#HashEx
BLOCKCHAIN SECURITY

Fieres Deposit Manager

smart contracts
final audit report

July 2024

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 contract. The audit was conducted between 30/06/2024 and 02/07/2024.

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 code is available at 0x470F10637DF1124596b916357542e0D4AfE34423 in the Fieres network.

2.1 Summary

Project name	Fieres Deposit Manager
URL	https://fieres.io/
Platform	Fieres Network
Language	Solidity
Centralization level	High
Centralization risk	High

2.2 Contracts

Name	Address
DepositManager	

3. Project centralization risks

C1dCR2a Owner privileges

The DepositManager contract allows users to deposit native EVM coins but does not provide any methods to withdraw. The contract's owner can withdraw all contract's balance at once.

4. Found issues

High	1 (17%)
Medium	1 (17%)
Low	3 (50%)
Info	1 (16%)

C1d. DepositManager

ID	Severity	Title	Status
C1dIfc	High	Lack of tests and documentation	Open
C1dI01	Medium	Incorrect usage of blockhash	Open
C1dI00	Low	Deterministic random	Open
C1dIff	Low	Unaddressed funds	Open
C1dIfd	Low	Gas optimizations	Open
C1dI02	Info	Inconsistent comment	Open

5. Contracts

C1d. DepositManager

Overview

The contract allows users to deposit coins for desired beneficiary. The only outcome is deposit event and stored value and time of deposit. The contract has a helper function to generate a pseudo random address (without a corresponding private key) passing a nonce parameter to it. According to the Fieres team this address is used as a key to store deposit records for a user. This is how the contract handles user's deposits from different accounts. It's crucially important that no generated address is used for accounts of different users. This logic is handled off-chain and could not be verified via this smart contract audit.

Issues

C1dIfc Lack of tests and documentation

High Open

The project doesn't contain any tests and documentation. We urgently recommend increasing test coverage. We also suggest providing the documentation section and add in-code descriptions using the NatSpec Format.

C1dI01 Incorrect usage of blockhash

Medium Open

Blockhash of current block is 0, so returned address of the `createWallet()` function is derived only from input parameter.

```
function createWallet(uint256 nonce) public view returns (address) {
    ...
    keccak256(abi.encodePacked(nonce, blockhash(block.number)))
}
```

Recommendation

BLOCKHASH is available only for 256 most recent blocks already mined.

C1dI00 Deterministic random

Low Open

The `createWallet()` function uses weak on-chain source of randomness to return address.

```
    function createWallet(uint256 nonce) public view returns (address) {
        address randomAddress = address(
            uint160(
                uint(
                    keccak256(abi.encodePacked(nonce, blockhash(block.number)))
                )
            )
        );
        return randomAddress;
    }
```

C1dIff Unaddressed funds

Low Open

Unaddressed coins can be sent to the contract using the `receive()` function.

```
function deposit(address userDepositAddress) public payable {
    ...
}

receive() external payable {}
```

Recommendation

Remove empty receive function or redirect it to internal `_deposit(msg.sender)` function.

C1dIfd Gas optimizations

Low Open

1. Unused code: the `wallets` variable and its getter are never used and remain zeroed.

2. The `depositHistory[]` array has no clear use case and is duplicated by events.

3. Multiple reads from storage in the `deposit()` function: `userDepositAddresses[msg.sender].length` variable.

4. Multiple reads from storage in the `withdrawAll()` and `emergencyDrain()` functions: `admin` variable.

5. Duplicated events: the `DepositMade` and `WalletEvent` events are emitted in the same context and contain same data.

6. Duplicated code: the `withdrawAll()` and `emergencyDrain()` functions shares the identical logic.

7. Duplicated code: the `totalDeposited()` and `getTotalDeposited()`, and the `lastDepositAmount()` and `getLastDepositAmount()` pair of functions are duplicated getters for the same storage data.

C1dI02 Inconsistent comment

Info Open

The comment L6 is not the part of `wallets` description.

```
    // uint256 public nonce;
    address[] public wallets;
```

6. Conclusion

1 high, 1 medium, 3 low severity issues were found during the audit. No issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

The crucial logic is of mapping user accounts to deposit address is handled off-chain and could not be verified in the contract. See the contract description for more details.

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. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. 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

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY