Security Assessment

IFPool

Sep 10th. 2021

CYBERHUNTER

Contents

Introduce **3**
About IFPool 3
About CyberHunter 4
Evaluation methods and vulnerability levels 5
Summary of audit **6**
List of Bugs 7
General Vulnerabilities **8**
No.1 Array out-of-bounds check 8
No.2 TimeStamp may be tampered 9
No.3 Input parameter check 10
No.4 Meaningless value check 11
Business Problems **13**
No.1 Event notification omission 13
Suggestions **14**
No.1 Copy third-party library 14
No.2 Function parameter naming 15
Audit Conclusion **16**

Introduce

We are commissioned by the customer's IFPool R&D team to conduct a comprehensive audit of the smart contract code of the IFPool Staking Protocol.

According to our security audit specifications and customer needs, we will list in the report systematic methods for detecting potential security issues, and according to the test results, give corresponding suggestions or recommendations to repair safety problems or improve safety.

About IFPool

The IFPool project is a decentralized DeFi staking protocol on CoinEx Smart Chain(CSC). IFPool supports users to staking CET token to CSC nodes, so as to obtain the node's output bonus. The user deposit amount is recorded through the smart contract, and the PoS income obtained by the node is distributed according to the share ratio. In addition, using IFPool, the protocol will charges a 10% fee to enter the distribution pool, and users can mine additional IFT . By staking IFT, users can get profit dividends from IFPool.

The IFPool system is written in Solidity language, deployed on the CSC , and runs in an EVM–compatible manner.

Project	Content
Issuer	IFPool R&D Team
Issuing platform	CSC
Programming language	Solidity
Audit method	Code Review
Audit completion time	2021–09–10

Audit code base and contract conditions on the chain:

Contracts list:

名称	合约地址
IFPool	0x633acb5ca22c5851b4278B062AA6B567791F2C5B
IFT Vault	0x918F0ec3d0cdb94e39fCad6dE40365b5f85c699A
IFT Staker	0xDfEcB6584366f3111e930fEf5A3E921896C90d65
IFT Token	0x1D7C98750A47762FA8B45c6E3744aC6704F44698

Code Base：

Code base address：https://github.com/IFWallet/ifpool_staking

File	Commit
IFPool.sol	602caad2d14da30e5965808d297fbb69e8694002
IFTStaker.sol	602caad2d14da30e5965808d297fbb69e8694002
IFTVault.sol	602caad2d14da30e5965808d297fbb69e8694002
Wallet.sol	602caad2d14da30e5965808d297fbb69e8694002
ValidatorInterface.sol	602caad2d14da30e5965808d297fbb69e8694002
ERC20.sol	602caad2d14da30e5965808d297fbb69e8694002

About CyberHunter

CyberHunter is a global-oriented professional audit team founded by Silicon Valley engineer Gessi Kok. Team members have many years of security work experience and have extensive experience in the development process of smart contracts and audit methods. CyberHunter is committed to improving industry security practices, disclosing risks in advance for customers, and providing a full range of security audits, vulnerability tracking and other services for DApp applications. If you have business needs, welcome to contact us.

Evaluation methods and vulnerability levels

In order to test and evaluate the standardization, we define the following terms according to the OWASP Risk Rating Methodology：

• **Possibility**: Indicates the possibility of a particular vulnerability being discovered and exploited;
• **Influence**: Measures the loss caused by a successful attack through a vulnerability;
• **Harmfulness**: show the severity of the harm of the bug;

Possibility and influence are divided into three levels: **High**, **Medium** and **Low**.
The Harmfulness is determined by the possibility and influence, and is divided into four levels: **Severe**, **High-Risk**, **Medium-Risk**, and **Low-Risk**.

For this audit, we will use the following inspection methods:：

- Basic vulnerability detection: First, analyze the smart contract with the self-developed automatic static detection tool, and then manually confirm whether the vulnerability exists.
- Code and business security testing: We conduct further review of business logic, system operation and other related content to discover potential hazards or vulnerabilities.
- Other suggestions: Practitioners has been proved by practice good go to and fro hair practice of view, for the preparation and deployment of smart contract codes give advice or opinions.

The specific test items are as follows：

Detection type	Test items
Basic vulnerability detection	Slither static tool scan
	Familiar with citation checking
	Boundary condition detection
	Permission check
	Reentrancy check
	Transaction Block Attack
	Transaction Rollback Attack
	Fake Transfer Attack

	Overflows & Underflows
	Costly Loop
Business model checking	Construct test cases and conduct code coverage testing
	Vault Asset Security checking
	Business Logic Bugs checking
	Roles Management checking
	Emergency Stop checking
	Deployment Consistency
Other suggestions	Solidity Coding Style
	Engineering structure suggestion

Summary of audit

The audit results are summarized as follows:

Severity	Number of vulnerabilities
serious	0
high risk	0
Medium risk	1
low risk	5
Suggest	0
Total	6

List of Bugs

Serial number	Description	Security Level	Status
No.1	Array out of bounds check	Medium risk	Confirmed
No.2	TimeStamp may be tampered	Low risk	Confirmed
No.3	Input parameter check	Low risk	Confirmed
No.4	Meaningless value check	Low risk	Confirmed
No.5	Omission of event notification	Low risk	Confirmed
No.6	Copy third-party libraries	Low risk	Confirmed

General Vulnerabilities

No.1 Array out-of-bounds check

Security level: medium risk

File: IFTStaker.sol

Detailed description:

setPoolReward function, there is no check to the parameter "_pid" of whether the size of the array poolInfo within the legal size, if it is greater than or equal to the array poolInfo size, it will cause an array of cross-border issues.

```
1.  function setPoolReward(uint256 _pid, uint256 _tokenPerDaily, bool _isOpenReward, bool _isRewardCet, ui
    nt256 _startTime, uint256 _endTime) public onlyOwner {
2.          poolInfo[_pid].tokenPerDaily = _tokenPerDaily;
3.          poolInfo[_pid].isOpenReward = _isOpenReward;
4.          poolInfo[_pid].isRewardCet = _isRewardCet;
5.          poolInfo[_pid].startTime = _startTime;
6.          poolInfo[_pid].endTime = _endTime;
        }
```

Here, when obtaining the pool object, the validity of the following table of the array is not judged . For example, the _pid passed in here is greater than the length of the poolInfo array, which will result in the fetched object being nil, which will cause the corresponding setPool to fail. If there is no clear prompt here, it may cause the management to ignore the error and cause the setting The reward information fails, which in turn causes the player's income data to be abnormal.

Suggestions :

Before operating the subscript of the array poolInfo, use "require" to check whether the _pid is within the legal range.

```
1.              require(_pid<poolInfo.length, “_pid is too large”)
```

No.2 TimeStamp may be tampered

Security level: medium risk

File: IFStaker.sol

Detailed description:

In the function pendingTokenReward, depending the block.timestamp to calculate the user's income, because here timestamp can be set by miners. A malicious attacker, you can set this up to the value 900s greater than real value. If the attacker's staking position is big enough, the 900s may have a huge difference in revenue.

```
1.  uint256 remain = 0;
2.  uint256 poolReward = tokenRewardPerSecondForPool(_pid).mul(getTimeCount(_pid, user.lastHarvestTime
     block.timestamp));
    uint256 userReward = user.amount.mul(100).div(pool.amount).mul(poolReward).div(100);
```

The poolReward here depends on the time of the user's last operation and the timestamp of the current block. If the block generator modifies the timestamp, it will cause the revenue to be magnified.

Ethereum network timestamp reasonable requirements:

- current block timestamp must be greater than the timestamp of last block
- current block should not greater than last block's timestamp + 900s
- miners can arbitrarily set the timestamp within the "reasonable" range

This leads to an unstable dependency on block.timestamp.

Suggestions:

You can use block.number to do replacement. Or when the overall revenue has little impact in the 900s, it does not cause much substantial impact.

No.3 Input parameter check

Security level: medium risk

File: IFPool.sol

Detailed description:

In setIftBonusRatio function, there is no check for whether _iftBonusRatio is in the range. Here IFT dividend is in accordance with the denominator 10000 to be calculated, if _iftBonusRatio greater than 10000 will cause the incoming currency is not enough.

```
1.  function setIftBonusRatio(uint256 _iftBonusRatio) public onlyOwner {
2.      iftBonusRatio = _iftBonusRatio;
3.  }
```

Here the code and no range judgement for incoming _iftBonusRatio parameter, Once manager set it wrong, according to the logic of contracts, as long as this is not equal to 10000 will lead to the relevant calculation error. like：

```
iftBonusAmount = iftPoolRewardAmount.mul(iftBonusRatio).div(ratioDivBase);
```

The bonus amount here is calculated by using the iftBonusRatio as the denominator to calculate a ratio. If it setup wrong, the ratio will be incorrect.

Suggestions :

Use require of _iftBonusRatio parameter range determined.

No.4 Meaningless value check

Security level: low risk

File: IFPool.sol

Detailed description:

setValidatorInfo function, did not check _rewardType whether 0, as enumerated RewardType type contains 0,1,2 three values, when strong convert, it may result in 3 values. If the result is NONE, it will cause other logic to fail to handle the None branch.

```
1.  function setValidatorInfo(address _validator, uint256 _rewardType, bool _enabled) public onlyOwner {
2.    ValidatorInfo storage info = validatorInfos[_validator];
3.    info.rewardType = RewardType(_rewardType);
4.    info.enabled = _enabled;
5.  }
```

_rewardType do type casting, and conversion type definition:

```
1.  enum RewardType {
2.        NONE,                  // No use, just for padding 0th of enum
3.        ONLY_CET,              // Only redistribute CET. E.g for cetfans validator
4.        IFT_BONUS              // Redistribute CET with IFT as bonous. E.g for IF validator
5.     }
```

NONE value is 0, if the incoming parameter is 0, then the corresponding is obtained None value. In the subsequent logic:

```
1.  if (validatorInfo.rewardType == RewardType.IFT_BONUS) {
2.      // Revert if CET balance not enough to withdraw
3.      payable(msg.sender).transfer(rewardAmount);
4.      payable(iftPoolAddress).transfer(iftPoolRewardAmount);
5.      totalRewardAmount = totalRewardAmount.add(rewardAmount.add(iftPoolRewardAmount));
6.
7.    } else if (validatorInfo.rewardType == RewardType.ONLY_CET) {
8.      Wallet wallet = wallets[msg.sender];
9.      wallet.withdraw(msg.sender, rewardAmount);
10.     wallet.withdraw(iftPoolAddress, iftPoolRewardAmount);
11.   }
```

Only two types of values were judged, and no else branch for rest conditions.

Suggestions :

Since None is a reserved meaningless parameter, try to avoid its existence when passing in parameters. Otherwise, it is necessary to ensure that the processing of this branch is added to all logic processing.

Business Problems

No.1 Event notification omission

Security level: Low risk

File: IFTVault.sol

Detailed description:

In IFTVault's withdraw function, when the user withdraw pledges, the corresponding Withdraw event was not sent, but the equivalent transferTo have sent.

```
1.  function withdraw(address _address, uint256 _amount) public onlyRole(TRANSFER_ROLE) {
2.      require(_amount <= address(this).balance, "amount not enough");
3.      payable(_address).transfer(_amount);
4.  }
```

After the transfer is completed, there is no event notification corresponding to emit, and the same, The transfer of ERC20 tokens does:

```
1.  function transferTo(address _address, uint256 _amount) public onlyRole(TRANSFER_ROLE) {
2.      require(amount >= _amount, "amount not enough");
3.
4.      amount = amount.sub(_amount);
5.      token.safeTransfer(_address, _amount);
6.      emit Withdraw(_address, _amount);
    }
```

Here as an ordinary tokens withdraw while the former is the native currency withdraw, in order to unify the performance, we should have sent the corresponding event notification.

Suggestions:

From the perspective of consistency, the two functions correspond to Withdraw ordinary tokens and Native tokens, and both should send this event.

Suggestions

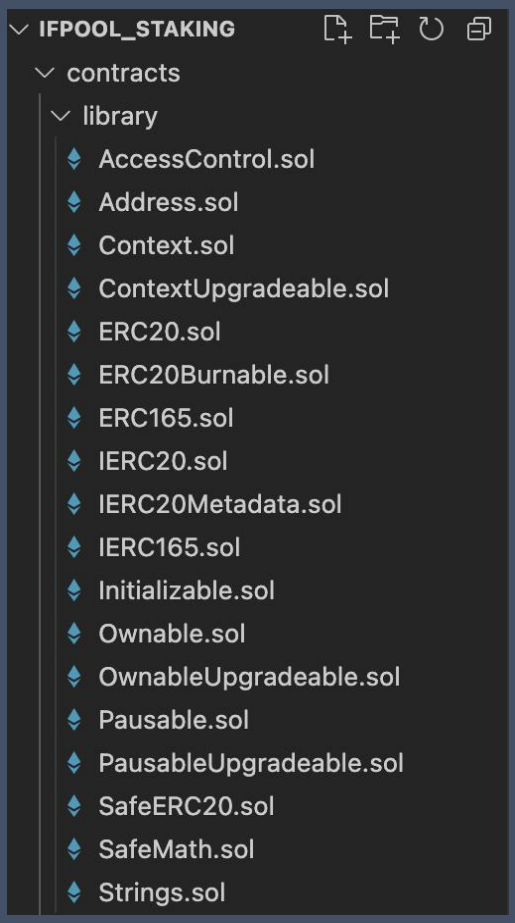
No.1 Copy third-party library

Security level: Low risk

File: library

Detailed description:

The library directory in the current project is the referenced third-party library code. This method of directly copying the third-party library files into the project may cause the library files to be updated untimely, and the bugfix of the upstream project cannot be tracked in time.



Suggestions :

Use npm packages to integrate third-party libraries and track upstream updates in real time.

No.2 Function parameter naming

Security level: Low risk

Files: All files

Detailed description :

All function parameters in the code have a prefix of _:

```
1.  // Add a new pool. Can only be called by the owner.
2.    function addPool(
3.        IERC20  _lpToken,
4.        uint256 _allocPointBase100
      ) public onlyOwner
```

Underline means internal variables, should not use as parameter.

Suggestion:

In order to be easy to read and understand, it is more appropriate to use the camel case naming method according to the Solidity Style suggestion.

Naming Styles

To avoid confusion, the following names will be used to refer to different naming styles.

- `b` (single lowercase letter)
- `B` (single uppercase letter)
- `lowercase`
- `lower_case_with_underscores`
- `UPPERCASE`
- `UPPER_CASE_WITH_UNDERSCORES`
- `CapitalizedWords` (or CapWords)
- `mixedCase` (differs from CapitalizedWords by initial lowercase character!)
- `Capitalized_Words_With_Underscores`

Audit Conclusion

According to the audit of the IFPool project contract code, the above 6 risky issues were discovered and confirmed with the IFPool R&D Team. It has avoided issues such as judgment of reward types and overflow through related configurations, and the damage of the loopholes is limited. Other issues outside of the configuration, such as the vulnerability that the timestamp can be tampered with within the scope of the consensus, will be completely repaired by the IFPool R&D Team by arranging contract upgrades. We will bring the above issues to the attention of project team or users until the review is passed again after the repair.

This audit report represents our audit opinions and is open to the public, and constructive feedback or comments are welcome. This audit only represents our professional advice and delivery responsibilities. For events in the actual operation of the project, the project team must bear corresponding responsibilities, and we are exempted from related responsibilities.