Gemini Smart Wallet

Security Assessment

August 15, 2025

Prepared for:
Daniel Buchner
Gemini

Prepared by: **Anish Naik and Coriolan Pinhas**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Emily Doucette, Project Manager
emily.doucette@trailofbits.com

The following engineering director was associated with this project:

Benjamin Samuels, Engineering Director, Blockchain
benjamin.samuels@trailofbits.com

The following consultants were associated with this project:

Anish Naik, Consultant
anish.naik@trailofbits.com

Coriolan Pinhas, Consultant
coriolan.pinhas@trailofbits.com

Project Timeline

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

Date	Event
June 4, 2025	Pre-project kickoff call
June 23, 2025	Delivery of report draft and report readout meeting
August 15, 2025	Delivery of final comprehensive report with fix review addendum

Executive Summary

Engagement Overview

Gemini engaged Trail of Bits to review the security of the Gemini Smart Wallet, a self-custodial, embedded cryptocurrency wallet. The wallet does not store a user's private key but instead deploys a Gemini smart account on-chain that uses account abstraction to perform user operations. The wallet also supports passkey authentication and signing.

A team of two consultants conducted the review from June 9 to June 23, 2025, for a total of four engineer-weeks of effort. Our testing efforts focused on reviewing the forked smart contract logic to assess whether user funds are at risk; if the WebAuthn protocol is integrated correctly; and whether traditional wallet operations (signing raw and typed data, switching chains, transaction simulation, etc.) are properly implemented. With full access to source code and documentation, we performed static and dynamic testing of the codebase, using automated and manual processes.

Observations and Impact

We identified two issues that could significantly affect the integrity of the system. A malicious dapp can open the wallet as an iframe and perform a clickjacking attack or other UI-related attacks (TOB-GSW-9). Similarly, TOB-GSW-5 allows a malicious dapp to manipulate transaction parameters without the user knowing.

More generally, we observed that the system lacks sufficient unit testing and user acceptance/interface testing. For example, the issue described in TOB-GSW-8—an incorrect usage of the storage API—can be caught through either a unit test or end-user testing. Similarly, we identified issues in the wallet's UI (TOB-GSW-5). Improving user interface testing and even comparing it to other industry-standard solutions, like MetaMask, will aid in identifying issues like this.

Finally, it is worth noting two other issues that can affect the confidentiality and availability of the system, respectively. First, since the client side is responsible for sending transactions on-chain, an attacker can intercept API keys to services like Pimlico (TOB-GSW-4). This is connected to TOB-GSW-13, which highlights that the system is reliant on third-party services like Pimlico without any redundant, backup services. Thus, if Pimlico gets compromised or has downtime, the Gemini wallet service has to be stopped.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Gemini take the following steps prior to deployment:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or any refactor that may occur when addressing other recommendations.

- **Improve user interface testing.** The audit identified a variety of issues with the user interface of the Gemini wallet. Since what the user sees on this wallet is the last line of defense before the transaction is sent on-chain, it is critical that this is thoroughly tested. We also recommend comparing the user experience of the Gemini wallet to other industry-standard wallets, like MetaMask.

- **Enhance the wallet's UI.** In particular, provide better user insight on specific transaction types (e.g., contract deployment details). Also consider adding warnings for suspicious transactions, such as a transfer to an address or contract previously unknown to the wallet or to a spender address that is an externally owned account.

- **Implement the TODOs in the code.** There are a variety of TODOs that have security implications (e.g., improved error handling).

- **Add a logging and monitoring solution.** There is currently no logging or monitoring solution to track user requests, API usage, or errors. Integrating a solution like Sentry is critical to responding quickly to incidents or bugs in production.

- **Support multiple redundant third-party services.** The system currently relies only on Alchemy and Pimlico. If either service is compromised or has downtime, the Gemini wallet service is also directly affected. Having redundant third-party services will aid in maintaining the availability of the system.

- **Improve documentation and diagrams.** The lack of documentation and workflow diagrams significantly deters manual review and understanding of the system.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	3
Medium	0
Low	4
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Access Controls	2
Configuration	2
Data Exposure	1
Data Validation	4

Project Goals

The engagement was scoped to provide a security assessment of the Gemini Smart Wallet. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can an attacker steal funds from a user's Gemini smart account?
- Does the SDK service leak any sensitive information or data?
- Are traditional wallet operations implemented correctly?
- Does the WebAuthn integration sufficiently support authentication and signing of transactions?
- Do users have all of the information required to safely sign a message or a transaction?

Project Targets

The engagement involved reviewing and testing the following target.

gemini-wallet

Repository	N/A
Version	`40747ad1b918f4b7638f81fb90fc98ad752b85b5`
Type	TypeScript, Solidity
Platform	Web browsers and mobile devices

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:

- **Smart contract logic.** The smart contracts are forked from Nexus, which is a ERC-7579 modular smart contract framework. The core components include a factory contract for deterministic wallet deployment, WebAuthn validators for passkey authentication, and JWT-based social recovery mechanisms. We performed a manual review of the contracts and investigated the following:
    - We reviewed the smart wallet factory to assess whether account collisions can be forced.
    - We reviewed access controls to investigate if they can be bypassed, which led to the discovery of TOB-GSW-1.
    - We reviewed the wallet integration with Nexus for general correctness.
    - We reviewed the version management for general correctness.
    - We reviewed the JWT Recovery module installation and interactions with the wallet for general correctness.
- **Gemini Wallet SDK.** The Gemini Wallet SDK is a TypeScript library that provides EIP-1193 provider functionality for dapps to integrate with Gemini's smart account wallet. It implements ERC-4337 account abstraction with WebAuthn-based authentication (through EIP-7579). The SDK handles cross-origin communication with the popup-based SDK server, manages smart account creation and signing, and provides a standard Ethereum provider interface. We performed a manual review of this logic and investigated the following:
    - We reviewed the implementation of the Gemini custom smart account to assess whether it correctly allows for the submission of user operations on-chain, always generates unique nonces, and correctly signs raw and typed messages. Additionally, we reviewed the implementation of the various signers (e.g., K-1 and WebAuthn) for general correctness.
    - We reviewed the provider and wallet implementation to determine if it is EIP-1193 compliant. We also reviewed the wagmi connector integration for general correctness.
    - We reviewed the use of storage API to assess whether all operations correctly use this API. This led to the discovery of TOB-GSW-8, which highlights that the

local storage is directly accessed (not through the API), which prevents module installation.

  - We reviewed the integration of the WebAuthn protocol for user registration, authentication, and transaction signing for general correctness.

- **Gemini SDK server.** The SDK server is a Next.js web application that runs in a popup window and provides the user interface for wallet operations. It handles user authentication through WebAuthn passkeys, displays transaction approval screens, and manages the onboarding flow for new users. All wallet operations requiring user consent (signing, transactions, chain switching) are processed through this popup-based interface. We performed a manual review of this logic and investigated the following:
  - We reviewed the UI components to understand and validate the various user operations. This led to the discovery of a variety of issues (TOB-GSW-5 and TOB-GSW-6) that showcase limitations in the validation of what a user signs and what the user sees when they are approving an operation.
  - We reviewed the communication between the SDK server and the SDK wrapper. This led to the discovery of TOB-GSW-4, which highlights that API keys are leaked on the client-side, and TOB-GSW-12, which highlights a misuse of the `window.postMessage` API function.
  - We reviewed the integration with WalletConnect and Blockaid for general correctness.

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 following packages were considered out of scope:**
  - `apps/demo-dapp`
  - `apps/demo-mobile`
  - `apps/onchain-dapp`
  - `packages/shared-ui`
  - `packages/subgraph`
  - `packages/zkVM`

- **Blockaid integration:** We briefly reviewed the Blockaid integration for general correctness but, due to time constraints, did not review it in depth for parsing return data and error handling.
- **Public key recovery:** We briefly reviewed the public key recovery logic for general correctness but, due to time constraints, did not review it in depth for specialized cryptographic considerations.
- **Currency conversions:** During the audit, we noticed that the gas fees are not correctly denominated in the selected currency. Since this is not a security risk and more of a user interface limitation, we are noting it here.
- **Mobile wallet:** We did not evaluate the wallet service under the threat model of a mobile device. We performed manual review and user interface testing within the context of connecting to the wallet in a web browser.
- **Dynamic testing of WalletConnect integration:** We were unable to dynamically test the WalletConnect integration through the UI since the provided QR code was not readable.
- **JWT signature verification:** We reviewed the `JWTRecovery` contract, but since the zkVM package was considered out of scope, we were unable to validate the signature validation in the contract.

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 system under review did not include any complex arithmetic operations that could lead to precision loss, underflows, or division by zero.	**Not Applicable**
Auditing	The codebase does not have any audit logging and monitoring capabilities. Currently, the system is unable to effectively respond to any bugs or issues that may arise in production.	**Weak**
Authentication / Access Controls	A minor issue in the smart contract logic allows an attacker to manipulate a user's nonce (TOB-GSW-1). The WebAuthn protocol was correctly implemented to support user authentication and transaction signing.	**Satisfactory**
Complexity Management	The system exhibits reasonable architectural separation between the smart contract logic, SDK wrapper, and the SDK service. However, the exposure of API keys (TOB-GSW-4) highlights that certain components may have to be redesigned. Additionally, the system is configured to work with only a single data provider for reading on-chain data and sending user operations. The lack of redundant services creates single points of failure (TOB-GSW-13). More generally, the code is incomplete with a large number of TODOs that would implement important security features (e.g., a passkey service or improved error handling).	**Moderate**
Cryptography and Key Management	The wallet itself does not manage any private keys and uses the WebAuthn protocol for user authentication and transaction signing.	**Further Investigation Required**

	As highlighted in the project coverage section, we did not fully assess the public key recovery logic, and the smart contract zkVM recovery module was considered out of scope.	
Documentation	The provided documentation is insufficient. The system has many complex codepaths that navigate back and forth between the smart contracts, SDK wrapper, and SDK server. Tracing these codepaths is time consuming and reduces the efficiency of code reviews. Having high-level documentation on the system layout and diagrams that highlight the most critical workflows (e.g., sending a transaction) is important. The inline documentation is also sparse and does not sufficiently highlight how, why, and when various functions should be used.	**Weak**
Low-Level Manipulation	There was no low-level assembly usage in the smart contract scope.	**Not Applicable**
Testing and Verification	The assessment revealed multiple basic implementation errors that should have been caught by comprehensive testing, including configuration parsing failures (TOB-GSW-8) and missing access controls (TOB-GSW-1). Additionally, there is a large gap in user acceptance and interface testing. We identified a number of security and usability issues when interacting with the wallet's UI (TOB-GSW-3, TOB-GSW-5, TOB-GSW-6).	**Moderate**
Transaction Ordering	While we identified no direct transaction-ordering vulnerabilities, the nonce manipulation issue (TOB-GSW-1) could potentially be exploited to affect transaction sequencing and execution order.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Missing access control allows nonce manipulation	Access Controls	Low
3	Missing notification about chain switching capability during connection	Data Validation	Low
4	API keys exposure in client-side code enables API abuse	Data Exposure	High
5	Wallet interface fails to display critical transaction parameters for user verification	Data Validation	High
6	Wallet displays non-printable characters as whitespace in signature requests	Data Validation	Low
8	Module installation fails due to incorrect local storage parsing	Configuration	Low
9	Wallet can be embedded in iframe, enabling clickjacking attacks	Access Controls	High
12	Malicious listener can intercept messages	Data Validation	Informational
13	Lack of redundant third-party services	Configuration	Informational

Detailed Findings

1. Missing access control allows nonce manipulation	
Severity: **Low**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-GSW-1
Target: `packages/contract/src/recovery/JWTRecovery.sol`	

Description

The `validateUserOp` function in the `JWTRecovery` module lacks access control, allowing any caller to validate a user's recovery operation signature and increment their nonce.

When a malicious actor calls the `validateUserOp` function of the JWTRecovery module before a legitimate user, it increments the account's nonce, causing the user's subsequent transaction to fail due to a nonce mismatch. The impact is limited to nonce manipulation and does not allow unauthorized access to account funds or other critical operations.

```
nonce[account] = currentNonce + 1;
```

Figure 1.1: The nonce is incremented in the `validateUserOp` function.

Exploit Scenario

A malicious user monitors the mempool for recovery transactions from a target account. Before the legitimate recovery transaction is processed, the attacker calls `validateUserOp` directly with the same parameters. This increments the target account's nonce, causing the legitimate user's recovery transaction to fail when it is processed due to the nonce mismatch. The user must then create a new recovery transaction with the updated nonce.

Recommendations

Short term, add an access control check requiring `msg.sender == userOp.sender` to ensure that only the account owner can validate their own recovery operations.

Long term, implement comprehensive access control testing in the test suite to verify that all validation functions properly restrict caller permissions and test edge cases where unauthorized callers attempt to manipulate the account state.

3. Missing notification about chain switching capability during connection

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-GSW-3
Target: `apps/sdk-server/app/(pages)/connect/page.tsx`	

Description
The Gemini Wallet interface does not inform users that connected dapps can switch chains when establishing a connection. Unlike other wallets, which explicitly warns users about chain switching capabilities during the connection process, Gemini's connection flow omits this critical information. This lack of transparency could lead to unexpected chain switches that users may not be aware they authorized.

```
<div className="flex gap-8 items-center justify-start">
  <CircleCheckIcon size="lg" />
  <p className="text-sm">Allows app to see your balances and activity</p>
</div>

<div className="flex gap-8 items-center justify-start">
  <CircleCheckIcon size="lg" />
  <p className="text-sm">Allows app to request approval for transactions</p>
</div>

<div className="flex gap-8 items-center justify-start">
  <CircleInfoIcon size="lg" />
  <p className="text-sm">Funds can't leave wallet without your approval</p>
</div>
```

Figure 3.1: Warning about what the user accepts when connecting their wallet

Exploit Scenario
A user connects their Gemini wallet to a dapp. The dapp, having received permission to switch chains, automatically changes the user's network to a different chain without any explicit warning or confirmation from the user. The user may not realize this has happened until they notice their transactions are being processed on an unexpected network, potentially leading to confusion or unintended interactions.

Recommendations
Short term, implement a clear warning message during the wallet connection process that explicitly informs users about the dapp's ability to switch chains, similar to MetaMask's implementation.

Long term, establish a comprehensive UI/UX review process to ensure that all critical security and functionality information is properly communicated to users during wallet interactions.

4. API keys exposure in client-side code enables API abuse	
Severity: **High**	Difficulty: **Low**
Type: Data Exposure	Finding ID: TOB-GSW-4
Target: `apps/sdk-server/app/config/smartAccountConfig.ts`	

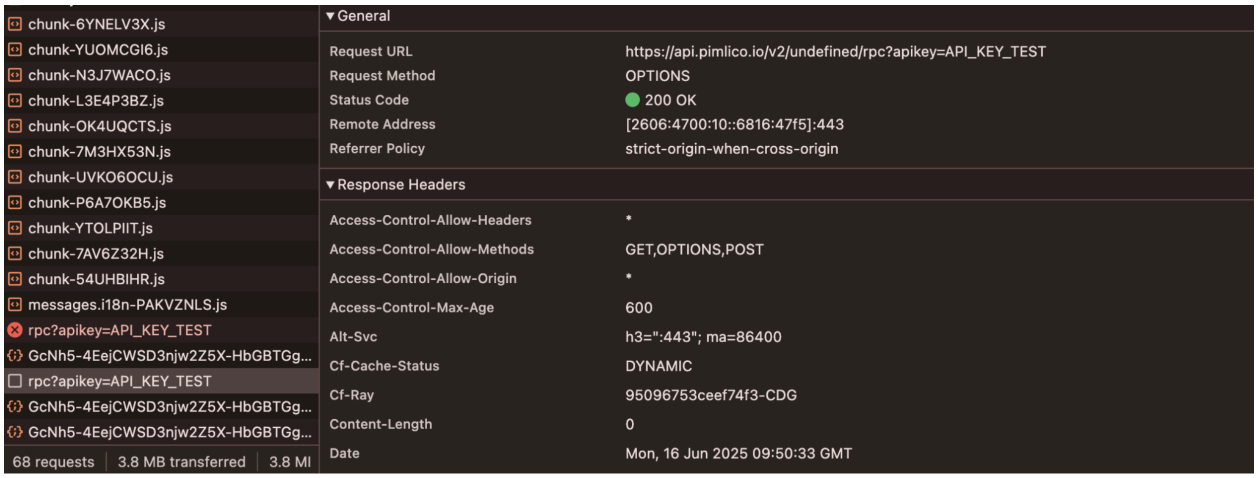
Description

The API keys are exposed in client-side code, allowing malicious users to extract and abuse them.

The API keys are used to create a Pimlico client for bundling transactions and to get on-chain information using Alchemy, for example. But to do this, the API key is currently being sent to the client browser, where it can be accessed through browser developer tools. This enables unauthorized users to make API calls using Gemini's API key, potentially exhausting the API quota or incurring unexpected costs.

```
// Create Pimlico configuration for bundling transactions
const pimlicoUrl =
`https://api.pimlico.io/v2/${chainId}/rpc?apikey=${process.env.NEXT_PUBLIC_PIMLICO_A
PI_KEY}`;
const pimlicoTransport = http(pimlicoUrl);
```

Figure 4.1: The Pimlico URL containing the API key is shared with the client.



```
chunk-6YNELV3X.js
chunk-YUOMCGI6.js
chunk-N3J7WACO.js
chunk-L3E4P3BZ.js
chunk-OK4UQCTS.js
chunk-7M3HX53N.js
chunk-UVKO6OCU.js
chunk-P6A7OKB5.js
chunk-YTOLPIIT.js
chunk-7AV6Z32H.js
chunk-54UHBIHR.js
messages.i18n-PAKVZNLS.js
rpc?apikey=API_KEY_TEST
GcNh5-4EejCWSD3njw2Z5X-HbGBTGg...
rpc?apikey=API_KEY_TEST
GcNh5-4EejCWSD3njw2Z5X-HbGBTGg...
GcNh5-4EejCWSD3njw2Z5X-HbGBTGg...
68 requests | 3.8 MB transferred | 3.8 MI
```

```
▾General
Request URL                        https://api.pimlico.io/v2/undefined/rpc?apikey=API_KEY_TEST
Request Method                     OPTIONS
Status Code                        ● 200 OK
Remote Address                     [2606:4700:10::6816:47f5]:443
Referrer Policy                    strict-origin-when-cross-origin
▾Response Headers
Access-Control-Allow-Headers       *
Access-Control-Allow-Methods       GET,OPTIONS,POST
Access-Control-Allow-Origin        *
Access-Control-Max-Age             600
Alt-Svc                            h3=":443"; ma=86400
Cf-Cache-Status                    DYNAMIC
Cf-Ray                             95096753ceef74f3-CDG
Content-Length                     0
Date                               Mon, 16 Jun 2025 09:50:33 GMT
```

Figure 4.2: Chrome developer tool on the client side

Exploit Scenario

A malicious user connects to the Gemini wallet and inspects the network requests or browser developer tools. They extract the Pimlico API key from the client-side code. Using this key, they can make direct API calls to Pimlico's services, potentially exhausting the API quota or incurring unexpected costs for the Gemini wallet service.

Recommendations

Short term, the following security measures should be implemented:

- Keys should have minimal privileges, if possible
- Use different keys for debug and prod builds
- Rotate keys
- Enable rate-limiting or monitoring

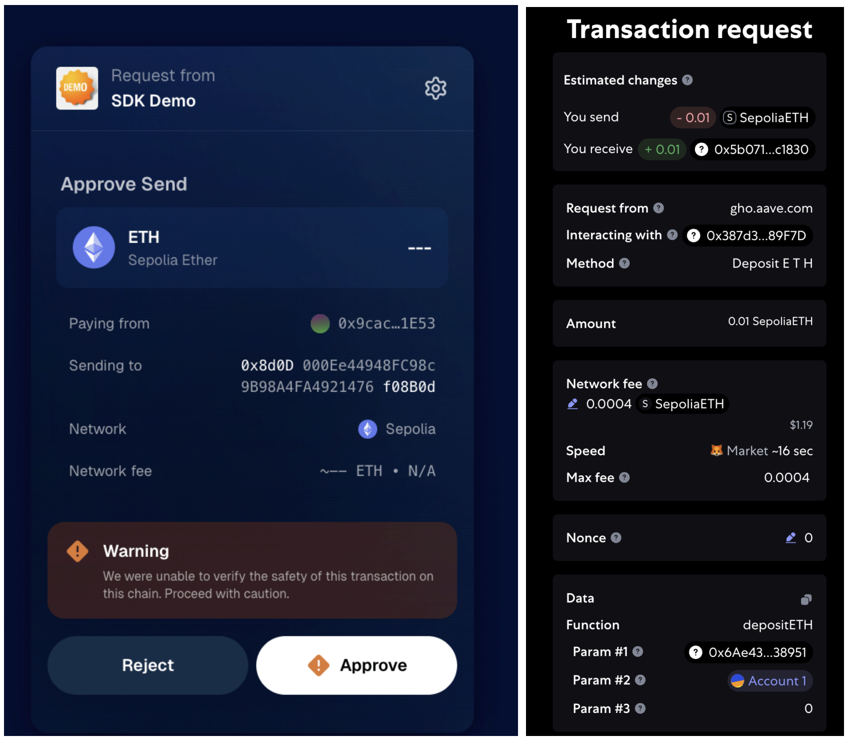
Long term, implement a proper API gateway pattern where all third-party API calls are proxied through the SDK server, with appropriate rate limiting and monitoring to prevent abuse. Note that this recommendation has the side effect of slowing down the connection.

5. Wallet interface fails to display critical transaction parameters for user verification

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-GSW-5
Target: Wallet interface/transaction approval screen	

Description

The wallet interface does not display critical transaction parameters, including `data`, `gas limit`, `maxFeePerGas`, `maxPriorityFeePerGas`, and nonce, preventing users from manually verifying complete transaction details before signing. Users can see only basic information like recipient address and amount, but lack visibility into essential transaction mechanics that could indicate malicious behavior or allow for proper transaction verification. This represents a significant security gap compared to established wallets like MetaMask, which display all these parameters for user review.



Request from
SDK Demo

Approve Send

ETH
Sepolia Ether ---

Paying from 0x9cac…1E53
Sending to 0x8d0D 000Ee44948FC98c 9B98A4FA4921476 f08B0d
Network Sepolia
Network fee ~-- ETH • N/A

Warning
We were unable to verify the safety of this transaction on this chain. Proceed with caution.

Reject | Approve

Transaction request

Estimated changes
You send - 0.01 SepoliaETH
You receive + 0.01 0x5b071...c1830

Request from gho.aave.com
Interacting with 0x387d3...89F7D
Method Deposit E T H

Amount 0.01 SepoliaETH

Network fee
0.0004 SepoliaETH
$1.19
Speed Market ~16 sec
Max fee 0.0004

Nonce 0

Data
Function depositETH
Param #1 0x6Ae43...38951
Param #2 Account 1
Param #3 0

Figure 5.1: Gemini wallet validation window (left) compared to Metamask (right)

Exploit Scenario

A malicious dapp presents what appears to be a simple ETH transfer to the user, showing only the recipient address and amount in the wallet interface. However, the transaction includes a hidden data payload that contains smart contract function calls. Since the wallet does not display the transaction data field, the user believes they are making a simple ETH transfer and approves the transaction. The hidden data payload executes additional malicious actions beyond the apparent ETH transfer, potentially compromising the user's entire wallet.

Recommendations

Short term, implement a comprehensive transaction parameter display in the wallet approval interface to show `data`, `gas limit`, `maxFeePerGas`, `maxPriorityFeePerGas`, and `nonce` fields matching industry standards.

Long term, add transaction parameter validation and warnings for unusual values (such as extremely high gas limits or fees) to help users identify potentially malicious transactions before signing.

6. Wallet displays non-printable characters as whitespace in signature requests	
Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-GSW-6
Target: Wallet signature interface/message display component	

Description

The wallet displays messages containing non-printable or binary characters as whitespace in signature request dialogs, masking the true content from users. When a dapp requests message signing through the JavaScript Ethereum API, non-printable characters in the message are rendered as blank spaces rather than being properly encoded or rejected. This prevents users from accurately verifying the content they are signing and creates opportunities for deceptive signature requests.

Exploit Scenario

A malicious dapp crafts a signature request containing a message with embedded non-printable characters that appear as whitespace to the user. The displayed message looks benign or incomplete. However, the actual message contains hidden binary data specifying malicious actions or different parameters than what the user perceives. The user, seeing what appears to be a standard or incomplete message, signs the request without realizing they are authorizing something entirely different from what is displayed.

Recommendations

Short term, implement detection of non-printable characters and also display messages containing such characters in hex encoding rather than as UTF-8 text.

Long term, establish consistent message handling standards that always display both plaintext and hex representations of signature requests, and implement warnings for messages containing non-printable characters to alert users of potentially deceptive content.

8. Module installation fails due to incorrect local storage parsing

Severity: **Low**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-GSW-8
Target: `apps/sdk-server/app/(pages)/settings/_views/advanced.view.tsx`	

Description

The module installation process fails because the chain ID is incorrectly parsed from local storage.

The code attempts to retrieve and parse the chain ID using the code shown in figure 8.1.

```
const currentChain = JSON.parse(localStorage.getItem(STORAGE_ETH_ACTIVE_CHAIN_KEY)
?? "{}")
```

Figure 8.1: The chain ID is parsed using this method.

However, this results in an empty object. The storage value is in a format shown in figure 8.2.

```
{
    "@gemini.wallet.settings": "{\n  \"fiatDisplayCurrency\": \"USD\"\n}",
    "@gemini.wallet.eth-active-chain": "{'id': '[CHAIN_ID]'}",
    "@gemini.wallet.passkey-credential": "[...]",
    "@gemini.wallet.smart-account": "[...]"
}
```

Figure 8.2: Storage value JSON

As a result, subsequent operations receive an undefined chain ID, causing downstream failures such as invalid API requests.

Exploit Scenario

Alice, a user of the Gemini Smart Wallet, attempts to download a module to use with her wallet. However, the module installation fails since the API request is malformed.

Recommendations

Short term, use the `loadObject` method to retrieve the chain ID from storage.

Long term, add unit tests to ensure that every feature works correctly.

9. Wallet can be embedded in iframe, enabling clickjacking attacks

Severity: **High**	Difficulty: **Low**
Type: Access Controls	Finding ID: TOB-GSW-9
Target: `packages/gemini-wallet-sdk/src/utils/popup.ts`	

Description
The wallet SDK allows the wallet UI to be embedded in an iframe instead of opening in a popup window.

This creates a clickjacking vulnerability where a malicious dapp can overlay the wallet interface with hidden elements or misleading UI, tricking users into signing transactions or messages without their knowledge. Any dapp could make a user sign a malicious message, thinking he is signing a safe message.

Exploit Scenario
A malicious dapp embeds the wallet in an iframe and overlays it with transparent elements or misleading UI. The user sees what appears to be a legitimate wallet interface, but is actually interacting with hidden elements that trigger unwanted actions, such as signing a transaction with different parameters than what is displayed.

Figure 9.1 provides a proof of concept showing the possibility of opening the wallet in an iframe.

```
import { rpcErrors } from "@metamask/rpc-errors";

const POPUP_WIDTH = 420;
const POPUP_HEIGHT = 650;

export const openPopup = (url: URL): Window => {
  const left = (window.innerWidth - POPUP_WIDTH) / 2 + window.screenX;
  const top = (window.innerHeight - POPUP_HEIGHT) / 2 + window.screenY;

  const popupId = `wallet_${window?.crypto?.randomUUID()}`;

  // Try to find an existing iframe with this name/id
  let iframe = document.querySelector(`iframe[name='${popupId}']`) as
HTMLIFrameElement | null;
  if (!iframe) {
    iframe = document.createElement('iframe');
    iframe.id = popupId;
    iframe.name = popupId;
    iframe.allow = "publickey-credentials-get *";
    iframe.style.width = `${POPUP_WIDTH}px`;
    iframe.style.height = `${POPUP_HEIGHT}px`;
    iframe.style.position = 'fixed';
    iframe.style.right = '20px';
    iframe.style.bottom = '20px';
    iframe.style.zIndex = '9999';
    iframe.style.border = '1px solid #ccc';
    document.body.appendChild(iframe);
  }
  iframe.style.display = 'block';

  // Use window.open with the popupId as the target to load the URL in the iframe
  const popup = window.open(url.toString(), popupId);

  if (!iframe.contentWindow || !popup) {
    throw rpcErrors.internal('Pop up window failed to open');
  }

  // Focus is not relevant for iframes, but you can scroll to it if needed
  // iframe.contentWindow?.focus();

  return iframe.contentWindow as Window;
};

export const closePopup = (popup: Window | null) => {
  if (popup && !popup.closed) {
    popup.close();
  }
};
```

Figure 9.1: Modified `popup.ts` file that creates an iframe instead of a popup window

Recommendations

Short term, implement `Content-Security-Policy` headers with the `frame-ancestors` directive set to none and `X-Frame-Options` headers set to DENY to prevent the wallet from being embedded in iframes.

Long term, ensure that the wallet can be opened only in a popup window or new tab, and implement additional security measures to prevent any form of embedding.

12. Malicious listener can intercept messages

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-GSW-12
Target: `apps/sdk-server/app/contexts/smartAccount/smartAccount.context.tsx`	

Description

A malicious listener on the user's device can intercept messages being sent from the SDK server to the dapp.

The system uses the `window.postMessage` function to communicate between the dapp and the SDK server (figure 12.1). When using the `postMessage` function, a target origin can be specified to ensure that only the window with the given origin can receive the event.

```
const postMessageToParent = <M extends GeminiSdkMessageResponse>({
  message,
  closeWindow = true,
  isWalletConnect = false,
}: {
  message: M;
  closeWindow?: boolean;
  isWalletConnect?: boolean;
}) => {
  if (window.opener) {
    // wallet loaded as iframe, post message and close
    window.opener.postMessage(
      {
        data: message.data,
        event: message.event,
        requestId: message.requestId ?? window?.crypto?.randomUUID(),
      },
      "*",
    );
    if (closeWindow) window.close();
    return;
  }
  // wallet not loaded inside iframe (i.e. via WC or direct url access)
  // if wallet connect, route back to initial page
  if (isWalletConnect) {
    return router.push("/wc");
  }
  // if not wallet connect and it was a connect message, route to settings
  if (message.event === GeminiSdkEvent.SDK_CONNECT) {
    return router.push("/settings");
  }
};
```

Figure 12.1: A target origin is not specified when using `window.postMessage`.

However, as highlighted in figure 12.1, a wildcard (`*`) is used as the target origin, meaning that any listener can subscribe and intercept the events emitted by the SDK server. This would allow a malicious listener to intercept potentially sensitive data on a user's device.

Recommendations

Short term, use the stored app metadata to specify the target origin of the message.

Long term, review the security guidelines of the Web APIs used to ensure that all critical functions are used correctly.

13. Lack of redundant third-party services	
Severity: **Informational**	Difficulty: **Low**
Type: Configuration	Finding ID: TOB-GSW-13
Target: `packages/gemini-wallet-sdk/src/smartAccount/utils/signers/webAuthnSigner.ts`	

Description

The system is not configured to have redundant third-party services to use in case a service such as Alchemy or Pimlico is compromised or has downtime.

The Gemini Wallet uses Alchemy to read on-chain data and uses Pimlico to submit user operations. If an attacker exploits the vulnerability highlighted in TOB-GSW-4, the wallet may be unable to service user requests.

In this scenario, or in the scenario where Alchemy or Pimlico are compromised, it is beneficial to have backup or redundant services for reading or writing to the chain.

More generally, it is beneficial to use multiple data sources to reduce the dependency on a single service and to verify the returned values. For example, the system could query both Alchemy and Infura and cross-reference the responses.

Exploit Scenario 1

A malicious attacker captures the production-level API key for Pimlico (see TOB-GSW-4). They use this key to hit the rate-limit of the service. Alice, a user, is unable to submit her transaction since Pimlico is throttling incoming requests.

Exploit Scenario 2

A malicious attacker compromises Pimlico. Gemini is notified of this compromise and must shut down service of the wallet since there is no backup bundler service configured.

Recommendations

Long term, integrate redundant services for reading and writing to the chain. Additionally, it may be beneficial to deploy and maintain blockchain nodes to reduce the reliance on third-party services such as Alchemy or Infura.

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
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system’s resistance to transaction-ordering attacks

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 does not apply 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.

C. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From July 21 to July 22, 2025, Trail of Bits reviewed the fixes and mitigations implemented by the Gemini team for the issues identified in this report. We reviewed additional fixes on August 13. We reviewed each fix to determine its effectiveness in resolving the associated issue.

We reviewed a zip file provided directly by the Gemini team. The commit hash associated with the zip is `b0b39e2684cbc3454c520f79821735a2887bf8fb`.

In summary, Gemini has resolved all of the nine issues described in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Severity	Status
1	Missing access control allows nonce manipulation	Low	Resolved
3	Missing notification about chain switching capability during connection	Low	Resolved
4	API keys exposure in client-side code enables API abuse	High	Resolved
5	Wallet interface fails to display critical transaction parameters for user verification	High	Resolved
6	Wallet displays non-printable characters as whitespace in signature requests	Low	Resolved
8	Module installation fails due to incorrect local storage parsing	High	Resolved
9	Wallet can be embedded in iframe, enabling clickjacking attacks	High	Resolved
12	Malicious listener can intercept messages	Informational	Resolved

13	Lack of redundant third-party services	Informational	Resolved

Detailed Fix Review Results

TOB-GSW-1: Missing access control allows nonce manipulation
Resolved. A validation on the `msg.sender` has been added at the beginning of the `validateUserOp` function.

TOB-GSW-3: Missing notification about chain switching capability during connection
Resolved. A message has been added on the wallet connection screen to warn the user about the chain switching capability.

TOB-GSW-4: API keys exposure in client-side code enables API abuse
Resolved. The Gemini team has added a proxy layer that prevents the exposure of any production keys.

TOB-GSW-5: Wallet interface fails to display critical transaction parameters for user verification
Resolved. The send transaction page now has a “Show Details” panel that allows a user to see all fields of the transaction they are sending.

TOB-GSW-6: Wallet displays non-printable characters as whitespace in signature requests
Resolved. An option has been added to show the hexadecimal data directly.

TOB-GSW-8: Module installation fails due to incorrect local storage parsing
Resolved. The chain ID is now correctly retrieved from local storage, allowing for successful module installation.

TOB-GSW-9: Wallet can be embedded in iframe, enabling clickjacking attacks
Resolved. `Content-Security-Policy` headers have been implemented with the `frame-ancestors` directive set to `self`, and `X-Frame-Options` headers have been set to `SAMEORIGIN`.

TOB-GSW-12: Malicious listener can intercept messages
Resolved. The `postMessage` implementation now uses the app metadata stored in `appContext?.origin` to specify the target origin instead of always using a `*`.

TOB-GSW-13: Lack of redundant third-party services
Resolved. The Gemini team has added redundant third-party bundler services and has added a fallback mechanism in case the default bundler’s configuration fails.

D. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.

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 assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up to date with our latest news and announcements, please follow @trailofbits on X or LinkedIn, 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

© 2025 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

Trail of Bits considers this report public information; it is licensed to Gemini under the terms of the project statement of work and has been made public at Gemini's request. Material within this report may not be reproduced or distributed in part or in whole without Trail of Bits' express written permission.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through sources other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information 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 software controls and security properties. 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. A project's time and resource constraints also limit their use.