DeHacker

Code Security Assessment

TrustFi Farmer AFP

Mar. 14th, 2021

Contents

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. 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	TrustFi Farmer AFP
Platform	BSC
website	https://www.trustfi.org/
Language	Solidity
Codebase	https://github.com/TrustFiNetwork/TrustFi-Farmer-AFP
Commit	ec525d786518905c37bcf3142901a50d5344abc4

Audit Summary

Delivery Date	Mar. 14, 2021
Audit Methodology	Static Analysis, Manual Review

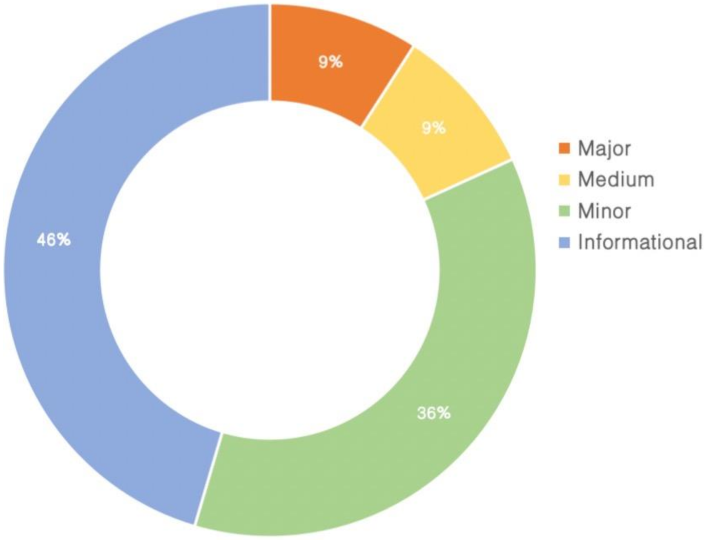
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	1	0	0
Medium	1	0	0	0	0	1
Minor	4	0	0	0	0	4
Informational	5	0	0	0	0	5
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
TFS	TrustFiStakingFactory.sol	b3f4eb75079a8bee85da4bc321ffbea7929bf14893da2cd146a40a64160674ef
TFF	TrustFiStakingFactoryCore.sol	5bd4194ace27430f8b851bd5c27ceb93d897834ca63215485d78bb9c6126bb13
TFC	TrustFiStakingFactoryCreator.sol	2f8c6d2cc16e09e24567d9e933931678a2e3897db02fbc57c10135e354117bbc

Findings



ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	Centralization / Privilege	Major	Acknowledged
TFA-01	Potential Reentrancy Attack	Logical Issue	Minor	Resolved
TFC-01	Missing Input Validation	Volatile Code	Informational	Resolved
TFF-01	Missing Update poolStakeInfo.lastRewardBlock	Logical Issue	Minor	Resolved
TFF-02	Missing Update the Reward of Pools	Logical Issue	Minor	Resolved
TFF-03	Missing Validation for Pool Status	Logical Issue	Informational	Resolved
TFF-04	Improper Usage of memory	Logical Issue	Medium	Resolved
TFF-05	Usage of Functions	Logical Issue	Informational	Resolved
TFS-01	Incorrect Calculation foruserMaxStakeAmount	Logical Issue	Minor	Resolved
TFS-02	Unused Variable	Coding Style	Informational	Resolved
TFS-03	Missing Validation for Pool Status	Logical Issue	Informational	Resolved

Major

GLOBAL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	Global	Acknowledged

Description

In the contract TrustFiStakingFactory, the role owner has authority over the following functions:

- function transferOwnership (address _owner)
- function setOperateOwner (address user, bool state)

In the contract TrustFiStakingFactory, the role platform has authority over the following functions:

- function withdrawRewards2(uint256[] memory _poolIds, address user)
- function addPool (uint256 range, address stakedToken, uint256[] memory poolParams, addressrewardToken, uint256 rewardTotals, uint256 rewardPerBlocks, string [] memory poolViewParams,address feeAddress, address commissionToken, address [] memory pairs)
- function editPool (uint256 poolId, address stakedToken, uint256[] memory poolParams, addressrewardToken, uint256 rewardTotals, uint256 rewardPerBlocks, string [] memory poolViewParams,address feeAddress, address [] memory pairs)
- function close Pool (uint256 poolId) function platformSafeTransfer(address token,address to,uint256 amount)
- function setWhiteList (address _super, bool _state)

In the contract TrustFiStakingFactoryCreator, the role admin has authority over the following functions:

- function transferOwnership (address _admin)
- function setOperateOwner (address user, bool state)
- function setFinanceOwner (address user, bool state)
- function setPoolFactoryTemplate (TrustFiStakingFactory _newTemplate)
- function setCoreTemplate (address _newCore)

In the contract TrustFiStakingFactoryCreator, the role operateOwner has authority over the followingfunctions:

- function setSupportCommTokens (address _token, uint256 _poolCommAmount, uint256_editFeeValue, uint256 _closeFeeValue, bool _state)
- function setName (address _creator, uint256 _poolId, string memory _name)
- function setPriority (address _creator, uint256 _poolId, uint256 _priority)
- function setWhiteList (address _super, bool _state)
- function setUnStakeFeePercent (uint256 _unStakeFeePercent)

In the contract TrustFiStakingFactoryCreator, the role financeOwner has authority over the followingfunctions:

- function withdrawal Commission (address _dst)
- function withdrawCommission (address token,address _dst,uint256 amount)

In the contract TrustFiStakingFactory, the role factory has authority over the following functions:

- function stake (uint256 poolId, uint256 amount, address user)
- function addPool (uint256 range, address stakedToken, address feeAddress, uint256[] memorypoolParams, address rewardToken, uint256 rewardTotals, uint256 rewardPerBlocks, string [] memorypoolViewParams, address commissionToken, address [] memory pairs)
- function editPool (uint256 poolId, address stakedToken, address feeAddress, uint256[] memorypoolParams, address rewardToken, uint256 rewardTotals, uint256 rewardPerBlocks, string [] memorypoolViewParams, address [] memory pairs)
- function closePool (uint256 poolId) function platformSafeTransfer (address token,address to,uint256 amount)

- function setRewardPerBlock (uint256 poolId, address token, uint256 rewardPerBlock)
- function setRewardTotal (uint256 poolId, address token, uint256 rewardTotal)
- function setMaxStakeAmount (uint256 poolId, uint256 maxStakeAmount)
- function _unStake (uint256 poolId, uint256 amount, address user)
- function _withdrawReward (uint256 poolId, address user)

In the contract TrustFiStakingFactoryCore, the role platform has authority over the following functions:

- function setName (uint256 poolId, string memory name)
- function setPriority (uint256 poolId, uint256 priority)

Any compromise to these owner/platform/factory/financeOwner/operateOwner accounts may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privilege account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also 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 key management 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 key compromised; 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.

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.

Alleviation

[Team]: We allow our clients to customize and use our products, but as contract builder we have no right to interfere with the content created by each client. This is a web3.0 system, and every client has the right to create and modify his own pools in the contract before its launch. The modification of the clients using our product can be modified or closed before the user staking tokens, so there is no risk problem

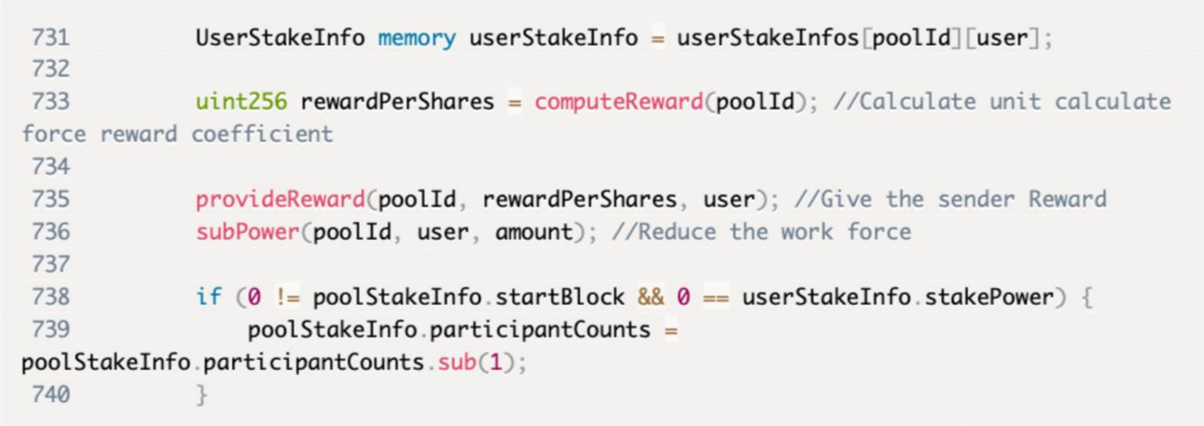
Medium

TFF-04 | Improper Usage of memory

Category	Severity	Location	Status
Logical Issue	Medium	TrustFiStakingFactoryCore.sol: 731	Resolved

Description

When executing subPower(), the stakePower of userStakeInfo would be updated in L736. In this case, the userStakeInfo should be stored in the storage instead of memory as the memory will not save the modified and will be completely wiped off for the next execution.



```
731         UserStakeInfo memory userStakeInfo = userStakeInfos[poolId][user];
732
733         uint256 rewardPerShares = computeReward(poolId); //Calculate unit calculate
force reward coefficient
734
735         provideReward(poolId, rewardPerShares, user); //Give the sender Reward
736         subPower(poolId, user, amount); //Reduce the work force
737
738         if (0 != poolStakeInfo.startBlock && 0 == userStakeInfo.stakePower) {
739             poolStakeInfo.participantCounts =
poolStakeInfo.participantCounts.sub(1);
740         }
```

Recommendation

We advise the team to apply storage instead of memory in L731 or move L731 to L737.

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

Minor

TFA-01 | Potential Reentrancy Attack

Category	Severity	Location	Status
Logical Issue	Minor	TrustFiStakingFactory.sol TrustFiStakingFactoryCreator.sol	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

- stake () in TrustFiStakingFactory
- unStake () in TrustFiStakingFactory
- unStakes () in TrustFiStakingFactory
- closePool () in TrustFiStakingFactory
- CreatoreditPool () in TrustFiStakingFactory
- CreatoraddPool () in TrustFiStakingFactoryCreator

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin

ReentrancyGuard library - nonReentrant modifier for theaforementioned functions to prevent reentrancy attack.

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFF-01 | Missing Update poolStakeInfo.lastRewardBlock

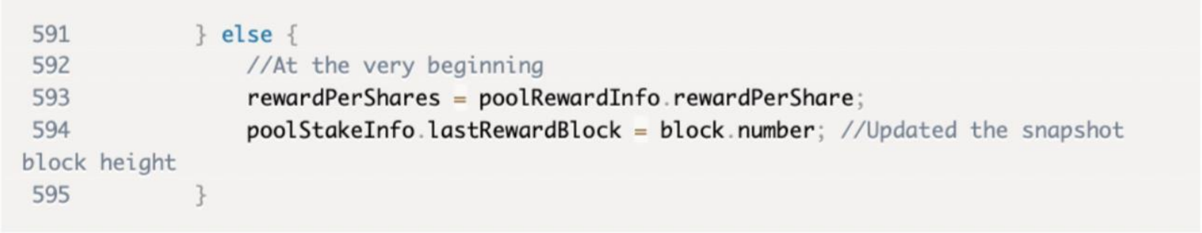
Category	Severity	Location	Status
Logical Issue	Minor	TrustFiStakingFactoryCore.sol: 546	Resolved

Description

Missing update poolStakeInfo.lastRewardBlock when poolStakeInfo.totalPower is zero.

Recommendation

We advise the client to update poolStakeInfo.lastRewardBlock as below:



```
        } else {
            //At the very beginning
            rewardPerShares = poolRewardInfo.rewardPerShare;
            poolStakeInfo.lastRewardBlock = block.number; //Updated the snapshot block height
        }
```

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFS-01 | Incorrect Calculation for userMaxStakeAmount

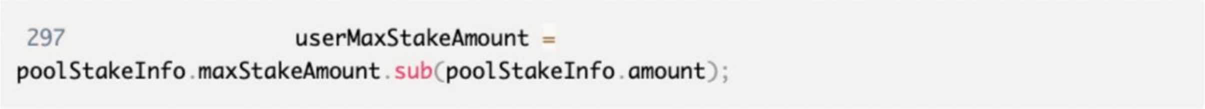
Category	Severity	Location	Status
Logical Issue	Minor	TrustFiStakingFactory.sol: 297	Resolved

Description

When poolStakeInfo.userMaxStakeAmount is zero and poolStakeInfo.maxStakeAmount is greater than 0, userMaxStakeAmount should be poolStakeInfo.maxStakeAmount.sub(poolStakeInfo.amount).

Recommendation

We advise the client to correct as below:



```
297                    userMaxStakeAmount =
poolStakeInfo.maxStakeAmount.sub(poolStakeInfo.amount);
```

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFF-02 | Missing Update the Reward of Pools

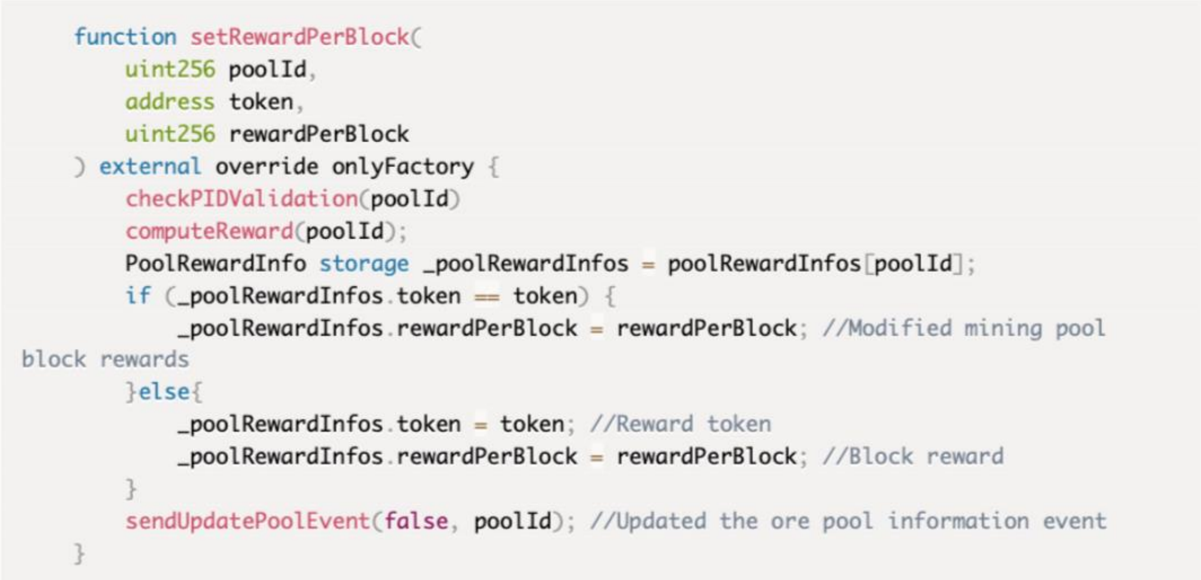
Category	Severity	Location	Status
Logical Issue	Minor	TrustFiStakingFactoryCore.sol: 496	Resolved

Description

When updating the rewardPerBlock for poolId, missing update the reward of the pool.

Recommendation

We advise the client to update reward as below:



```
    function setRewardPerBlock(
        uint256 poolId,
        address token,
        uint256 rewardPerBlock
    ) external override onlyFactory {
        checkPIDValidation(poolId)
        computeReward(poolId);
        PoolRewardInfo storage _poolRewardInfos = poolRewardInfos[poolId];
        if (_poolRewardInfos.token == token) {
            _poolRewardInfos.rewardPerBlock = rewardPerBlock; //Modified mining pool
block rewards
        }else{
            _poolRewardInfos.token = token; //Reward token
            _poolRewardInfos.rewardPerBlock = rewardPerBlock; //Block reward
        }
        sendUpdatePoolEvent(false, poolId); //Updated the ore pool information event
    }
```

Alleviation

The development team removed this function in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

Informational

TFC-01 | Missing Input Validation

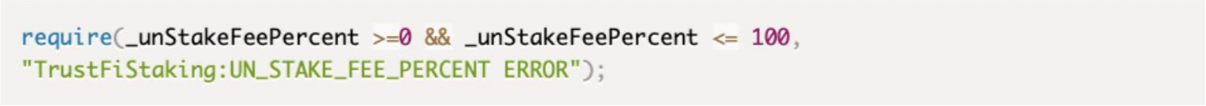
Category	Severity	Location	Status
Volatile Code	Informational	TrustFiStakingFactoryCreator.sol: 86, 200, 290	Resolved

Description

The given input is missing the sanity check.

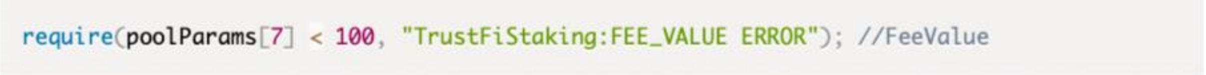
Recommendation

We advise adding the check for the passed-in values to prevent unexpected error as below:constructor():



```
require(_unStakeFeePercent >=0 && _unStakeFeePercent <= 100,
"TrustFiStaking:UN_STAKE_FEE_PERCENT ERROR");
```

createPool()/editPool():



```
require(poolParams[7] < 100, "TrustFiStaking:FEE_VALUE ERROR"); //FeeValue
```

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFF-03 | Missing Validation for Pool Status

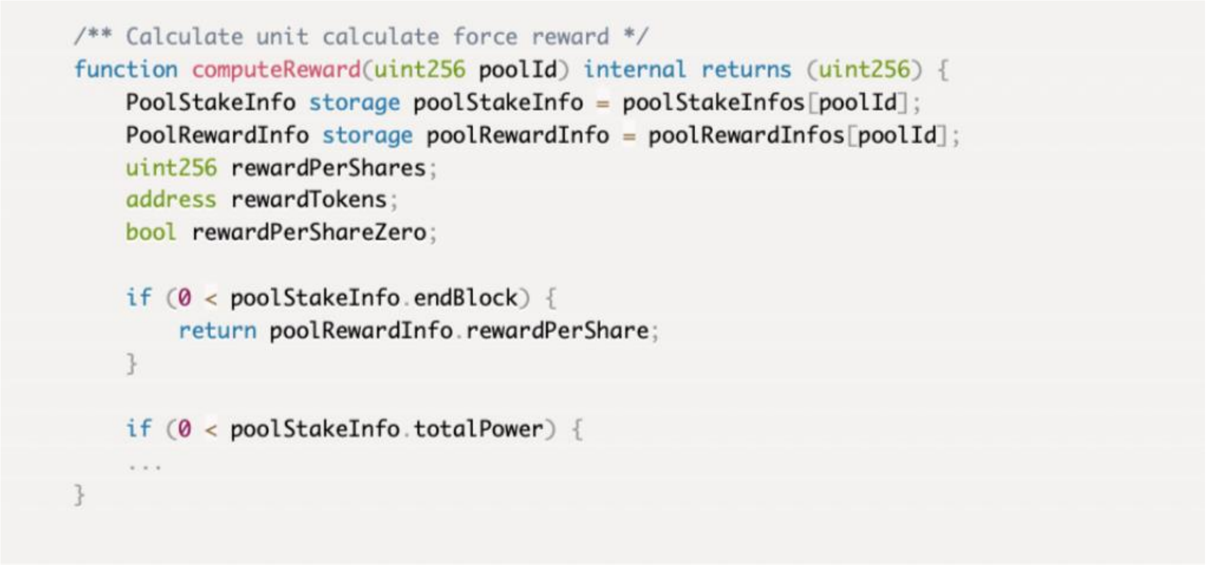
Category	Severity	Location	Status
Logical Issue	Informational	TrustFiStakingFactoryCore.sol: 546	Resolved

Description

There is no validation to check whether the pool has been closed.

Recommendation

We advise the client to add validtion as below:



```
/** Calculate unit calculate force reward */
function computeReward(uint256 poolId) internal returns (uint256) {
    PoolStakeInfo storage poolStakeInfo = poolStakeInfos[poolId];
    PoolRewardInfo storage poolRewardInfo = poolRewardInfos[poolId];
    uint256 rewardPerShares;
    address rewardTokens;
    bool rewardPerShareZero;

    if (0 < poolStakeInfo.endBlock) {
        return poolRewardInfo.rewardPerShare;
    }

    if (0 < poolStakeInfo.totalPower) {
    ...
}
```

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFF-05 | Usage of Functions

Category	Severity	Location	Status
Logical Issue	Informational	TrustFiStakingFactoryCore.sol: 496, 513, 535	Resolved

Description

These functions setRewardPerBlock(), setRewardTotal() and setMaxStakeAmount() only can be called by the factory. According to the logic, the factory is contract TrustFiStakingFactory. But they are not becalled in TrustFiStakingFactory. Do these functions need to be called in the factory? If so, add them to the factory. If not, remove these functions.

Recommendation

We advise the client to remove these functions.

Alleviation

The development team removed these functions in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFS-02 | Unused Variable

Category	Severity	Location	Status
Coding Style	Informational	TrustFiStakingFactory.sol: 24	Resolved

Description

The variable lastSetRewardPerBlockTime in L24 is declared but never used or updated.

Recommendation

We advised the client to remove the unused variable if it is not intended to be used.

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

TFS-03 | Missing Validation for Pool Status

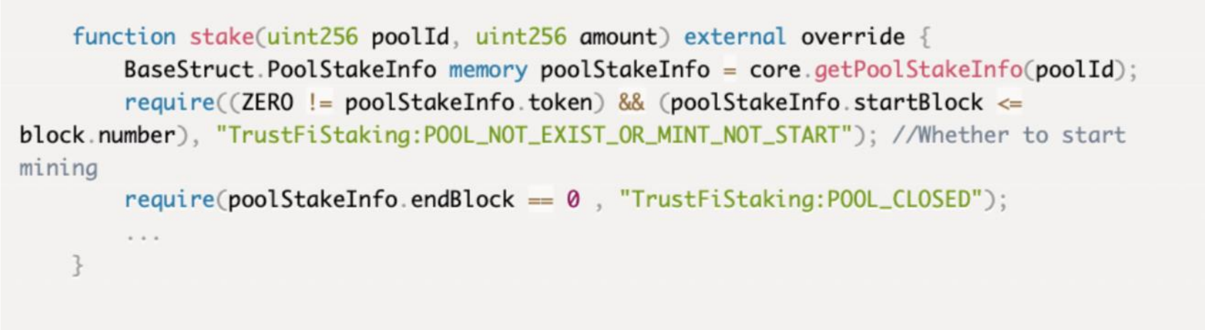
Category	Severity	Location	Status
Logical Issue	Informational	TrustFiStakingFactory.sol: 111	Resolved

Description

There is no validation to check whether the pool has been closed.

Recommendation

We advise the client to add validation as below:



```
    function stake(uint256 poolId, uint256 amount) external override {
        BaseStruct.PoolStakeInfo memory poolStakeInfo = core.getPoolStakeInfo(poolId);
        require((ZERO != poolStakeInfo.token) && (poolStakeInfo.startBlock <=
block.number), "TrustFiStaking:POOL_NOT_EXIST_OR_MINT_NOT_START"); //Whether to start
mining
        require(poolStakeInfo.endBlock == 0 , "TrustFiStaking:POOL_CLOSED");
        ...
    }
```

Alleviation

The development team resolved this issue in commit f2f0bdc34387e0136df2e82cde84fe05b583f373.

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 ORMATERIALS, 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
- Eos
- Cosmos
- Substrate

TECH STACK

- Python
- Rust
- Solidity
- c++

CONTACTS

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

DeHacker

March 2022