Guard

Smart contracts security assessment

Final report

Tariff: Standard

Atropine

October 2023

0xguard.com

hello@0xguard.com

Contents

Introduction

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

The audited project is Masterchef-like farm minting ERC20 token as a reward.

The PineToken contract includes pre-mint in the constructor.

The MasterChef contract charges a pool-specific fee on both deposit and withdrawal.

The code is available at the @MathAsgard/AtropineContracts Github repo and was audited in the 972a3e4 commit.

The updated code was rechecked after the commit 77aa37e.

Name	Atropine
Audit date	2023-10-03 - 2023-10-05
Language	Solidity
Platform	Pulse Chain

Contracts checked

Name	Address
PineToken	
MasterChef	

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Outdated Compiler Version	passed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. Mint is open for owner (PineToken)

Status: Fixed

The contract owner can mint an arbitrary number of tokens alongside the typical minter MasterChef contract.

```
    function mint(uint256 amount) public onlyOwner returns (bool) {
        _mint(_msgSender(), amount);
        return true;
    }

    function mint(address _to, uint256 _amount) public {
        require(msg.sender == minter, "PINE: Permission declined");
        _mint(_to, _amount);
    }
```

Recommendation: Remove owner minting or secure ownership by transferring it to a well-known contract, e.g., Timelock with Multisig admin.

2. Owner controls emission rate (MasterChef)

Status: Open

The reward emission rate is calculated using the PinePerBlock, stakingPercent and BONUS_MULTIPLIER variables. PinePerBlock and BONUS_MULTIPLIER are updatable by the contract owner without any safety restrictions. This allows owner to increase emission rate to absurdly high value to receive massive reward in a single block and manipulate the PineToken's price on DEX.

Recommendation: Add safety limit to reward emission rate.

3. Owner can steal user's funds (MasterChef)

Status: Fixed

The `migrate` function calls for migrator contract, which is controlled by the owner. It also approves migrator to access all staked funds from selected pool. This means that owner is able to transfer all

staked funds to an arbitrary address without user's permission.

```
    function setMigrator(IMigratorChef _migrator) public onlyOwner {
        migrator = _migrator;
    }

    // Migrate lp token to another lp contract. Can be called by anyone. We trust that
migrator contract is good.
    function migrate(uint256 _pid) public {
        require(address(migrator) != address(0), "migrate: no migrator");
        PoolInfo storage pool = poolInfo[_pid];
        IERC20 lpToken = pool.lpToken;
        uint256 bal = lpToken.balanceOf(address(this));
        lpToken.safeApprove(address(migrator), bal);
        IERC20 newLpToken = migrator.migrate(lpToken);
        require(bal == newLpToken.balanceOf(address(this)), "migrate: bad");
        pool.lpToken = newLpToken;
    }
```

Function `migrate` is not going to work as expected in general. Requirement of balance equality for different LP tokens before and after switching DEX is very strict condition which is hard to meet.

Recommendation: Remove `migrate` function.

Medium severity issues

1. Duplicated pools are not supported (MasterChef)

Status: Fixed

Pools with the same staking token should be avoided by the contract owner. Such pools receive lower reward than it should be because the `updatePool` function calculates the pool supply as `pool.lpToken.balanceOf(address(this))` which is incorrect in case of duplicated pools.

Recommendation: Consider storing pool's `lpSupply` in the `PoolInfo` struct.

Low severity issues

1. Irrelevant comment (PineToken)
Status: Fixed

The L875 comment with @notice description is misplaced and should be moved to the description of the `mint` function.

```
// @notice Creates `_amount` token to `_to`. Must only be called by the minter
(MasterChef).
```

2. Typos (PineToken)
Status: Fixed

Typo in 'adysd' (L921).

3. Typos (MasterChef)
Status: Fixed

Typos in 'avaliable', 'vairables', 'alrady'.

4. Constant variables (MasterChef)
Status: Fixed

Some variables (`PINE`, `stakingPercent`, `devPercent`, `percentDec`, `startBlock`) are constant or immutable and should be marked with an appropriate keyword and named according to the Solidity naming conventions.

5. Non-standard tokens aren't supported (MasterChef)
Status: Open

Tokens with transfer hooks and tokens with transfer tax are not supported and must be avoided by the owner.

Tokens with transfer hooks are susceptible to reentrancy problem. Tokens with transfer tax are not checked during the deposit.

6. Division before multiplication (MasterChef)

Status: Fixed

Division before multiplication reduces precision of calculations.

```
function updatePool(uint256 _pid) public {
  ...
  uint256 PineReward = multiplier.mul(PinePerBlock).mul(pool.allocPoint).div(totalAllocP
oint).mul(stakingPercent).div(percentDec);
  ...
}
```

Also, the total emission rate is calculated as `PinePerBlock * BONUS_MULTIPLIER * stakingPercent / percentDec`. It can be reduced to a single `PinePerBlock`.

Conclusion

Atropine PineToken, MasterChef contracts were audited. 3 high, 1 medium, 6 low severity issues were found.
2 high, 1 medium, 5 low severity issues have been fixed in the update.

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.

0x Guard