Stealing Trust: Unraveling Blind Message Attacks in Web3 Authentication

Kailun Yan*
School of Cyber Science and
Technology, Shandong University
Qingdao, China
kailun@mail.sdu.edu.cn

Xiaokuan Zhang†
Department of Computer Science,
George Mason University
Fairfax, VA, United States
xiaokuan@gmu.edu

Wenrui Diao†
School of Cyber Science and
Technology, Shandong University
Qingdao, China
diaowenrui@link.cuhk.edu.hk

ABSTRACT

As the field of Web3 continues its rapid expansion, the security of Web3 authentication, often the gateway to various Web3 applications, becomes increasingly crucial. Despite its widespread use as a login method by numerous Web3 applications, the security risks of Web3 authentication have not received much attention. This paper investigates the vulnerabilities in the Web3 authentication process and proposes a new type of attack, dubbed *blind message attacks*. In blind message attacks, attackers trick users into blindly signing messages from target applications by exploiting users' inability to verify the source of messages, thereby achieving unauthorized access to the target application. We have developed Web3AuthChecker, a dynamic detection tool that interacts with Web3 authentication-related APIs to identify vulnerabilities. Our evaluation of real-world Web3 applications shows that a staggering 75.8% (22/29) of Web3 authentication deployments are at risk of blind message attacks. In response to this alarming situation, we implemented Web3AuthGuard on the open-source wallet MetaMask to alert users of potential attacks. Our evaluation results show that Web3AuthGuard can successfully raise alerts in 80% of the tested Web3 authentications. We have responsibly reported our findings to vulnerable websites and have been assigned two CVE IDs.

CCS CONCEPTS

• **Security and privacy** → **Authentication**.

KEYWORDS

Web3, Authentication, Blockchain, Security

---

*Part of Kailun Yan's work was done when visiting George Mason University.
†Corresponding authors.

---

1 INTRODUCTION

Blockchain technology has been evolving rapidly since its first introduction by Satoshi Nakamoto in 2008 [51]. Web3 applications, taking advantage of the decentralized nature of blockchain technology, have garnered significant attention from both investors and users [45]. Decentralized Finance, or DeFi, is a prominent example of the Web3 application, and its Total Value Locked (TVL) was around 52 billion USD as of December 2023 [70]. The total value of the Non-Fungible Token (NFT) marketplaces has exceeded two billion USD [20]. In January 2024, the total value of assets in the Web3 game *Midas Miner* was over one billion USD [22]. The considerable financial activities underscore the enormous potential of the Web3 ecosystem.

Web3 applications need to perform user authentication to grant the correct permissions for end users to access off-chain resources. Web3 authentication is a challenge-response protocol where the user is identified by a public key (wallet address): the Web3 application sends a specific message to the crypto wallet, signed by the user, and sent back to the application. The application verifies the signature and authenticates the user if it is valid [74]. This decentralized approach differs markedly from the traditional authentication protocols of Web2. The most common way to perform Web3 authentication is through MetaMask [3], a famous Web3 wallet for users to manage their digital assets and perform DeFi transactions. As per statistics, MetaMask had 30 million monthly active users [25]. Since the Web3 application is fundamentally a website, in this paper, we use the terms *application* and *website* interchangeably.

Due to the popularity of Web3 and the large amount of money involved, the security of Web3 is paramount. Existing research mainly focuses on the security risks associated with Web3 applications. These include vulnerabilities in smart contracts [14, 19, 27, 33, 60, 61, 63–65, 81], malicious DeFi attacks [10, 11, 21, 49, 58, 72, 82, 83], and the security of NFT marketplaces [23, 76]. The security and anonymity of crypto wallets [62, 71, 78], as well as cryptocurrency scams [32, 42] have also raised concerns. However, to the best of our knowledge, almost no work has paid attention to the security of Web3 authentication.

To bridge the gap, this paper takes the *first* step toward understanding the security risks of Web3 authentication. We find that during the Web3 authentication process, there is no proper way for the user to identify the origin of the message to be signed. Also, users use crypto wallets to manage key pairs, and one key pair is often used across multiple applications [18]. As a result, a malicious application A can present a valid signing message from application B for the user to sign (assuming the user uses the same key pair on both applications, which is very common [18]). The user

blindly signs the message, unwittingly granting application A the authorization to access application B. We call such attacks **Blind Message Attacks**. We have systematically analyzed the attack surfaces during the Web3 authentication process, which leads to the identification of several categories of potential vulnerabilities in the message design and server verification.

Web3AuthChecker. To detect Blind Message Attacks, we have designed and implemented Web3AuthChecker, a dynamic detection tool that examines the security of the Web3 authentication process. Web3AuthChecker bypasses the web page (front-end) and directly tests the API (back-end). It requests Web3 authentication-related APIs and checks the responses to find vulnerabilities. The tool consists of two modules: 1) *Checker*, which defines a set of general attack payloads and rules for testing APIs to find potential vulnerabilities, and 2) *FlexRequest*, an HTTP library designed to streamline testing identical APIs on different websites. Specifically, to test a website, we load the authentication-related APIs of the website to *Checker*. *Checker* then instructs *FlexRequest* to make a series of requests to these APIs. Finally, *Checker* examines the data returned by *FlexRequest* to identify potential vulnerabilities.

To evaluate the impact of Blind Message Attacks, we collected 29 real-world cases of Web3 authentication, including 25 cases of using Web3 authentication for user login and 4 cases for profile update. The 29 cases are from 27 websites, with some encompassing both types. The websites we studied, which span marketplaces, games, and services, were sourced from DappRadar and Google searches. In January 2024 alone, these sites reported a total transaction volume exceeding 592 million US dollars and over 1.29 million unique active wallets (UAW), underscoring the widespread nature of these vulnerabilities. By testing the APIs of 29 cases, Web3AuthChecker reports that 75.8% (22/29) of Web3 authentications belonging to 20 websites are vulnerable to Blind Message Attacks. We manually checked all 29 cases and confirmed that the results were accurate. Building upon the foundation of the Blind Message Attack, we have developed more advanced attacks: the Replay Attack renders the session mechanism ineffective, while the Blind Multi-Message Attack allows attackers to acquire user identities across multiple websites in a single attack. In our study of 29 cases, we identified 11 instances of the Replay Attacks and 7 of the Blind Multi-Message Attacks.

Web3AuthGuard. To address these vulnerabilities, websites and crypto wallets must implement a complete Web3 authentication update, but this is difficult to implement immediately. Therefore, we proposed a user-side mitigation solution, Web3AuthGuard, to help users immediately mitigate Blind Message Attacks in crypto wallets. When a user logs into a website B, the wallet equipped with Web3AuthGuard will extract a message template from B's message. Later, when the user attempts to log into a new website A, the wallet will perform a regex match between the message from A and the message template from B. If a match is successful, the wallet will alert the user about the potential Blind Message Attack on that website. We implement Web3AuthGuard in MetaMask's open-source code. We tested 25 user logins, and Web3AuthGuard successfully addressed 20 of them. Web3AuthGuard is unable to handle the remaining five cases because their vulnerabilities allow an attacker to modify the message body at will.

Responsible Disclosure. Following the responsible disclosure policy, we reported the vulnerabilities we discovered to the corresponding websites. One of them (*LearnBlockchain*) has acknowledged and fixed the vulnerability. Additionally, since some vendors did not respond to our security reports, we submitted our findings directly to the CVE program, and two CVE IDs (CVE-2023-50053 and CVE-2023-50059) have been assigned.

Demos. The PoC demos of Blind Message Attacks and mitigation are available at https://sites.google.com/view/web3auth.

Contributions. Our contributions are as follows:

- **New Vulnerabilities and Attacks.** We conducted an in-depth investigation of the vulnerabilities in Web3 authentication and identified a new type of attack, called Blind Message Attacks, and two advanced attacks – Replay Attacks and Blind Multi-Message Attacks (Section 4).
- **New Detection Tool (Web3AuthChecker).** We implemented a dynamic detection tool, Web3AuthChecker, to detect such vulnerabilities. It sends requests to Web3 backends and checks the responses to identify vulnerabilities (Section 5).
- **Empirical Analysis.** We performed detection on 29 Web3 authentication cases using Web3AuthChecker, and found that 75.8% of them are at risk of Blind Message Attacks. We also conducted two case studies (Section 6).
- **User-side Mitigation (Web3AuthGuard).** We designed and implemented a user-side mitigation solution, Web3AuthGuard, which helps alert users about a potential Blind Message Attack when the user attempts to log into a new website (Section 7).
- **Open-source Release.** To benefit the Web3 community, we will open-source our code on GitHub, including the detection tool Web3AuthChecker[1] and a MetaMask-based crypto wallet with built-in Web3AuthGuard[2].

2 BACKGROUND

This section introduces the background of Web3, Web3 authentication, and crypto wallets, then explores the potential impacts of unauthorized access in Web3 applications.

2.1 Web3

Web3, also known as Web 3.0, signifies the upcoming evolution of the Internet. It will move away from centralized servers and data centers owned by a few large corporations and instead use a distributed network, blockchain, to host the data [45]. Blockchains are decentralized ledgers that record transactions across many nodes (miners). This ensures that the data cannot be altered without the consensus of the network.

Web3 Application. Applications built on the blockchain network are called Web3 applications or decentralized applications (DApps). Web3 applications cover a broad scope, including NFT marketplaces (NFTM), which allow creators and buyers to trade digital assets efficiently and securely; decentralized exchanges (DEX), which enable trading of digital currencies without a central authority; and decentralized finance (DeFi) platforms provide financial services such as lending and borrowing. Each Web3 application typically

---

[1]https://github.com/d0scoo1/Web3AuthChecker
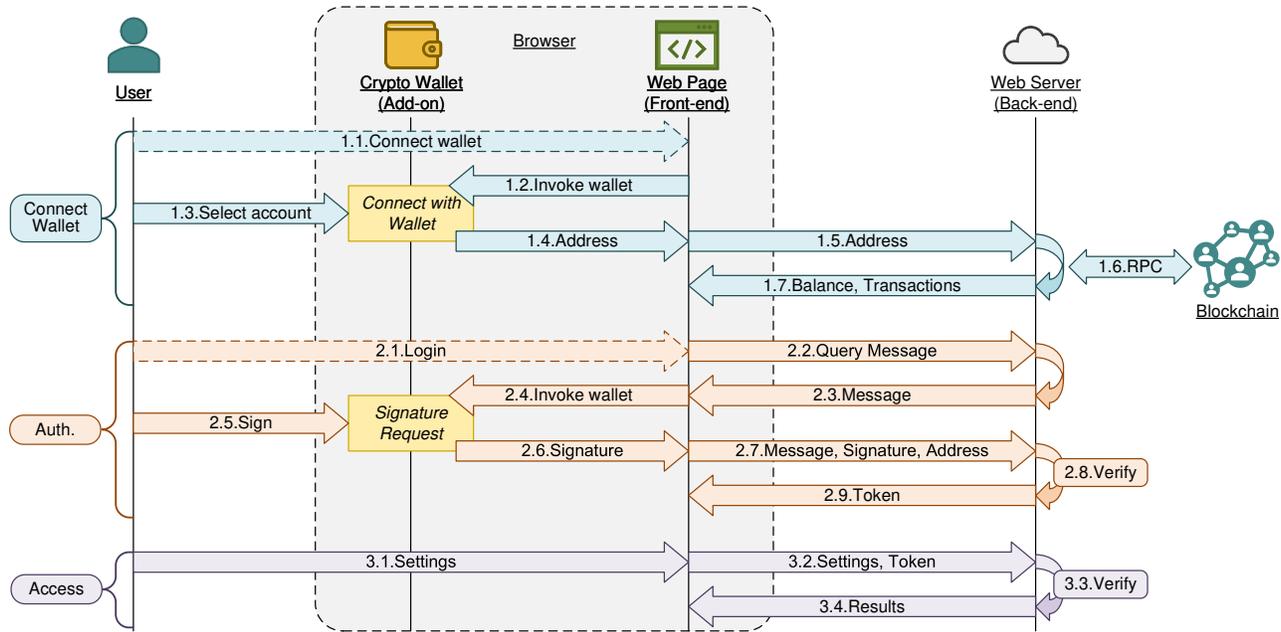[2]https://github.com/d0scoo1/Web3AuthGuard

Figure 1: Web3 Authentication Process.

consists of a smart contract and a website. Smart contracts automate operations (transactions) and store critical data on the blockchain, enhancing transparency and security. Websites serve as the user interface and manage off-chain data. While some Web3 applications are embedded within mobile apps like crypto wallets, they fundamentally function as websites. This paper focuses on websites of Web3 applications and uses the terms *application* and *website* interchangeably.

Web3 Authentication. Web3 authentication is a decentralized off-chain authentication technology based on cryptography. Web3 authentication offers benefits such as anonymity, security, and decentralization and is widely used in Web3 applications [56]. In these applications, users can employ their public keys as identifiers to manage their off-chain data. Specifically, it uses asymmetric encryption, where the client (crypto wallet) signs a specific message with their private key, and the server (Web3 application) verifies the message and its signature to authenticate the user's identity (public key) [35]. Compared to Web2 authentication technologies, Web3 employs elliptic curve algorithms, which provide enhanced security. Moreover, in the Web3 ecosystem, public keys serve as identifiers, offering better anonymity. In contrast, Web2 authentication methods [30, 34, 50, 57] often require users to provide personally identifiable information (PII), such as phone numbers and email addresses. These are stored on centralized servers, exposing users to risks like privacy breaches. Furthermore, Web3 authentication allows users to maintain a unified identity (public key) across multiple applications and services. Users no longer need to manage accounts and passwords for each website, which simplifies the user experience and reduces security risks.

Crypto Wallet. Crypto wallets are a crucial component of the Web3 ecosystem, allowing users to securely store and manage their

digital assets (key pairs) using hardware or software [71]. Most Web3 applications (websites) are compatible with popular crypto wallets and interact with them through APIs such as *Web3.js* [7] or *ethers.js* [1]. When a user accesses a website, the website examines the browser for the presence of a Web3 provider, which is the API interface for crypto wallets. If the Web3 provider is detected, the website establishes a connection to it and prompts the user to connect their wallet. After user approval, the wallet shares the account addresses (public keys) with the website. This facilitates subsequent interactions, including transactions, smart contract operations, and Web3 authentication.

2.2 Impacts of Unauthorized Access in Web3

Asset Loss. Web3 authentication only grants users access to off-chain data, and on-chain transactions usually require additional signatures so that Web3 authentication will not impact users' on-chain assets. However, attackers can still profit from unauthorized access to user accounts. Here, we highlight two significant asset losses: *Unlocked Content* and *Unfair Trading*.

Unlocked Content. In NFT marketplaces, some items are valued from *unlocked content*. If attackers gain unauthorized access to these accounts, they can directly access the *locked content* without any cost. Moreover, most Web3 websites do not have abnormal login detection, meaning that the owners may never be notified when their locked content is leaked or stolen.

Unfair Trading. An NFT's characteristics directly influence its value, with rarer traits often commanding higher prices [23]. For instance, in a popular game, CryptoKitties, the cost of each cat is determined by 13 distinct properties. While an attacker generally cannot manipulate the price of an item without signature authorization, they could alter the properties without restrictions

in collections that support *lazy minting*. This could allow an attacker to change the properties of a low-value item to a rare one and then purchase it, resulting in an *unfair trade*. Even if the owner detects the attack afterward, they cannot reverse the transaction. Lazy minting allows creators to produce NFTs without upfront gas costs, significantly reducing the barriers to selling NFTs.

Compromised Anonymity and Reputation. The key pair in the crypto wallet is randomly generated, so the user's public key (address) has good anonymity, but an attacker who gains access to a user's account can link the user's address to personal information such as email, effectively breaking the anonymity. Moreover, unauthorized access can damage users' reputations, as attackers may use the user's account for illegal activities. NFT marketplaces allow users to link their social media accounts, such as Twitter or Facebook. These social media accounts represent important channels for artists to strengthen their connections with users and promote their artworks [76], which attackers can exploit to encourage illegal activities, causing reputational damage.

3 WEB3 AUTHENTICATION

This section briefly introduces the Web3 authentication process, then specifically focuses on *message design* and *server verification* in authentication.

3.1 Overview

Web3 authentication is primarily utilized in two scenarios: *user login* and *profile update*. In the first scenario, the client needs to sign a specific message, which is then verified by the server. Once authenticated, the server will issue a token to the client to sustain a session. In the second scenario, the user signs the updated profile, and the server checks this signature before updating the user's profile. These two scenarios are independent; some websites may include only one of them, while others incorporate both. The user login is the most common scenario of Web3 authentication, and therefore, this paper mainly focuses on this scenario. Figure 1 illustrates the process of a user logging into a website by Web3 authentication. The process is described below.

Connect Wallet. Connecting a wallet with the website is the first step for a user to interact with a Web3 website. The user clicks the *Connect Wallet* button on the web page (front-end) (1.1), then the wallet pops up a window asking the user to select the account they want to connect to the website (1.2). The user selects an account (address, i.e., public key) (1.3), and the wallet returns the address to the web page (1.4). The web page forwards this address to its server (back-end) (1.5). The server, using RPC, fetches the balance and transactions of the address from the blockchain (1.6), then returns this information to the web page (1.7). The web page displays the user's balance and transactions on the page. It is worth noting that while multiple accounts may be connected, the website will interact with the first account by default. For simplicity, we assume that only one account is selected in this paper.

Authenticate. Upon the user clicking the *login* button (2.1), the web page requests a message from the server (2.2). Then, the web page forwards the message to the wallet to prompt a signature request (2.3-2.4). A window pops up from the wallet, asking the user to sign

the message. After the user approves the signature request (2.5), the wallet returns the signature to the web page (2.6). Then, the web page sends the signature, message, and address to the server for verification (2.7-2.8). If authenticated successfully, the server returns a token to the web page (2.9).

Access. The user (front-end) holds the token to access protected information (3.1-3.4). To keep the session, the website may store the token in the request header or cookie. This paper only focuses on Web3 authentication, so the session details are omitted.

```
message = 1*field
    ; Message consists of multiple fields.

field = statement / domain / name / nonce / ext

statement = *( reserved / unreserved / " " )
    ; See RFC 3986 for the definition of "reserved" and "unreserved".

domain = authority
    ; See RFC 3986 for the definition of "authority".

name = *( ALPHA / DIGIT / "-" )
    ; Website Name.

nonce = timestamp / date-time / rnd
    ; See RFC 3339 for the definition of "date-time".

timestamp = 10( DIGIT ) / 13( DIGIT )

rnd = *( ALPHA / DIGIT )
    ; Random Number.

ext = address / version / chain-id / issued-at / expiration-time /
    not-before / request-id
    ; Extension Fields.
```

Listing 1: ABNF Message Format

```
Welcome to OpenSea!

This request will not trigger a blockchain transaction or
cost any gas fees.
Click to sign in and accept the OpenSea Terms of Service:
https://opensea.io/tos

Your authentication status will reset after 24 Hours.

Wallet address: 0x36e7c6feb20a90b07f63863d09cc12c4c9f39064
Nonce: 66ffb8f1-5eb1-4477-9558-36a60eb1b51f
```

Listing 2: A Message from Opensea.io (Good Design)

```
Please sign this message to connect to Foundation.
```

Listing 3: A Message from Foundation.app (Bad Design)

3.2 Message Design

In Steps 2.3-2.6, Web3 authentication normally uses the protocol *Personal Sign* (EIP-191) [68]. This protocol does not impose any requirement on the format or content of a message, so Web3 applications must implement their own messages for authentication. We conducted an extensive survey and identified common fields in messages, represented in Augmented Backus–Naur Form (ABNF) as shown in Listing 1. The fields referenced from other RFCs [5] are annotated in the ABNF. These fields can be categorized as variable fields, such as nonce and address, and static fields, like domain and statement. For convenience, the content composed of static fields in the message is referred to as the *message body*.

In Web3 authentication, each field serves a distinct function, and the design of the message differs across various websites. Listing 2 and Listing 3 provide two examples from well-known NFT marketplaces, *opensea.io* and *foundation.app*, respectively. The message from *opensea.io* is well-designed, including fields such as domain and nonce. In contrast, the message from *foundation.app* contains only name and statement, thereby posing security vulnerabilities. Next, we will explain the meaning and function of each field in Listing 1 in detail, using these two examples as references.

statement. The statement field provides a human-readable text that explains the message. For instance, the message in Listing 3 contains the statement *"Please sign..."*, prompting users to sign.

domain. The domain field is a critical element of the message, representing the website's domain name. It plays a pivotal role in message security, enabling users to verify the source of the message. For example, the message in Listing 2 includes the domain field *opensea.io*. If a malicious website prompts the user to sign this message, the user can quickly recognize that it comes from *opensea.io* and refuse to sign it.

name. The name field, indicating the website name, offers identifiability but may be unreliable due to potential duplication.

nonce. The nonce field is vital for fortifying security against replay attacks. Nonces can be classified into two categories:

- *Time-based nonces*, usually *timestamp* or *datetime*, are generated by the front-end and embedded into the message. The server then verifies if the nonce is within its acceptable time range.
- *Record-based nonces*, typically *random numbers*, require the front-end to query the backend before signing. Record-based nonces can be further divided into *one-time nonces* per address and *temporary nonces*, reusable within a specific validity period.

The choice between these nonces depends on factors such as distribution and storage, and no absolute preference exists.

ext. The ext field serves as an extension and is not central to this work. For more details, please refer to EIP-4361 [17].

3.3 Server Verification

Steps 2.7-2.9 outline the verification process of Web3 authentication. The front-end submits the *message*, *signature*, and *address* to the server, which then scrutinizes these data to authenticate the user's authority to log into the website. The server's verification process typically involves the following three steps:

Static Field Verification. The server first checks the validity of the message body (static fields). It disregards variable fields, such as nonce, and compares the remaining content of the message with its local message body. If a discrepancy exists between the two, the authentication process is terminated, returning a failure response.

Variable Field Verification. The server verifies the nonce in the message to avoid replay attacks. Depending on the nonce type, the server might need to query the existence of a nonce record (for record-based nonces) or verify whether the timestamp is within an acceptable time range relative to its internal clock (for time-based nonces). Other variable fields on the server, such as address, will also be verified during this process.
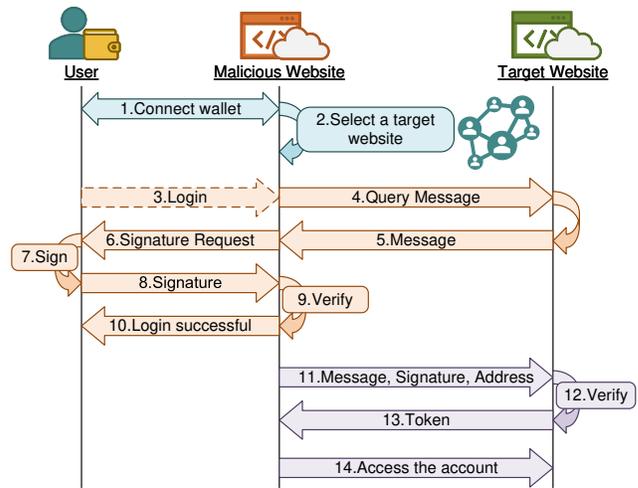

Figure 2: Blind Message Attack.

Signature Verification. The server verifies the validity of the signature: it uses the user's address (public key) to decrypt the signature. Then, it compares the result with the hash of the message. If they match, the signature is deemed valid. Once verified, the server issues an authentication token to authorize user access.

4 BLIND MESSAGE ATTACK

This section presents Blind Message Attacks, which exploit vulnerabilities in the Web3 authentication implementation described in Section 3 to gain unauthorized access. Specifically, when a user logs into a malicious website using Web3 authentication, it tricks the user into signing a message from another website, thereby gaining access to the user's account on that other website. This section first defines the threat model, followed by a motivating example. Then, we analyzed the potential vulnerabilities in the Web3 authentication process. Finally, according to the impact of these vulnerabilities, we classify the risk levels of Blind Message Attacks and discuss advanced attacks. Attack demos are available at https://sites.google.com/view/web3auth.

Threat Model. In our threat model, the attacker aims to steal users' identities on target websites, thereby gaining unauthorized access. This occurs when a user logs into the attacker's (malicious) website using Web3 authentication. The attacker exploits this opportunity to steal the user's identity (signature) of the target website. In our model, the attacker cannot exploit browser vulnerabilities or crack users' crypto wallets. Also, attackers cannot intercept and decrypt network packets, such as man-in-the-middle (MitM) attacks. The attacker's capabilities are limited to exploiting vulnerabilities in the Web3 authentication process of the target website.

4.1 Motivating Example

Figure 2 shows in detail how an attacker steals a user's identity on the target website when the user logs in using Web3 authentication on the malicious website. The attack process is as follows:

(a) Select a Target Website. In Steps 1-2, a user visits a malicious website and connects to the wallet. The malicious website queries the transaction record of the address (public key) in the blockchain, identifies the websites that the user has logged into before by contract addresses, and then selects the most valuable target website based on information such as balance and trading volume.

(b) Login. In Steps 3-6, when the user logs in, the malicious website impersonates the user to obtain a message from the target website and prompts the user to sign it. Here, the malicious website can bypass the target website's front-end and obtain the message directly from the back-end through an HTTP request. In Steps 7-10, the user signs the message and returns the signature, after which the malicious website then verifies the signature using the user's public key and prompts that the login is successful.

(c) Unauthorized Access. In Steps 11-14, the malicious website uses the signature, message, and address previously obtained to request a token from the target website and then holds the token to access the user's account.

Root Cause. According to the above attack process, the root cause of the Blind Message Attack lies in the user's action of blindly signing a message intended for a legitimate (target) website while on a malicious website during the Web3 authentication process. The attacker then uses this leaked signature to bypass the target website's Web3 authentication. Notably, the malicious website does not need to impersonate a specific target website — it can be any website that successfully entices users to log in. The success of a Blind Message Attack hinges on *the victim's unawareness that the message being signed is from another website*, and this vulnerability is dependent on the security flaws of the target website.

4.2 Vulnerability Analysis

Vulnerabilities in the Web3 authentication process are crucial in facilitating Blind Message Attacks. According to Section 3, these vulnerabilities stem from two primary sources: 1) flaws in message design resulting in the exclusion of essential fields (V1) and 2) inadequate verification allows security measures to be circumvented (V2, V3). Therefore, the vulnerabilities of Web3 authentication mainly include the following three aspects:

(V1) Lack of Essential Fields. In the EIP-191 protocol, developers can customize the message in Web3 authentication. Typically, domain serves as an identifier of a message to indicate the issuer; if it is missing, users cannot determine the precise source of the message. Moreover, the potential duplication of name renders it unreliable as an identifier. For instance, the *foundation.app* message in Listing 3 includes only the name *Foundation*, enabling malicious sites (e.g., *foundation.com*) to imitate legitimate ones. Furthermore, the lack of the nonce field makes it possible for a leaked signature to be reused for token refreshment, enabling replay attacks.

(V2) Unchecked Fields. Failure of the server to check the fields in the message can cause security risks. For instance, some servers fail to check static fields, such as the message body. This oversight allows malicious websites to alter fields like domain or name, misleading users about the message's issuer. Additionally, neglecting to verify the nonce field opens the possibility of replay attacks.

(V3) Verification Flaws. Verification flaws on the server allow specially crafted messages to evade checks. A common error is using regular expressions (regex) to match the message body with variable fields without confirming the message body's exact match with the issued message. Since regex operates on the principle of *inclusion* rather than *equality*, messages with extra content can also pass verification. Moreover, risks may also arise when checking the time-based nonce, as an appropriate time range is not set.

4.3 Security Risks

This section discusses the impact of vulnerabilities on Blind Message Attacks and discusses two advanced attacks, namely Replay Attacks and Blind Multi-Message Attacks.

4.3.1 Risk Levels. Based on the vulnerability analysis in Section 4.2, we categorize Blind Message Attacks into four levels. In critical to medium-risk scenarios, users are often unable to detect the attack or find it challenging to do so. In low-risk scenarios, attentive users might identify the message as originating from another website, thereby preventing the attack.

Critical Risk. In critical-risk scenarios, the attacker can directly launch a Blind Message Attack and steal the user's identity without even interacting with the victim (i.e., skipping 3-7 in Figure 2). This risk arises from the absence of verification for signatures, addresses, and messages. Without the server checking the signature or the address, an attacker only needs to set the address in the request as the user to impersonate them without requiring the user's signature. Suppose the server only verifies the signature and address but not the content of the message. In that case, the attacker does not need to obtain the user's specific signature on the website; he can impersonate the user using any of the user's messages and signatures.

High Risk. In high-risk scenarios, the user would not be able to recognize that the malicious website's message came from another website, so once the user intends to log into the malicious website, the Blind Message Attack will be successful. This vulnerability arises from shortcomings in message design (V1) or server verification (V2). Specifically, if messages from the target website lack the domain and name, users cannot identify the message's source in (malicious) websites. Furthermore, if the target website does not verify the message body, a malicious website can alter the domain and name to its own, thereby misleading users.

Medium Risk. In medium-risk scenarios, users may find that the malicious website's message comes from the target website and refuse to sign it, thereby avoiding the attack. Specifically, if the message from the target website contains name but lacks a domain (V1), the malicious website has to register a similar domain name to mislead the user. If the message body check is sloppy (V3), such as only regex matching, attackers can hide the target website's message body within their own messages. Although both situations may be noticed by experienced users, in fact, most users are not aware of it, as demonstrated by the second case in Section 6.2.

Low Risk. In low-risk scenarios, the message contains a domain, and there are no flaws in the server verification. An attack will only succeed if the user disregards the signed message. Due to the common lack of Web3 expertise among users and the similarity in messages across various websites, the attack might still succeed.

4.3.2 Advanced Attacks. Serious vulnerabilities in Web3 authentication also provide the basis for advanced attacks, specifically Replay Attack and Blind Multi-Message Attack. Through replay attacks, attackers can maintain prolonged unauthorized access. Through blind multi-message attacks, attackers can obtain a user's identity across various websites within a single Blind Message Attack.

Replay Attack. Generally, an authentication token has a specified expiration time, and once it expires, the attacker cannot access the user's account without securing a new signature. However, if the message lacks nonce (V1) or the verification process is flawed (V2, V3), an attacker can replay the signature repeatedly to refresh the token, thus maintaining unauthorized access. It is important to note that while broader replay attacks could be carried out by intercepting signatures, such as man-in-the-middle attacks, such cases fall outside the threat model this paper considers. Our focus remains on assessing the impact of replay attacks within the framework of Blind Message Attacks.

Blind Multi-Message Attack. Attackers can exploit vulnerabilities across multiple websites to create a crafted message that simultaneously passes Web3 authentication on all these websites. As a result, with a single signature from the user, the attacker can access the user's accounts on several websites. The crafted message is equivalent to multiple messages on websites, so we call the attack Blind Multi-Message Attack. Specifically, for websites that do not verify the message body (V2) or only verify its presence rather than strict equality (V3), an attacker can bypass the server verification of these websites by constructing a message that contains the required fields. We will describe this attack in detail in Section 6.2.

5 WEB3AUTHCHECKER

This section presents WEB3AUTHCHECKER, a tool for automatically detecting Blind Message Attacks through dynamic analysis of target websites. We start with an overview of the architecture, followed by a detailed implementation of three checkers (message, nonce, and signature). Finally, we introduce *FlexRequest*, an HTTP request tool designed to streamline the testing process.

5.1 Overview

As depicted in Figure 2, the interaction between a malicious website and a target website involves three requests:

- QUERY. In Step 4, the malicious website queries a message from the target website's server.
- AUTH. In Step 11, the malicious website forwards the message, signature, and address to the target website's server to authenticate and obtain a token.
- ACCESS. In Step 14, the malicious website holds the authentication token to access the user's account.

Our tool, WEB3AUTHCHECKER, interfaces with these requests to identify vulnerabilities. In simple terms, it injects different attack payloads into HTTP requests, then analyzes whether responses are as expected. The architecture, depicted in Figure 3, is composed of *Checker* and *FlexRequest*. There are some predefined parsers in *FlexRequest* to support converting different types of API and configuration files into a unified *Request items* object. The vulnerability detection process is as follows:
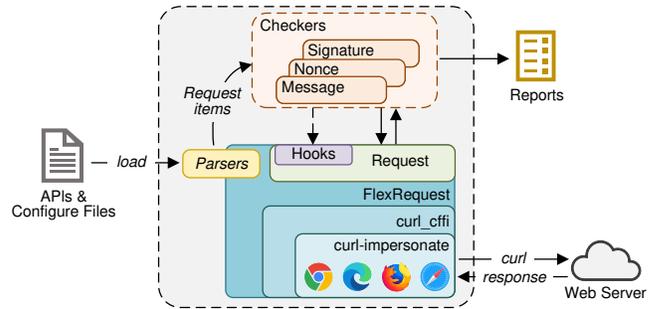

Figure 3: Architecture of Web3AuthChecker.

(1) First, parsers convert loaded APIs and configuration files into *Request items* and send them to *Checker*.
(2) *Checker* guides *FlexRequest* to perform a series of API requests. For each request, *Checker* supplies attack payloads and API parameters to *FlexRequest*. *FlexRequest* utilizes its auto-replacement mechanism to insert the attack payloads into the proper slots within the API parameters, subsequently forwarding the request to the website under test.
(3) *FlexRequest* parses the website's response according to the predefined configuration and returns the parsed results back to *Checker*.
(4) After completing a series of test requests, *Checker* scrutinizes the data returned by *FlexRequest* to detect vulnerabilities.
(5) Finally, *Checker* generates a test report for each website, highlighting any identified vulnerabilities.

Through the process mentioned above, WEB3AUTHCHECKER efficiently evaluates the security of Web3 authentication implementations on target websites, identifying potential risks. Subsequently, we will elaborate on the two components of WEB3AUTHCHECKER.

5.2 Checker Implementation

We developed three checkers to detect vulnerabilities in Section 4.2. Each checker focuses on one aspect of Web3 authentication – message, nonce, and signature – to identify potential vulnerabilities. The checkers assess the integrity of the message design by examining specific fields (V1) and employ various attack payloads to uncover vulnerabilities in verification (V2, V3).

Message Checker. Message Checker begins by sending a QUERY request to retrieve a message from the website. Then, it matches the presence of domain names (domain) and website names (name). To detect vulnerabilities in verification, the checker modifies the message to a random string as the attack payload and signs it. If a subsequent AUTH request returns a token, it indicates that the server does not adequately check the message. To further verify, the checker 1) empties the message body and 2) appends random characters at the beginning of the message to assess any vulnerabilities in the server's message body verification. If the message fails any of the above checks, it is considered vulnerable to Blind Message Attacks, and the checker returns specific security risk levels based on the type of vulnerability.

Nonce Checker. Nonce Checker sends QUERY requests from multiple addresses to obtain messages. It then compares these messages

Table 1: AUTH Request Status vs. Nonce Type

| Nonce Type | AUTH Request | | | | |
	1st	2nd	3rd	4th	5th
One-time	✓	✗	-	-	-
Temporary	✓	✓	✓	✗	-
Time-based	✓	✓	✓	✓	✗
Invalid Nonce	✓	✓	✓	✓	✓

✓: Request successful; ✗:Request failed.

to identify variable fields, excluding non-nonce fields. If variable fields persist, it suggests the presence of nonce. Subsequently, the checker executes the following sequence of requests:

(1) It sends a QUERY and AUTH request to the server.
(2) It repeats the first AUTH request.
(3) It uses a new address to retrieve another message by a QUERY request. It then uses the old address to sign this new message and sends the third AUTH.
(4) It generates a value similar to the nonce and replaces it before sending the fourth AUTH.
(5) It removes the nonce before sending the fifth AUTH.

If a token is still returned after these requests, it implies that the server has failed nonce verification. The type of nonce can also be inferred from these requests, as detailed in Table 1.

Signature Checker. Signature Checker sends three QUERY requests to obtain three different messages. These messages are then used to perform three independent tests: 1) setting the signature to null, 2) setting the signature to an invalid value, and 3) replacing the address with a different one. If the token can be obtained from any of these tests, it indicates that the server does not properly verify the signature or address.

Furthermore, for convenience, an additional ACCESS request is sometimes sent following the AUTH request. *Checker* confirms the vulnerability by checking that the return value of a ACCESS request (instead of AUTH) is as expected.

5.3 FlexRequest: A Python HTTP Library

To detect vulnerabilities, *Checker* tests the authentication-related APIs of the website. Given that the differences in the APIs of each website, existing testing tools or libraries, such as *Postman*[4] and *Requests*[2], require the development of separate test scripts for each site, leading to issues of code duplication and maintainability. To address these challenges, we developed a specialized HTTP library for WEB3AUTHCHECKER. *FlexRequest*, a Python-based HTTP library, features an automatic replacement mechanism to align with the variations in APIs, providing a flexible and adaptable solution for testing APIs on various websites. As illustrated in Figure 3, *FlexRequest* utilizes *cffi_curl* for HTTP requests. This Python library creates bindings for *curl-impersonate* through the C Foreign Function Interface (CFFI). *curl-impersonate* is a special build of *curl* that can impersonate the four main browsers. By performing TLS and HTTP handshakes identical to a real browser, *curl-impersonate* ensures that websites do not block requests.

FlexRequest supports using keys to replace dynamic parameters in the API. Before executing a request, it automatically substitutes these keys with the corresponding values. After receiving the response, *FlexRequest* retrieves the values at the specified positions

according to a predefined configuration, binding them to the corresponding keys. By managing the values of these keys, developers can perform unified testing across various APIs.

Furthermore, *FlexRequest* maintains a *session context* throughout a session to pass the previous response values between a series of requests. For example, the message obtained from the Query response will be stored in the session context and bound to the key *msg*. When conducting an Auth request, *FlexRequest* automatically populates the request with the value of *msg* from the session context. For detailed insights into API testing challenges and *FlexRequest*'s unique solutions, please refer to our open-source tool, WEB3AUTHCHECKER[3].

6 FINDINGS

This section conducts a comprehensive evaluation of Web3 authentication. Our analysis uncovers the extensive occurrence of Blind Message Attacks. Additionally, we delve into two specific cases and evaluate the efficacy of WEB3AUTHCHECKER.

Dataset. Given the limited research on Web3 authentication, no relevant datasets exist. DappRadar, a well-known Web3 dapp distribution platform, provided a basis for our collection: we selected 18 marketplaces that support Web3 authentication from the top 50 in DappRadar's *Top Decentralized Marketplaces* list [6]. Additionally, we identified nine websites that support Web3 authentication through Google searches. Two of these 27 websites required Web3 authentication for user login and profile updates (4 test cases). Among the remaining 25 websites, 23 utilized Web3 authentication solely for user login (23 test cases), and two employed it specifically for profile updates (2 test cases). Therefore, we have a total of 29 test cases. According to DappRadar's statistics, in January 2024 alone, the total transaction volume of 16 websites exceeded 592 million US dollars, and the number of unique active wallets (UAW) surpassed 1.29 million.

Our dataset includes various categories such as games, services, and forums. Notably, NFT marketplaces are the primary entities of Web3 authentication. On the contrary, DEXs (Decentralized Exchanges) and DeFi (Decentralized Finance) typically interact directly with the blockchain, making Web3 authentication unnecessary. Our examination of the top 25 DEXs and the top 25 DeFi applications revealed that none employ Web3 authentication; therefore, they have been excluded from our dataset.

Setup. We thoroughly examined each website and collected the relevant APIs, exporting them as JSON files in Postman format. We also prepared corresponding configuration files for WEB3AUTHCHECKER. We deployed WEB3AUTHCHECKER on Github Codespaces, configured with 4 cores and 8GB RAM. We set Chrome (v110) as the *curl-impersonate* browser and limited the API response timeout to 10 seconds. To avoid being blocked due to frequent requests, we instituted a one-minute interval between each request.

6.1 Analysis of Results

Table 2 provides detailed information and detection results for all 29 test cases. WEB3AUTHCHECKER examines whether messages include domain, name, and nonce, and it verifies if the server checks

---

[3]https://github.com/d0scoo1/Web3AuthChecker

Table 2: Detection Results of Websites that Support Web3 Authentication

#	Website	Category	Message Design (V1)			Server Verification					Security Risk		
			Domain	Name	Nonce	Message	Body	Nonce	Signature	Address	BMA	RA	BMMA
1	Blur	Marketplace	✗	✓	✓	✓	✓	✓	✓	✓	M	○	○
2	OpenSea	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
3	LooksRare	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
4	Foundation	Marketplace	✗	✓	✗	✓	✗(V3)	N/A	✓	✓	M	●	●
5	Element	Marketplace	✓	✓	✓	✓	✗(V3)	✓	✓	✓	M	○	●
6	Rarible	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
7	Joepegs	Marketplace	✗	✓	✓	✓	✓	✓	✓	✓	M	○	○
8	Quix	Marketplace	✗	✗	✓	✓	✗(V2)	✗(V3)	✓	✓	H	●	●
9	Minted Network	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
10	Campfire	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
11	Moonflow NFT	Marketplace	✗	✓	✓	✓	✓	✓	✓	✓	M	○	○
12	Galler	Marketplace	✓	✓	✓	✗(V2)	✗(V2)	✗(V2)	✓	✓	C	●	●
13	PlayDapp	Marketplace	✗	✗	✗	✓	✓	N/A	✓	✓	H	●	○
14	Refinable	Marketplace	✗	✗	✓	✓	✓	✓	✓	✓	H	○	○
15	Apeiron	Marketplace	✗	✗	✓	✓	✓	✓	✓	✓	H	○	○
16	Lifty	Marketplace	✓	✓	✓	✓	✓	✓	✓	✓	L	○	○
17	LearnBlockchain	Community	✗	✓	✗	✗(V2)	✗(V2)	N/A	✓	✓	C	●	●
18	DappRadar	Ranking	✗	✗	✓	✓	✓	✓	✓	✓	H	○	○
19	QuestN	Service	✗	✓	✓	✓	✗(V2)	✓	✓	✓	H	○	●
20	Galxe	Social	✓	✓	✓	✓	✓	✗(V2)	✓	✓	L	●	○
21	Planetix	Game	✓	✓	✓	✓	✗(V2)	✗(V2)	✓	✓	H	●	●
22	MOBOX	Game	✗	✗	✓	✓	✓	✓	✓	✓	H	○	○
23	Bomb Crypto 2	Game	✗	✗	✓	✓	✓	✓	✓	✓	H	○	○
24	Decert	Service	✗	✓	✓	✓	✓	✓	✓	✓	M	○	○
25	Paragraph	Media	✗	✓	✓	✓	✓	✓	✓	✓	M	○	○
26	Campfire	Marketplace	✗	✗	✗	✓	✓	N/A	✓	✓	H	●	○
27	Lifty	Marketplace	✗	✗	✗	✓	✓	N/A	✓	✓	H	●	○
28	NFTmall	Marketplace	✗	✗	✗	✓	✓	N/A	✓	✓	H	●	○
29	Babylons	Marketplace	✗	✗	✗	✓	✓	N/A	✓	✓	H	●	○

In this table, BMA = Blind Message Attack, RA = Replay Attack, BMMA = Blind Multi-Message Attack.
In the Web3 authentication process, cases #1~#25 are for user login, while cases #26~#29 are for profile update.
✓: No vulnerability found; ✗: Vulnerability found; N/A: Not applicable; **C**: Critical; **H**: High; **M**: Medium; **L**: Low; ●: Risk exists; ○: No risk.

the message, message body (body), nonce, signature, and address to identify the risks defined in Section 4. The results are alarming, with 22 out of the 29 test cases found to be subject to medium to critical risks, making them vulnerable to Blind Message Attacks. At the same time, we identified critical risks in two cases, and we have informed the respective vendors. One of them has already acknowledged the risk and fixed it. Also, 11 cases are at the risk of Replay Attacks, and seven have Blind Multi-Message Attacks. Table 2 shows that each website has its own distinct vulnerabilities, indicating that their Web3 authentication implementations are independent. Therefore, we conclude that these websites have no common server-side SDKs.

Subsequently, we will analyze the security risks of 25 cases of using Web3 authentication for user login (#1~#25). Then, we will address the four remaining cases (#26~#29) that employ this authentication method for profile updates (note that two websites are included in both categories).

Blind Message Attack. In the 25 user login cases (#1~#25), the *BMA* column of Table 2 reveals that two of them are at critical risk (**C**), nine at high risk (**H**), seven at medium risk (**M**), and only seven are evaluated as low risk (**L**). While all cases verify the signature and address, two cases do not perform any checks on the message, thus resulting in critical risks. In two critical-risk cases, an attacker could use a user's arbitrary signature to log in to the user's account. One of these cases, learnblockchain, has over 2 million users, highlighting a significant risk.

In nine high-risk cases, either the `domain` and `name` fields do not exist in the message (V1), or the server does not verify the message

body so the attacker can tamper with them (V2). The nine high-risk websites provide attractive targets for attackers. By exploiting the users' inability to identify the source of messages, malicious websites can effortlessly trick users into signing messages from these websites. Notably, Apeiron (#15), QuestN (#19), and MOBOX (#22), all identified as high-risk websites, reported 89.21k, 54.94k, and 27.26k Unique Active Wallets (UAW), respectively, in January 2024. These UAWs underscore the substantial user engagement and the potential impact of such vulnerabilities.

In the seven medium-risk cases, the attacker may need more sophisticated techniques to implement attacks, such as using a domain name identical to the target website (V1) or hiding the target website's message body within the malicious website's message (V3). As discussed in the case study section (Section 6.2), these hurdles may be relatively easy for experienced attackers. Importantly, even cases categorized as low risk are not necessarily exempt from Blind Message Attacks, as attackers still have opportunities to deceive unwary users.

Vulnerabilities in verification can weaken a well-structured design. For instance, Element (#5), Galler (#12), and Planetix (#21) all have appropriately structured messages containing the essential fields. However, Element does not improperly verify the message body (V3), thus placing it at medium risk. In contrast, the other two do not verify the message body (V2), which puts them at high risk. As of January 2024, Element reported 158.63k UAWs and a transaction volume of $18.91M. These security risks emphasize the importance of server verification.

Table 3: Nonce Types of the 25 Websites

Nonce Type	Replay	Websites	Total
One-time	○	#2, #3, #5, #14, #16, #24	6
Temporary	◑	#1, #7, #10, #11, #18, #25	6
Time-based	◐	#6, #9, #15, #19, #22, #23	6
Invalid Nonce	●	#8, #12, #20, #21	4
No Nonce	●	#4, #13, #17	3

In this table, ●:Risk exist; ◐: Short-term risk; ○: No risk.

Replay Attack. In the 25 user login cases (#1~#25), as indicated in the *RA* column of Table 2, seven out of these 25 cases are vulnerable to replay attacks. This vulnerability implies that an attacker could reuse a signature, thereby bypassing the session's security mechanism. Three cases can be attributed to the message not containing nonce (V1). The remaining four cases do include nonce, but it is either unverified on the server (✘(V2)) or the server check contains a vulnerability (✘(V3)), as shown in the *Nonce* column of Table 2. For instance, Quix (#8) uses a timestamp as a nonce. If the timestamp in the message is ahead of the current time, Quix's server rejects it. However, the servers do not define an expiration time, meaning used messages and signatures remain valid.

Further investigation into the nonces used by these cases is presented in Table 3. Among them, 12 cases employ temporary and time-based nonces. These nonces could technically be replayed within their valid period; however, we measured their expiration time and found that they typically have short validity periods, often not exceeding fifteen minutes. As a result, we conclude that these cases are not at significant risk of replay attacks.

Blind Multi-Message Attack. There are five cases failed to verify the message body (✘(V2)), while two cases improperly checked the message body (✘(V3)), as shown in the *Body* column of Table 2. For the former, the high-risk scenario allows an attacker to construct a message body freely, removing crucial details such as the `domain` and `name`. In the latter scenario, characterized as medium-risk, the message body is improperly checked, leaving an opportunity for the attacker to exploit this vulnerability by adding misleading information or cleverly hiding the original message. Overall, these two vulnerabilities could be exploited for blind multi-message attacks, suggesting that all seven websites are at such risk as shown in the *BMMA* column of Table 2.

Profile Update. In the four profile update cases (#26~#29), all four cases are vulnerable to high-risk blind message attacks and replay attacks. NFTmall (#28) and BabyIons (#29) do not require user login, but they require Web3 authentication when users update their profiles. Campfire (#26) and Lifty (#27) require additional Web3 authentication for profile updates even after the user has logged in. However, Campfire and Lifty do not verify the tokens obtained from the prior login during profile updates. As a result, the security of profile updates relies entirely on the current Web3 authentication implementation. Both Campfire and Lifty employ well-structured messages without any evident security vulnerabilities during user login. Regrettably, they fail to maintain this level of security in the profile update. For instance, Campfire uses the simple phrase *update_profile* as the message for profile update. If an attacker obtains a user's signature for this phrase, they could freely update the user's profile.

6.2 Case Study

This section presents two case studies highlighting the security issues. The first case study examines the unchecked message vulnerability (V1) and has been acknowledged by LearnBlockchain. The second case study underscores a blind multi-message attack, demonstrating how it is possible to bypass Web3 authentication on three distinct websites utilizing just a single message.

Unchecked Message. LearnBlockchain (#17) is a well-known blockchain community with more than 2 million users. It supports both traditional password-based authentication and Web3 authentication. In the Web3 authentication, LearnBlockchain's message is simply *learnblockchain*. Moreover, the server only verifies the signature and does not perform any checks on the message, allowing any signature of the user to pass the Web3 authentication. This vulnerability provides an easy way for an attacker to gain unauthorized access to a user account, which can lead to the exposure of important personal information and potential asset losses. Specifically, LearnBlockchain's points system supports points exchange for currency, so attackers can transfer points from the victim's account to their own account and then withdraw them.

We promptly reported blind message attack and replay attack to LearnBlockchain, who fixed their Web3 authentication based on our suggestions and awarded us 2500 points for our responsible disclosure. Another website, Galler (#12), facing similar security issues, was also notified about the vulnerabilities. However, at the time of writing, we have not received any response from Galler.

Blind Multi-Message Attacks. The vulnerabilities associated with message body verification can cause more extensive damage than expected. An attacker can create a message while bypassing the Web3 authentication of multiple websites. In this way, attackers can steal a user's corresponding identity on multiple websites at once. We refer to this situation as a Blind Multi-Message Attack (BMMA).

```
/** Modified based on Foundation's message. **/
Welcome! Please sign this message to connect to Foundation.com.

/** Planetix **/
Web3 Token Version: 2
Nonce: 84800972
Issued At: 2024-01-13T03:59:00.000Z
Expiration Time: 2024-01-14T03:59:00.000Z

/** QuestN **/
Timestamp: 1706389762
```

Listing 4: A Malicious Message For the BMMA

Listing 4 presents an example of a malicious message that can bypass the authentication of three different websites: Foundation (#4), QuestN (#19), and Planetix (#21). The attacker first registers a domain on *foundation.com*. Exploiting the vulnerability of *foundation.app*, which only checks for the existence of a message body without verifying its content, the attacker cleverly appends *Welcome!* to the beginning of the foundation.app's message (Listing 3) and *com.* to the end. Additionally, Planetix and QuestN do not check the message body. Planetix verifies four fields: *Web3 Token Version*, *Nonce*, *Issued At*, and *Expiration Time*, while QuestN merely verifies the timestamp as nonce. The attacker appends fields required by Planetix and QuestN to the end of the message. The malicious message is obviously different from the messages on the three websites,

and it contains the domain of foundation.com, thereby misleading users into believing it originates from the malicious website (foundation.com). As a result, users trust that the message is safe and easily sign the message on the malicious website. This example underscores the severity of blind multi-message attacks.

Besides, the Foundation's message lacks a nonce, implying that once the attacker acquires a user's signature, they can maintain unauthorized access through replay attacks.

6.3 Evaluation of Web3AuthChecker

Web3AuthChecker tests each attack payload from scratch every time, with multiple payloads per vulnerability. Therefore, in the experiment, Web3AuthChecker sent a total of 1,319 requests, an average of 44 requests per website. We manually checked all the results of Web3AuthChecker and found that it did not produce any false positives or false negatives. We also compared *FlexRequest*'s performance with that of the widely-used Python HTTP library, *Requests* [2], and the tool *Postman* [4]. The results indicated that six websites rejected requests from both *Requests* and *Postman*, even when the request headers were modified to mimic other browsers. In contrast, *FlexRequest* successfully completed requests to all APIs of these websites, demonstrating its superior capability in handling APIs from a diverse range of websites.

7 MITIGATION: WEB3AUTHGUARD

Given the vulnerabilities identified by Web3AuthChecker, extensive server-side updates are necessary but challenging to implement quickly. This section introduces Web3AuthGuard, a solution designed to detect potential attacks on the user (wallet) side. Compared to Web3AuthChecker, Web3AuthGuard automatically detects potential attacks during wallet operation (signature request) without the need for pre-configuration.

7.1 Design of Web3AuthGuard

Web3AuthGuard is designed to detect suspicious messages in crypto wallets, to prevent Blind Message Attacks during the Web3 authentication process. The workflow of Web3AuthGuard is depicted in Figure 4. During the Web3 authentication process, a website triggers the wallet's signature API (*Personal Sign*). Consequently, the wallet prompts a *Signature Request* page for the user to sign the message. For wallets featuring Web3AuthGuard, the first step involves comparing the new message with previously signed messages (message templates). If the new message significantly resembles a previously signed message from a different website, Web3AuthGuard promptly raises an alert on the top of the signature request page. This alert notifies the user of a potential Blind Message Attack, enabling them to decide whether to proceed with the authentication process. The above process mainly relies on three components of Web3AuthGuard: *template extraction*, *fuzzy matching*, and *wallet alerts*. Below, we introduce them in detail.

Template Extraction. Each time the user logs in, Web3AuthGuard extracts the message template from the message. It then stores this template in the wallet with the domain name for subsequent fuzzy matching. We avoid directly storing the message, as it often contains variable fields (such as nonce). Web3AuthGuard extracts a message template from multiple messages originating from each
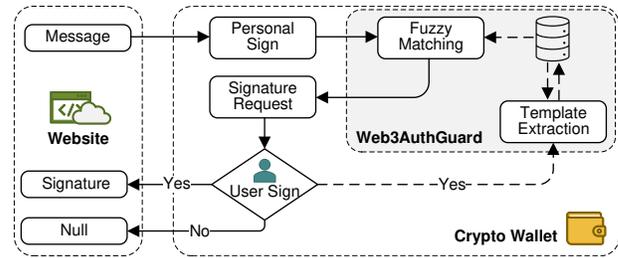

Figure 4: Workflow of Web3AuthGuard.

website. The template captures the static fields of the message, substituting variable fields with wildcards.

Template extraction process is as follows: when the user logs into the website for the first time, the wallet will record the message and mark it with the domain name. When the user logs in to the website again, Web3AuthGuard will compare the new message with the previously stored message to extract the message template. Specifically, the process involves a word-by-word comparison of the two messages. Words that match are kept and incorporated into the template. In contrast, non-matching words, pinpointed by predefined variable fields, are substituted with corresponding wildcards in the template. This updated template then supersedes the previously saved message and becomes the reference for detecting blind message attacks.

The template extraction is a dynamic and adaptive process; it can capture changes when a website's message is updated, eliminating the need for manual intervention. Web3AuthGuard only retains the latest template for each website. A few megabytes are generally sufficient for wallets. In experiments, the total size of templates for 25 websites was less than 10 KB, with the largest being 1.21 KB. Consequently, 1MB can accommodate at least 800 such templates.

Fuzzy Matching. Web3AuthGuard employs regular expression matching to compare the new message with the stored message template during the authentication process. If a match is found, indicating similarity between the new message and the template, Web3AuthGuard further checks if they originate from different websites. If they do, Web3AuthGuard adds an alert on the signature request page, indicating a potential blind message attack.

Wallet Alerts. Web3AuthGuard will add alerts to the signature request page to inform users of potential security risks. There are two types of alerts, *blind message attack* and *lack of domain name*, which are independent and may occur at the same time. If fuzzy matching detects a blind message attack, Web3AuthGuard will add a red alert, as shown in Figure 5(a). The alert will prompt for the domain names of the current and victim websites. The user should check whether the domain name (domain) or the similar website name (name) of the victim website exists in the message. If it exists, it is a blind message attack, and the user should click the *Reject* button on the signature page to refuse the signature. If the domain and name do not exist in the message, this may be a potential attack. Once the user signs, the website will *intentionally* or *unintentionally* gain access to the user's account on the victim's website. Therefore, we still recommend that users do not sign it.
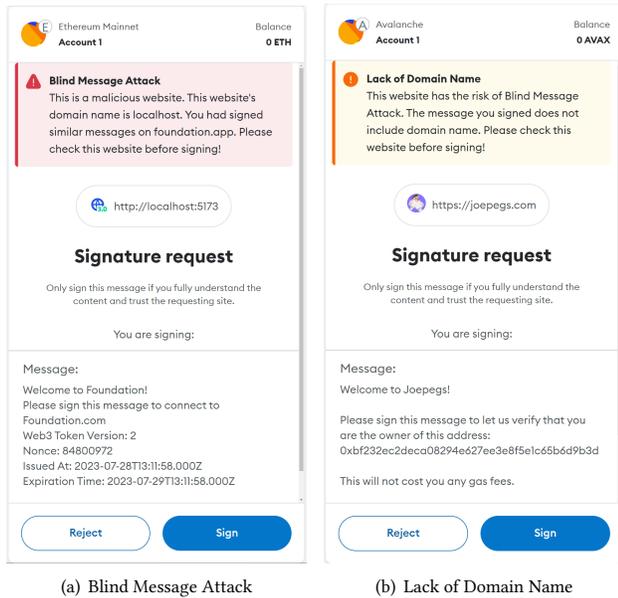
(a) Blind Message Attack

(b) Lack of Domain Name

Figure 5: Alerts in Signature Requests.

In addition, WEB3AUTHGUARD will match whether the `domain` of the visited website appears in the message. If it does not exist, WEB3AUTHGUARD will raise a yellow alert (*lack of domain name*), as shown in the figure 5(b). We recommend that users do not sign the message as it may originate from other websites or be exploited by other malicious websites. Finally, users should report any malicious activity on the website to their wallets, and most wallets will then warn other users about these dangerous websites.

7.2 Evaluation of WEB3AUTHGUARD

To evaluate the effectiveness, we implemented WEB3AUTHGUARD in the open-source code of the MetaMask browser extension for Chrome, the most widely used crypto wallet with over 30 million monthly users [25]. Our evaluation involved 25 websites (#1~#25) listed in Table 2 that employ Web3 authentication for the user login. We excluded four cases of updated profiles because the content of the messages in these cases was the updated profile rather than standard message fields.

For the 25 websites, we collected two sets of messages: one for template extraction (comprising five messages) and another for testing (also comprising five messages). Initially, we embedded the templates extracted after four rounds into MetaMask and started testing. We evaluated WEB3AUTHGUARD's effectiveness by observing if it generated an alert on the MetaMask signature page during Web3 authentication on a malicious website.

To ensure a comprehensive evaluation, we set up a local malicious website capable of sending signature requests to Meta-Mask and conducted three rounds of testing: In the first round, the malicious website directly used a second set of five messages from each website to send signature requests, testing whether WEB3AUTHGUARD could detect blind message attacks. In the second round, we assessed whether WEB3AUTHGUARD could identify

attacks in the presence of verification flaws; the malicious website randomly embedded each test message into a message from another website before sending signature requests. In the third round, we directly visited 25 websites and performed Web3 authentication to evaluate whether WEB3AUTHGUARD generated false positives.

In the 25 user-login cases, WEB3AUTHGUARD successfully passed all three rounds of testing without false positives or negatives. These tests encompass low-risk, medium-risk, and certain high-risk scenarios, thoroughly examining WEB3AUTHGUARD's effectiveness across various potential threats.

Limitation. Our evaluation shows that WEB3AUTHGUARD is robust without false positives or negatives. In addition, WEB3AUTHGUARD effectively handles server-side *verification flaws* (V3), and its fuzzy matching can detect attacks even when malicious websites embed others' messages into their own. However, WEB3AUTHGUARD is not entirely free of false positives or negatives. In two situations, WEB3AUTHGUARD might produce false positives (Blind Message Attack) in cases where messages from two different websites, lacking `domain` and `name` fields, have the same message structures. Also, WEB3AUTHGUARD is ineffective against the vulnerability *unchecked fields* (V2); this vulnerability allows attackers to alter the message content (message body) at will, rendering any user-side detection ineffective. In our testing of 25 cases, five exhibited this vulnerability, as detailed in the *body* column of Table 2. Consequently, we have manually classified these five cases as failures. In conclusion, WEB3AUTHGUARD was effective in detecting Blind Message Attacks on 20 of the 25 tested websites. Another limitation is that WEB3AUTHGUARD requires multiple messages from the same website to extract a valid template when the website's message contains variable fields. A practical solution is to preload templates of popular websites into crypto wallets.

8 DISCUSSION

Comparing Threats in Web2. Ensuring the security of the Web3 ecosystem is paramount for maintaining trust in the decentralized vision, and Blind Message Attacks represent a significant threat. Similar to traditional Web2 authentication threats like phishing attacks and man-in-the-middle (MitM) attacks, the ultimate goal of blind message attacks is to gain unauthorized access by deceiving users. However, unlike phishing attacks, which masquerade as the target website, or MitM attacks that intercept or alter communications, blind message attacks exploit vulnerabilities in the Web3 authentication process. This exploitation offers greater flexibility and effectiveness in deceiving users, as a single malicious website can target different users with varying websites.

Protocol Flaws & Solutions. The main reason for the flaws in the Web3 authentication protocol is that the website uses EIP-191 [68] for signature. The EIP-191 protocol is not specifically designed for Web3 authentication but is only used to implement the *signature* function. EIP-191 allows a user to sign arbitrary content as the message for Web3 authentication, and the client (user or wallet) cannot effectively confirm whether the message was issued by the visiting website, which allows malicious websites to launch Blind Message Attacks. We propose two solutions to address the flaws in the Web3 authentication protocol, including 1) designing a new

end-to-end protocol or 2) using unique public keys on each website. These two solutions are orthogonal.

1) New Protocol. The EIP-191 protocol was not designed for Web3 authentication, so a new security protocol is needed. This new protocol should cover both the client and server and meet the following security policies:

- *Verifiable Source.* The client can easily verify the source of the message. Meanwhile, each website's message should be unique and cannot be impersonated.
- *Integrity.* The server can easily verify the integrity of a message to ensure that it was issued by itself and has not been modified.

To implement the above security policy, the new protocol should mandate the presence of `domain` and `nonce` security fields in the message. At the same time, the protocol should specify the message format so that the client and server can automatically parse these fields. During the Web3 authentication process, the client (wallet) can automatically check these security fields in the message and prompt the user of potential security risks. During the verification process, except to verify the validity of the signature, the server also parses the fields in the message. It verifies them one by one to ensure the integrity of the message. Although the end-to-end protocol can effectively solve the flaws in the current Web3 authentication protocol, it is difficult to implement in the short term because it requires upgrading both the server and the wallet.

2) Unique Identity. The private key can generate multiple public keys, allowing users to use a unique key for Web3 authentication on each website, effectively eliminating blind message attacks. This also enhances user anonymity, as attackers cannot link a user's public key to all their visited websites. This solution can be implemented by upgrading the wallet. When a user uses Web3 authentication, the wallet checks for an existing public key for the website and generates a new one if none exists. However, this approach isolates the user's digital assets across different websites. This issue can be addressed with account abstraction (EIP-4337) [15], although it requires additional transaction fees.

Ethical Concerns. Our study conducted experiments in the wild, which may raise ethical concerns. We critically analyzed our work using the ethics framework [37] to assess our study's ethical considerations and potential risks. During the experiment, we used randomly generated accounts from a wallet to detect vulnerabilities in Web3 authentication; no normal users were affected. The APIs and functions we tested are all public functions of the websites, and we actively notified the websites with vulnerabilities and offered suggestions for improvements. To demonstrate the attacks, we built non-public websites and used randomly generated accounts representing both attackers and victims. We conducted the demonstration locally to ensure that no malicious websites were created online to mislead or deceive users. We adhered to Responsible Disclosure, and because some websites had not yet responded, we submitted our security report to the CVE database. We understand that once vulnerabilities in decentralized services are disclosed, attackers can exploit them. Therefore, we will only release the code of WEB3AUTHGUARD and not disclose the test scripts of websites.

9 RELATED WORK

To the best of our knowledge, we are the first to perform an in-depth study of security in Web3 authentication. Our research relates to Web authentication and distributed digital identity.

Web2 User Authentication. Web2 Authentication security focuses on verifying user identities to prevent unauthorized access, which includes Password-based Authentication [47, 50, 52], Multi-Factor Authentication (MFA) [30, 39, 59], Single Sign-On (SSO) [16, 29, 34], and Fast IDentity Online 2 (FIDO2) [31, 41, 46]. Despite its security criticisms, password-based authentication remains prevalent due to its simplicity. Multi-Factor Authentication, particularly two-step verification, increases password-based authentication's security [39] by requiring supplemental factors like a one-time password via SMS or email. Single Sign-On streamlines authentication by enabling a single credential's use across numerous platforms. Common SSO standards include OpenID Connect (OIDC) [48], OAuth 2.0 [57], and SAML [66]. FIDO2, which uses hardware tokens, offers an attractive alternative to password-based authentication. Lyastani et al. [46] provide an overview of user authentication technologies, focusing on the potential of FIDO2 for passwordless authentication.

Web2 authentication is vulnerable to common security threats like password guessing [54, 55, 77, 80], session hijacking [24, 28, 67], phishing [40][8], man-in-the-middle (MitM) attacks [12][36], and credential stuffing [53][75]. Web3 authentication, which is based on cryptography and effectively reduces threats such as password guessing, may still be vulnerable to session hijacking since it commonly uses traditional session management. Blind message attacks pose a unique challenge to Web3, as they can automatically select target websites based on address and blockchain data, unlike phishing, which can only mimic a specific website. Moreover, they do not require interception of communication (MitM), but rather bypass authentication by stealing user identities.

Distributed Digital Identity. Decentralized Identifiers (DIDs) are user-generated identifiers that circumvent the need for a centralized registry [44][43]. DIDs play a crucial role in the Web3 ecosystem, where through public key authentication technology, Web3 applications can easily verify user identities. Other techniques, such as zero-knowledge proofs [26][79] also hold relevance in this field. MP-CAuth [69] is a multi-factor authentication system for distributed-trust applications that addresses ease-of-use and privacy challenges. Ansaroudi et al.[9] systematically analyze multiple digital identity wallets. Korir et al.[38] conducted a study on decentralized identity wallets, revealing user misconceptions about DIDs.

Web3 Authentication Protocol. Sign-In with Ethereum (SIWE, EIP-4361)[17] requires the inclusion of specific fields such as `domain`, `nonce` in the message, and crypto wallets supporting the SIWE protocol will verify these fields. Our research complements SIWE; of the 27 websites we examined that use the SIWE protocol (#10, #20), one (#20) had a replay attack risk. Some EIPs (Ethereum Improvement Proposals) provide signed data and identity standards. EIP-191 [68] and EIP-712 [13] improve message readability and complex data signing, respectively, while EIP-725 [73] suggests a multi-key control scheme for proxy smart contracts. However, they have yet to gain widespread adoption due to issues like distributed server incompatibility (SIWE) and transaction fees (EIP-725).

10 CONCLUSION

This paper investigates the security risks associated with Web3 authentication and introduces the concept of Blind Message Attack. This attack enables attackers to gain unauthorized access to user accounts. Meanwhile, we propose advanced attacks, namely Replay Attack and Blind Multi-Message Attack. Our dynamic detection tool, WEB3AUTHCHECKER, successfully identifies 22 out of 29 real-world deployments of Web3 authentication that are at risk of Blind Message Attacks. To better alert users, we have developed WEB3AUTHGUARD, a client-side solution to detect Blind Message Attacks in crypto wallets, and showed that it can alert users immediately of potential attacks. WEB3AUTHCHECKER and WEB3AUTHGUARD are open source to facilitate future research.

ACKNOWLEDGMENTS

REFERENCES

[1] 2023. Ethers.js. https://docs.ethers.org/v6/. (Accessed on 01/27/2024).
[2] 2023. Requests. https://github.com/psf/requests. (Accessed on 01/27/2024).
[3] 2024. Metamask. https://metamask.io/. (Accessed on 01/27/2024).
[4] 2024. Postman. https://www.postman.com/. (Accessed on 01/27/2024).
[5] 2024. RFCs. https://www.rfc-editor.org/. (Accessed on 01/27/2024).
[6] 2024. Top Decentralized Marketplaces. https://dappradar.com/rankings/category/marketplaces. (Accessed on 01/27/2024).
[7] 2024. Web3.js. https://web3js.org/. (Accessed on 01/27/2024).
[8] Areej Abdullah Alhogail and Afrah Alsabih. 2021. Applying machine learning and natural language processing to detect phishing email. *Comput. Secur.* (2021).
[9] Zahra Ebadi Ansaroudi, Roberto Carbone, Giada Sciarretta, and Silvio Ranise. 2023. Control is Nothing Without Trust a First Look into Digital Identity Wallet Trends. In *Proc. of the 37th Data and Applications Security and Privacy (DBSec 2023)*.
[10] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. 2023. Clockwork Finance: Automated Analysis of Economic Security in Smart Contracts. In *Proc. of the 44th IEEE Symposium on Security and Privacy (S&P 2023)*.
[11] Kushal Babel, Mojan Javaheripi, Yan Ji, Mahimna Kelkar, Farinaz Koushanfar, and Ari Juels. 2023. Lanturn: Measuring Economic Security of Smart Contracts Through Adaptive Learning. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2023)*.
[12] Henry Birge-Lee, Liang Wang, Daniel McCarney, Roland Shoemaker, Jennifer Rexford, and Prateek Mittal. 2021. Experiences Deploying Multi-Vantage-Point Domain Validation at Let's Encrypt. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.
[13] Remco Bloemen, Leonid Logvinov, and Jacob Evans. 2017. EIP-712: Typed structured data hashing and signing. https://eips.ethereum.org/EIPS/eip-712. (Accessed on 01/27/2024).
[14] Priyanka Bose, Dipanjan Das, Yanju Chen, Yu Feng, Christopher Kruegel, and Giovanni Vigna. 2022. SAILFISH: Vetting Smart Contract State-Inconsistency Bugs in Seconds. In *Proc. of the 43rd IEEE Symposium on Security and Privacy, (S&P 2022)*.
[15] Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. 2021. Account Abstraction Using Alt Mempool. https://eips.ethereum.org/EIPS/eip-4337. (Accessed on 04/05/2024).
[16] Stefano Calzavara, Riccardo Focardi, Matteo Maffei, Clara Schneidewind, Marco Squarcina, and Mauro Tempesta. 2018. WPSE: Fortifying Web Protocols via Browser-Side Security Monitoring. In *Proc. of the 27th USENIX Security Symposium (USENIX Security 2018)*.
[17] Wayne Chang, Gregory Rocco, Brantly Millegan, Nick Johnson, and Oliver Terbu. 2021. ERC-4361: Sign-In with Ethereum. https://eips.ethereum.org/EIPS/eip-4361. (Accessed on 01/27/2024).
[18] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding Ethereum via Graph Analysis. *ACM Trans. Internet Techn.* (2020).

[19] Ting Chen, Yufei Zhang, Zihao Li, Xiapu Luo, Ting Wang, Rong Cao, Xiuzhuo Xiao, and Xiaosong Zhang. 2019. TokenScope: Automatically Detecting Inconsistent Behaviors of Cryptocurrency Tokens in Ethereum. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2019)*.
[20] Coinmarketcap. 2024. Highest Price NFT Stats. https://coinmarketcap.com/. (Accessed on 01/27/2024).
[21] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *Proc. of the 41th IEEE Symposium on Security and Privacy (S&P 2020)*.
[22] DappRadar. 2024. Midas Miner. https://dappradar.com/dapp/midas-miner. (Accessed on 01/27/2024).
[23] Dipanjan Das, Priyanka Bose, Nicola Ruaro, Christopher Kruegel, and Giovanni Vigna. 2022. Understanding Security Issues in the NFT Ecosystem. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2022)*.
[24] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. 2020. The Cookie Hunter: Automated Black-box Auditing for Web Authentication and Authorization Flaws. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2020)*.
[25] EarthWeb. 2022. Ethereum Wallet MetaMask Passes 30M Users, Plans DAO and Token. https://decrypt.co/95039/metamask-consensys-30-million-users. (Accessed on 01/27/2024).
[26] Uriel Feige, Amos Fiat, and Adi Shamir. 1988. Zero-Knowledge Proofs of Identity. *J. Cryptol.* (1988).
[27] Joel Frank, Cornelius Aschermann, and Thorsten Holz. 2020. ETHBMC: A Bounded Model Checker for Smart Contracts. In *Proc. of 29th USENIX Security Symposium (USENIX Security 2020)*.
[28] Mohammad Ghasemisharif, Chris Kanich, and Jason Polakis. 2022. Towards Automated Auditing for Account and Session Management Flaws in Single Sign-On Deployments. In *Proc. of the 43rd IEEE Symposium on Security and Privacy (S&P 2022)*.
[29] Mohammad Ghasemisharif, Amrutha Ramesh, Stephen Checkoway, Chris Kanich, and Jason Polakis. 2018. O Single Sign-Off, Where Art Thou? An Empirical Analysis of Single Sign-On Account Hijacking and Session Management on the Web. In *Proc. of the 27th USENIX Security Symposium (USENIX Security 2018)*.
[30] Maximilian Golla, Grant Ho, Marika Lohmus, Monica Pulluri, and Elissa M. Redmiles. 2021. Driving 2FA Adoption at Scale: Optimizing Two-Factor Authentication Notification Design Patterns. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.
[31] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. 2023. Token meets Wallet: Formalizing Privacy and Revocation for FIDO2. In *Proc. of the 44th IEEE Symposium on Security and Privacy (S&P 2023)*.
[32] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang, and Yajin Zhou. 2023. TxPhishScope: Towards Detecting and Understanding Transaction-based Phishing on Ethereum. In *Proc. of the ACM Conference on Computer and Communications Security (CCS 2023)*.
[33] Ningyu He, Ruiyi Zhang, Haoyu Wang, Lei Wu, Xiapu Luo, Yao Guo, Ting Yu, and Xuxian Jiang. 2021. EOSAFE: Security Analysis of EOSIO Smart Contracts. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.
[34] Louis Jannett, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. 2022. DISTINCT: Identity Theft using In-Browser Communications in Dual-Window Single Sign-On. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2022)*.
[35] Don Johnson, Alfred Menezes, and Scott A. Vanstone. 2001. The Elliptic Curve Digital Signature Algorithm (ECDSA). *Int. J. Inf. Sec.* 1, 1 (2001), 36–63. https://doi.org/10.1007/S102070100002
[36] Nikolaos Karapanos and Srdjan Capkun. 2014. On the Effective Prevention of TLS Man-in-the-Middle Attacks in Web Applications. In *Proc. of the 23rd USENIX Security Symposium (USENIX Security 2014)*.
[37] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. 2023. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *Proc. of the 32nd USENIX Security Symposium (USENIX Security 2023)*, Joseph A. Calandrino and Carmela Troncoso (Eds.).
[38] Maina Korir, Simon Parkin, and Paul Dunphy. 2022. An Empirical Study of a Decentralized Identity Wallet: Usability, Security, and Perspectives on User Control. In *Proc. of the 8th Symposium on Usable Privacy and Security (SOUPS 2022)*.
[39] Katharina Krombholz, Karoline Busse, Katharina Pfeffer, Matthew Smith, and Emanuel von Zezschwitz. 2019. "If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS. In *Proc. of the IEEE Symposium on Security and Privacy (S&P 2019)*.
[40] Daniele Lain, Kari Kostiainen, and Srdjan Capkun. 2022. Phishing in Organizations: Findings from a Large-Scale and Long-Term Study. In *Proc. of the 43rd IEEE Symposium on Security and Privacy (S&P 2022)*.
[41] Leona Lassak, Annika Hildebrandt, Maximilian Golla, and Blase Ur. 2021. "It's Stored, Hopefully, on an Encrypted Server": Mitigating Users' Misconceptions About FIDO2 Biometric WebAuthn. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.

[42] Xigao Li, Anurag Yepuri, and Nick Nikiforakis. 2023. Double and Nothing: Understanding and Detecting Cryptocurrency Giveaway Scams. In *Proc. of the 30th Annual Network and Distributed System Security Symposium (NDSS 2023)*.

[43] Chia-Hung Liao, Xue-Qin Guan, Jen-Hao Cheng, and Shyan-Ming Yuan. 2022. Blockchain-based identity management and access control framework for open banking ecosystem. *Future Gener. Comput. Syst.* (2022).

[44] Yang Liu, Debiao He, Mohammad S. Obaidat, Neeraj Kumar, Muhammad Khurram Khan, and Kim-Kwang Raymond Choo. 2020. Blockchain-based identity management systems: A review. *J. Netw. Comput. Appl.* (2020).

[45] Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, Qi Li, and Yih-Chun Hu. 2022. Make Web3.0 Connected. *IEEE Trans. Dependable Secur. Comput.* (2022).

[46] Sanam Ghorbani Lyastani, Michael Schilling, Michaela Neumayr, Michael Backes, and Sven Bugiel. 2020. Is FIDO2 the Kingslayer of User Authentication? A Comparative Usability Study of FIDO2 Passwordless Authentication. In *Proc. of the 41th IEEE Symposium on Security and Privacy (S&P 2020)*.

[47] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. 2014. A Study of Probabilistic Password Models. In *Proc. of the IEEE Symposium on Security and Privacy (S&P 2014)*.

[48] Christian Mainka, Vladislav Mladenov, Jörg Schwenk, and Tobias Wich. 2017. SoK: Single Sign-On Security - An Evaluation of OpenID Connect. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P 2017)*.

[49] Robert McLaughlin, Christopher Kruegel, and Giovanni Vigna. 2023. A Large Scale Study of the Ethereum Arbitrage Ecosystem. In *Proc. of the 32nd USENIX Security Symposium (USENIX Security 2023)*.

[50] Collins W. Munyendo, Philipp Markert, Alexandra Nisenoff, Miles Grant, Elena Korkes, Blase Ur, and Adam J. Aviv. 2022. "The Same PIN, Just Longer": On the (In)Security of Upgrading PINs from 4 to 6 Digits. In *Proc. of the 31st USENIX Security Symposium (USENIX Security 2022)*.

[51] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. (Accessed on 01/27/2024).

[52] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. 2019. Beyond Credential Stuffing: Password Similarity Models Using Neural Networks. In *Proc. of the IEEE Symposium on Security and Privacy (S&P 2019)*.

[53] Bijeeta Pal, Mazharul Islam, Marina Sanusi Bohuk, Nick Sullivan, Luke Valenta, Tara Whalen, Christopher A. Wood, Thomas Ristenpart, and Rahul Chatterjee. 2022. Might I Get Pwned: A Second Generation Compromised Credential Checking Service. In *Proc. of the 31st USENIX Security Symposium (USENIX Security 2022)*.

[54] Dario Pasquini, Marco Cianfriglia, Giuseppe Ateniese, and Massimo Bernaschi. 2021. Reducing Bias in Modeling Real-world Password Strength via Deep Learning and Dynamic Dictionaries. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.

[55] Dario Pasquini, Ankit Gangwal, Giuseppe Ateniese, Massimo Bernaschi, and Mauro Conti. 2021. Improving Password Guessing via Representation Learning. In *Proc. of the 42nd IEEE Symposium on Security and Privacy (S&P 2021)*.

[56] Williams Peter. 2023. Web3 Authentication: What is it? What Use Can It Have? https://medium.com/coinmonks/web3-authentication-what-is-it-what-use-can-it-have-71bca4a70895. (Accessed on 04/17/2024).

[57] Pieter Philippaerts, Davy Preuveneers, and Wouter Joosen. 2022. OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem. In *Proc. of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2022)*.

[58] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. 2021. Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit. In *Proc. of the 25th Financial Cryptography and Data Security (FC 2021)*.

[59] Joshua Reynolds, Trevor Smith, Ken Reese, Luke Dickinson, Scott Ruoti, and Kent E. Seamons. 2018. A Tale of Two Studies: The Best and Worst of YubiKey Usability. In *Proc. of the IEEE Symposium on Security and Privacy (S&P 2018)*.

[60] Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi. 2019. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks. In *Proc. of the 26th Annual Network and Distributed System Security Symposium (NDSS 2019)*.

[61] Christoph Sendner, Huili Chen, Hossein Fereidooni, Lukas Petzi, Jan König, Jasper Stang, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2023. Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning. In *Proc. of the 30th Annual Network and Distributed System Security Symposium (NDSS 2023)*.

[62] I Wayan Budi Sentana, Muhammad Ikram, and Mohamed Ali Kâafar. 2023. An Empirical Analysis of Security and Privacy Risks in Android Cryptocurrency Wallet Apps. In *Proc. of the 21st Applied Cryptography and Network Security (ACNS 2023)*.

[63] Sven Smolka, Jens-Rene Giesen, Pascal Winkler, Oussama Draissi, Lucas Davi, Ghassan Karame, and Klaus Pohl. 2023. Fuzz on the Beach: Fuzzing Solana Smart Contracts. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2023)*.

[64] Sunbeom So, Seongjoon Hong, and Hakjoo Oh. 2021. SmarTest: Effectively Hunting Vulnerable Transaction Sequences in Smart Contracts through Language Model-Guided Symbolic Execution. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.

[65] Sunbeom So, Myungho Lee, Jisu Park, Heejo Lee, and Hakjoo Oh. 2020. VERISMART: A Highly Precise Safety Verifier for Ethereum Smart Contracts. In *Proc. of the IEEE Symposium on Security and Privacy (S&P 2020)*.

[66] Juraj Somorovsky, Andreas Mayer, Jörg Schwenk, Marco Kampmann, and Meiko Jensen. 2012. On Breaking SAML: Be Whoever You Want to Be. In *Proc. of the 21th USENIX Security Symposium (USENIX Security 2012)*.

[67] Avinash Sudhodanan and Andrew Paverd. 2022. Pre-hijacked accounts: An Empirical Study of Security Failures in User Account Creation on the Web. In *Proc. of the 31st USENIX Security Symposium (USENIX Security 2022)*.

[68] Martin Holst Swende and Nick Johnson. 2016. ERC-191: Signed Data Standard. https://eips.ethereum.org/EIPS/eip-191. (Accessed on 01/27/2024).

[69] Sijun Tan, Weikeng Chen, Ryan Deng, and Raluca Ada Popa. 2023. MPCAuth: Multi-factor Authentication for Distributed-trust Systems. In *Proc. of the 44th IEEE Symposium on Security and Privacy (S&P 2023)*.

[70] theblock.co. 2024. Total value locked across DeFi protocols has surged to levels last seen before the collapse of FTX. https://www.theblock.co/post/269669/total-value-locked-across-defi-protocols-has-surged-to-levels-last-seen-before-the-collapse-of-ftx. (Accessed on 01/27/2024).

[71] Md Shahab Uddin, Mohammad Mannan, and Amr M. Youssef. 2021. Horus: A Security Assessment Framework for Android Crypto Wallets. In *Proc. of the 17th Security and Privacy in Communication Networks (SecureComm 2021)*.

[72] Friedhelm Victor and Andrea Marie Weintraud. 2021. Detecting and Quantifying Wash Trading on Decentralized Cryptocurrency Exchanges. In *Proc. of the ACM Web Conference (WWW 2021)*.

[73] Fabian Vogelsteller and Tyler Yasaka. 2017. ERC-725: General data key/value store and execution. https://eips.ethereum.org/EIPS/eip-725. (Accessed on 01/27/2024).

[74] Shicheng Wan, Hong Lin, Wensheng Gan, Jiahui Chen, and Philip S. Yu. 2023. Web3: The Next Internet Revolution. *CoRR* abs/2304.06111 (2023). arXiv:2304.06111

[75] Ke Coby Wang and Michael K. Reiter. 2021. Using Amnesia to Detect Credential Database Breaches. In *Proc. of the 30th USENIX Security Symposium (USENIX Security 2021)*.

[76] Bryan White, Aniket Mahanti, and Kalpdrum Passi. 2022. Characterizing the OpenSea NFT Marketplace. In *Proc. of the Companion of The Web Conference (WWW Companion 2022)*.

[77] Ming Xu, Chuanwang Wang, Jitao Yu, Junjie Zhang, Kai Zhang, and Weili Han. 2021. Chunk-Level Password Guessing: Towards Modeling Refined Password Composition Representations. In *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS 2021)*.

[78] Kailun Yan, Jilian Zhang, Xiangyu Liu, Wenrui Diao, and Shanqing Guo. 2023. Bad Apples: Understanding the Centralized Security Risks in Decentralized Ecosystems. In *Proc. of the ACM Web Conference (WWW 2023)*.

[79] Xiaohui Yang and Wenjie Li. 2020. A zero-knowledge-proof-based digital identity management scheme in blockchain. *Comput. Secur.* (2020).

[80] Fangyi Yu and Miguel Vargas Martin. 2022. GNPassGAN: Improved Generative Adversarial Networks For Trawling Offline Password Guessing. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P 2022)*.

[81] Mengya Zhang, Xiaokuan Zhang, Yinqian Zhang, and Zhiqiang Lin. 2020. TXSPECTOR: Uncovering Attacks in Ethereum from Transactions. In *Proc. of the 29th USENIX Security Symposium (USENIX Security 2020)*.

[82] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. 2021. On the Just-In-Time Discovery of Profit-Generating Transactions in DeFi Protocols. In *Proc. of the 42nd IEEE Symposium on Security and Privacy (S&P 2021)*.

[83] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2023. SoK: Decentralized Finance (DeFi) Attacks. In *Proc. of the 44th IEEE Symposium on Security and Privacy (S&P 2023)*.

A EXTENSION FIELD OF THE MESSAGE

The extension fields, as shown in Listing 5, include many useful optional fields with limited security impact on Web3 authentication. Here, we briefly explain them.

```
address = "0x" 40*40HEXDIG
  ; Must also conform to captilization
  ; checksum encoding specified in EIP-55 (EOAs).

version = "1"

chain-id = 1*DIGIT
  ; See EIP-155 for valid CHAIN_IDs.

issued-at = date-time
```
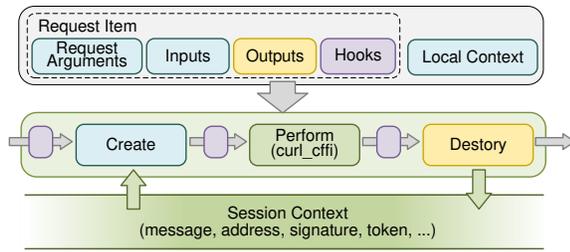
Figure 6: FlexRequest in Operation.

```
expiration-time = date-time

not-before = date-time

request-id = *pchar
  ; See RFC 3986 for the definition of "pchar".
```

Listing 5: Extension Fields

address. Listing 2 has an address field to tell the user which account to sign the message. However, the address field is unnecessary because the crypto wallet UI has more straightforward prompts, usually including account avatars and aliases.

version. The version field indicates the version of the message format. Also, the server can determine the version of the message by parsing the message format.

chain-id. The chain-id field indicates the chain ID of the blockchain network. The server can determine the chain ID of the message by parsing the message format. The chain ID is usually used to prevent replay attacks across different blockchain networks. For example, if the user signs a message on the Ethereum *testnet*, the attacker cannot use the signature to invade the user's account on the Ethereum *mainnet*.

issued-at, expiration-time, not-before-time. The three fields prompt the validity period of the message. The issued-at field is a datetime string of the current time. The expiration-time field indicates when the signed message is invalid. The not-before-time field indicates when the signed message becomes valid. The server must have the corresponding verification if these fields appear in a message.

request-id. Users may have multiple devices to log in to the website using Web3 authentication. The request-id field is a system-specific identifier that uniquely refers to the login request.

B FURTHER DETAILS ON FLEXREQUEST

To illustrate the challenges of API testing, consider the QUERY responses and AUTH requests from two different websites as shown in Listing 6 and Listing 7. In the case of galler.io, the QUERY returns

a message directly, whereas, for element.market, the QUERY only returns a nonce. The AUTH request parameters also differ in both examples, requiring distinct test scripts for each website. This would typically lead to considerable code duplication and challenges in maintaining the codebase.

```
1  QUERY Response:
2  {'data':{'auth':{'message':'This is Galler, welcome...
3  timestamp: 1625468800000'}}}
4
5  AUTH Request:
6  {method:'POST', url:'https://www.galler.io/api/v1',
7  headers:{...}, data:{address:'{{ addr }}',
8  message:'{{ msg }}', signature:'{{ sig }}'}}
```

Listing 6: The Response and Request of galler.io

```
1  QUERY Response:
2  {'data':{'auth':{'nonce':'3deca92b'}}}
3
4  AUTH Request:
5  {method:'POST', url:'https://api.element.market/graphql',
6  headers:{'x-viewer-addr':'{{ addr }}',...}, data:{message:
7  '{{ msg }}', nonce:'{{ nonce }}', signature:'{{ sig }}'}}
```

Listing 7: The Response and Request of element.market

However, FlexRequest's automatic replacement mechanism harmonizes these differences. As seen in Listings 6 and 7, keys such as *addr*, *msg*, etc., are set in the Auth requests. By managing the values of these keys, developers can perform unified testing across multiple APIs. For instance, to test whether a token can still be obtained with an incorrect signature, one needs to set the value of the *sig* key to be empty, which would put an empty signature in all Auth requests.

FlexRequest in Operation. As shown in Figure 6, FlexRequest operates in three phases for each request:

- CREATE. FlexRequest substitutes keys in the request with values from the *local context*, *session context*, and *inputs*.
- EXECUTE. FlexRequest carries out the request using *curl_cffi*.
- DESTORY. FlexRequest retrieves values from the specified position in the response according to *outputs*, and stores them in *the session context* in a key-value format.

A Request Item encapsulates all the pertinent details of a request, including the URL, headers, and so forth. It also includes two special parameters, *inputs* and *outputs*. Developers can define default values (*inputs*) and return values (*outputs*) in the API's configuration file and set test values in each checker's *local context*. The local context is only valid for the current request, and those values are first filled into the request.

The key-value replacement function of FlexRequest is very powerful, and the value of a key can even be an executable expression. FlexRequest uses Python's *eval* function to evaluate expressions and return results. Therefore, developers can even simulate front-end JavaScript execution by executable expressions. Besides, FlexRequest also provides hooks for complex API testing.