CERTIK

Security Assessment

Filet Staking

Jun 10th, 2021

Table of Contents

Summary

This report has been prepared for Filet Staking smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Additionally, this audit is based on a premise that all external smart contracts are implemented safely.

The security assessment resulted in 14 findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Filet Staking
Description	Decentralized mining activity.
Platform	BSC, Heco
Language	Solidity
Codebase	https://github.com/fltproject/Filet/tree/9adb9f79d17ed6001e2e0992aa37dad607a10a39 https://github.com/fltproject/Filet/tree/8c8a8ea503aa67649c03aa21df60df42749e8639
Commit	<9adb9f79d17ed6001e2e0992aa37dad607a10a39> <8c8a8ea503aa67649c03aa21df60df42749e8639>

Audit Summary

Delivery Date	Jun 10, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Total Issues	14
● Critical	0
● Major	0
● Medium	0
● Minor	5
● Informational	9
● Discussion	0

Audit Scope

ID	file	SHA256 Checksum
SCF	StakingCon.sol	c8cf78b706c79107afb3516d8ebbbce7c5179c1de34f2cd5f0fc6fed390525ba

Understandings

Overview

The Filet protocol is a decentralized mining activity deployed on the Binance smart chain(BSC) and Heco smart chain(Heco). Filet employs a novel feature in its protocol.

Users can stake some form of tokens to get the mining power of `FIL`. Filet will distribute mined profit to users based on their mining power. But when users would like to `redeem` from the contract in advance, they must pay the aforementioned `Fee`. Users will not lose any staking asset since this `Fee` will not exceed their profit.

Privileged Functions

The project contains the following privileged functions that are restricted by some modifiers. They are used to modify the contract configurations and address attributes. We grouped these functions below:

The `onlyOwner` modifier:

Contract `StakingCon`:

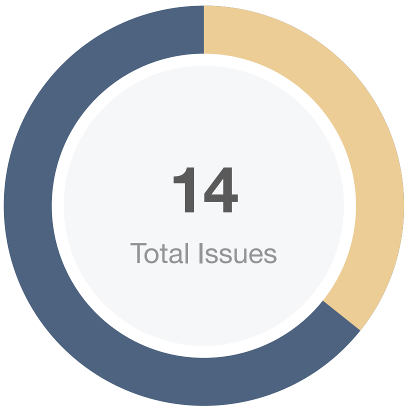
```
function setAdmin(address newAdminUser)
function transferOwnership(address newOwner)
```

The `ownerAndAdmin` modifier:

Contract `StakingCon`:

```
function switchOnContract(bool op)
function updateMinePool(
    updateMineInput memory updateParas,
    uint256[] memory poolThredhold,
    uint[] memory serviceFeePercent
)
function updateOrderFee(updateUserOrderType[] memory updateOrders)
function addFLTTokenContract(address fltToken)
function addFILTokenContract(address filTokenCon)
```

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**0** (0.00%)
🟨	**Medium**	**0** (0.00%)
🟨	**Minor**	**5** (35.71%)
🟦	**Informational**	**9** (64.29%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
SCF-01	State Variables Could Be Declared Constant	Language Specific	● Informational	✓ Resolved
SCF-02	Remove Redundant State Variables And Modifier	Gas Optimization	● Informational	✓ Resolved
SCF-03	Missing Emit Events	Coding Style	● Informational	✓ Resolved
SCF-04	Proper Usage Of `view` And `pure` Type	Gas Optimization	● Informational	✓ Resolved
SCF-05	Missing Zero Address Validation	Logical Issue	● Informational	✓ Resolved
SCF-06	Proper Usage of `public` And `external` Type	Gas Optimization	● Informational	✓ Resolved
SCF-07	Unreasonable Restrictions	**Centralization / Privilege**	● **Minor**	✓ **Resolved**
SCF-08	Divide Before Multiply	Mathematical Operations	● Minor	✓ Resolved
SCF-09	Check If Order Is Exists	Logical Issue	● Informational	✓ Resolved
SCF-10	Redundant Check In `redeem()`	Logical Issue	● Informational	✓ Resolved
SCF-11	Incorrect Assignment Of `stopDayTime`	Logical Issue	● Minor	✓ Resolved
SCF-12	Optimization For `checkisPremium()`	Gas Optimization	● Informational	✓ Resolved
SCF-13	Fee Need to Pay When Early Redemption	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**

ID	Title	Category	Severity	Status
SCF-14	Potential Cross-chain Data Integrity Issue	Centralization / Privilege	● Minor	ⓘ Acknowledged

SCF-01 | State Variables Could Be Declared Constant

Category	Severity	Location	Status
Language Specific	● Informational	StakingCon.sol: 13~16	✓ Resolved

Description

The variables that are not modified throughout the contract should be declared as `constant` variables.

Recommendation

We advise the client to consider modifying the variable `secondsForOneDay` and `timeZoneDiff` as following:

```
//一天的秒数
uint private constant secondsForOneDay = 86400;

//时区调整
uint private constant timeZoneDiff = 28800;
```

Alleviation

The team heeded our advice and added `constant` attribute. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-02 | Remove Redundant State Variables And Modifier

Category	Severity	Location	Status
Gas Optimization	● Informational	StakingCon.sol: 28, 71~79, 183~190	✓ Resolved

Description

The state variables `StakingCon._cfltTokenContract`, `StakingCon.minerInfoList`, `StakingCon.minerInterest`, and modifier `onlyAdmin()` are never used in contract `StakingCon.sol`.

Recommendation

We advise the client to consider removing the redundant state variables and modifier.

Alleviation

The team heeded our advice and removed the unused variables and modifier. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-03 | Missing Emit Events

Category	Severity	Location	Status
Coding Style	● Informational	StakingCon.sol: 203, 208, 550, 588, 594, 600, 633	✓ Resolved

Description

The functions that affect the status of sensitive variables should be able to emit events as notifications to customers.

- `setAdmin()`
- `swithOnContract()`
- `updateMinePool()`
- `updateOrderFee()`
- `addFLTTokenContract()`
- `addFILTokenContract()`
- `minerRetrieveToken()`

Recommendation

We advise the client to consider adding events for sensitive actions and emit them in the functions.

```
event SetAdmin(address indexed user, address indexed _admin);

function setAdmin(address newAdminUser) external onlyOwner{
    _admin = newAdminUser;
    emit SetAdmin(msg.sender, _admin);
}
```

Alleviation

The team heeded our advice and added events. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-04 | Proper Usage Of `view` And `pure` Type

Category	Severity	Location	Status
Gas Optimization	● Informational	StakingCon.sol: 666, 672, 686	✓ Resolved

Description

The functions that are not written to the storage of the smart contract can be defined as the `view` function. If no read or write happen in the function, this function can be defined as a `pure` function.

Recommendation

We advise the client to consider modifying as demonstrated below:

```
function convertToDayTime(uint forConvertTime) internal view returns (uint){
    ...
}

function checkisPremium(uint256 amount,uint256[] memory levelThredhold) internal pure returns (uint){
    ...
}

function convertTokenToPower(uint256 amount, uint poolID) internal view returns (uint256){
    ...
}
```

Alleviation

The team heeded our advice and added attribute `view` or `pure` to the functions. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-05 | Missing Zero Address Validation

Category	Severity	Location	Status
Logical Issue	● Informational	StakingCon.sol: 203, 593, 599	✓ Resolved

Description

The assigned values to `_admin`, `_fltTokenContract` and `_filTokenContract` should be verified as non-zero values to prevent being mistakenly assigned as `address(0)` in the `setAdmin()`, `addFLTTokenContract()` and `addFILTokenContract()` functions respectively.

Recommendation

We advise the client to add input validators to guarantee the addresses are not zero in `setAdmin()`, `addFLTTokenContract()` and `addFILTokenContract()` functions as demonstrated below.

```
require(newAdminUser != address(0), "newAdminUser is a zero address");
```

Alleviation

The team heeded our advice and added a zero check. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-06 | Proper Usage of `public` And `external` Type

Category	Severity	Location	Status
Gas Optimization	● Informational	StakingCon.sol: 212, 221, 307, 386, 430, 563, 592, 598, 618	✓ Resolved

Description

`public` functions that are never called by the contract could be declared `external`.

Recommendation

We advise the client to consider using the `external` attribute for functions never called from the contract.

Alleviation

The team heeded our advice and changed attribute `public` to `external`. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-07 | Unreasonable Restrictions

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	StakingCon.sol: 307, 386	✓ **Resolved**

Description

Functions `redeem()` and `getProfit()` are decorated by modifier `swithOn` in which the value of `_swithOn` can be decided by calling function `swithOnContract()` by `_owner` or `_admin`. Once users stake in the `minePool`, if the `_swithOn` is set to `false`, users can not redeem or collect profit.

Recommendation

We advise the client to make sure users will not be blocked by `swithOn` when they redeem and collect profit.

Alleviation

The team had removed the modifier. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-08 | Divide Before Multiply

Category	Severity	Location	Status
Mathematical Operations	● Minor	StakingCon.sol: 688	✓ Resolved

Description

According to the logic of function `StakingCon.convertTokenToPower()`, `amount` divides`10**minePoolMap[poolID].mPool.tokenPrecision` before engaging in multiplication. In this case `amount` may lose accuracy. If the value of `amount` is less than `10**minePoolMap[poolID].mPool.tokenPrecision`, it will return 0.

Recommendation

We advise the client to consider ordering multiplication before division to prevent any loss of arithmetical operation accuracy.

Alleviation

The team heeded our advice and ordered multiplication before division. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-09 | Check If Order Is Exists

Category	Severity	Location	Status
Logical Issue	● Informational	StakingCon.sol: 311~312, 387~388	✓ Resolved

Description

Before using `userOrder`, it is better to check if `userOrder` exists.

Recommendation

We advise the client to consider adding a require check in functions `redeem()` and `getProfit()` as demonstrated below:

```
require(userData[msg.sender][orderID].user!=address(0));
```

Alleviation

The team heeded our advice and added a require check for `userOrder`. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-10 | Redundant Check In `redeem()`

Category	Severity	Location	Status
Logical Issue	● Informational	StakingCon.sol: 340, 344	✓ Resolved

Description

In line 344, the validation that `curSubDayTime >= minePoolMap[uOrder.poolID].mPool.lockInterval` had already been executed in line 340.

Recommendation

We advise the client to consider modifying the codes as demonstrated below:

```
if(curSubDayTime < minePoolMap[uOrder.poolID].mPool.expireType){
    ...
} else {
    ...
}
```

Alleviation

The team heeded our advice and changed the if and else statements. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-11 | Incorrect Assignment Of `stopDayTime`

Category	Severity	Location	Status
Logical Issue	● Minor	StakingCon.sol: 355, 367	✓ Resolved

Description

In line 355, the variable `stopDayTime` is assigned with the value of `curSubDayTime`, which is the number of days between `curDayTime` and `userCreateDayTime`, like `30 days`. But in line 367, the `stopDayTime` is assigned with the value of `curDayTime` which represents a certain day like the value of `10.4.2021`. According to the logic, `stopDayTime` should be assigned with a value of a certain day.

Recommendation

We advise the client to consider assigning `stopDayTime` with the value of `curDayTime` in line 355.

Alleviation

The team heeded our advice and corrected the assignment. Code change was applied in commit : 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-12 | Optimization For `checkisPremium()`

Category	Severity	Location	Status
Gas Optimization	● Informational	StakingCon.sol: 674~681	✓ Resolved

Description

Based on the observation of the logic of the smart contract, the values of the members in the array `levelThredhold` are stored in ascending order. In the function `checkisPremium()`, the variable `isPrem` was assigned multiple times. If traversed in reverse order, the variable `isPrem` can be assigned only once, which will save gas cost for the project.

Recommendation

We advise the client to consider modifying as below:

```
uint isPrem = 0;
for (uint i = levelThredhold.length - 1 ; i >= 0 ; i--){
    if (amount >= levelThredhold[i]){
        isPrem = i;
        break;
    }
}
return isPrem;
```

Alleviation

The team heeded our advice and changed the traversing order. Code change was applied in commit: 801e07e1338d9ffd063e87fb2d4debcb5728e376.

SCF-13 | Fee Need to Pay When Early Redemption

Category	Severity	Location	Status
Centralization / Privilege	● Minor	StakingCon.sol: 366~374	ⓘ Acknowledged

Description

When users would like to `redeem` from the contract in advance, they need to pay the `Fee` which is decided by `_owner` or `_admin`. If the `Fee` is greater than the profit the users will get back their adjusted principal.

Recommendation

We advise the client to carefully manage the private keys of `_owner` role's and `_admin` role's accounts and avoid any potential risks of being hacked. In our professional opinion, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to a private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement.

Alleviation

The team response: After users staked `FIL`, miners need to invest a certain amount to start packaging. If the user redeems in advance after the packaging is completed, they need to bear a certain handling fee, and there is no handling fee for the redemption at the time of expiration. Deciding to pay the handling fee for early redemption is entirely up to the end user. The earlier you redeem, the higher the handling fee. The amount of handling fee is relative to the packaging cost and maintenance cost (i.e., logistics, cost of operation, etc.). The amount of the fee will not be more than the user's revenue. A such users' principal will not get lost. The relevant data can be queried from Filecoin browsers such as Filfox or Filscan as a reference, so the project party cannot set Fee's arbitrarily.

SCF-14 | Potential Cross-chain Data Integrity Issue

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	StakingCon.sol: 563	ⓘ **Acknowledged**

Description

According to the cross-chain design of the project, the data will transfer between HECO/BSC chain and the Filecoin chain. This must be done in a centralized cross-chain implementation. The owner of the cross-chain implementation has the responsibility and privilege to handle the process of cross-chain data transfers. Therefore, a centralized threat concern may be raised by the community on how data integrity can be guaranteed in this implementation.

Recommendation

We strongly advise the client to make more efforts on improving transparency on the cross-chain transaction process. For example, the client can opt to show more details about the miner's status on Filecoin chain as currently they only address the miners and the distribution rate for the minter which can be observed in Filscan.

Alleviation

The team response: The profit is generated on the Filecoin chain, and the profit distribution is on the HECO/BSC chain, so `Filet` needs to transfer `FIL` across the chain by `_owner` or `_admin`. Users can query the total income and unit computing power income data of the Filecoin node of their chosen miner on Filecoin browsers such as Filfox or Filscan, and calculate the income allocated by themselves according to their own computing power, thereby verifying the accuracy of the data calculated by Filet.

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.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

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.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

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

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.

Disclaimer

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.