Guard

Smart contracts security assessment

Final report

Tariff: Standard

Pcap

September 2024

0xguard.com

hello@0xguard.com

Contents

Introduction

The report has been prepared for **Pcap**.

The Pcap project is pepresented by a mintable token PcapToken, a liquidity lock token StockToken, and farming strategies using MasterChef contracts.

The md5 sum of the files under review:

1e8356b60cbf69c6b8e6c6811ef6fdeb Farms.sol

b897a3dc6217cc958228e7460d48873d PcapToken.sol

4ec088b8a1c8e46a877f0644692e90fc RhFarms.sol

21bf91f8fe915f5aa99e44df36fbc35e StockPool.sol

e83e639c1372903199682fc5d62cde66 StockToken.sol

6659ce131dff9fcb6b33778f51de332b Zapper.sol

Report Update

The contracts code was updated according to this report.

The md5 sum of the updated files:

dd2f082ad2b5623c1ccb083117a6eb93 Farms.sol

6b6ba1853cc745db579e26686313b264 PcapToken.sol

f3686d731a6bdc2faac01f40af016c67 RhFarms.sol

128d6aa5d3e894b5c3b2ee2e2a18cd4c StockPool.sol

142c27a089bfd047b4ae2886beeb290b StockToken.sol

224e39d2b5da61f4ff2d35ff73d9a8c9 Zapper.sol

Report Update 2

The contracts code was updated according to this report.

The md5 sum of the updated files:

48ce7c9093a340d791ba9c34ff115d50 Farms.sol

6b6ba1853cc745db579e26686313b264 PcapToken.sol

5e19cee0acf0f356e56baa56870e1e8e RhFarms.sol

f6601eb24b4f25a938ab9ea9371e47a1 StockPool.sol

142c27a089bfd047b4ae2886beeb290b StockToken.sol

30dc51d28236247f9aaa7add90e8ae75 Zapper.sol

Name	Pcap
Audit date	2024-09-05 - 2024-09-10
Language	Solidity
Platform	Binance Smart Chain

Contracts checked

Name	Address
PcapToken	
StockToken	
Farms (MasterChef)	

RhFarms (MasterChef)

StockPool

Zapper

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Manually analyze smart contracts for security vulnerabilities
- Smart contracts' logic check

Known vulnerabilities checked

Title	Check result
Unencrypted Private Data On-Chain	passed
Code With No Effects	passed
Message call with hardcoded gas amount	passed
Typographical Error	passed
DoS With Block Gas Limit	passed
Presence of unused variables	passed
Incorrect Inheritance Order	passed
Requirement Violation	passed

Weak Sources of Randomness from Chain Attributes	passed
Shadowing State Variables	passed
Incorrect Constructor Name	passed
Block values as a proxy for time	passed
Authorization through tx.origin	passed
DoS with Failed Call	passed
Delegatecall to Untrusted Callee	passed
Use of Deprecated Solidity Functions	passed
Assert Violation	passed
State Variable Default Visibility	passed
Reentrancy	passed
Unprotected SELFDESTRUCT Instruction	passed
Unprotected Ether Withdrawal	passed
Unchecked Call Return Value	passed
Floating Pragma	passed
Outdated Compiler Version	not passed
Integer Overflow and Underflow	passed
Function Default Visibility	passed

Classification of issue severity

High severity High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.

Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

1. Using transferFrom() without approve (StockToken)

Status: Fixed

In the `rehyph()` modifier, the `transferFrom()` function is called without a preceding `approve()` function call. This approach will block the execution of the modifier and the associated functions.

Recommendation: We recommend using the `safeTransfer()` function on L57.

2. Validation of `slippage` (StockToken)

Status: Fixed

Low `slippage` values can result in liquidity addition being blocked, causing the transaction to fail. This can lead to the following scenario: a user deposits their tokens by calling the `mint()` function. Subsequently, the contract owner significantly reduces the `slippage` value (or current value is not enough), which blocks the user from withdrawing their funds. This is because when the `burnLocked()` or `burnUnlocked()` functions are called, an attempt to add liquidity will be made, which will fail due to the low `slippage` setting.

Recommendation: We recommend preemptively increasing the slippage and incorporating validation in the `updateSlippage()` function to prevent setting excessively low values.

3. Using transferFrom() without approve (Farms (MasterChef))

Status: Fixed

In the `rehyph()` modifier, the `transferFrom()` function is called without a preceding `approve()` function call. This approach will block the execution of the modifier and the associated functions.

Recommendation: We recommend using the `safeTransfer()` function on L196.

4. Wrong function implementations (Farms (MasterChef))

Status: Fixed

The contract cannot be successfully compiled because the functions `getMultiplier()`, `setDevAddress()` have wrong implementation.

```
    function setDevAddress(address _devaddr) public onlyOwner {
        require(_devaddr != address(0), "Invalid address");
        totalAllocPoint =
totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
         return _to.sub(_from).mul(BONUS_MULTIPLIER);
        devaddr = _devaddr;
    }
```

Medium severity issues

1. Blocked swaped token (StockToken)

Status: Fixed

The `rehyph()` modifier allows for exchanging `inc` token to another token and subsequently adding such tokens to the LP pair. During this operation, token remnants may occur, which will be transferred from the `Zapper` contract (L366-367) to the `StockToken` contract . One of these tokens (not the `inc` token) will be permanently locked in the StockToken contract.

Recommendation: We recommend implementing functionality to withdraw the remaining tokens or redirect them to the LP pair in the future.

2. Unsafe emergencyWithdraw (Farms (MasterChef))
Status: Fixed

The function `emergencyWithdraw()` is used to withdraw funds without considering rewards. However, in the current version, the function `emergencyWithdraw()` calls the regular `withdraw()` function of the third-party contract which may fail.

Recommendation: We recommend calling the corresponding `emergencyWithdraw()` function on the third-party `masterchef` contract (if such function exist).

Developer comment: According developer current implementation is used to prevent withdawing of all funds from the third-party masterchef contract.

3. Blocked swaped token (Farms (MasterChef))
Status: Fixed

The `rehyph()` modifier allows for exchanging `inc` token to another token and subsequently adding such tokens to the LP pair. During this operation, token remnants may occur, which will be transferred from the `Zapper` contract (L366-367) to the `StockToken` contract . One of these tokens (not the `inc` token) will be permanently locked in the StockToken contract.

Recommendation: We recommend implementing functionality to withdraw the remaining tokens or redirect them to the LP pair in the future.

4. Unsafe emergencyWithdraw (RhFarms (MasterChef))
Status: Fixed

The function `emergencyWithdraw()` is used to withdraw funds without considering rewards. However, in the current version, the function `emergencyWithdraw()` calls the regular `withdraw()` function of the third-party contract which may fail.

Recommendation: We recommend calling the corresponding `emergencyWithdraw()` function on the third-party `masterchef` contract (if such function exist).

Developer comment: According developer current implementation is used to prevent withdawing of all funds from the third-party masterchef contract.

5. Possible underflow (StockPool)
Status: Fixed

In the `withdraw()` function, an underflow might occur when updating the `pendingDividends` variable due to potential rounding inaccuracies when calculating the `_dividends` variable.

This will block withdrawal functionality.

Recommendation: We recommend adding a check that if `_dividends > pendingDividends`, then the value of `pendingDividends` should be set to zero.

Low severity issues

1. Gas optimization (PcapToken)
Status: Fixed

1. Visibility of the functions `initialize()`, `mint()`, `setNewFarms()`, `setNewHeartFarms()`, `updateMinterFarms()`, `updateMinterHeart()` can be declared as '`external`' to save gas.

2. The `setNewFarms()`, `setNewHeartFarms()`, `updateMinterFarms()`, `updateMinterHeart()` functions return default boolean value `false`. And such value never uses in the contract code. Consider removing it.

2. Parameters validation (PcapToken)
Status: Fixed

We recommend adding non-zero validation for the `_farms` and `_heartFarms` parameters of the `initialize()` function to avoid incorrect contract initialization

3. Lack of events (PcapToken)
Status: Fixed

The `initialize()`, `setNewFarms()`, `setNewHeartFarms()`, `updateMinterFarms()`, `updateMinterHeart()` functions of the contract lack events. Consider adding events for easier offchain tracking of the contract updates.

4. Outdated libraries and compiler (PcapToken)
Status: Fixed

Consider using latest versions of OpenZeppelin libraries and solidity compiler.

5. Source of the `inc` tokens (StockToken)
Status: Fixed

The source of the inc tokens is not explicitly specified in the contract code. However, these tokens are used for swapping and adding liquidity in the `rehyph()` modifier call.

Recommendation: We recommend specifying the source of the inc tokens in the documentation (including NatSpec). Additionally, it is important to note that any user can donate inc tokens to the StockToken contract at any time to activate the `rehyph()` modifier. We recommend ensuring that the `rehyph()` modifier functions as intended.

6. Parameters validation (StockToken)
Status: Fixed

To avoid incorrect contract initialization we recommend adding non-zero validation

1) for the `initialOwner` and `_devAddress` parameters of the contract constructor

2) for the `setZapper()`, `setStockPool()` functions.

7. Gas optimization (StockToken)
Status: Fixed

1. Visibility of the functions `mint()`, `burnUnlocked()`, `lock()`, `burnLocked()`, `userLocksLength()`, `userLocksArray()`, `setZapper()`, `setStockPool()`, `updateMinRehyphAmount()`, `updateSlippage()` can be declared as '`external`' to save gas.

2. The `burntAmount` variable is never used and can be removed.

3. The `lpReserve`, `dev` variables can be declared as `` `immutable` ``.

4. The `percentDec` variable can be declared as `` `constant` `` instead of `` `immutable` ``.

8. Lack of events (StockToken)
Status: Fixed

The `lock()`, `burnLocked()`, `setZapper()`, `setStockPool()`, `updateMinRehyphAmount()`, `updateSlippage()` functions of the contract lack events. Consider adding events for easier offchain tracking of the contract updates.

9. Variable default visibility (Farms (MasterChef))
Status: Fixed

The variable `slippage` has default visibility, which might lead to inconvenience in using the contract.

Recommendation: We recommend explicitly specifying the visibility for this variable.

10. stakingPercent value (Farms (MasterChef))
Status: Fixed

The value of the immutable variable `stakingPercent` is set to 9500000 in the contract constructor. At the same time contract logic uses `percentDec = 1000000` for calculation. Since the `9500000` is significantly greater than `1000000` we recommend making sure that the value of stakingPercent was correctly set.

11. Using safe transfers (Farms (MasterChef))
Status: Fixed

Since the contract uses `SafeERC20` library we recommend using it for all transfers.

Consider replacing `transfer()` function with `safeTransfer()` in the `safePcapTransfer()` function.

12. Gas optimization (Farms (MasterChef))
Status: Fixed

Visibility of the functions `updateMultiplier()`, `withdrawDevFee()`, `add()`, `set()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `withdrawAllTaxes()` can be declared as '`external`' to save gas.

13. Parameters validation (Farms (MasterChef))
Status: Fixed

We recommend adding parameter validation for the contract constructor anf for the `setDevAddress()`, `updateMinRehyphAmount()`, `updateSlippage()` functions to prevent incorrct setting of state variables.

14. Unused rewards from masterchef (Farms (MasterChef))
Status: Fixed

The functions `deposit()` and `withdraw()` interact with the third-party contract `masterchef` (L349, L366, L386). Typically, with such interactions (`deposit()` and `withdraw()`), the calling account or contract (`msg.sender`) is awarded rewards. However, the contract does not explicitly mention the use of such rewards, implying that these rewards might be locked in the contract's balance.

Recommendation: We recommend implementing functionality for distributing the received rewards or documenting this behavior.

15. Using safe transfers (RhFarms (MasterChef))
Status: Fixed

Since the contract uses `SafeERC20` library we recommend using it for all transfers.

Consider replacing `transfer()` function with `safeTransfer()` in the `safePcapTransfer()` , `safeStockTransfer()`, `rehyph()` functions.

16. Gas optimization (RhFarms (MasterChef))
Status: Fixed

Visibility of the functions `add()`, `setTaxFee()`, `pendingPcap()`, `pendingStock()`, `deposit()` , `withdraw()` , `emergencyWithdraw()` can be declared as '`external`' to save gas.

17. Unused variable (RhFarms (MasterChef))
Status: Fixed

The `minRehyphAmount` variable is never used. Make sure that contract works as intended without using this variable.

18. Unused function parameter (StockPool)
Status: Fixed

Consider using the `_customerAddress` instead of `msg.sender` in the `createStake()` function (L141) for the correct code consistency.

19. Gas optimization (StockPool)
Status: Fixed

1. All `public` functions of the contract (except the `dividendsOf()` function) can be delared as `extarnal`.

2. Consider adding return statement to the `updatePool()` modifier in case `block.timestamp == lastDripTime`

20. Variable default visibility (StockPool)
Status: Fixed

The variable `STOCK` has default visibility, which might lead to inconvenience in using the contract.

Recommendation: We recommend explicitly specifying the visibility for this variable.

21. lastDripTime updating (StockPool)
Status: Fixed

In the `updatePool()` modifier, there may be a situation where the value of `distRate` is greater than `secondsPassed * dividendPool`. This will result in `dividends` being zero.

In such a case, it makes sense to exit the modifier without updating the `lastDripTime` variable.

22. Whitelist for function execution (Zapper)
Status: Fixed

Since the contract may contain a certain amount of tokens left over from swap, addLiquidity, and other operations, we recommend adding a whitelist of addresses that are allowed to call functions of this contract. This will prevent external users from invoking the functions and taking over of control of tokens.Also we recommend adding owner functionality to withraw staked tokens in the contract.

23. Mapping default visibility (Zapper)

Status: Fixed

The mapping `tokens` has default visibility, which might lead to inconvenience in using the contract.

Recommendation: We recommend explicitly specifying the visibility for this mapping.

24. Gas optimization (Zapper)

Status: Fixed

The `tokenA`, `tokenB`, `pair`, `wpls`, `stockToken`, `pulsexV2Router` variables can be declared as immutable.

Conclusion

Pcap PcapToken, StockToken, Farms (MasterChef), RhFarms (MasterChef), StockPool, Zapper contracts were audited. 4 high, 5 medium, 24 low severity issues were found.
4 high, 5 medium, 24 low severity issues have been fixed in the update.

We strongly recommend writing unit tests to have extensive coverage of the codebase minimize the possibility of bugs and ensure that everything works as expected.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.
OxGuard retains exclusive publishing rights for the results of this audit on its website and social networks.

0x Guard