VIDMA

FERRUM NETWORK

SMART CONTRACT AUDIT

Project: Ferrum Network
Date: September 15th, 2022

TABLE OF CONTENTS

Summary . . . 03

Scope of Work . . . 06

Workflow of the auditing process . . . 07

Structure and organization of the findings . . . 09

Manual Report . . . 11

Critical | MC – 01 | Resolved

A user from one quorum can add a new subscriber to another . . . 11

Critical | MC – 02 | Resolved

There is a chance to add one user to different quorums in contract MultiSigCheckable . . . 11

Critical | MC – 03 | Resolved

Problem with adding a user to the one quorum a few times which may lead to making this quorum unusable . . . 12

Critical | MC – 04 | Resolved

Incorrect veto right managing . . . 13

High | MH – 01 | Resolved

Signature recovering is incorrect . . . 14

Medium | MM – 01 | Resolved

Shadowing of existing declaration . . . 14

Low | ML – 01 | Resolved

Useless *if* statement . . . 15

Low | ML – 02 | Resolved

Lock pragma to a specific version . . . 15

Low | ML – 03 | Resolved

State variables visibility state . . . 16

Informational | MI – 01 | Unresolved
Order of Layout 16
Informational | MI – 02 | Resolved
NatSpec annotations 17
Informational | MI – 03 | Resolved
Debugging tools and comments in production contracts 17
Informational | MI – 04 | Resolved
Inappropriate use of require statement 17
Informational | MI – 05 | Resolved
Unused function parameter 18
Informational | MI – 06 | Resolved
Typo in the codebase 18
Test Results 19
Tests written by Ferrum Network 20
Tests written by Vidma auditors 21

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

During the audit, we analyzed the Basic implementation of IronSafe. Ironsafe is a product similar to Gnosis safe with additional functionality. It is a multi-sig product that allows you to set multiple wallet addresses to be required as signatories for each transaction with a key additional feature of Veto wallets. If at least one wallet is granted veto rights then the transaction will be successful only when one of the signers will be a wallet with veto rights and the minimum number of signatories will be reached. For signature verification, EIP-712 is used.

During the audit process, the Vidma team found several issues, including those with critical severity. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	4 issues	4 issues	0 issues
High	1 issue	1 issue	0 issues
Medium	1 issue	1 issue	0 issues
Low	3 issues	3 issues	0 issues
Informational	6 issues	5 issues	1 issue
Total	**15 issues**	**14 issues**	**1 issue**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

FERRUM NETWORK

With the mission of breaking down barriers to mass adoption, Ferrum empowers the industry by reducing friction and bringing startups and established networks closer together.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from August 1, 2022 to September 15, 2022. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contracts:

- BasicIronSafe;
- PublicMultiSigCheckable;
- MultiSigCheckable;
- WithAdmin;
- MultiSigLib;
- SafeAmount.

The source code was taken from the following **source**:
https://github.com/ferrumnet/open-staking/blob/v2/v2/contracts/ironSafe/BasicIronSafe.sol

Initial commit submitted for the audit:
72687a084cee449bafcb50f49ecae02d663cfc3c

Last commit reviewed by the auditing team:
e5e8383b0356d39795d686451faa13c94243faaf

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contracts.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by Ferrum Network or not. The issues with “Not Valid” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

A user from one quorum can add a new subscriber to another

Critical | MC – 01 | Resolved

MultiSigCheckable contract has a possibility to initialize new quorums. When the admin will add quorums A and B with the same groupId or *ownerGroupId* then users from quorum A will have the possibility to add a new quorum subscriber to quorum B.

Recommendation:

Consider checking if the groupId and *ownerGroupId* don't duplicate for the further added quorums.

There is a chance to add one user to different quorums in contract MultiSigCheckable

Critical | MC – 02 | Resolved

MultiSigCheckable contract has a possibility to initialize new quorums. Via function *addToQuorum()* there is a possibility to add one user to different quorums which will lead to incorrect *quorumsSubscribers* variable accumulation. Each time when the user is added to a new quorum he is de-facto deleted from the previous but *quorumsSubscribers* still stay the same. It may to a critical problem when the user is removed from the quorum.

Recommendation:

Consider checking if the address which is added to the quorum doesn't have any quorum subscription yet

Problem with adding a user to the one quorum a few times which may lead to making this quorum unusable

Critical | MC – 03 | Resolved

There is a possibility to add to quorum the same user a few times via the function *addToQuorum()* here the number of quorumSubscribers will increase each time the user will be added but the actual subscribers won't increase. In this situation, there is a possibility to delete more than allowed users from the quorum based on *minSignatures* required and make this quorum not usable anymore.

Steps to reproduce:

1) Call function *addToQuorum()* - add user A. to *quorumId 0x…1*
   Data will changed to next one: *quorumId: 0x…1, quorumsSubscribers: 3 (users A, B), minSignatures: 2*;
2) Call *removeFromQuorum()* - remove user A
   Data will changed to next one: *quorumId: 0x…1, quorumsSubscribers: 2 (users B), minSignatures: 2*;
3) After this step, any data related to the quorum with *id 0x…1* can't be changed anymore.

Recommendation:

Consider checking if the address which is added to the quorum doesn't have any quorum subscription yet.

Incorrect veto right managing

Critical | MC – 04 | Resolved

When a user receives a veto right and is deleted from quorum he still has a veto right which may lead to the next critical issues:

1) When initially there is only one user A with veto right and this user A is deleted from quorum *vetoRightsLength* variable still equal 1 and when will need to sigh any transaction it will become impossible because of at least one of signer should be with veto right. As only user A has veto rights but he is not in quorum he can't sign transactions and there can't be any possibility to add a user to the quorum or set veto to the other users, or event unset veto for user A.
2) When initially there is not only one user with veto rights. If user B with veto right is deleted from the quorum he still has his veto right, *vetoRightsLength* points to an invalid number. And when in some time quorum participants will return user B to the quorum he will have his veto right by default from the first time he received it. This may point to unexpected logic in the contract.

Recommendation:

Consider unset veto right when the user is deleted from quorum to not broke contract logic.

Signature recovering is incorrect

High | MH – 01 | Resolved

In the contract MultiSigCheckable in the function *verifyUniqueSaltWithQuorumId()* signature is recovered and checked. The function *tryVerifyDigestWithAddress()* is called here where the first parameter should be the hash of the fully encoded EIP712 message (digest) but instead an encoded struct message hash is passed.

Line 352:

```
(bool result, address[] memory signers) =
tryVerifyDigestWithAddress(message, expectedGroupId,
multiSignature);
```

Recommendation:

Consider fixing the signature checking by composing correct digest in the *verifyUniqueSaltWithQuorumId()* funciton as follow:

```
 bytes32 digest = _hashTypedDataV4(message);
       (bool result, address[] memory signers) =
tryVerifyDigestWithAddress(digest, expectedGroupId, multiSignature);
```

Shadowing of existing declaration

Medium | MM – 01 | Resolved

In the constructor of contract BasicIronSafe variable *deploySalt* is shadowing an existing declaration on L16. Additionally, *deploySalt (L16)* is not initialized.

Recommendation:

Consider changing the variable name in contract constructor *deploySalt* to *_deploySalt* and initialize state variable.

Useless *if* statement

Low | ML – 01 | Resolved

In the contract MultiSigCheckable function *tryVerifyDigestWithAddress()* there is an *if* statement in line 413 which checks if the loop iterator *i* is bigger than 0. Iterator *i* is initialized as 1 so *if* is useless here because the *else* branch will never appear.

Recommendation:

Consider removing the useless *if* statement

Lock pragma to a specific version

Low | ML – 02 | Resolved

At the current state for part of the contracts pragma is set to version range ^0.8.0. It's best practice to lock the pragma to a specific version since not all the EVM compiler versions support all the features, especially the latest one which are kind of beta versions, so the intended behavior written in code might not be executed as expected.

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs.

Recommendation:

Consider locking pragma for all contracts to a specific version.

State variables visibility state

Low | ML – 03 | Resolved

In contract BasicIronSafe (L16, L20, L50, L68, L83, L101) defined state variables without explicit visibility state.

Recommendation:

Explicitly mark visibility of state variables.

Order of Layout

Informational | MI – 01 | Unresolved

The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation:

Consider changing the order of layout according to solidity documentation: Order of Layout / Order of Functions.

NatSpec annotations

Informational | MI – 02 | Resolved

Smart contracts are not fully covered by NatSpec annotations.

Recommendation:

Consider covering by NatSpec all contract's methods.

Debugging tools and comments in production contracts

Informational | MI – 03 | Resolved

In contract BasicIronSafe (L8), MultiSigLib (L3), MultiSigCheckable (L8, L374-L375, L389-L391, L398, L415) there is hardhat/console.sol which is used for development propose.

Recommendation:

Remove debugging tools from audited contracts.

Inappropriate use of require statement

Informational | MI – 04 | Resolved

In functions *forceRemoveFromQuorum()* and *cancelSaltedSignature()* in a PublicMultiSigCheckable contract require is used to prevent calling those functions. Require should be used to ensure valid conditions that cannot be detected until execution time. This includes conditions on inputs or return values from calls to external contracts.

Recommendation:

Change require to revert statement.

Unused function parameter

Informational | MI – 05 | Resolved

In function *initialize()* of BasicIronSafe contract all function parameters is unused.

In function *verifyUniqueSalt()* of BasicIronSafe expectedGroupId parameter is not used.

Recommendation:

To silence compile warning comment out the parameters names. Change *initialize()* function state mutability to pure.

```
function initialize(
        address /*quorumId*/,
        uint64 /*groupId*/,
        uint16 /*minSignatures*/,
        uint8 /*ownerGroupId*/,
        address[] calldata /*addresses*/
    ) public pure override {
        revert("BIS: not supported");
    }
```

Typo in the codebase

Informational | MI – 06 | Resolved

There is a typo in the contract MultiSigCheckable in the constant variable *UPDATE_MIN_SIGNATURE_MEHTOD -> ..._METHOD*

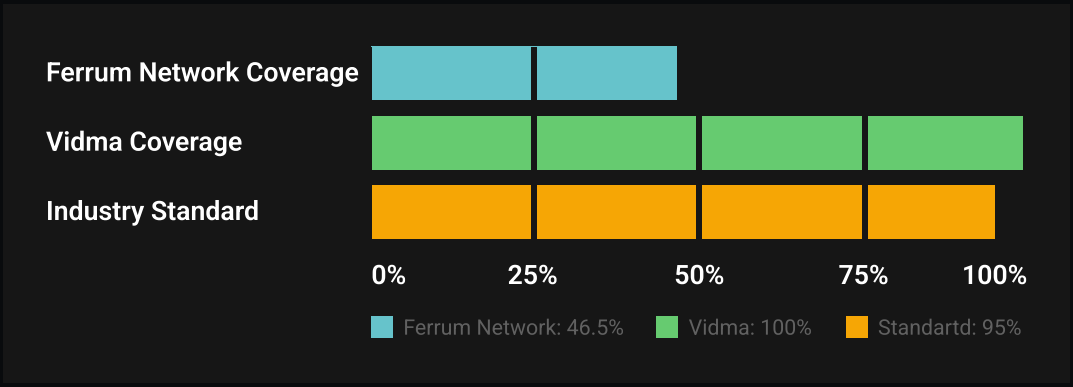
Recommendation:

Consider fixing the typo.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by Ferrum Network and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the Ferrum Network repository. We write totally separate tests with code coverage of a minimum of 95% to meet industry standards.

Tests written by Ferrum Network

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
SafeAmount.sol	0.00	0.00	0.00	0.00
WithAdmin.sol	0.00	0.00	0.00	0.00
MultiSigCheckable.sol	35.87	18.75	13.33	34.74
MultiSigLib.sol	100.00	50.00	100.00	100.00
PublicMultiSigCheckable.sol	0.00	0.00	0.00	0.00
BasicIronSafe.sol	64.71	50.00	44.44	61.11
All Files	**46.5**	**22.22**	**21.88**	**45.12**

Test Results

```
Transfers only on multisig
  ✓ Create a safe with a veto (5035ms)

1 passing (5s)
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
SafeAmount.sol	100.00	75.00	100.00	100.00
WithAdmin.sol	100.00	100.00	100.00	100.00
MultiSigCheckable.sol	100.00	92.86	100.00	100.00
MultiSigLib.sol	100.00	100.00	100.00	100.00
PublicMultiSigCheckable.sol	100.00	100.00	100.00	100.00
BasicIronSafe.sol	100.00	95.00	100.00	100.00
All Files	**100.00**	**92.86**	**100.00**	**100.00**

Test Results

```
Contract: BasicIronSafe
  initialization
    ✓ should set default quorum id correct (78ms)
    ✓ should revert call of 'initialize' function
  setVeto
    ✓ should set veto correct (377ms)
    ✓ should not set veto twice (89ms)
    ✓ should not set veto to a not quorum subscriber
    ✓  should not set veto if signature is incorrect (101ms)
    ✓ should not set veto if signature salt was used already (100ms)
    ✓ should revert if there is no veto signature when requested
      at least one (142ms)
```

```
  unsetVeto
    ✓ should unset veto correct (74ms)
    ✓ should not unset veto if the user initially doesn't have it
    ✓ should not unset veto if signature is incorrect (52ms)
    ✓ should not unset veto if signature salt was used already
    ✓ should revert if there is no veto signature when requested
      at least one
  sendEthSigned
    ✓ should send eth correct (201ms)
    ✓ should not send eth if signature is incorrect (42ms)
    ✓ should not send eth if signature salt was used already (65ms)
    ✓ should revert if there is no veto signature when requested
      at least one (118ms)
  sendSigned
    ✓ should send token correct (76ms)
    ✓ should not send token if signature is incorrect (43ms)
    ✓ should not send token if signature salt was used already (76ms)
    ✓ should revert if there is no veto signature when requested
      at least one (88ms)
  verifyUniqueSalt
    ✓ should verify unique salt correct (55ms)
    ✓ shouldn't verify unique salt message if the group id is not
      supported (41ms)
  SafeAmount
    ✓ should safe transfer from token correct (105ms)

Contract: MultiSigCheckable
  initialize
    ✓ should initialize by the admin correct
    ✓ should not initialize by not the admin or owner
    ✓ should not initialize when passed params are wrong (51ms)
    ✓ should not initialize quorum with same id (72ms)
    ✓ should not add same address to few quorums (61ms)
  updateMinSignature
    ✓ should update minSignature correct (167ms)
    ✓ should not update minSignature if quorumId does not exist
    ✓ should not update minSignature if new amount if too large
  addToQuorum
    ✓ should add to quorum correct (108ms)
    ✓ should not add address to quorum if quorumId does not exist
    ✓ shouldn't add to quorum the same user twice (115ms)
    ✓ shouldn't add to different quorums the same user (382ms)
```

```
removeFromQuorum
  ✓ should remove from quorum correct (92ms)
  ✓ should not remove from quorum if the user is not subscribed
    to that quorum
  ✓ should not remove from quorum too many users (128ms)
  ✓ should delete veto after the user is deleted from
    quorum (346ms)
signature verification
  ✓ should verifyUniqueMessageDigest without salt correct (291ms)
  ✓ should revert if expiry date is timed out
  ✓ should revert if expiry date is more than week from now
  ✓ should revert if signer is not in quorum (61ms)
  ✓ should revert if signers length is zero
  ✓ should revert signers are from different quorums (205ms)
  ✓ shouldn't add to quorum if signers are from different
    quorums (206ms)
  ✓ shouldn't remove from quorum if signers are from different
    quorums (185ms)
  ✓ should add to owned quorum only by signers from owner
    quorum (186ms)
  ✓ should revert if invalid groupId for signer (189ms)
  ✓ should revert if sigs in multisig are not sorted (147ms)
  ✓ should revert there are not enough signatures (171ms)
admin functionality
  ✓ should set admin by the owner correct
  ✓ should not set admin by not the owner
forceRemoveFromQuorum
  ✓ should forceRemoveFromQuorum by the admin correct
  ✓ should forceRemoveFromQuorum if the user is not the
    subscriptor
  ✓ should not forceRemoveFromQuorum by not the admin or owner
  ✓ should revert forceRemoveFromQuorum in BasicIronSafe
cancelSaltedSignature
  ✓ should cancelSaltedSignature correct (207ms)
  ✓ should revert cancelSaltedSignature in BasicIronSafe
modifier governanceGroupId
  ✓ should check if the passed groupId is correct (93ms)
  ✓ should revert if passed groupId is bigger than
    GOVERNANCE_GROUP_ID_MAX (91ms)

Contract: PublicMultiSigCheckable
  ✓ should initialize correct (64ms)
```

```
    ✓ should revert on call forceRemoveFromQuorum
    ✓ should revert on call cancelSaltedSignature

65 passing (23s)
```

We are delighted to have a chance to work with the Ferrum Network team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA