RISK AUDIT

for

sumvin

on

Feb 10, 2026

FIDESIUM

Executive Summary

Report

11
/100

TOTAL
Low risk
Feb 10, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on Sumvin **contracts**

Contract Link: N/A - Zip File provided

Issue Summary

Critical	High	Medium	Low	Info
0 Issues	1 Issues	2 Issues	2 Issues	4 Issues

Caveats

Sumvin's codebase is generally well written, clean, and follows good standard architectural pattenrs, but does incur a handful of flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vulnerability	Description	Risk	Probability	Status
Missing Deployment Chain Guard	Deployment Scripts do not assert chain-id	High	Info	Active
Centralization	MINTER_ROLE has significant modification rights	Medium	Low	Active
Front runnable initialization	`initialize` can be frontrun	Medium	Low	Active
Souldbound approvals allowed	Contract allows soulbound approvals	Low	Low	Active
Insufficient metadata validation	Metadata validation is insufficient	Low	Low	Active
ERC-7201 noncompliance	Protocol claims to follow ERC-7201 but does not utilize correct storage slot	Info	Certain	Active
Gas Inefficiency: Repeated `string.concat` calls	`tokenUri` makes repeated string.concat calls	Info	Info	Active
Gas Inefficiency: Repeated storage reads in `tokenURI`	`tokenURI` repeatedly reads `keys[i]` and `dat[tokenId][keys[i]]` directly from storage inside the loop	Info	Info	Active
Gas Inefficiency: Duplicate byte conversions in `_setDat` and `removeDat`	`_setDat` and `removeDat` recompute `bytes(key)` and `keccak256` on the same input	Info	Info	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Smart Contract Risks

Risk summary: 14

The contracts are well written, and secure with only a few minor issues..

Vulnerabilities Critical

Current scan crits Clear

During this scan no critical security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities High

Missing Deployment Chain Guard

Vulnerability severity: **High**

Vulnerability probability: **Info**

Missing Deployment Chain Guard

Deployment Scripts do not assert chain-id

An accidental oversight could allow for deployment to a non intentionally supported chain.

Recommendations:

Assert chain-id in `deploy-production.ts`

Vulnerabilities Medium

Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Centralization

`MINTER_ROLE` has significant modification rights

A compromised `MINTER_ROLE` could alter identity data for all users.

While deployment scripts set this to a Safe, the security of this is ultimately dependent on the Safe's opsec (multi-sig, hardware-backed, threshhold >1)

Recommendations:

Ensure the Safe is a well managed, hardware-backed multi-sig with a sufficient threshhold

Require multiple `MINTER_ROLE` signatures for updates

Consider changing to an owner only update model, allowing token owners to manager their own `dat`

Consider making critical identity data immutable

Front runnable initialization

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Front runnable initialization

`initialize` can be frontrun

An attacker could monitor the mempool for a pending `initialize` call.

`_disableInitializers` protects against direct initialization, however does not protet from initialization on the proxy

Recommendations:

Restrict the initializer call

```
constructor() {
    _deployer = msg.sender;
    _disableInitializers();
}

//...

function initialize(string memory name, string memory symbol, address admin) public initializer {
    require(msg.sender == _deployer, "Only deployer can initialize");
    //...
}
```

Vulnerabilities Low

Souldbound approvals allowed

Vulnerability severity: **Low**

Vulnerability probability: **Low**

Contract allows soulbound approvals

Users can still call `approve()` and `setApprovalForAll()` on tokens that can never be transferred.

These functions are inherited via ERC-721, and will waste gas for users

Recommendations:

Overwrite these functions to revert with `Soulbound()`

```
function approve(address, uint256) public pure override {

    revert Soulbound();

}
function setApprovalForAll(address, bool) public pure override {

    revert Soulbound();

}
```

Insufficient metadata validation

Vulnerability severity: **Low**

Vulnerability probability: **Low**

Metadata validation is insufficient

`mint` does not validate `sub` and `schema`

This could lead to a number of downstream effects

- Off-chain consumers may receive unexpected or malformed values
- Invalid tokenUri formats (e.g. does not start with https://) could render metadata nonfunctional
- Maliciously injected tokenUri data could be used as part of a multistep attack
- tokenURI could become large, gas constrained, and potentially even revert

Since this is set by an authorized minter, this is largely an exercise in Defence in Depth

Recommendations:

Enforce Length Limits on `sub` and `schema`, and enforce any other further validations such as a valid URL format for schema, ideally limiting to a specific set of whitelisted hostnames.

Vulnerabilities Info

ERC-7201 noncompliance

Vulnerability severity: **Info**

Vulnerability probability: **Certain**

Protocol claims to follow ERC-7201 but does not utilize correct storage slot

ERC-7201 should follow the formulat `keccak256(abi.encode(uint256(keccak256("sumvin.identity.storage")) - 1)) & ~bytes32(uint256(0xff))`, however `SumvinProxy.sol` sets the storage slot to `0x6d616e1d27b9fc2e3c8e7d3c4f9a00e8b1d2c3f4a5b6c7d8e9f0a1b2c3d4e500`

This will limit the visibility for ERC-7201 aware tools/debuggers, and will require future upgrades to follow the incorrect path to avoid storage corruption

This could additionally reduce public trust due to incorrect documentation

Recommendations:

Use the correct ERC-7201 compliant slot `0xee7e2ca0980d73ff25b9303dd708960c6fde53107631639262d122fdbc686f00`

Include the compound value in the build scripts

Add a test to verify derivation

```
function testStorageLocation() public {
    bytes32 expected = keccak256(
        abi.encode(
            uint256(keccak256("sumvin.identity.storage")) - 1
        )
    ) & ~bytes32(uint256(0xff));

    assertEq(IDENTITY_STORAGE_LOCATION, expected);
}
```

Gas Inefficiency: Repeated `string.concat` calls

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`tokenUri` makes repeated `string.concat calls`

While bounded by `MAX_DAT_KEYS`, this ultimately interacts with the lack of limits on sub/schema (see rest of report). If values are very large, or if you want this function to be callable onchain, this could cause reverts

string.concat scales at O(n^2).

Recommendations:

Build JSON using either a bytes buffer and abi.encodePacked, or better yet, an assembly string builder with precomputed length

Vulnerabilities Info

Gas Inefficiency: Repeated storage reads in `tokenURI`

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`tokenURI` repeatedly reads `keys[i]` and `dat[tokenId][keys[i]]` directly from storage inside the loop

This creates repeated storage reads and escaping work per iteration.

Recommendations:

Consider caching values in memory (e.g., pre-escaped strings)

Gas Inefficiency: Duplicate byte conversions in `_setDat` and `removeDat`

Vulnerability severity: **Info**

Vulnerability probability: **Info**

`_setDat` and `removeDat` recompute `bytes(key)` and `keccak256` on the same input

This adds overhead and can be avoided by caching `bytes(key)` once per call.

Recommendations:

Cache `bytes(key)` and reuse its length and hash.

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.