# Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Daimo

Veridise Inc.
November 7, 2023

► **Prepared For:**

Daimo
`https://daimo.xyz`

► **Prepared By:**

Jacob Van Geffen
Bryan Tan

► **Contact Us:** `contact@veridise.com`

► **Version History:**

Nov. 07, 2023 V1

# Contents



 

# 1 Executive Summary

From Nov. 1, 2023 to Nov. 3, 2023, Daimo engaged Veridise to review the security of an update to their Daimo project. Compared to the previous version, which Veridise has audited previously in Sep. 2023*, the new version has been modified to support passkey authentication. Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `0d8ff1c` of `daimo` and `f13149e` of `p256-verifier`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

**Project Summary.** The security assessment covered changes made to the Daimo smart contracts that are needed to support passkeys, along with accompanying changes to the Daimo developers' `p256-verifier` library. Specifically, the signature format has been modified so that it is now embedded within a WebAuthn authenticator assertion (as its challenge field), such that the entire assertion must be signed with a public key registered to the account. To accommodate this change, the `p256-verifier` library has been extended with two new files `WebAuthn.sol` and `Base64.sol` which implement the "Verifying an Authentication Assertion" procedure described in Section 7.2 of the Web Authentication Level 2 Specification.

**Code assessment.** The Daimo developers provided the updated source code of Daimo for review†. The updates include new functionalities but otherwise do not introduce any significant changes to the overall structure of the project. Additional test cases were added for the new functionalities, providing test coverage for both successful and unsuccessful usage scenarios.

**Summary of issues detected.** The audit uncovered 3 issues, of which 1 is assessed to be a warning and 2 are assessed to be informational findings by the Veridise auditors. Specifically, there is a array length check that is inconsistent with the WebAuthn specification but does not appear to have security impact (V-DWA-VUL-001) , and there are two code locations that can be clarified with comments (V-DWA-VUL-002, V-DWA-VUL-003).

**Disclaimer.** We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

* The previous audit report can be found on Veridise's website at `https://veridise.com/audits/`
† The source code is publicly available at `https://github.com/daimo-eth/daimo`.

# Project Dashboard 2

**Table 2.1:** Application Summary.

| Name | Version | Type | Platform |
|---|---|---|---|
| Daimo | `0d8ff1c` | Solidity | Ethereum |
| `p256-verifier` | `f13149e` | Solidity | Ethereum |

**Table 2.2:** Engagement Summary.

| Dates | Method | Consultants Engaged | Level of Effort |
|---|---|---|---|
| Nov. 1 - Nov. 3, 2023 | Manual & Tools | 2 | 6 person-days |

**Table 2.3:** Vulnerability Summary.

| Name | Number | Resolved |
|---|---|---|
| Critical-Severity Issues | 0 | 0 |
| High-Severity Issues | 0 | 0 |
| Medium-Severity Issues | 0 | 0 |
| Low-Severity Issues | 0 | 0 |
| Warning-Severity Issues | 1 | 1 |
| Informational-Severity Issues | 2 | 2 |
| TOTAL | 3 | 3 |

**Table 2.4:** Category Breakdown.

| Name | Number |
|---|---|
| Maintainability | 2 |
| Data Validation | 1 |

# Audit Goals and Scope 3

## 3.1 Audit Goals

The engagement was scoped to provide a security assessment of Daimo's smart contracts. In our audit, we sought to answer questions such as:

- Does Daimo's WebAuthn authentication assertion verification implementation follow the recommended procedure in the WebAuthn specification?
- Does the updated DaimoAccount signature validation code sufficiently guard against replay attacks?
- Is the base64url encoding implementation correct?

## 3.2 Audit Methodology & Scope

**Audit Methodology.** To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.
- *Fuzzing/Property-based Testing.* We also leverage fuzz testing to determine if the protocol may deviate from the expected behavior. To do this, we formalize the desired behavior of the protocol as [V] specifications and then use our fuzzing framework OrCa to determine if a violation of the specification can be found.

*Scope*. The scope of this audit is limited to the following files of the source code provided by the Daimo developers:

- `DaimoAccount.sol` from `daimo`
- `DaimoAccountFactory.sol` from `daimo`
- `WebAuthn.sol` from `p256-verifier`
- `Base64URL.sol` from `p256-verifier`

Other files within the `daimo` and `p256-verifier` repositories, as well as third-party dependencies such as OpenZeppelin, are not in the scope of this audit. During the audit, the Veridise auditors referred to the excluded files but assumed that they have been implemented correctly.

*Methodology*. Veridise auditors reviewed the report of the previous audit for Daimo, inspected the provided tests, and read the Daimo documentation. They then began a manual audit of the code assisted by both static analyzers and automated testing. During the audit, the Veridise auditors regularly met with the Daimo developers to ask questions about the code.

## 3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

**Table 3.1:** Severity Breakdown.

| | Somewhat Bad | Bad | Very Bad | Protocol Breaking |
|---|---|---|---|---|
| Not Likely | Info | Warning | Low | Medium |
| Likely | Warning | Low | Medium | High |
| Very Likely | Low | Medium | High | Critical |

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

**Table 3.2:** Likelihood Breakdown

| | |
|---|---|
| Not Likely | A small set of users must make a specific mistake |
| Likely | Requires a complex series of steps by almost any user(s)<br>- OR -<br>Requires a small set of users to perform an action |
| Very Likely | Can be easily performed by almost anyone |

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

**Table 3.3:** Impact Breakdown

| | |
|---|---|
| Somewhat Bad | Inconveniences a small number of users and can be fixed by the user |
| Bad | Affects a large number of people and can be fixed by the user<br>- OR -<br>Affects a very small number of people and requires aid to fix |
| Very Bad | Affects a large number of people and requires aid to fix<br>- OR -<br>Disrupts the intended behavior of the protocol for a small group of users through no fault of their own |
| Protocol Breaking | Disrupts the intended behavior of the protocol for a large group of users through no fault of their own |

# Vulnerability Report 4

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

**Table 4.1:** Summary of Discovered Vulnerabilities.

| ID | Description | Severity | Status |
|---|---|---|---|
| V-DWA-VUL-001 | Authenticator length check inconsistent with ac. . . | Warning | Fixed |
| V-DWA-VUL-002 | Consider noting the draft version of WebAuthn i. . | Info | Fixed |
| V-DWA-VUL-003 | Inconsistent doc comment about signature format | Info | Acknowledged |

 

## 4.1 Detailed Description of Issues

### 4.1.1 V-DWA-VUL-001: Authenticator length check inconsistent with actual length

| | | | |
|---|---|---|---|
| **Severity** | Warning | **Commit** | f13149e |
| **Type** | Data Validation | **Status** | Fixed |
| **File(s)** | `WebAuthn.sol` | | |
| **Location(s)** | verifySignature() | | |
| **Confirmed Fix At** | aa1ce73 | | |

The `verifySignature()` implements the "Verifying an Authentication Assertion" procedure described in WebAuthn Level 2 Specification. To check whether the `authenticatorData` is well-formed, the `verifySignature()` function will return `false` if the `authenticatorData` is less than 32 bytes long. However, the WebAuth Level 2 Specification states that the `authenticatorData` is at least 37 bytes long:

> The authenticator data structure is a byte array of 37 bytes or more, laid out as shown [. . . ]

```
// Check that authenticatorData has good flags
if (
    authenticatorData.length < 32 ||
    !checkAuthFlags(authenticatorData[32], requireUserVerification)
) {
    return false;
}
```

**Snippet 4.1:** Relevant lines in `verifySignature()`

**Impact** The check that the length is at least 32 bytes long is inconsistent with the WebAuthn specification, but there should be no security impact. The first 32 bytes consist of the RP ID, the 33rd byte is the authenticator flags, and the 34th-37th bytes are the signature counter. If the authenticator data is exactly 32 bytes long, then the index into `authenticatorData[32]` will be out-of-bounds and correctly trigger a revert. Furthermore, the signature counter is irrelevant to the intended behavior of the Daimo application and can be safely ignored.

**Recommendation** To be more consistent with the WebAuthn specification, change the check to `authenticatorData.length < 37` to ensure that the `authenticatorData` is the correct length.

#### 4.1.2 V-DWA-VUL-002: Consider noting the draft version of WebAuthn in a comment

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | f13149e |
| **Type** | Maintainability | **Status** | Fixed |
| **File(s)** | `WebAuthn.sol` | | |
| **Location(s)** | See description | | |
| **Confirmed Fix At** | 671cc1b | | |

The current implementation of the `WebAuthn` library appears to be based on the WebAuthn Level 2 specification published on April 8, 2021. This may be confusing to readers of the code because there are multiple versions of WebAuthn, such as the upcoming editor's draft of the level 3 specification. We recommend inserting a documentation comment that indicates which version of the WebAuthn specification that is being used.

 

### 4.1.3 V-DWA-VUL-003: Inconsistent doc comment about signature format

| | | | |
|---|---|---|---|
| **Severity** | Info | **Commit** | 0d8ff1c |
| **Type** | Maintainability | **Status** | Acknowledged |
| **File(s)** | `DaimoAccount.sol` | | |
| **Location(s)** | _verifySignature() | | |
| **Confirmed Fix At** | | | |

The signature format used by the `DaimoAccount` is described in a documentation comment on the `_validateSignature()` method:

```
// Signature structure: [uint8 keySlot, uint8 signatureType, bytes signature]
// - keySlot: 0-255
// - signature: abi.encode form of Signature struct
```

However, the auditors were unable to find any code that decodes or checks this `signatureType`. Based on the code in `DaimoVerifier.verifySignature()`, it appears that the second byte of the signature is the start of the `Signature` struct.

**Recommendation** Clarify this inconsistency in the signature format and make the code consistent with the intended behavior.

**Developer Response** The developers confirmed that the comment is outdated and needs to be updated; the intended behavior is that the `signatureType` field is not part of the signature format. They further noted:

> **We will update the comment in our next DaimoAccount upgrade.**
>
> Changes (including comments) in DaimoAccount require that all existing users upgrade the implementation used by their proxy account contracts since it would change the CREATE2 address (Solidity compiler hashes in the comments to obtain the bytecode).
>
> Since there are no functional changes at the moment, we will batch this change with our next upgrade in future.