SECURITY AUDIT OF

GOEN VAULT SMART CONTRACTS

Public Report

Jul 18, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Jul 18, 2022. We would like to thank the Goen for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Goen Vault Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Goen Vault Smart Contracts

Goen Finance is a DAO. You also can earn GOEN - the governing token of Goen Finance by depositing your funds into Goen's pools.

Then you can stake GOEN to get veGOEN (vote-escrowed GOEN) back in order to decide on GOEN reward emission of each pool, significant improvements of protocol, and get rebates in BTC, ETH, BNB, DOT...

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the smart contracts of Goen Vault Smart Contracts. It was conducted on the source code provided by the Goen team.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
829c275b630010e3167e82857ba16d681de082773c8eaa5373ad241474e460cc	VaultBase.sol
4a562837154effe0995e1f0bd73955a2ca0c88d207ca05272b21c28c6a95b0c4	VaultBSW.sol
8a339839a1769b1e5551ac0e404be79c1f9ab67ea039891ba031e6fdfeac98b4	VaultController.sol
6cc4719074434d072463fe9e8fc2c6052b64eb752c7c7d6bcde0a981dbf36b63	VaultCake.sol
12b4c13fa7279dc0b7bdd784a396e939e97b73e87c3a5d5cb5ef8706508ea807	VaultCakeKNC.sol

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence

- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Goen Vault Smart Contracts was written in `Solidity` language, with the required version to be `^0.6.12`.

2.1.1. VaultBase contract

The base of the vault contracts implements the main logic of the product. The users may deposit a specific token to the vaults for profit. The vault represents the users depositing the `staking` contract to gain the reward tokens.

The vault will distribute the reward token and its reward (goen tokens) to the user. When the users withdraw their deposited tokens, the latest profit from the `staking` contract may be kept in the contract like the penalty, it will be distributed to other deposited users.

2.1.2. VaultBSW contract

One of the latest contracts inherits the `VaultBase` contract and implements some functions (including `_distributeReward`, `_swapRewardToTokens`, ...) for a specific token.

2.1.3. VaultCake contract

One of the latest contracts inherits the `VaultBase` contract and implements some functions (including `_distributeReward`, `_swapRewardToTokens`, ...) for a specific token.

2.1.4. VaultCakeKNC contract

One of the latest contracts inherits the `VaultBase` contract and implements some functions (including `_distributeReward`, `_swapRewardToTokens`, ...) for a specific token.

2.2. Findings

This section contains a detailed analysis of all the vulnerabilities that were discovered by the audit team during the audit process.

2.2.1. VaultBase - Reentrancy in `withdrawAll` function if the rewardToken is WBNB CRITICAL

The `withdrawAll` function uses the `call` method to transfer the native token and update the state variable after the transfer action. Therefore, the attackers may use reentrancy attack to gain `profit` multiple times in a transaction.

```
function withdrawAll() external override updateReward(msg.sender)
updateGOENReward(msg.sender) {
    IERC20Metadata poolRewardToken = getRewardToken();
    uint256 amount = principal[msg.sender];
    //Update userTotalDeposit24h
    uint256 amount24h = userTotalDeposit24h[msg.sender];
    if (amount24h >= amount) { //always true in the reentrancy flow
        userTotalDeposit24h[msg.sender] = amount24h.sub(amount);
    } else {
        uint256 before = poolRewardToken.balanceOf(address(this));
        _withdrawFromPool(amount.sub(amount24h));
        if (address(vaultStakingToken) == address(poolRewardToken)) {
            before = before.add(amount.sub(amount24h));
        }
        uint256 withdrawProfit = poolRewardToken.balanceOf(address(this)).sub(before);

        totalShares = totalShares.sub(amount.sub(amount24h));
        userTotalDeposit24h[msg.sender] = 0;
        profitInterval = profitInterval.add(withdrawProfit);
    }
    totalDeposit = totalDeposit.sub(amount);
    totalDeposit24h = totalDeposit24h.sub(amount24h);

    vaultStakingToken.transfer(msg.sender, amount);

    delete principal[msg.sender];
    uint256 profit = rewards[msg.sender];
    _transferReward(msg.sender, profit); // <- Trigger reentrancy here
    if (config.goenReleased()) {
        schedule.getGoen(msg.sender, goenRewards[msg.sender]);
        delete goenRewards[msg.sender];
        delete goenUserRewardPerSharePaid[msg.sender];
    }
    delete rewards[msg.sender];
    delete userRewardPerSharePaid[msg.sender];
    delete balances[msg.sender];
    emit Withdrawn(msg.sender, amount, 0);
}

function _transferReward(address receiver, uint256 amount) internal {
    if (amount == 0) return;
    if (address(rewardToken) == WBNB) {
        SafeToken.safeTransferETH(receiver, amount); // <- Trigger reentrancy here
    } else {
        SafeToken.safeTransfer(address(rewardToken), receiver, amount);
    }
}

//Safetoken.sol
function safeTransferETH(address to, uint256 value) internal {
    (bool success, ) = to.call{value: value}(new bytes(0)); // <- Trigger reentrancy here
    require(success, "!safeTransferETH");
}
```

RECOMMENDATION

Add `nonReentrant` modifier to all external functions to avoid attackers. Besides, the state variable should update before the `transfer` action.

UPDATES

- *Jul 18, 2022*: This issue has been acknowledged and fixed by the Goen team.

2.2.2. VaultBase.sol - Reentrancy in `harvest` function if the rewardToken is WBNB CRITICAL

The `harvest` function uses the `_distributeReward` function implemented by the child contracts. In the child contracts, the `_distributeReward` function uses the `_transferReward` like the above issue which triggers reentrancy. The attacker can use reentrancy attack to swap tokens in the vault multiple times to get more harvestBounty. The reward factor may be affected.

```
function harvest()
    public
    override
    notPaused
    updateReward(address(0))
    updateGOENReward(address(0))
    returns (uint256 poolReceivedAmount)
{
    // GET REWARDS
    IERC20Metadata poolRewardToken = getRewardToken();
    uint256 before = poolRewardToken.balanceOf(address(this));
    _depositInPool(totalDeposit24h);
    if (address(vaultStakingToken) == address(poolRewardToken)) {
        before = before.sub(totalDeposit24h);
    }
    uint256 withdrawProfit = poolRewardToken.balanceOf(address(this)).sub(before);

    uint256 reward = 0;
    uint256 rebateReceive = 0;
    if (profitInterval.add(withdrawProfit) > 0) {
        poolReceivedAmount = _swapRewardToToken(address(poolRewardToken),
profitInterval.add(withdrawProfit));
        (reward, rebateReceive) = _distributeReward(poolReceivedAmount);
    }
    // // BATCH DEPOSIT
    rewardPerShareStored = rewardPerShare();
    lastTimeReward = totalReward;
    for (uint256 i = 0; i < depositedUsers.length; i++) {
        address userAddress = depositedUsers[i];
        rewards[userAddress] = earned(userAddress);
        userRewardPerSharePaid[userAddress] = rewardPerShareStored;

        uint256 userDeposited = userTotalDeposit24h[userAddress];
        balances[userAddress] = balances[userAddress].add(userDeposited);
        delete userTotalDeposit24h[userAddress];
        delete mappingUser[userAddress];
    }
    totalShares = totalShares.add(totalDeposit24h);

    // // CLEAR DEPOSIT
    delete depositedUsers;
    totalDeposit24h = 0;
    profitInterval = 0;

    emit VaultHarvested(reward, rebateReceive);
}

//VaultBSW.sol
function _distributeReward(uint256 poolReceivedAmount)
internal
override
returns (uint256, uint256) {
    ...
    if (totalShares > 0) {
        totalReward += reward;
        _transferReward(address(TREASURY), treasuryReceive);
    } else {
        _transferReward(address(TREASURY), treasuryReceive.add(reward));
        reward = 0;
    }
    _transferReward(helper.tokenAtPool(address(rewardToken)), rebateReceive);
    _transferReward(msg.sender, harvestBounty);  // <- Trigger reentrancy here
    _transferReward(address(GOV), govReceive);
    return (reward, rebateReceive);
}
```

RECOMMENDATION

Add `nonReentrant` modifier to all external functions to avoid attackers. Besides, the state variable should update before the `transfer` action.

UPDATES

- *Jul 18, 2022*: This issue has been acknowledged and fixed by the Goen team.

2.2.3. VaultBase.sol - Attacker may spam `addUser` function to break the `harvest` and `reinvest` logics HIGH

The `addUser` function is a public function with no limitation. So, the attacker may spam `addUser` to increase the length of the `depositUsers` array. When the length of `depositUsers` is too large, the for-loop in both `harvest` and `reinvest` may cause run out of gas in the transactions.

There is no flow which allows erasing the `depositUsers` directly. Therefore, both `harvest` and `reinvest` logics will always be stuck.

```
function addUser(address userAddress) public returns (uint256) {
    require(userAddress != address(0));

    if (!userExist(userAddress)) {
        newUser = User(depositedUsers.length, userAddress);

        mappingUser[userAddress] = newUser;
        depositedUsers.push(userAddress);

        return newUser.userId;
    }
}
```

RECOMMENDATION

The `addUser` public function should be changed to `internal` function. Besides, the `depositedUser` should have a length constraint to avoid the problem when the `depositedUser` is not erased in time.

UPDATES

- *Jul 18, 2022*: This issue has been acknowledged and fixed by the Goen team.

2.2.4. VaultBase.sol - Attacker may spam `deposit` function to increase the length of `depositUsers` array MEDIUM

There is no limitation with the `_amount` parameter in the `deposit` function. Therefore, the attacker may deposit `0` token (or approximate `0`) to increase the length of `depositUsers`.

```
function deposit(uint256 _amount)
    public
    override
    notPaused
    nonReentrant
    updateReward(msg.sender)
    updateGOENReward(msg.sender)
{
    vaultStakingToken.transferFrom(msg.sender, address(this), _amount);
    totalDeposit = totalDeposit.add(_amount);

    addUser(msg.sender);
    userTotalDeposit24h[msg.sender] = userTotalDeposit24h[msg.sender].add(_amount);
    principal[msg.sender] = principal[msg.sender].add(_amount);
    totalDeposit24h = totalDeposit24h.add(_amount);

    emit Deposited(msg.sender, _amount);
}
```

RECOMMENDATION

The function should add a require statement which compares `_amount` parameter with a threshold constant.

UPDATES

- *Jul 18, 2022*: This issue has been acknowledged and fixed by the Goen team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jul 18, 2022*	Public Report	Verichains Lab

Table 2. Report versions history