interaction between the protocol and the platform; Secondly, it provided a machine-verifiable strict proof for a specific and common implementation defect of the OIDC protocol and its patch solution, surpassing the traditional empirical analysis depth; Finally, it provided key suggestions and theoretical basis for the secure implementation of the protocol. The research work highlights the important value of formal methods in enhancing the security of single sign-on systems.

Acknowledgement: We acknowledge the use of Gemini 2.5 Pro (Google, Mountain View, USA; https://gemini.google.com (accessed on 21 December 2025)) for limited language polishing and refinement during the preparation of this manuscript. All scientific content and conclusions are entirely the responsibility of the authors.

Funding Statement: This work is supported by National Key Research and Development Program of China No. 2023YFB2705000, Blockchain System Security Key Technology Research of Henan Province Major Public Welfare Project No. 201300210200 and National Engineering Laboratory for Big Data Distribution and Exchange Technologies.

Author Contributions: The authors confirm their contribution to the paper as follows: Xingyun Hu: Conceptualization, Investigation, Formal analysis, Writing—original draft, Project administration. Siqi Lu: Conceptualization,

Writing—original draft, Supervision. Liujia Cai: Investigation, Writing—review & editing, Formal analysis, Project administration. Ye Feng: Writing—original draft, Project administration. Shuhao Gu: Conceptualization, Writing—review & editing. Tao Hu: Project administration, Writing—review & editing. Yongjuan Wang: Investigation, Writing—review & editing, Supervision. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Fett D, Küsters R, Schmitz G. A comprehensive formal security analysis of OAuth 2.0. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; 2016 Oct 24–28; Vienna, Austria. p. 1204–15.
2. Sakimura N, Bradley J, Jones M, De Medeiros B, Mortimore C. OpenID connect core 1.0 incorporating errata set 1. The OpenID Foundation Specification. 2014;335:1–73.
3. Bertocci V, Chiarelli A. OAuth2 and OpenID connect: the professional guide-beta. Bellevue, WA, USA: Auth0; 2020.
4. Ferry E, O Raw J, Curran K. Security evaluation of the OAuth 2.0 framework. Inf Comput Sec. 2015;23:73–101.
5. Al Shabi M, Marie RR. Analyzing privacy implications and security vulnerabilities in single sign-on systems: a case study on OpenID Connect. Int J Adv Comput Sci Appl. 2024;15(4):637–46. doi:10.14569/ijacsa.2024.0150465.
6. Sun S-T, Beznosov K. The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security; 2012 Oct 16–18; Raleigh, NC, USA. p. 378–90.
7. Li W, Mitchell CJ. Security issues in OAuth 2.0 SSO implementations. In: International Conference on Information Security. Cham, Switzerland: Springer International Publishing; 2014. p. 529–41.
8. Arshad E, Benolli M, Crispo B. Practical attacks on Login CSRF in OAuth. Comput Sec. 2022;121(4):102859. doi:10.1016/j.cose.2022.102859.
9. Benolli M, Mirheidari SA, Arshad E, Crispo B. The full gamut of an attack: an empirical analysis of OAuth CSRF in the wild. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Cham, Switzerland: Springer International Publishing; 2021. p. 21–41.
10. Likaj X, Khodayari S, Pellegrino G. Where we stand (or fall): an analysis of CSRF defenses in web frameworks. In: Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses; 2021 Oct 6–8; San Sebastian, Spain. p. 370–85.
11. Mainka C, Mladenov V, Schwenk J, Wich T. SoK: single sign-on security—an evaluation of openID connect. In: Proceedings of the 2017 IEEE European Symposium on Security and Privacy; 2017 Apr 26–28; Paris, France. p. 251–66.
12. Morkonda SG, Chiasson S, van Oorschot PC. 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; 2021 Nov 15; Seoul, Republic of Korea. p. 195–208.
13. Fett D, Küsters R, Schmitz G. The web SSO standard openid connect: in-depth formal security analysis and security guidelines. In: Proceedings of the 2017 IEEE 30th Computer Security Foundations Symposium; 2017 Aug 21–25; Santa Barbara, CA, USA. p. 189–202.
14. Qiu K, Liu Q, Liu J, Yu L, Wang Y. An empirical study of OAuth-Based SSO system on web. In: Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications; 2018 Jun 20–22; Tianjin, China. p. 400–11.
15. Soleimani H, Hadavi MA, Bagherdaei A. WAVE: black box detection of XSS, CSRF and information leakage vulnerabilities. In: Proceedings of the 2017 14th International Iranian Society of Cryptology Conference on Information Security and Cryptology; 2017 Jun 6–7; Shiraz, Iran. p. 19–24.

16. Calzavara S, Conti M, Focardi R, Rabitti A, Tolomei G. Mitch: a machine learning approach to the black-box detection of CSRF vulnerabilities. In: Proceedings of the 2019 IEEE European Symposium on Security and Privacy; 2019 Jun 17–19; Stockholm, Sweden. p. 528–43.
17. Basin D, Cremers C, Dreier J, Sasse R. Symbolically analyzing security protocols using tamarin. ACM SIGLOG News. 2017;4(4):19–30. doi:10.1145/3157831.3157835.
18. Naresh S, Jevitha K. Formal analysis of openid connect protocol using tamarin prover. In: International Conference on Advances in Electrical and Computer Technologies. Singapore: Springer Nature Singapore; 2020. p. 297–309.
19. Hofmeier X. Formal analysis of web single-sign on protocols using Tamarin [bachelor's thesis]. Zurich, Switzerland: Swiss Federal Institute of Technology; 2019.
20. Baelde D, Delaune S, Jacomme C, Koutsos A, Moreau S. An interactive prover for protocol verification in the computational model. In: 2021 IEEE Symposium on Security and Privacy; 2021. p. 537–54.
21. Dünki S. Modelling and analysis of web applications in tamarin [master thesis]. Zürich, Switzerland: ETH Zurich; 2019.
22. Fett D, Küsters R, Schmitz G. An expressive model for the web infrastructure: definition and application to the browser id SSO system. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy; 2014 May 18–21; SAN JOSE, CA, USA. p. 673–88.
23. Bansal C, Bhargavan K, Delignat-Lavaud A, Maffeis S. Discovering concrete attacks on website authorization by formal analysis. J Comput Sec. 2014;22(4):601–57. doi:10.3233/jcs-140503.
24. Bansal C, Bhargavan K, Delignat-Lavaud A, Maffeis S. Keys to the cloud: formal analysis and concrete attacks on encrypted web storage. In: International Conference on Principles of Security and Trust. Berlin/Heidelberg, Germany: Springer; 2013. p. 126–46.
25. Akhawe D, Barth A, Lam PE, Mitchell J, Song D, editors. Towards a formal foundation of web security. In: Proceedings of the 2010 23rd IEEE Computer Security Foundations Symposium; 2010 Jul 17–19; Edinburgh, UK. Edinburgh, UK. p. 290–304.
26. Bugliesi M, Calzavara S, Rabitti A. Cryptographic web applications: from security engineering to formal analysis. In: Handbook of Formal Analysis and Verification in Cryptography. Boca Raton, FL, USA: CRC Press; 2023. p. 275–318.
27. Basin D, Dreier J, Hirschi L, Radomirovic S, Sasse R, Stettler V. A formal analysis of 5G authentication. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security; 2018 Oct 15–19; Toronto, ON, Canada. p. 1383–96.
28. Singh J, Chaudhary N. OAuth 2.0: architectural design augmentation for mitigation of common security vulnerabilities. J Inform Sec Appl. 2022;65:103091.
29. Li CT, Lee CC, Weng CY, Fan CI. An extended multi-server-based user authentication and key agreement scheme with user anonymity. KSII Trans Internet Inf Syst. 2013;7(1):119–31. doi:10.3837/tiis.2013.01.008.
30. Lee CC, Li CT, Chen SD. Two attacks on a two-factor user authentication in wireless sensor networks. Parallel Process Letters. 2011;21(1):21–6. doi:10.1142/s0129626411000047.
31. Philippaerts P, Vanhoof J, Van Cutsem T, Joosen W. Is your OAuth middleware vulnerable? Evaluating open-source identity providers' security. In: GLOBECOM 2024-2024 IEEE Global Communications Conference. Piscataway, NJ, USA: IEEE; 2024. p. 3607–12.
32. Innocenti T, Golinelli M, Onarlioglu K, Mirheidari A, Crispo B, Kirda E. Oauth 2.0 redirect URI validation falls short, literally. In: Proceedings of the 39th Annual Computer Security Applications Conference; 2023 Dec 4–8; Austin, TX, USA. p. 256–67.
33. Pai S, Sharma Y, Kumar S, Pai RM, Singh S. Formal verification of OAuth 2.0 using alloy framework. In: Proceedings of the 2011 International Conference on Communication Systems and Network Technologies; 2011 Jun 3–5; Jammu, India. p. 655–9.
34. Bugliesi M, Calzavara S, Focardi R. Formal methods for web security. J Log Algebr Methods Program. 2017;87(2):110–26. doi:10.1016/j.jlamp.2016.08.006.
35. Dowling B, Fischlin M, Günther F, Stebila D. A cryptographic analysis of the TLS 1.3 handshake protocol. J Cryptol. 2021;34(4):37. doi:10.1007/s00145-021-09384-1.

36. Hartl Z, Derek A. Towards automated formal security analysis of SAML V2. 0 web browser SSO standard—the POST/Artifact use case. IEEE Access. 2025;13:180126–180144. doi:10.1109/access.2025.3622379.
37. Do QH, Hosseyni P, Küsters R, Schmitz G, Wenzler N, Würtele T. A formal security analysis of the W3C web payment APIs: attacks and verification. In: 2022 IEEE Symposium on Security and Privacy. Piscataway, NJ, USA: IEEE; 2022. p. 215–34.
38. Primbs J, Menth M. OIDC^2: open identity certification with OpenID connect. IEEE Open J Commun Soc. 2024;5:1880–98.
39. Ali A, Lin YD, Liu J, Huang CT. The universal federator: a third-party authentication solution to federated cloud, edge, and fog. J Netw Comput Appl. 2024;229:103922.
40. Diaz Rivera JJ, Akbar W, Khan TA, Muhammad A, Song WC. Secure enrollment token delivery mechanism for zero trust networks using blockchain. IEICE Trans Commun. 2023;106(12):1293–301. doi:10.1587/transcom.2022tmp0005.
41. Fugkeaw S, Rattagool S, Jiangthiranan P, Pholwiset P. FPRESSO: fast and privacy-preserving SSO authentication with dynamic load balancing for multi-cloud-based web applications. IEEE Access. 2024;12:157888–900.
42. Alsumayt A, Almalki A, Almushraf F, Almansori H, Alfaraj L, Almulla S, et al. RASID: a secure UAV-based platform for intelligent traffic accident assessment with cryptographic verification and AI-driven analysis. Front Comput Sci. 2025;7:1709565. doi:10.3389/fcomp.2025.1709565.
43. Jillepalli AA, Conte de Leon D, Steiner S, AlvesFoss J. Analysis of web browser security configuration options. KSII Trans Internet & Inf Syst. 2018;12(12):6139–60. doi:10.3837/tiis.2018.12.028.
44. Kim S, Lee S, Kim M, Kwon Y. A web application framework for battery health prediction in industrial IoT networks. J Web Eng. 2025;24(6):943–72. doi:10.13052/jwe1540-9589.2464.
45. Schmitz G. Privacy-preserving web single sign-on: Formal security analysis and design. It-Inf Technol. 2022;64(1–2):43–8.
46. Li W, Mitchell CJ. User access privacy in OAuth 2.0 and OpenID connect. In: 2020 IEEE European Symposium on Security and Privacy Workshops. Piscataway, NJ, USA: IEEE; 2020. p. 664–6732.
47. He J, Lei L, Wang Y, Wang P, Jing J. ARPSSO: an OIDC-compatible privacy-preserving SSO scheme based on RP anonymization. In: European Symposium on Research in Computer Security. Cham, Switzerland: Springer Nature; 2024. p. 268–88.