#HashEx
BLOCKCHAIN SECURITY

Fieres Bridge

smart contracts
final audit report

April 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Fieres team to perform an audit of their smart contracts. The audit was conducted between 31/03/2023 and 05/04/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The audited contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

The code is available at @fiereschain/fieresbridge GitHub repository and was audited after the commit 2967915.

Update. A recheck was done after the commit f11e28e.

2.1 Summary

Project name	Fieres Bridge
URL	https://fieroscan.com/
Platform	Fieres Network
Language	Solidity

2.2 Contracts

Name	Address
WFIERO	
CoinSwapAgentImpl	
TokenSwapAgentImpl	
BEP20TokenImplementation	
Imports and interfaces	
Centralization issues	

3. Found issues

11
Total issues

●	High	1 (9%)
●	Low	5 (45%)
●	Info	5 (46%)

C2. CoinSwapAgentImpl

ID	Severity	Title	Status
C2-01	● Low	Gas optimizations	Partially fixed
C2-02	● Info	Lack of events	✓ Resolved

C3. TokenSwapAgentImpl

ID	Severity	Title	Status
C3-01	● Low	Gas optimizations	✓ Acknowledged
C3-02	● Info	Lack of events	✓ Resolved
C3-03	● Info	Multiple pairs with the same address aren't supported	✓ Acknowledged

C4. BEP20TokenImplementation

ID	Severity	Title	Status
C4-01	● High	Transfer blacklist	✓ Resolved
C4-02	● Low	Gas optimizations	✓ Acknowledged
C4-03	● Info	Lack of events	✓ Partially fixed

C5. Imports and interfaces

ID	Severity	Title	Status
C5-01	● Low	Proxy implementations have no storage gaps	✓ Acknowledged
C5-02	● Low	SafeMath is outdated	✓ Resolved
C5-03	● Info	Lack of documentation (NatSpec)	✓ Acknowledged

4. Contracts

C1. WFIERO

Overview

A simple wrapper contract for EVM currency to a ERC20 standard token with 18 decimals.

No issues were found.

C2. CoinSwapAgentImpl

Overview

The CoinSwapAgentImpl is part of a cross-chain bridge to receive native EVM currency in order to generate events for bridge's backend, and to be operated by a single privileged account for swaps in other direction.

CoinSwapAgentImpl is intended to be used behind a proxy contract.

Issues

C2-01 Gas optimizations ● Low ✓ Partially fixed

1. The `wfiero` variable should be declared immutable.

2. `Address.isContract()` method has to be used only to check non-zero bytecode of the address in question. It's unreliable in check if target is not a contract and should be removed.

3. Multiple reads from storage of the `swapFee` variable in the `swapCoin2Token()` function.

4. Double check of `msg.value > swapFee` in the `swapCoin2Token()` function.

C2-02 Lack of events ● Info ✓ Resolved

The function `updateSwapFee()` doesn’t emit events, which complicates the off-chain tracking of important changes.

C3. TokenSwapAgentImpl

Overview

The TokenSwapAgentImpl is part of a cross-chain bridge to burn ERC20 tokens in order to generate events for bridge's backend, and to be operated by a single privileged account for swaps in other direction, i.e. minting ERC20 tokens.

TokenSwapAgentImpl is intended to be used behind a proxy contract.

Issues

C3-01 Gas optimizations ● Low ✓ Acknowledged

1. `Address.isContract()` method has to be used only to check non-zero bytecode of the address in question. It's unreliable in check if target is not a contract and should be removed.

2. The SafeMath library is imported but not used.

C3-02 Lack of events ● Info ✓ Resolved

The function `setSwapFee()` doesn’t emit events, which complicates the off-chain tracking of important changes.

C3-03 Multiple pairs with the same address aren't supported ● Info ✓ Acknowledged

The `createSwapPair()` function populates pair mappings for both directions: `swapMappingCoin2Token[]` and `swapMappingToken2Coin[]`. Thus, multiple pairs with the same coin address, e.g. WFIERO, can't be created.

```
    function createSwapPair(bytes32 _coinTxHash, address _wfieroAddr, ...) onlyOwner
external returns (address) {
        require(swapMappingCoin2Token[_wfieroAddr] == address(0x0), "Duplicated swap
pair");

        BEP20UpgradeableProxy proxyToken = new BEP20UpgradeableProxy(tokenImplementation,
tokenProxyAdmin, "");
        IProxyInitialize token = IProxyInitialize(address(proxyToken));
        token.initialize(_name, _symbol, _decimals, 0, true, address(this));

        swapMappingCoin2Token[_wfieroAddr] = address(token);
        swapMappingToken2Coin[address(token)] = _wfieroAddr;

        emit SwapPairCreated(_coinTxHash, address(token), _wfieroAddr, _symbol, _name,
_decimals);
        return address(token);
    }
```

C4. BEP20TokenImplementation

Overview

A BEP20 standard token, which is an extension of the ERC20 token standard. Supports privileged minting as well as white- and blacklisting.

BEP20TokenImplementation is intended to be used behind a proxy contract.

Issues

C4-01 Transfer blacklist ● High ✓ Resolved

The blacklist of transfer prohibited users is implemented. The major holders, e.g. swap pairs may become inaccessible, if the owner's key would be compromised.

At the same time, blacklisting is not working since it can be bypassed via transferFrom() function.

```
    function transfer(address recipient, uint256 amount) external override returns (bool)
{
        require(!blacklisted[msg.sender], "User blacklisted");
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) external
override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
"BEP20: transfer amount exceeds allowance"));
        return true;
    }
```

Recommendation

Move check for blacklist in the `_transfer()` function.

C4-02 Gas optimizations ● Low ✓ Acknowledged

1. Search over `nomineeList[]` array is ineffective, better to use EnumerableSet from OpenZeppelin.

2. The `whitelisted` and `nomineeList` variables aren't used in the contract.

C4-03 Lack of events

● Info — Partially fixed

The functions `whiteList()`, `removeWhiteList()`, `blackList()`, and `removeBlackList()` don't emit events, which complicates the off-chain tracking of important changes.

C5. Imports and interfaces

Overview

Libraries: Address, SafeERC20, SafeMath, StorageSlot.

Contracts: Initializable, Context, Ownable, ERC20, Proxy, ERC1967Upgrade, ERC1967Proxy, Tr ansparentUpgradeableProxy, BEP20UpgradeableProxy.

Interfaces: IERC20, IERC20Permit, ISwap, IBeacon, IERC1822Proxiable, IBEP20.

Issues

C5-01 Proxy implementations have no storage gaps

● Low — Acknowledged

The CoinSwapAgentImpl, TokenSwapAgentImpl, and BEP20TokenImplementation contracts lack storage gaps for possible proxy upgrades. The owner must pay attention for storage layout during implementation changes.

C5-02 SafeMath is outdated

● Low — Resolved

Unlike BEP20TokenImplementation, the CoinSwapAgentImpl contract inherits the outdated SafeMath library to be used with pre-0.8 pragma version of Solidity.

```
// BEP20TokenImplementation_flat (2).sol
library SafeMath {
  ...
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        return a + b;
    }
  ...
}

// CoinSwapAgentImpl_flat.sol
library SafeMath {
  ...
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
  ...
}
```

C5-03 Lack of documentation (NatSpec) ● Info ✓ Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

C6. Centralization issues

Overview

The BEP20TokenImplementation, CoinSwapAgentImpl, and TokenSwapAgentImpl are going to be deployed behind the proxies, supposedly TransparentUpgradeableProxy, which allows implementation upgrade. Users have to trust the project owners.

Both parts of the bridge, CoinSwapAgentImpl and TokenSwapAgentImpl, rely on the owner accounts, whose private key has to be stored at the backend. Compromising these keys may lead to a massive loss of the collected funds of the CoinSwapAgentImpl contract.

All the BEP20TokenImplementation ERC20 tokens are deployed in the TokenSwapAgentImpl contract with the external proxy admin, who can update token implementations and gain access to the minting.

All the BEP20TokenImplementation ERC20 tokens contain transfer blacklisting functionality, which may be used maliciously by the owner.

The CoinSwapImpl contract owner may withdraw all native currency wrapped into WFIERO with the `failSafe()` function.

5. Conclusion

1 high, 5 low severity issues were found during the audit. 1 high, 1 low issues were resolved in the update.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured. Review the "Centralization Issues" section of the report to understand the risks associated with a malicious owner or a compromised owner's private key.

The audited contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will. The audit was done on the contracts published in the GitHub repo. Users must check if they are interacting with the same contracts as were audited.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

#HashEx
BLOCKCHAIN SECURITY