DeHacker

Code Security Assessment

VOLAARK

July 27th, 2024

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best prac tices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	THE VOLAARK
Platform	Tron
website	https://volaark.io/
Type	DeFi
Language	Solidity
Codebase	https://github.com/ARKFLEET/ark_fx/tree/main/contracts

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	2	0	0	2	0	0
Medium	0	0	0	0	0	0
Minor	4	0	0	4	0	0
Informational	1	0	0	1	0	3
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
MTL	TonarkLabs/ark_fx	cf935f53a46e2d0fc12088fdb4756d97baa8d379ae5aefd0abb74ac3cd5344fe
RPT	TonarkLabs/ark_fx	2b78e580a026f956bfb4aedbb82c5d86f8948fd549b1583f7c58d2339f10f6d0
TTL	TonarkLabs/ark_fx	063f85cf1e159912eb6fd3e8e6c10013faaea2d6785d461c299bc3df3b

Findings

ID	Issue	Severity	Status
TLB-01	Centralized Control Of Contract Upgrade	Major	Acknowledged
TLB-05	Centralization Risks	Informational	Resolved
TLB-02	Initialize Functions Are Unprotected	Informational	Resolved
TLB-03	Out of Scope Dependency Usage	Informational	Resolved
TLB-04	Missing Zero Address Validation	Major	Acknowledged
TTL-06	Immutable Incompatible With Upgradeable Contracts	Informational	Resolved
TTL-04	Unused Return Value	Informational	Resolved

MAJOR

TLB-01|Centralization Control Of Contract Upgrade

Issue	Severity	Location	Status
Centralization	Major	Contracts/Market.sol:18; contracts/RebalancePool.sol: 60; contracts/Treasury.sol:27	Acknowledged

Description

The contracts Market, RebalancePool, and Treasury serve as the implementation contracts for proxy contracts. The proxy contract allows interaction with the implementation contract while preserving the state of the proxy contract. The admin of the proxy contract has the authority to change the address of the implementation contract to a new address. Any compromise to the proxy admin account may allow a hacker to take advantage of this authority and change the implementation contracts which are pointed by proxies and therefore execute potential malicious functionality in the implementation contracts.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining atime-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private keycompromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrateto a new implementation contract.Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level andsuggestions that would permanently fully resolve the risk.

Recommendation

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation andavoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private keycompromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:Provide the deployed time-lock address

- Provide the deployed time-lock address
- Provide the gnosis address with ALL the multi-signer addresses for the verification process.
- Provide a link to the medium/blog with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate thecontract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with thecommunity.

Recommendation

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the gnosis address with ALL the multi-signer addresses for the verification process.
- Provide a link to the medium/blog with all of the above information included.

P**ermanent:**

Renouncing ownership of the admin account or removing the upgrade functionality can fully resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shallmake a decision based on the current state of their project, timeline, and project resources.

MAJOR

TBL-05|Centralization Risks

Issue	Severity	Location	Status
Centralization	Major	contracts/Market.sol: 544, 569, 595, 618, 638, 646, 654, 663,671, 679, 687, 695; contracts/RebalancePool.sol: 475, 528,545, 553, 568, 576, 586, 604, 628; contracts/Treasury.sol: 341, 384, 414, 438, 463, 481, 488, 494, 507, 515, 523, 531, 540,548, 556	Acknowledged

Description

In the contract Treasury the role market has authority over the functions shown in the diagram below. Any compromise to the market account may allow the hacker to take advantage of this authority and:

- function mint(), to mint fToken or xToken.
- function redeem(), to burn fToken/xToken from an account and transfer baseToken out to the market.
- function addBaseToken(), to mint xToken with incentive.
- function liquidate(), to burn fToken from an account and transfer baseToken out to the market .

Note that in practice, the role should be the Market contract.

Description

In the contract Treasury the role owner has authority over the functions shown in the diagram below. Any compromise tothe owner account may allow the hacker to take advantage of this authority and:

- function initializePrice() , to initialize the lastPermissionedPrice .
- function updateStrategy() , to change address of strategy contract.
- function updateBeta() , to change the value of fToken beta.
- function updatePriceOracle() , to change address of price oracle contract.
- function updateRateProvider() , to change address of rate provider contract.
- function updateSettleWhitelist() , to update the whitelist status for settle account.
- function updateBaseTokenCap() , to update the baseToken cap.
- function updateEMASampleInterval() , to update the EMA sample interval.

In the contract Treasury the role strategy has authority over the functions shown in the diagram below. Any compromiseto the strategy account may allow the hacker to take advantage of this authority and:

- function transferToStrategy() , to transfer baseToken to strategy.
- function notifyStrategyProfit() , the function currently does not have any code.

In the contract Treasury the role settleWhitelist has authority over the function shown in the diagram below. Anycompromise to the settleWhitelist account may allow the hacker to take advantage of this authority and:

- function protocolSettle() , to settle the nav of baseToken, fToken and xToken

In the contract Market the role DEFAULT_ADMIN_ROLE has authority over the functions shown in the diagram below. Anycompromise to the DEFAULT_ADMIN_ROLE account may allow the hacker to take advantage of this authority and:

- function updateRedeemFeeRatio() , to update the fee ratio for redeeming.
- function updateMintFeeRatio() , to update the fee ratio for minting.
- function updateMarketConfig() , to update the market config.

Description

- function updateIncentiveConfig() , to update the incentive config.
- function updatePlatform() , to change address of platform.
- function updateReservePool() , to change address of reserve pool contract.
- function updateRebalancePoolRegistry() , to change address of RebalancePoolRegistry contract.
- function updateLiquidationWhitelist() , to update the whitelist status for self liquidation account

In the contract Market the role EMERGENCY_DAO_ROLE has authority over the functions shown in the diagram below. Anycompromise to the EMERGENCY_DAO_ROLE account may allow the hacker to take advantage of this authority and:

- function pauseMint() , to pause minting in this contract.
- function pauseRedeem() , to pause redeeming in this contract.
- function pauseFTokenMintInSystemStabilityMode() , to pause fToken minting in system stability mode.
- function pauseXTokenRedeemInSystemStabilityMode() , to pause xToken redeeming in system stability mode.

In the contract RebalancePool the role owner has authority over the functions shown in the diagram below. Any compromise to the owner account may allow the hacker to take advantage of this authority and:

- function updateLiquidator() , to update the address of liquidator.
- function updateWrapper() , to update the address of reward wrapper.
- function updateLiquidatableCollateralRatio() , to update the liquidatable collateral ratio.
- function updateUnlockDuration() , to update the unlock duration after unlocking.
- function addReward() , to add a new reward token to this contract.
- function removeReward() , to remove an existed reward token.
- function updateReward() , to update the reward distribution for some reward token.

In the contract RebalancePool the role rewardManager has authority over the function shown in the diagram below. Anycompromise to the rewardManager account may allow the hacker to take advantage of this authority and:

- function depositReward() , to transfer tokens to the contract as rewards.

Description

In the contract RebalancePool the role liquidator has authority over the function shown in the diagram below. Anycompromise to the liquidator account may allow the hacker to take advantage of this authority and

- function liquidate() , to liquidate asset for baseToken.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level ofdecentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefullymanage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommendcentralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accountswith enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that wouldalso mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination mitigate by delaying the sensitive operation and avoiding a single point of keymanagement failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private keycompromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, mitigate by applying decentralization and transparency.

Recommendation

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

AND

- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered fully resolved.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality

MINOR

TLB-02|Initialize Functions Are Unprotected

Issue	Severity	Location	Status
Access Control	Minor	contracts/Market.sol: 221; contracts/RebalancePool.sol: 242; contracts/Treasury.sol: 160, 185	Acknowledged

Description

The initialization functions for the Treasury , Market , and RebalancePool contracts are not adequately protected. Thisvulnerability allows an attacker to initialize these contracts by front-running the functions and set critical variables in the proxycontracts. Additionally, leaving the logic contracts' initialization functions unprotected allows an attacker to call these functionsand assume ownership of the logic contracts, enabling them to perform privileged operations and deceive users.

Affected Functions:

- Treasury.initialize()
- Treasury.initializeV2()
- Market.initialize()
- RebalancePool.initialize()

Recommendation

Add checks to verify that the sender is authorized to perform initialization

MINOR

TLB-03|Out Of Scope Dependency Usage

Issue	Severity	Location	Status
Design Issue	Minor	contracts/Market.sol; contracts/RebalancePool.sol; contracts/Treasury.sol	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more out-of-scope protocols. The scope of the audittreats out-of-scope entities as black boxes and assumes their functional correctness. However, in the real world, out-of-scopeentities can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possiblycreate severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
198 address public reservePool;
199 if (_bonus > 0 && IFxRebalancePoolRegistry(registry).totalSupply() == 0) {
200 _bonus = IFxReservePool(reservePool).requestBonus(baseToken, _recipient,
_bonus);
201     }
```

The contract Market interacts with third party contract with IFxReservePool interface via reservePool.

```
201 address public registry;
202 if (_bonus > 0 && IFxRebalancePoolRegistry(registry).totalSupply() == 0) {
203 _bonus = IFxReservePool(reservePool).requestBonus(baseToken, _recipient,
_bonus);
204 }
```

Description

- The contract Market interacts with third party contract with IFxRebalancePoolRegistry interface via registry.

```
198 address public wrapper;
```

- The contract RebalancePool interacts with third party contract with IFxTokenWrapper interface via wrapper .

```
22 import { FxLowVolatilityMath } from "./math/FxLowVolatilityMath.sol";
23 using FxLowVolatilityMath for FxLowVolatilityMath.SwapState
```

- The contract Treasury interacts with third party contract with IAssetStrategy interface via strategy .

```
120 address public override strategy;
```

- The contract Treasury interacts with third party contract with IAssetStrategy interface via strategy .

```
129 address public rateProvider;
```

- The contract Treasury interacts with third party contract with IFxRateProvider interface via rateProvider .

```
620 ExponentialMovingAverageV7.EMAStorage memory cachedEmaLeverageRatio =
emaLeverageRatio;
621 uint256 _ratio = _state.leverageRatio(beta, _earningRatio);
622 cachedEmaLeverageRatio.saveValue(uint96(_ratio));
```

Description

The contract Treasury interacts with third party contract with ExponentialMovingAverageV7 interface via cachedEmaLeverageRatio .

```
655 (bool _isValid, uint256 _safePrice, uint256 _minPrice, uint256 _maxPrice) =
IFxPriceOracle(priceOracle).getPrice();
```

- The contract Treasury interacts with third party contract with IFxPriceOracle interface via priceOracle .

Recommendation

The auditors acknowledge that the business logic requires interaction with third parties. It is recommended that the team:

- Constantly Monitor Third-Party Statuses: Regularly check the status and reliability of third-party services and contracts.
- Mitigate Side Effects: Develop strategies to handle unexpected activities or issues arising from third-party interactions to minimize adverse impacts on your contract.

MINOR

TLB-04|Missing Zero Address Validation

Issue	Severity	Location	Status
Volatile Code	Minor	contracts/Market.sol: 639, 647, 655; contracts/RebalancePool.sol: 546; contracts/Treasury.sol: 172, 173, 174, 175, 176, 508, 524	Acknowledged

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leadingto unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent lossof those tokens

```
639 platform = _platform
```

- _platform is not zero-checked before being used.

```
647 reservePool = _reservePool;
```

- _reservePool is not zero-checked before being used.

```
655 registry = _registry;
```

Description

- _registry is not zero-checked before being used.

```
172 market = _market;
```

- _market is not zero-checked before being used.

```
173 baseToken = _baseToken;
```

- _baseToken is not zero-checked before being used.

```
174 fToken = _fToken;
```

- _fToken is not zero-checked before being used.

```
175 xToken = _xToken;
```

- _xToken is not zero-checked before being used.

```
176 priceOracle = _priceOracle;
```

- _priceOracle is not zero-checked before being used.

```
508 strategy = _strategy;
```

Description

- _strategy is not zero-checked before being used.

```
524 priceOracle = _priceOracle;
```

- _priceOracle is not zero-checked before being used.

```
546 liquidator = _liquidator;
```

- _liquidator is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

MINOR

TTL-06|Immutable Incompatible With Upgradeable Contracts

Issue	Severity	Location	Status
Logical Issue	Minor	contracts/Treasury.sol: 157	Acknowledged

Description

The immutable keyword is incompatible with the way upgradeable contracts work for two reasons:

1. Upgradeable contracts have no constructors but initializers, therefore they can't handle immutable variables.
2. Since the immutable variable value is stored in the bytecode its value would be shared among all proxies pointing toa given contract instead of each proxy's storage.

In some cases, immutable variables are upgrade-safe. The plugins cannot currently detect these cases automatically so theywill point it out as an error anyway. You can manually disable the check using the option unsafeAllow: ['state-variable-immutable'], or in Solidity >=0.8.2 placing the comment /// @custom:oz-upgrades-unsafe-allow state-variable-immutablebefore the variable declaration.

In the Treasury contract, the solidity version is 0.7.6 and the immutable keyword is used together with the constructor.

Description

```
pragma solidity ^0.7.6;
pragma abicoder v2;

    /// @dev The initial mint ratio for fToken.
    uint256 private immutable initialMintRatio;

    constructor(uint256 _initialMintRatio) {
        require(0 < _initialMintRatio && _initialMintRatio < PRECISION, "invalid initial
mint ratio");
        initialMintRatio = _initialMintRatio;
    }
```

Recommendation

We recommend the team to be cautious if the compliation error occurs.

INFORMATIONAL

TTL-04|Unused Return Value

Issue	Severity	Location	Status
Coding Issue, Volatile Code	Informational	contracts/Treasury.sol: 591	Acknowledged

Description

The return value _amount of the function _transferBaseToken() is not used in the contract. The function _transferBaseToken() is called in redeem and liquidate, in both places the return value is not used.

Recommendation

We advise to remove the return value if it's redundant.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege
Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style
Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code
Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue
Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

Ethereum
Cosmos
Eos
Substrate

TECH STACK

Python
Solidity
Rust
c++

CONTACTS

https://dehacker.io
https://twitter.com/dehackerio
https://github.com/dehacker/audits_public
https://t.me/dehackerio
https://blog.dehacker.io/

July 27th 2024