Smart Contract Audit Report

Cyclone Protocol v2

CHAINSHIELD

ChainShield Team (https://chainshield.io)

12th March, 2021

Table of Contents

Introduction

Cyclone Protocol

Cyclone is a cross-chain, non-custodial, universal privacy-preserving protocol with decentralized governance. Cyclone applies zkSNARKs to enable transactional privacy for all DeFi components by breaking the on-chain link between depositor and recipient addresses. It uses a smart contract that accepts coins/tokens deposits, which can be withdrawn by a different address. Whenever an asset is withdrawn from Cyclone, there is no way to link the withdrawal to the deposit for absolute privacy.

While Cyclone's zkSNARKs part is based on the attested implementation of tornado.cash, it offers unique values in supporting cross-chain and being the universal privacy-preserving layer for almost all DeFi components with the decentralized governance by CYC holders.

So far, Cyclone has 4 anonymity pools on IoTeX blockchain and will have 3 new anonymity pools on Binance Smart Chain.

ChainShield

ChainSheild Inc. is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products, including smart contract auditing, project consultation and penetration test of blockchain infrastructure. We can be reached via Website (https://chainshield.io), or Email (support@chainsheild.io).

Scope

This report covers the auditing methodology and finding of Cyclone Protocol v2 Smart Contract in https://github.com/cycloneprotocol/cyclone-contracts at commit 684887e. Smart contracts in this repo can be classified into three categories:

- Standard (or commonly used) contracts, such as Pausable.sol, Ownable.sol, SafeMath.sol, BasicToken.sol, Timelock.sol
- zkSNARK contracts from Tornado.cash, such as Verifier.sol, MerkleTreeWithHistory.sol
- Cyclone Protocol specific contracts, such as Cyclone.sol, CycloneV2.sol, Aeolus.sol, AeolusV2.sol

For standard (or commonly used) contracts, we have confirmed that they are unmodified from the original contracts; for zkSNARK contracts from Tornado.cash which have already been audited, we have confirmed that they are unmodified from the original contracts used by Tornado.cash; for Cyclone Protocol specific contracts, we have focused on v2 ones below

- CycloneV2.sol
- AeolusV2.sol
- UniswapV2CycloneRouter.sol

For these three contracts, we have audited common pitfalls of smart contract such as Reentrancy, Unchecked External Call, Costly Loop; readability such as Semantic Consistency Checks, code structures; gas optimization; and DeFi related vulnerabilities such as Oracle, Authentication Management, ERC20 Idiosyncrasies Handling, Deployment Consistency and so on. No critical issues have been found during our auditing.

Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contracts. In other words, the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Last but not least, this security audit should not be used as investment advice.

Findings

Summary

Here is a summary of our findings after analyzing the design and implementation of Cyclone Protocol v2. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify issues reported by our tools. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Finding	Category	Severity	Impact
1 - Possible Sandwich/MEV Attack For Reduced Returns	DeFi Common Pitfalls	Medium	Low
2 - Parameter “govDAO” is confusing	Readability	Low	None
3 - Constructor Input Parameters	Potential Inconsistency	Low	None
4 - Liquidity removed may be 0	DeFi Common Pitfalls	Low	Minor
5 - Unused Code Removal	Readability	Low	None

Overall, cyclone smart contracts are well-designed and well-engineered. We have so far identified a few non-critical issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are listed in the above table. More information can be found in the next subsection.

Key Findings

In this section, we list the key findings we have during the audit.

Finding 1: Possible Sandwich/MEV Attack For Reduced Returns

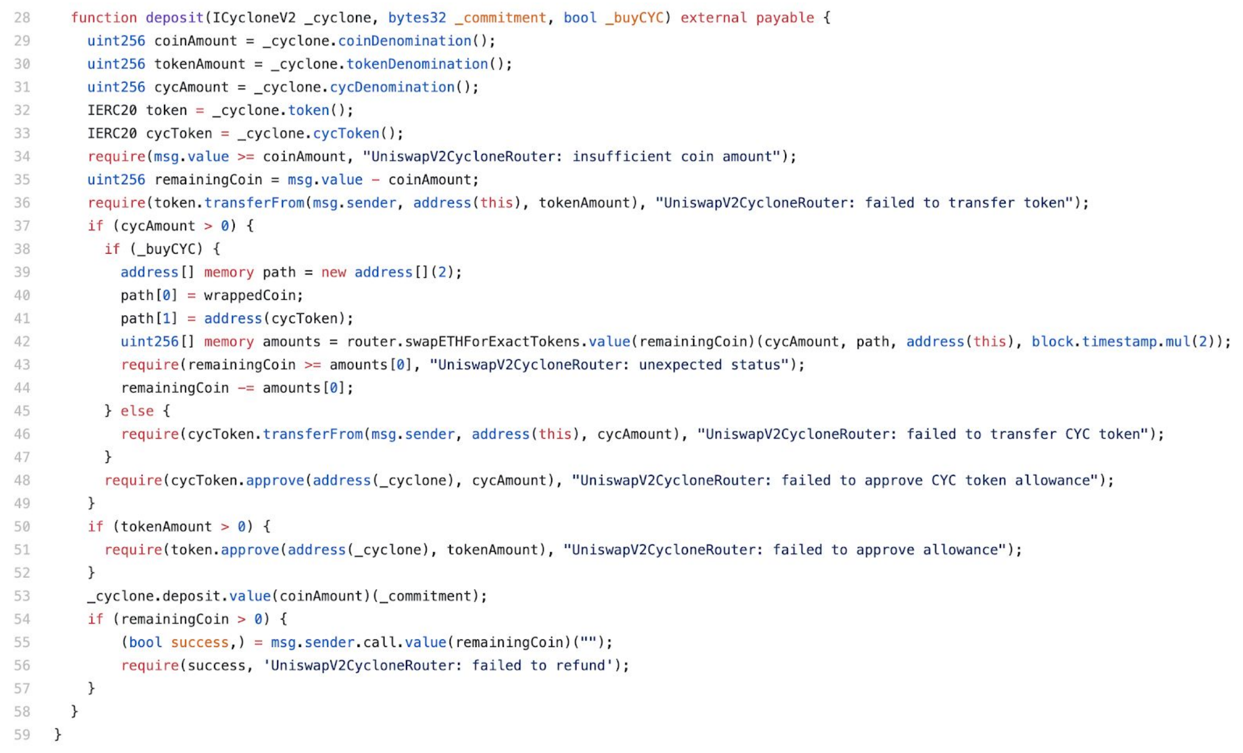
• Severity: Medium

• Impact: Low

• Category: DeFi Common Pitfalls

UniswapV2CycloneRouter.sol is a frontend contract for depositing into the anonymity pools. When a user deposits (for example) BUSD into a pool, and does not have enough CYC to start with, the contract will buy some CYC (by invoking Swap operation on Pancake) and processes forward with the deposit. If the user has enough CYC, there will be no buying action taken.

We notice the swap operation does not specify any restriction on possible slippage and is therefore vulnerable to possible front-running attacks, resulting in a smaller gain for this round of yielding for users.



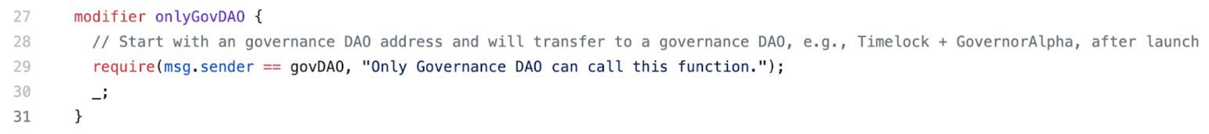
```
  function deposit(ICycloneV2 _cyclone, bytes32 _commitment, bool _buyCYC) external payable {
    uint256 coinAmount = _cyclone.coinDenomination();
    uint256 tokenAmount = _cyclone.tokenDenomination();
    uint256 cycAmount = _cyclone.cycDenomination();
    IERC20 token = _cyclone.token();
    IERC20 cycToken = _cyclone.cycToken();
    require(msg.value >= coinAmount, "UniswapV2CycloneRouter: insufficient coin amount");
    uint256 remainingCoin = msg.value - coinAmount;
    require(token.transferFrom(msg.sender, address(this), tokenAmount), "UniswapV2CycloneRouter: failed to transfer token");
    if (cycAmount > 0) {
      if (_buyCYC) {
        address[] memory path = new address[](2);
        path[0] = wrappedCoin;
        path[1] = address(cycToken);
        uint256[] memory amounts = router.swapETHForExactTokens.value(remainingCoin)(cycAmount, path, address(this), block.timestamp.mul(2));
        require(remainingCoin >= amounts[0], "UniswapV2CycloneRouter: unexpected status");
        remainingCoin -= amounts[0];
      } else {
        require(cycToken.transferFrom(msg.sender, address(this), cycAmount), "UniswapV2CycloneRouter: failed to transfer CYC token");
      }
      require(cycToken.approve(address(_cyclone), cycAmount), "UniswapV2CycloneRouter: failed to approve CYC token allowance");
    }
    if (tokenAmount > 0) {
      require(token.approve(address(_cyclone), tokenAmount), "UniswapV2CycloneRouter: failed to approve allowance");
    }
    _cyclone.deposit.value(coinAmount)(_commitment);
    if (remainingCoin > 0) {
        (bool success,) = msg.sender.call.value(remainingCoin)("");
        require(success, 'UniswapV2CycloneRouter: failed to refund');
    }
  }
}
```

Note that this is a common issue plaguing current AMM-based DEX solutions. Specifically, a large trade may be sandwiched by a preceding sell to reduce the market price, and a tailgating buyback of the same amount plus the trade amount. Such sandwiching behavior unfortunately causes a loss and brings a smaller return as expected to the trading user or the strategy contract in our case (because the swap rate is lowered by the preceding sell). As a mitigation, we may consider specifying the restriction on possible slippage caused by the trade or referencing

Finding 2: Parameter “govDAO” is confusing

- Severity: Low
- Impact: None
- Category: Readability

The owner of CycloneV2.sol is named as
“govDAO”. It is a common practice to name the owner or operator of a smart contract as “Owner” or “Operator”. Using a different word “govDAO” may confuse readers. However, we have communicated with the Cyclone Team and learned that the “Owner” of CycloneV2.sol will be passed over to GovernorAlpha.sol soon which is indeed a governance DAO used by Compound.finance. At this point, we recommended to have a seperate member variable “Owner” as the owner of this contract to be explicit.



```
modifier onlyGovDAO {
  // Start with an governance DAO address and will transfer to a governance DAO, e.g., Timelock + GovernorAlpha, after launch
  require(msg.sender == govDAO, "Only Governance DAO can call this function.");
  _;
}
```

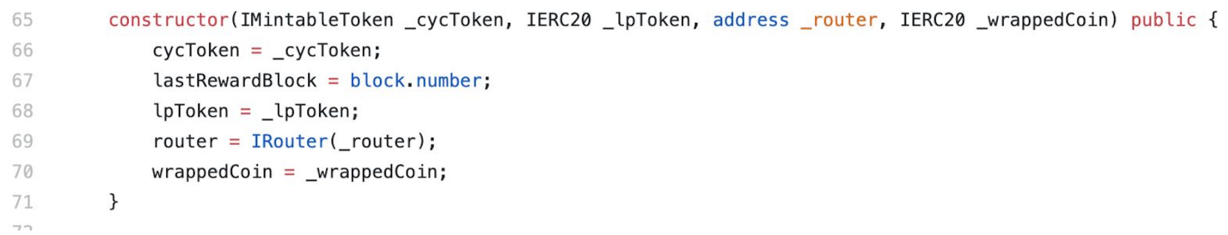
Finding 3: Constructor Input Parameters

• Severity: Low

• Impact: None

• Category: Potential inconsistency

AeolusV2.sol is implemented based on Sushiswap’s MasterChef contract, which handles the liquidity mining of CYC token. The last input parameter "_wrappedCoin" of the constructor basically refers to WBNB - the wrapped BNB coin on BSC chain. However, this parameter can be directly fetched from “_router” which points to the Pancake’s router contract. The manually input "_wrappedCoin" may be inconsistent with the wrapped coin used in Pancake. We recommend fetching from “_router” for better consistency.



```
    constructor(IMintableToken _cycToken, IERC20 _lpToken, address _router, IERC20 _wrappedCoin) public {
        cycToken = _cycToken;
        lastRewardBlock = block.number;
        lpToken = _lpToken;
        router = IRouter(_router);
        wrappedCoin = _wrappedCoin;
    }
```

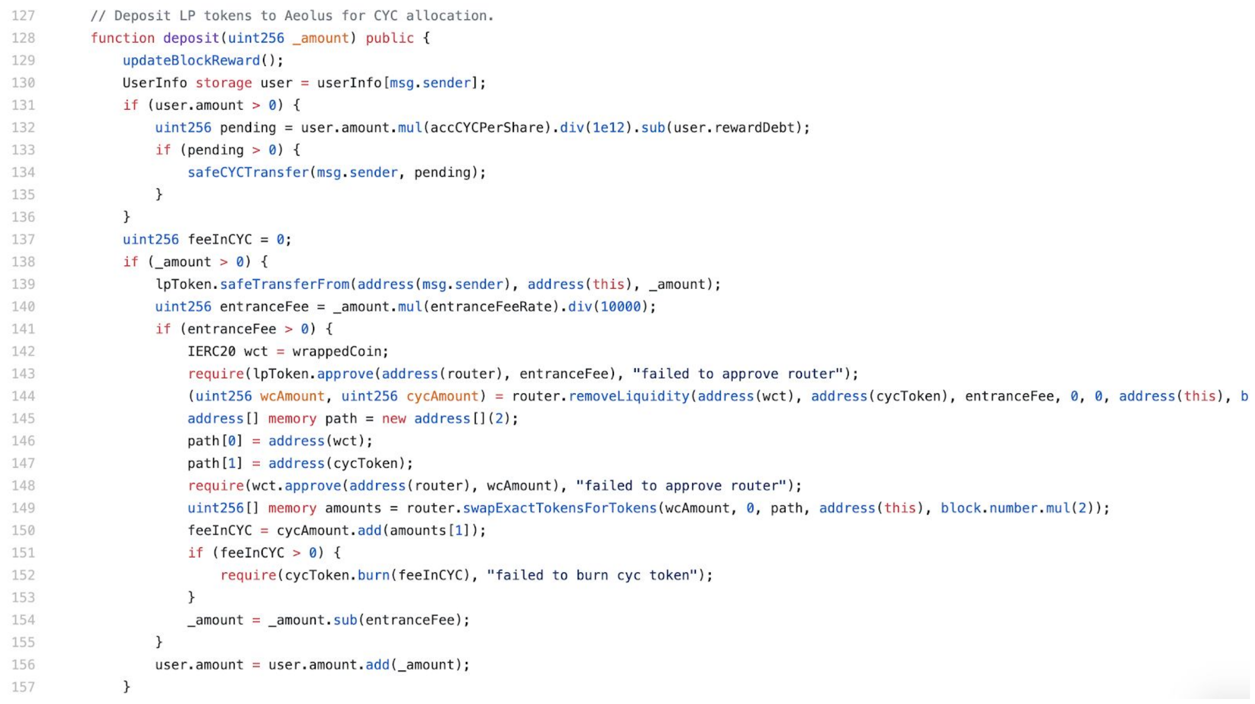
Finding 4: Liquidity removed may be 0

• Severity: Low

• Impact: Minor

• Category: DeFi Common Pitfalls

In the same AeolusV2.sol contract, when a user deposits, it When a user deposits CYC-BNB pool tokens from Pancake, it charges a small fee in terms of the pool token, unstakes to get CYC and BNB, buys CYC using BNB on Pancake and burns all CYC it has. In this process, if the degree of liquidity is too small on Pancake for the CYC-BNB pair, the removed liquidity may be zero. As a result, the swapping wrapped coin to CYC token will fail. We suggest adding proper error handling here.



```
    // Deposit LP tokens to Aeolus for CYC allocation.
    function deposit(uint256 _amount) public {
        updateBlockReward();
        UserInfo storage user = userInfo[msg.sender];
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(accCYCPerShare).div(1e12).sub(user.rewardDebt);
            if (pending > 0) {
                safeCYCTransfer(msg.sender, pending);
            }
        }
        uint256 feeInCYC = 0;
        if (_amount > 0) {
            lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
            uint256 entranceFee = _amount.mul(entranceFeeRate).div(10000);
            if (entranceFee > 0) {
                IERC20 wct = wrappedCoin;
                require(lpToken.approve(address(router), entranceFee), "failed to approve router");
                (uint256 wcAmount, uint256 cycAmount) = router.removeLiquidity(address(wct), address(cycToken), entranceFee, 0, 0, address(this), b
                address[] memory path = new address[](2);
                path[0] = address(wct);
                path[1] = address(cycToken);
                require(wct.approve(address(router), wcAmount), "failed to approve router");
                uint256[] memory amounts = router.swapExactTokensForTokens(wcAmount, 0, path, address(this), block.number.mul(2));
                feeInCYC = cycAmount.add(amounts[1]);
                if (feeInCYC > 0) {
                    require(cycToken.burn(feeInCYC), "failed to burn cyc token");
                }
                _amount = _amount.sub(entranceFee);
            }
            user.amount = user.amount.add(_amount);
        }
```

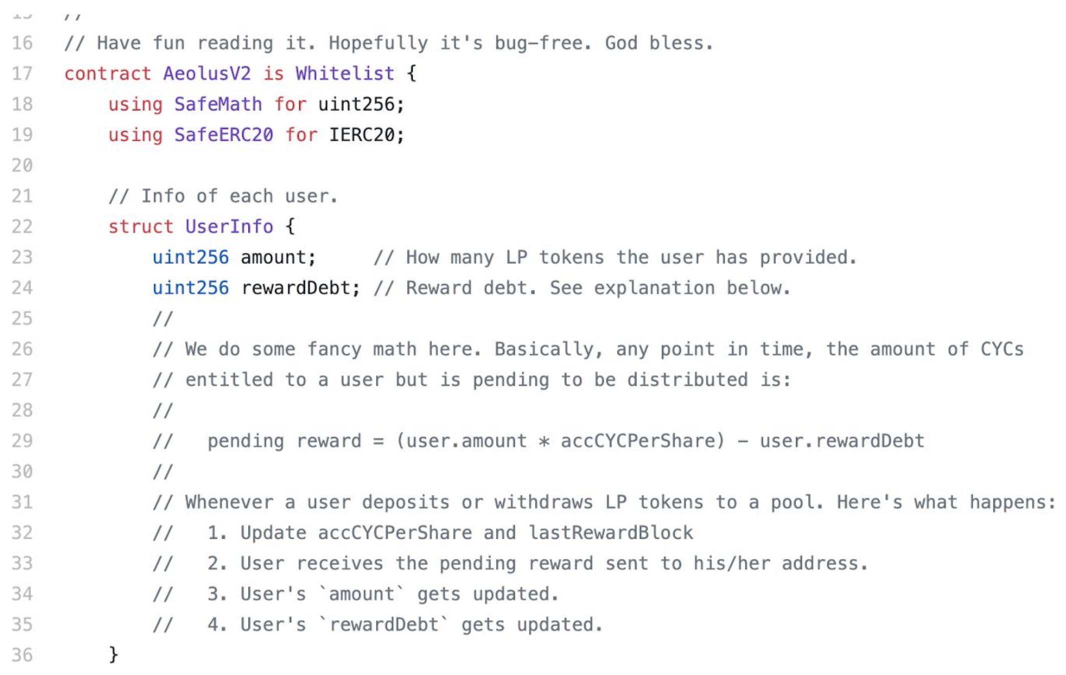
Finding 5: Unused Code Removal

• Severity: Low

- Impact: None
- Category: Readability

Cyclone Protocol makes good use of a number of reference contracts, such as ERC20, SafeERC20, SafeMath, Context, and Ownable, to facilitate its code implementation and organization. However, we observe the inclusion of certain unused code or the presence of unnecessary redundancies that can be safely removed.

In AeolusV2.sol, the reference contract Whitelist has been used to implement access control, which allows access to a method only from specific addresses. However, we found this access control is not actually used in this contract. Replacing “Whitelist” with “Ownerable” is what we recommend.



```
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract AeolusV2 is Whitelist {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;     // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.
        //
        // We do some fancy math here. Basically, any point in time, the amount of CYCs
        // entitled to a user but is pending to be distributed is:
        //
        //   pending reward = (user.amount * accCYCPerShare) - user.rewardDebt
        //
        // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
        //   1. Update accCYCPerShare and lastRewardBlock
        //   2. User receives the pending reward sent to his/her address.
        //   3. User's `amount` gets updated.
        //   4. User's `rewardDebt` gets updated.
    }
```

Conclusion

In this audit, we have analyzed the design and implementation of Cyclone Protocol v2. The audited system presents a unique addition to current DeFi offerings transactional privacy for users. Inspired by DeFi mining economics, the Cyclone Protocol v2 has been equipped with proper token economics to incentivize everyone to contribute to its anonymity pools. The current code base is clearly organized and those identified issues are promptly confirmed and fixed.

As a final precaution, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedback or suggestions on our methodology and findings.