VIDMA

FERRUM NETWORK

# SMART CONTRACT AUDIT

Project: Ferrum Network
Date: July 10th, 2023

# TABLE OF CONTENTS

# 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 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 | Not relevant | Unresolved |
|---|---|---|---|---|
| Critical | 6 issues | 4 issues | 2 issues | 0 issues |
| High | 1 issue | 1 issue | 0 issues | 0 issues |
| Medium | 3 issues | 3 issues | 0 issues | 0 issues |
| Low | 5 issues | 5 issues | 0 issues | 0 issues |
| Informational | 6 issues | 6 issues | 0 issues | 0 issues |
| **Total** | **21 issues** | **19 issues** | **2 issues** | **0 issues** |

After evaluating the findings in this report and the final state after fixes, the Vidma auditors insist on fixing the critical issue which is presented in the report. Otherwise, the issue bears the risk for the end user. Under the given circumstances, we set the following risk level:

To set the codebase quality mark, 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 evaluation codebase quality 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.*

**Codebase quality: 92.70**

Evaluating the **initial commit** and the **last commit with the fixes**, Vidma audit team set the following **codebase quality** mark.

## Score

Based on the **overall result of the audit** and the state of the final reviewed commit, 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 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 April 20, 2023 to May 3, 2023. The last version of fixes was reviewed on July 7, 2023. The outcome is disclosed in this document.

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

- contract\src\address.rs;
- contract\src\bridge_pool_contract.rs;
- contract\src\data.rs;
- contract\src\detail.rs;
- contract\src\error.rs;
- contract\src\event.rs;
- contract\src\lib.rs;
- contract\src\main.rs.

The source code was taken from the following **source**:
https://github.com/ferrumnet/bridge-casper-smart-contracts/

**Initial commit** submitted for the audit:
c6cf7ce7d23605a6c719e761160f7cb2a73844a5

**Last commit** reviewed by the auditing team:
fa334c470a4fedcdedf78b9a8a0a6a97972b1392

Last audited **tag**:
https://github.com/ferrumnet/bridge-casper-smart-contracts/releases/tag/v0.4.8

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the Ferrum Network team has provided the following documentation:
https://docs.ferrumnetwork.io/ferrum-network-ecosystem/v/multiswap-and-multichain-liquidity-pool-bridge/architecture-and-tech-stack/general-architecture-overview/core-components-of-bridging

# 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 relevant” 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

## Lack of check for ownership

Critical | MC – 01 | Resolved

The contract does not implement an ownership model, which is mandatory for the safe operation of the contract, in particular for functions: *allow_target*, *add_signer*, etc.

**Recommendation:**

Consider adding necessary checks for ownership to all management functions.

**UPD:**

Added function *remove_signer()* should be ownable too.

**UPD 2:**

Testing part: *add_signer*, *remove_signer*, *allow_target*, *check_signer* can be called only be admin role.

## Incorrect usage of signature

Critical | MC – 02 | Resolved

In the function *withdraw_signed() message_hash* is not recreated from passed arguments, such as salt, payee, amount, *token_contract_package_hash*, etc, but passed as a function argument. It will not guarantee that the passed argument, is data, that were signed by the FIBER engine. So anyone can take message hash and signature, and manipulate data to withdraw tokens.

**Recommendation:**

Consider following the best industry standards working with signature.

**UPD:**

The address of the token recipient is not in a signed message, so if someone intercepts the signature he can withdraw tokens (stole from user).

**UPD 2:**

*actor(msg.sender)* is not checked to be a *receiver*, if someone intercepts the signature he can withdraw tokens (stole from user). Tokens are sent to *client_addr* but in the event *TransferBySignature* emitted *receiver*.

## Wrong implementation for *withdraw_signed* function

Critical | MC – 03 | Invalid

Function *withdraw_signed()* invokes function *remove_liquidity_generic()* which is designed to remove liquidity. That makes it impossible to withdraw tokens in case the user has no specific liquidity since the error *ClientDoesNotHaveAnyKindOfLioquidity* occurs.

**Response:**

It's expected behavior since *withdraw_signed* designed to be called only by the Ferrum team.

**Recommendation:**

Consider deleting invokes function *remove_liquidity_generic()*.

## Wrong dictionary for checking used *message_hash*

Critical | MC – 04 | Resolved

Function *signer_unique()* uses the wrong dictionary to check if *message_hash* was already used to withdraw tokens, it supposes to use *used_hashes_dict* instead.

**Recommendation:**

Make sure you refer to the correct dictionary.

## Absence of the receiver address in function swap

Critical | MC – 05 | Resolved

Function swap uses "session caller" as *target_address*, which is wrong since the target chain can be EVM compatible or Cosmos, so the standard of target address can be different and it should be passed as String. Furthermore, the type of *target_address* is defined as an Address instead of a String in the *BridgeSwap* event.

**Recommendation:**

Add *target_address* argument and emit the correct address in event.

**UPD:**

bridge_pool_contract.rs::111, *target_address = target_token*
*target_address* should be a string address, in standard of target network

## Entrypoint *constructor()* can be called multiple time

Critical | MC – 06 | Invalid

No check that allows the *constructor()* entrypoint to be called once.

**Recommendation:**

Consider adding a necessary check for the constructor.

## Absence of the *remove_signer* function

High | MH – 01 | Resolved

According to the documentation of the MultiSwap, FIBER Engine validates transactions and sign messages which allow the withdrawal of tokens on the target chain. During the operational life cycle, the signer will be changing, which requires deleting outdated signer addresses from the contract.

**Recommendation:**

Consider adding a function to delete the outdated signer, it will bring more safety to your application.

## Lack of address validation

Medium | MM – 01 | Resolved

FIBER Engine used ECDSA signatures for signing messages, so the signer is a Secp256k1 type key.

Function *add_signer()* does not validate the address of the argument signer. Using the address of the token in mixed case or lower case can bring incorrect processing.

**Recommendation:**

Consider adding a necessary check to arguments with an address.

## Wrong parameter in event *TransferBySignature*

Medium | MM – 02 | Resolved

Event *TransferBySignature* has a wrong parameter signer. According to the documentation signer, it's an Ethereum standard address that signs a message for withdraw tokens, but in the function *withdraw_signed()* of bridge_pool_contract.rs::146, a “session caller” is emitted. Furthermore, the type of signer is defined as an Address instead of a String.

**Recommendation:**

Make sure you emit the correct values in the events.

**UPD:**

Receiver is not the payee, but *msg.sender*.

**UPD 2:**

Tokens are sent to *client_addr* but in the event *TransferBySignature* emitted *receiver*.

## Outdated crates dependencies

Medium | MM – 03 | Resolved

Project uses outdated crate dependencies, such as:

- casper-contract;
- casper-types;
- k256;
- ecdsa;
- secp256k1.

**Recommendation:**

It’s highly recommended to use the latest version of crates dependencies.

**UPD:**

k256 still outdated - invalid, updated to a latest supported version.

## Duplicates function

Low | ML – 01 | Resolved

Function *withdraw()* completely duplicates the functionality of function *remove_liquidity()*.

Path: contracts/src/data.rs::214

**Recommendation:**

Consider deleting useless functions, it will decrease gas consumption during deployment and make the code more clear.

**UPD:**

Entry point were removed but *withdraw()* in contracts/src/data.rs::211 is still present.

## Shadowing variable

Low | ML – 02 | Resolved

Function *add_liquidity()* has shadowed the local variable *bridge_pool_contract_package_hash*(bridge_pool_contract.rs line 54).

**Recommendation:**

Consider renaming the shadowed variable.

## Unused functions

Low | ML – 03 | Resolved

Some functions in *data.rs* after fixes are not used.

Unused functions: *withdraw_signed*, *withdraw_signed_message*, *signer_unique*.

**Recommendation:**

Use that functions as it was done in previous versions, or delete unused functions.

## Duplication of logic in *withdraw_signed*

Low | ML – 04 | Resolved

Function *withdraw_signed* in *bridge_pool_contract.rs* duplicate logic from function *withdraw_signed* in *data.rs*. Function *check_signer* to check signer is not used. Function *pay_from_me* to transfer tokens is not used.

**Recommendation:**

Consider replacing old code in *data.rs* and call relevant functions from *data.rs*.

## Variable duplicates

Low | ML – 05 | Resolved

Variable *client_address* (line 165) and *client_addr* (line 247) duplicates variable *actor* (line 162).

**Recommendation:**

Consider deleting useless variables.

## Typos

Informational | MI – 01 | Resolved

Typos in the contracts:

- contracts/src/data.rs::37, *BrigdePool -> BridgePool*;
- contracts/src/data.rs::94, *get_liquidity_added_by_client_genric - > get_liquidity_added_by_client_generic*;
- contracts/src/error.rs::25, *ClientDoesNotHaveAnyKindOfLioquidity -> ClientDoesNotHaveAnyKindOfLiquidity*;
- contracts/src/error.rs::26, *ClientDoesNotHaveSpecificKindOfLioquidity -> ClientDoesNotHaveSpecificKindOfLiquidity*;
- contracts/src/event.rs::31, *reciever -> receiver*.

**Recommendation:**

Consider renaming all variables with typos.

## Redundant code repetition

Informational | MI – 02 | Resolved

The code snippet below is used 5 times in the contract: contracts/src/data.rs.

```
let dict = match client_address {
    Address::Account(_) => &self.account_hash_liquidities_dict,
    Address::ContractPackage(_) => &self.hash_addr_liquidities_dict,
    Address::ContractHash(_) => return
Err(Error::UnexpectedContractHash),
};
```

**Recommendation:**

Consider writing a function and using it instead of redundant code repetition.

## Redundant code repetition

Informational | MI – 03 | Resolved

The code snippet below is used 4 times in contract: contracts/src/data.rs::524.

```
if actor.as_account_hash().is_some() { // TODO: use try_into
    param.insert("actor",
actor.as_account_hash().unwrap().to_string());
} else {
    param.insert(
        "actor",
        actor.as_contract_package_hash().unwrap().to_string(),
    );
};
```

That can be simply done by calling *actor.try_into().unwrap()*, which are overridden in *address.rs*, and implementing the same logic.

**Recommendation:**

Consider using *try_into()* to reduce redundant code repetition.

## Using string literals instead of constants

Informational | MI – 04 | Resolved

Use string for Key name can lead to issues if somewhere will be a different value. It's a good practice to create a constant for Key names, entry points, and argument names.

“Bridge_pool_contract_package_hash” -> const BRIDGE_POOL_CONTRACT_PACKAGE_HASH_KEY

Path:

- contracts/src/data.rs::446;
- contracts/src/main.rs::61;
- contracts/src/main.rs::58;
- contracts/src/main.rs::58.

**Recommendation:**

Consider using constants for Key name, entry points, and arguments name instead of string literals.

**UPD:**

Not fixed in main.rs:154, use *CHAIN_ID*.

**UPD 2:**

*data.rs*, line 299, 301, 469, 470, 493. Use one style, if you decide to replace string literal with constants, do it everywhere. To make it clear it's better to add prefixes/ suffixes to the name constants, such as “*ARG_NAME*”, “*ENTRY_POINT*”, etc.

Furthermore, in order to prevent issues it's better to move all constants to separate files and import them into a place where you need them, instead of creating "duplicates" in a few files.

## Commented code

Informational | MI – 05 | Resolved

Useless commented code in the data.rs::460-472, lib.rs::10-17.

**Recommendation:**

Consider deleting unused commented code.

## Unused functions

Informational | MI – 06 | Resolved

There are a few unused functions in the contracts:

- contracts/src/data.rs::481 – *name()*;
- contracts/src/data.rs::485 – *set_name()*;
- contracts/src/data.rs::489 – *address()*;
- contracts/src/data.rs::493 – *set_address()*.

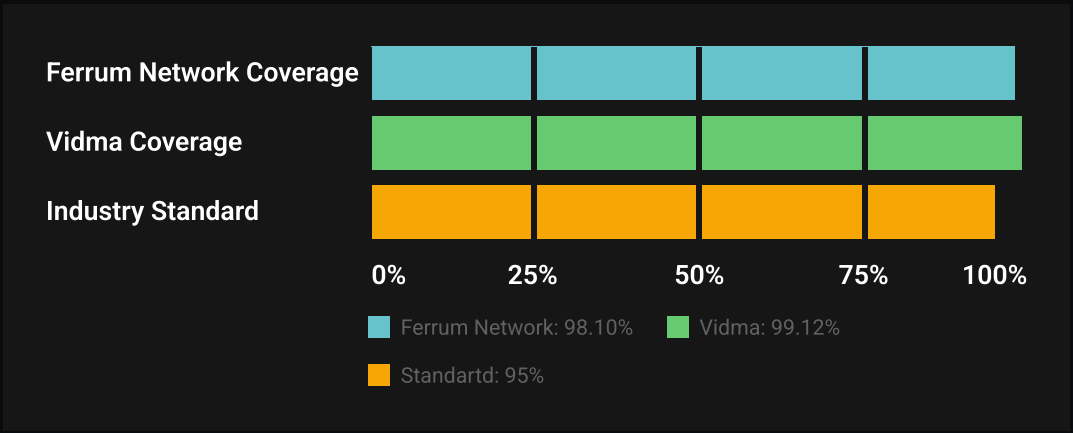
**Recommendation:**

Consider deleting unused functions.

# TEST RESULTS

To verify the security of contracts and their performance, a number of integration tests were carried out using the Cargo 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 the industry standards.

## Tests written by Ferrum Network

### Test Coverage

| File | % Stmts |
|---|---|
| contract/src/lib.rs | 100.00 |
| contract/src/event.rs | 100.00 |
| contract/src/consts.rs | 100.00 |
| contract/src/error.rs | 100.00 |
| contract/src/detail.rs | 100.00 |
| contract/src/address.rs | 100.00 |
| contract/src/bridge_pool_contract.rs | 90.23 |
| contract/src/main.rs | 100 |
| contract/src/data.rs | 92.68 |
| **All Files** | **98.10** |

## Test Results

```
running 8 tests
  test tests::should_be_able_to_install_and_add_signer ... ok
  test tests::should_be_able_to_install_and_allow_target ... ok
  test tests::should_be_able_to_install_and_add_liquidity ... ok
  test tests::should_be_able_to_install_add_and_remove_signer
  ... ok
  test tests::should_be_able_to_install_and_get_liquidity ... ok
  test tests::should_be_able_to_install_and_add_liquidity_and
  _remove_liquidity ... ok
  test tests::should_be_able_to_install_and_add_liquidity_and_swap
  ... ok
  test tests::should_be_able_to_install_and_add_liquidity_and
  _withdraw_signed ... ok

test result: PASSED. 8 passed; 0 failed; 0 ignored; 0 measured;
0 filtered out; finished in 4.08s
```

# Tests written by Vidma auditors

## Test Coverage

| File | % Stmts |
|---|---|
| contract/src/lib.rs | 100.00 |
| contract/src/event.rs | 100.00 |
| contract/src/consts.rs | 100.00 |
| contract/src/error.rs | 100.00 |
| contract/src/detail.rs | 100.00 |
| contract/src/address.rs | 100.00 |
| contract/src/bridge_pool_contract.rs | 95.49 |
| contract/src/main.rs | 100 |
| contract/src/data.rs | 96.61 |
| **All Files** | **99.12** |

## Test Results

```
running 17 tests
  test tests::allow_target_by_not_admin_failure ... ok
  test tests::add_signer_by_not_admin_failure ... ok
  test tests::add_liquidity_catching_err_not_contract_package
  _hash_for_token_argument ... ok
  test tests::allow_target ... ok
  test tests::add_liquidity_catching_err_not_bridge_pool_contract
  _package_hash ... ok
  test tests::add_signer ... ok
  test tests::add_liquidity ... ok
  test tests::add_liquidity_for_few_tokens_by_user ... ok
  test tests::build_bridge ... ok
  test tests::remove_signer_by_not_admin_failure ... ok
  test tests::allow_target_expect_adding_few_tokens_for_one_chain
  ... ok
  test tests::remove_liquidity ... ok
  test tests::remove_signer ... ok
  test tests::swap ... ok
  test tests::withdraw_signed ... ok
  test tests::withdraw_signed_by_anyone_with_signature ... ok
  test tests::swap_few_tokens ... ok

test result: PASSED. 17 passed; 0 failed; 0 ignored; 0 measured;
0 filtered out; finished in 7.48s
```

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