WalletConnect v2.0 SDK

Security Assessment

September 15, 2023

Prepared for:
Derek Rein and Pedro Gomes
WalletConnect

Prepared by: **Alex Useche and Emilio López**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

WalletConnect engaged Trail of Bits to review the security of v2.0 of its SDK. From March 20 to March 31, 2023, a team of two consultants conducted a security review and lightweight threat model of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with partial knowledge of the system, including access to the client-side source code and documentation of the SDK. We performed static and dynamic testing of the target system and its codebase, using both automated and manual processes. Additionally, we performed a lightweight threat model of the core WalletConnect functionality.

Summary of Findings

The audit did not uncover any significant flaws or defects that could impact system confidentiality, integrity, or availability. A summary of the findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	0
Informational	2
Undetermined	1

CATEGORY BREAKDOWN

Category	*Count*
Cryptography	2
Data Exposure	1
Patching	1

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Alex Useche, Consultant
alex.useche@trailofbits.com

Emilio López, Consultant
emilio.lopez@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
March 17, 2023	Pre-project kickoff call
March 21, 2023	Threat model discussion call
April 6, 2023	Delivery of report draft; report readout meeting
May 5, 2023	Delivery of final report
September 15, 2023	Delivery of revised final report

Project Goals

The engagement was scoped to provide a security assessment of the WalletConnect v2.0 SDK, including the Core, Sign, and Auth APIs. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can attackers impersonate connections by conducting attacks such as nonce replay?
- Can attackers impersonate dApps and trick users into approving illegitimate transactions?
- Is session data stored and handled correctly via storage logic?
- Is user data untrusted and effectively parsed and validated?
- What are common ways in which attackers can attempt to compromise WalletConnect and its users, given the system's design and architecture?
- Is the system susceptible to any known cryptographic attacks?

Project Targets

The engagement involved a review and testing of the targets listed below.

WalletConnect Monorepository

Repository	https://github.com/WalletConnect/walletconnect-monorepo
Version	`5faa1cc2a8936644f46d946cec08c8af63b082f7`
Types	JavaScript, TypeScript
Platforms	Web, mobile

WalletConnect Auth SDK

Repository	https://github.com/WalletConnect/auth-client-js
Version	`85406023fba0eb3ceb09822fca8bb51492095fe0`
Types	JavaScript, TypeScript
Platforms	Web, mobile

WalletConnect Utils Repository

Repository	https://github.com/WalletConnect/walletconnect-utils
Version	`d6df9698dcd2fe396382175fec7dc95eb50c8825`
Types	JavaScript, TypeScript
Platforms	Web, mobile

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- A review of the documentation provided by the client, including but not limited to the following:
    - A high-level overview of the system's design
    - Diagrams of the system's architecture
    - A detailed description of the system's design
    - An overview of the protocol
- A review of the documentation included in the README files for the various repositories in scope
- A manual code analysis of the Core, Sign, and Auth APIs
- Automated static analysis with tools such as Semgrep and CodeQL
- Dynamic analysis using the demo applications set up by the WalletConnect team
- A review of the use of cryptography to ensure that cryptographic primitives are used correctly, that cryptographic algorithm and parameter choices are sound, and that the system is not susceptible to known cryptographic attacks

Additionally, we focused primarily on concerns that could arise from potential malicious extensions, browser exploits, and other attacks that could allow malicious actors to obtain session data, replay requests, or impersonate dApps.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- The threat model was focused only on the Core, Sign, and Auth APIs.
- The Relay infrastructure, although mentioned in the threat model, was out of scope for this review.

- During the code review, we checked whether the codebase’s dependencies, such as the cryptography library, are used and integrated correctly. However, we did not perform a dedicated code review of the dependencies themselves.

Threat Model

As part of the audit, Trail of Bits conducted a lightweight threat model, drawing from Mozilla's "Rapid Risk Assessment" methodology and the National Institute of Standards and Technology's (NIST) guidance on data-centric threat modeling (NIST 800-154). We began our assessment of the design of the WalletConnect SDK, including the Auth and Sign APIs, by reviewing the documentation provided by the client.

Data Types

The primary data handled by WalletConnect includes session data from subscriptions to various topics, wallet addresses, and blockchain transaction data.

Data Flow

The following diagram presents the data flow that occurs during the use of WalletConnect.

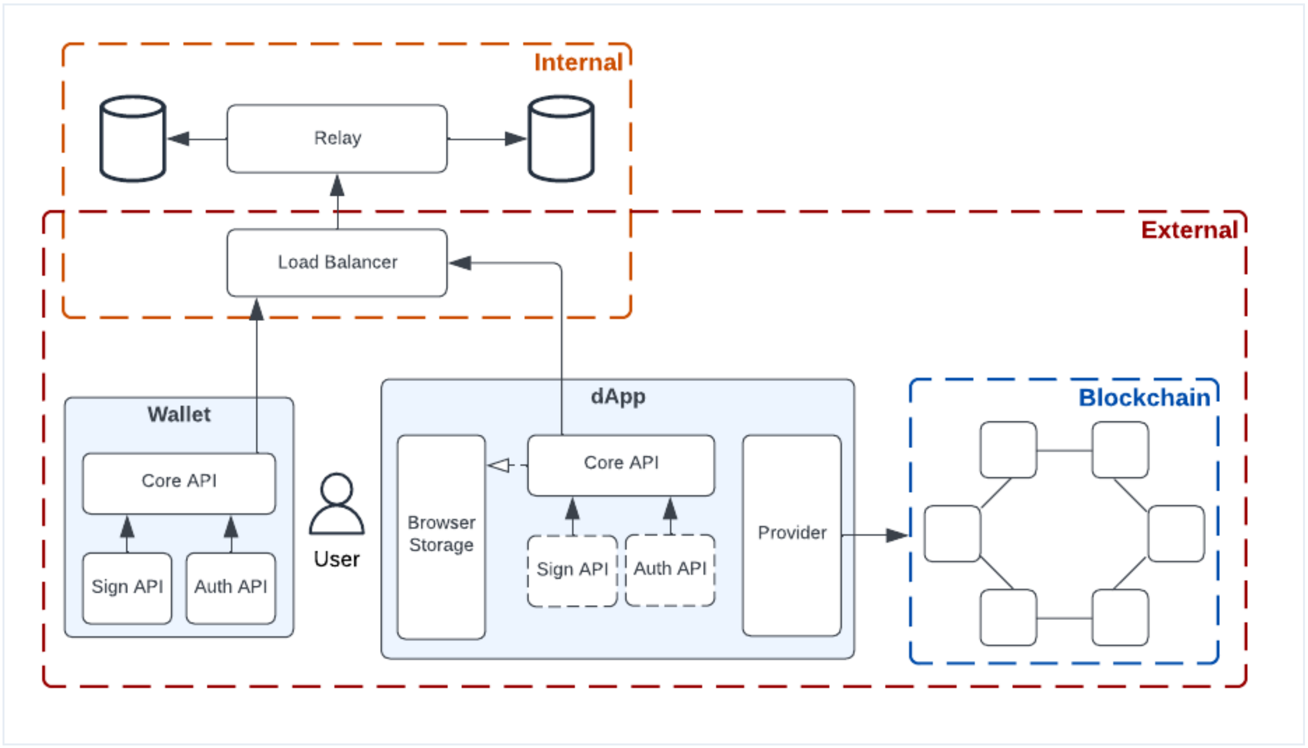


Figure 1: The WalletConnect data flow

Components

The following table describes each WalletConnect component and dependency identified for our analysis. It also indicates whether the component or dependency is *not* in scope; an asterisk (*) next a component's name indicates that it was out of scope for this assessment. We explored the implications of threats involving out-of-scope components that directly affect in-scope components, but we did not consider threats to the out-of-scope components themselves.

Component	Description
Relay Infrastructure (*)	The Relay infrastructure is the AWS infrastructure hosting the following components required for dApps to communicate with user wallets: ● Relay server: A WebSocket server that allows wallets and dApps to communicate ● Address book: Redis and MongoDB databases that indicate which client is connected to which WebSocket server ● Mailbox: Keeps track of messages when a given party is offline so that communications can result when a connection is reestablished
Wallet	A wallet is a user-owned wallet with public and private keys.
dApp	A dApp allows users to connect their wallets and perform transactions using WalletConnect.
Sign API	The Sign API allows dApps to establish sessions with wallets. Sessions are established through a relay using WebSocket and remote JSON-RPC transport with methods and events.
Auth API	The Auth API allows dApps to authenticate wallet users to log in with their wallets by automatically signing an authentication message. Authentication messages can also be used to verify address ownership via a single signature request.

Core Controllers

The Core API consists of a series of controllers with narrow sets of responsibilities. Each controller is listed and described below.

Component	Description	Interacts With
Crypto	Manages keys (generation, retrieval, derivation) and performs encryption and decryption	Keychain
Expirer	Allows the registration of keys (used in key-value pairs) with expirations and triggers events when they expire	Storage
History	Persists JSON-RPC requests and their responses	Storage
Keychain	Persistent setter/getter for keys	Storage
Messages	Persistent setter/getter for messages	Storage
Pairing	Allows users to initiate (create) pairing requests, accept (pair) and activate pairings, ping pairing peers, delete pairings, and perform some other operations	Crypto, Expirer, History, Relayer
Publisher	Publishes a message on the RPC channel and emits a success/failure event	Relayer
Relayer	Handles RPC subscription, unsubscription, transport, and publishing	Messages, Publisher, Subscriber
Store	A generic key-value store that writes to an underlying `Storage` object	-
Subscriber	Allows subscription to and unsubscription from topics	Relayer, TopicMap
TopicMap	A key-set store with set, get, item-in-key, and delete operations	-

Trust Zones

Trust zones capture logical boundaries where controls should or could be enforced by the system and allow developers to implement controls and policies between components' zones.

Zone	Description	Included Components
External	The wider external-facing internet zone	• Sign API • Auth API • Wallet • dApp
Internal	The internal cloud zone that allows the relay components to communicate with each other	• Relay infrastructure

Trust Zone Connections

This table describes the connections that occur between trust zones.

Originating Zone	Destination Zone	Description	Connection Types	Authentication Types
External	Internal	A wallet reaches out to the relay server through a load balancer to initiate a connection to a dApp using the Auth API. The wallet then listens to connection messages posted by dApps via the relay. Messages are signed using the Sign API.	HTTPS JSON-RPC over WSS	JWT
		A dApp reaches out to the relay server through a load balancer to listen for successful authentication and to subscribe to connection messages. The dApp then reaches out to the relay server and listens to messages posted by wallet applications via the relay. Messages are signed using the Sign API.	JSON-RPC over WSS	Key-based
External	External	A dApp reaches out to a blockchain RPC provider to synchronize blockchain transaction data via a blockchain RPC	HTTPS	N/A

		provider.		
		A user reaches out to a dApp using a web browser.	HTTPS	N/A
Internal	Internal	A load balancer passes communications from external components to the relay server.	HTTPS	N/A

Threat Actors

The following table describes actors who could be malicious, could be induced to undertake an attack, or could be impacted by an attack. Defining these actors is helpful in determining which protections, if any, are necessary to mitigate or remediate a vulnerability.

Actor	Description
WalletConnect User	WalletConnect users use WalletConnect to connect their wallets to supported dApps.
Wallet Developer	Wallet developers control code for wallet applications running on user devices that support WalletConnect.
dApp Developer	dApp developers control the code for dApps that support WalletConnect.
WalletConnect Engineer	Engineers are members of the WalletConnect team who have access to the cloud network hosting the relay server.
External Attacker	External attackers attempt to use vulnerabilities in WalletConnect to compromise users and to traverse the external zone boundary to gain access to the internal network hosting the relay server and other internal components.
Internal Attacker	Internal attackers traverse external boundaries and gain access to the Relay infrastructure in AWS.

Threat Scenarios

The following table describes possible threat scenarios that the system could be vulnerable to, given the design, architecture, and risk profile of the WalletConnect components in scope. The threats listed in this section are not an indication that we discovered a specific vulnerability that would facilitate the attack in the code. Rather, this table serves as a way to list potential attack vectors that could be taken by malicious actors.

Threat	Threat Scenario	Actors	Components
Denial of service	Given that the Relay is centralized in its current state, attackers could attempt to disrupt communications with WalletConnect by conducting denial-of-service attacks.	● External attackers	● Relay infrastructure
Session data exfiltration	Attackers could attempt to conduct malicious browser or browser extension exploits to obtain session data for a WalletConnect connection stored in `localStorage`, including keychain data.	● External attackers	● dApps ● Wallets
Cross-site scripting (XSS) of dApps	A dApp developed with XSS vulnerabilities could allow attackers to obtain WalletConnect session data, including keychain values, from the browser's local storage.	● dApp developers	● dApps ● Wallets
Browser attacks	Attackers could attempt to conduct malicious browser or browser extension exploits to propose new transactions with a wallet connected to a dApp via WalletConnect.	● External attackers	● dApps ● Wallets
Malicious dApp	A malicious developer could develop a dApp crafted to replicate the look and feel of a well-known, legitimate dApp in order to trick users into connecting their wallets, allowing the attacker to steal their funds through illegitimate transactions.	● Malicious dApp developers	● dApps ● Wallets

Supply chain attack	An attacker could compromise one of the open-source dependencies used by WalletConnect and include malicious code that proposes and/or signs malicious transactions.	● External attackers	● dApps ● Wallets
Shoulder surfing attack	An attacker in the vicinity of a pairing in progress could capture the shared key shown in the QR code and then attempt a cryptographic or man-in-the-middle attack.	● Local attackers	● dApps

Recommendations

Based on our assessment of the codebase's security based on the threat modeling exercise, we recommend that WalletConnect take the following actions:

- Decentralize the Relay infrastructure to minimize the risk that a denial-of-service attack could cause the loss of availability for all users of WalletConnect. WalletConnect shared with Trail of Bits engineers its work, along with design and architecture diagrams, on decentralizing the Relay infrastructure. This work was ongoing at the time of this audit.

- Consider adjusting the authentication protocol to allow dApps to authenticate themselves with a wallet or other long-term identifier like those required for users to authenticate themselves. Currently, the authentication protocol uses a challenge-response protocol to authenticate users by having them produce a signature using their wallet's private keys. However, there is not an analogous authentication for the dApps that users interact with, as all of the dApp keys in the authentication protocol are ephemeral. As a result, if a dApp is compromised, an attacker can impersonate the dApp to all connected users.

  This recommendation was discussed with the WalletConnect team, and in response, WalletConnect shared an ongoing plan in place to help address this concern.

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time
CodeQL	An open-source tool for finding bugs via interprocedural analysis of code paths
JetBrains Inspectors	Built-in JetBrains inspectors for TypeScript codebases

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The in-scope components do not perform many arithmetic operations.	Not Applicable
Auditing	The codebase implements various levels of logging.	Satisfactory
Authentication / Access Controls	The WalletConnect authentication protocol authenticates users by requiring them to sign a payload using their wallet's secret keys. However, the codebase is missing some replay protections (TOB-WCSDK-2).	Moderate
Complexity Management	The application consists of well-organized and documented TypeScript code. Logic is separated via various APIs and controllers with single responsibilities.	Satisfactory
Cryptography and Key Management	The codebase relies on cryptographically secure random number generators for generating random values and secrets, and it uses modern cryptographic algorithms for hashing, encrypting, and key exchange. However, the codebase relies on local storage for storing cryptographic secrets (TOB-WCSDK-4).	Satisfactory
Data Handling	Due to the nature of the protocol, all data is stored in `localStorage`, which is insecure as it does not offer the same protections against XSS attacks offered by cookies. However, most data validation is done via standard TypeScript libraries. Where data is validated using custom parsing and validation functions rather than TypeScript libraries, there are tests to ensure they behave correctly.	Satisfactory
Decentralization	Although we did not review the out-of-scope Relay	Further

	infrastructure, we note that it is currently centralized. However, there were ongoing, concrete plans to decentralize this component at the time of this review. For those reasons, we mark this category as "Further Investigation Required."	**Investigation Required**
Documentation	The application code is well documented. Each API is described in detail on the WalletConnect documentation page.	**Strong**
Maintenance	The code is organized into logical modules, making the code relatively straightforward to maintain.	**Satisfactory**
Memory Safety and Error Handling	All code for this review was written in TypeScript, a memory-safe language.	**Strong**
Testing and Verification	The codebase includes unit and integration tests for core functionality. We recommend developing additional testing focused on security-related edge cases. Additionally, consider using fuzzing for the custom parsing of data.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Use of outdated dependencies	Patching	**Informational**
2	No protocol-level replay protections in WalletConnect	Cryptography	**Undetermined**
3	Key derivation code could produce keys composed of all zeroes	Cryptography	**Informational**
4	Insecure storage of session data in local storage	Data Exposure	**Medium**

Detailed Findings

1. Use of outdated dependencies	
Severity: **Informational**	Difficulty: **Undetermined**
Type: Patching	Finding ID: TOB-WCSDK-1
Target: `walletconnect-monorepo`, `walletconnect-utils`	

Description

We used `npm audit` and `lerna-audit` to detect the use of outdated dependencies in the codebase. These tools discovered a number of vulnerable packages that are referenced by the `package-lock.json` files.

The following tables describe the vulnerable dependencies used in the `walletconnect-utils` and `walletconnect-monorepo` repositories:

walletconnect-utils			
Dependencies	**Vulnerability Report**	**Vulnerability Description**	**Vulnerable Versions**
`glob-parent`	CVE-2020-28469	Regular expression denial of service in enclosure regex	< 5.1.2
`minimatch`	CVE-2022-3517	Regular expression denial of service when calling the `braceExpand` function with specific arguments	< 3.0.5
`nanoid`	CVE-2021-23566	Exposure of sensitive information to an unauthorized actor in `nanoid`	3.0.0–3.1.30

walletconnect-monorepo			
Dependencies	**Vulnerability Report**	**Vulnerability Description**	**Vulnerable Versions**
`flat`	CVE-2020-36632	`flat` vulnerable to prototype pollution	< 5.0.1
`minimatch`	CVE-2022-3517	Regular expression denial of service when calling the `braceExpand` function with specific arguments	< 3.0.5
`request`	CVE-2023-28155	Bypass of SSRF mitigations via an attacker-controller server that does a cross-protocol redirect (HTTP to HTTPS, or HTTPS to HTTP)	<= 2.88.2

In many cases, the use of a vulnerable dependency does not necessarily mean the application is vulnerable. Vulnerable methods from such packages need to be called within a particular (exploitable) context. To determine whether the WalletConnect SDK is vulnerable to these issues, each issue will have to be manually triaged.

While these specific libraries were outdated at the time of the review, there were already checks in place as part of the CI/CD pipeline of application development of WalletConnect to keep track of these issues.

Recommendations

Short term, update the project dependencies to their latest versions wherever possible. Use tools such as `retire.js`, `npm audit`, and `yarn audit` to confirm that no vulnerable dependencies remain.

2. No protocol-level replay protections in WalletConnect

Severity: **Undetermined**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-WCSDK-2
Target: WalletConnect v2 protocol	

Description

Applications and wallets using WalletConnect v2 can exchange messages using the WalletConnect protocol through a public WebSocket relay server. Exchanged data is encrypted and authenticated with keys unknown to the relay server. However, using dynamic testing during the audit, we observed that the protocol does not protect against replay attacks.

The WalletConnect authentication protocol is essentially a challenge-response protocol between users and servers, where users produce signatures using the private keys from their wallets. A signature is performed over a message containing, among many other components, a nonce value chosen by the server. This nonce value is intended presumably to prevent an adversary from replaying an old signature that a user generated to authenticate themselves. However, there does not seem to be any validation against this nonce value (except validation that it exists), so the library would accept replayed signatures.

In addition to missing validation of the nonce value, the payload for the signature does not appear to include the pairing topic for the pairing established between a user and the server. Because the authentication protocol runs only over an existing pairing, it would make sense to include the pairing topic value inside the signature payload. Doing so would prevent a malicious user from replaying another user's previously generated signature for a new pairing that they establish with the server.

To repeat our experiment that uncovered this issue, pair the React App demo application with the React Wallet demo application and intercept the traffic generated from the React App demo application (e.g., use a local proxy such as BurpSuite). Initiate a transaction from the application, capture the data sent through the WebSocket channel, and confirm the transaction in the wallet. A sample captured message is shown in figure 2.1. Now, edit the `message` field slightly and add "==" to the end of the string ("=" is the Base64 padding character). Finally, replay (resend) the captured data. A new confirmation dialog box should appear in the wallet.

```
{
  "id": 1680643717702847,
  "jsonrpc": "2.0",
  "method": "irn_publish",
  "params": {
    "topic": "42507dee006fe8(...)2d797cccf8c71fa9de4",
    "message": "AFv70BclFEn6MteTRFemaxD7Q7(...)y/eAPv3ETRHL0x86cJ6iflkIww",
    "ttl": 300,
    "prompt": true,
    "tag": 1108
  }
}
```

Figure 2.1: A sample message sent from the dApp

This finding is of undetermined severity because it is not obvious whether and how an attacker could use this vulnerability to impact users.

When this finding was originally presented to the WalletConnect team, the recommended remediation was to track and enforce the correct nonce values. However, due to the distributed nature of the WalletConnect system, this could prove difficult in practice. In response, we have updated our recommendation to use timestamps instead. Timestamps are not as effective as nonces are for preventing replay attacks because it is not always possible to have a secure clock that can be relied upon. However, if nonces are infeasible to implement, timestamps are the next best option.

Recommendations

Short term, update the implementation of the authentication protocol to include timestamps in the signature payload that are then checked against the current time (within a reasonable window of time) upon signature validation. In addition to this, include the pairing topic in the signature payload.

Long term, consider including all relevant pairing and authentication data in the signature payload, such as sender and receiver public keys. If possible, consider using nonces instead of timestamps to more effectively prevent replay attacks.

3. Key derivation code could produce keys composed of all zeroes

Severity: **Informational**	Difficulty: **High**
Type: Cryptography	Finding ID: TOB-WCSDK-3
Target: `walletconnect-monorepo/packages/utils/src/crypto.ts`	

Description

The current implementation of the code that derives keys using the `x25519` library does not enable the `rejectZero` option. If the counterparty is compromised, this may result in a derived key composed of all zeros, which could allow an attacker to observe or tamper with the communication.

```
export function deriveSymKey(privateKeyA: string, publicKeyB: string): string {
  const sharedKey = x25519.sharedKey(
    fromString(privateKeyA, BASE16),
    fromString(publicKeyB, BASE16),
  );
  const hkdf = new HKDF(SHA256, sharedKey);
  const symKey = hkdf.expand(KEY_LENGTH);
  return toString(symKey, BASE16);
}
```

Figure 3.1: The code that derives keys using `x25519.sharedKey` (`walletconnect-monorepo/packages/utils/src/crypto.ts#35–43`)

The `x25519` library includes a warning about this case:

```
/**
 * Returns a shared key between our secret key and a peer's public key.
 *
 * Throws an error if the given keys are of wrong length.
 *
 * If rejectZero is true throws if the calculated shared key is all-zero.
 * From RFC 7748:
 *
 * > Protocol designers using Diffie-Hellman over the curves defined in
 * > this document must not assume "contributory behavior".  Specially,
 * > contributory behavior means that both parties' private keys
 * > contribute to the resulting shared key.  Since curve25519 and
 * > curve448 have cofactors of 8 and 4 (respectively), an input point of
 * > small order will eliminate any contribution from the other party's
 * > private key.  This situation can be detected by checking for the all-
 * > zero output, which implementations MAY do, as specified in Section 6.
 * > However, a large number of existing implementations do not do this.
 *
 * IMPORTANT: the returned key is a raw result of scalar multiplication.
 * To use it as a key material, hash it with a cryptographic hash function.
 */
```

Figure 3.2: Warnings in `x25519.sharedKey`
(`stablelib/packages/x25519/x25519.ts#595–615`)

This finding is of informational severity because a compromised counterparty would already allow an attacker to observe or tamper with the communication.

Exploit Scenario
An attacker compromises the web server on which a dApp is hosted and introduces malicious code in the front end that makes it always provide a low-order point during the key exchange. When a user connects to this dApp with their WalletConnect-enabled wallet, the derived key is all zeros. The attacker passively captures and reads the exchanged messages.

Recommendations
Short term, enable the `rejectZero` flag for uses of the `deriveSymKey` function.

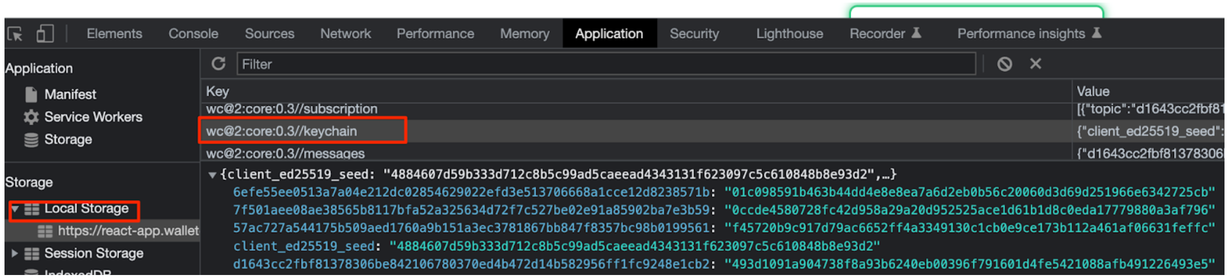
Long term, when using cryptographic primitives, research any edge cases they may have and always review relevant implementation notes. Follow recommended practices and include any defense-in-depth safety checks to ensure the protocol operates as intended.

4. Insecure storage of session data in local storage

Severity: **Medium**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-WCSDK-4
Target: Browser storage	

Description

HTML5 local storage is used to hold session data, including keychain values. Because there are no access controls on modifying and retrieving this data using JavaScript, data in local storage is vulnerable to XSS attacks.



Key	Value
wc@2:core:0.3//subscription	[{"topic":"d1643cc2fbf81
wc@2:core:0.3//keychain	{"client_ed25519_seed":
wc@2:core:0.3//messages	{"d1643cc2fbf81378306

```
▾{client_ed25519_seed: "4884607d59b333d712c8b5c99ad5caeead4343131f623097c5c610848b8e93d2",…}
   6efe55ee0513a7a04e212dc02854629022efd3e513706668a1cce12d8238571b: "01c098591b463b44dd4e8e8ea7a6d2eb0b56c20060d3d69d251966e6342725cb"
   7f501aee08ae38565b8117bfa52a325634d72f7c527be02e91a85902ba7e3b59: "0ccde4580728fc42d958a29a20d952525ace1d61b1d8c0eda17779880a3af796"
   57ac727a544175b509aed1760a9b151a3ec3781867bb847f8357bc98b0199561: "f45720b9c917d79ac6652ff4a3349130c1cb0e9ce173b112a461af06631feffc"
   client_ed25519_seed: "4884607d59b333d712c8b5c99ad5caeead4343131f623097c5c610848b8e93d2"
   d1643cc2fbf81378306be842106780370ed4b472d14b582956ff1fc9248e1cb2: "493d1091a904738f8a93b6240eb00396f791601d4fe5421088afb491226493e5"
```

Figure 4.1: Keychain data stored in a browser's `localStorage`

Exploit Scenario

Alice discovers an XSS vulnerability in a dApp that supports WalletConnect. This vulnerability allows Alice to retrieve the dApp's keychain data, allowing her to propose new transactions to the connected wallet.

Recommendations

Short term, consider using cookies to store and send tokens. Enable cross-site request forgery (CSRF) libraries available to mitigate these attacks. Ensure that cookies are tagged with `httpOnly`, and preferably `secure`, to ensure that JavaScript cannot access them.

References

- OWASP HTML5 Security Cheat Sheet: Local Storage

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories

Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.