TULIP DAO
X
DEFIMOON
KYC AUDIT DEVELOPMENT

CONTACTS
TWITTER TULIP DAO

TWITTER DEFIMOON

REPORT SMART CONTRACT AUDIT

AERARIUM FI
PROJECT

APRIL, 2022

TULIPDAO

X

DEFIMOON
be secure

AUDITOR
TEAM "DEFIMOON"

APPROVED BY
TULIP DAO

AERARIUM FI
APRIL, 2022

#SECURITY
AUDITED BY

DEFIMOON
be secure

TULIPDAO

REPORT
SMART CONTRACT
AUDIT

AUDIT OF PROJECT

HTTPS://AERARIUMFI.COM/

HTTPS://DOCS.AERARIUMFI.COM/AERARIUM-FI-NEXT-GENERATION-DAAS/THE-PROTOCOL/INTRODUCTION

Aerarium Fi Security Audit

Disclaimer

This audit is not financial, investment, or any other kind of advice and could be used for informational purposes only. This report is not a substitute for doing your own research and due diligence should always be paid in full to any project. Defimoon is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Defimoon has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Defimoon is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. Defimoon has no obligations other than to publish an objective report. Defimoon will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Defimoon assumes that the provided information and materials were not altered, suppressed, or misleading. This report is published by Defimoon, and Defimoon has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Defimoon. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Audit Information

Defimoon utilizes both manual and automated auditing approach to cover the most ground possible. We begin with generic static analysis automated tools to quickly assess the overall state of the contract. We then move to a comprehensive manual code analysis, which enables us to find security flaws that automated tools would miss. Finally, we conduct an extensive unit testing to make sure contract behaves as expected under stress conditions.

In our decision making process we rely on finding located via the manual code inspection and testing. If an automated tool raises a possible vulnerability, we always investigate it further manually to make a final verdict. All our tests are run in a special test environment which matches the "real world" situations and we utilize exact copies of the published or provided contracts.

While conducting the audit, the Defimoon security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Defimoon assesses the risks and assigns a risk level to each section together with an explanatory comment.

Contents

Project information

Name	Aerarium
Description	Aerarium Fi is a Treasury-as-a-Service protocol on the Metis chain. By purchasing a “fractal”, users are purchasing part of the protocol, and will earn real-time payouts as such.
Website	https://aerariumfi.com/
Twitter	https://twitter.com/aerariumfi
Token Name	Aerarium Fi
Token Short	$AERA
Total Supply	999899900000000000000000
Token Decimals	18
Contract address	0xFE540D6dbAD8C68928778AaF2Be828efA4b44Fa2
Code Language	Solidity

Name	Aerarium
Chain	Metis

Audit overview

During the audit 3 Low Risk issues were discovered

The contract has no majour security issues, however it is recommended to address 3 Low Risk issues documented in this report. Some points should be taken into the consideration. The current contract has a large number of state variables declared, some of which could be marked as constants, and some can be removed. This way, code quality and gas consumption efficiency could be improved.

Since the `NodeManager` contract is not verified on chain, the hash-sums of the bytecode of provided contract was compared with the hush-sum of the on-chain version. Provided `NodeManager` contract's bytecode and the bytecode of the deployed contract do not match. This does not explicitly mean that the deployed codebase differs from the audited and does not point to the presence of a malicious intent — bytecode could differ due to various compilation optimizations and use of different compilers. However, it does mean that Defimoon cannot guarantee that the deployed codebase matches the audited code and hence, due diligence should be paid.

Those findings represent a "good to know while interacting with the project" information, but do not and should not directly damage the project in its current state, hence it’s up to the project team to *resolve* those issues if they feel the need to do so.

The UI/UX, logic, team, or tokenomics of the Aerarium project were not reviewed by the Defimoon in the scope of this audition.

Please find the full report below for a complete understanding of the audit.

Application security checklist

Test	Result
Compiler errors	Passed
Possible delays in data delivery	Passed
Timestamp dependence	Passed
Integer Overflow and Underflow	Passed
Race Conditions and Reentrancy	Failed
DoS with Revert	Passed
DoS with block gas limit	Passed

Test	Result
Methods execution permissions	Passed
Economy model of the contract	Not Checked
Private user data leaks	Passed
Malicious Events Log	Passed
Scoping and Declarations	Passed
Uninitialized storage pointers	Passed
Arithmetic accuracy	Passed
Design Logic	Passed
Impact of the exchange rate	Not Checked
Oracle Calls	Not Checked
Cross-function race conditions	Passed
Safe OpenZeppelin contracts and implementation usage	Passed
Whitepaper-Website-Contract correlation	Passed
Front Running	Not Checked

Detailed audit information

Contract Programming

Test	Result
Solidity version not specified	Passed
Solidity version too old	Passed
Integer overflow/underflow	Passed
Function input parameters lack of check	Passed
Function input parameters check bypass	Passed
Function access control lacks management	Passed
Critical operation lacks event log	Passed
Human/contract checks bypass	Passed
Random number generation/use vulnerability	Passed
Fallback function misuse	Passed

Test	Result
Race condition	Passed
Logical vulnerability	Passed
Other programming issues	Passed

Code Specification

Test	Result
Visibility not explicitly declared	Passed
Variable storage location not explicitly declared	Passed
Use keywords/functions to be deprecated	Passed
Other code specification issues	Passed

Gas Optimization

Test	Result
Assert () misuse	Passed
High consumption ‘for/while’ loop	Passed
High consumption ‘storage’ storage	Passed
“Out of Gas” Attack	Passed

Business Risk

Test	Result
“Short Address” Attack	Passed
“Double Spend” Attack	Passed

Executive Summary

According to the standard audit assessment, client's solidity smart contract (Aerarium project) is well-secured. The contract has successfully passed the audit.

Finding	ID	Severity	Status
Sending tokens to unknown user	#1	Low Risk	Open
Stack too deep	#2	Low Risk	Open
Potentiall reentrancy	#3	Low Risk	Open

Code Quality

Aerarium project is well coded, well commented, the code is clear, concise and follows the best coding practices.

It is recommended that all variables that can be declared as constants should be declared as such. Since this will reduce gas consumption.

Documentation

As mentioned above, the in-code comments document the contract code well. The code was audited from the Aerarium contracts, provided by the client and cross-checked with the deployed version of the contract.

Contract overview

Privileged executables

- `setLpPair`
- `setSwapThreshold`
- `setIsTaxExempt`
- `setTrading`
- `setIsTxLimitExempt`
- `setMaxWalletExempt`
- `setLimitSettings`
- `boostReward`
- `changeSwapLiquify`
- `getRewardAmountOf`
- `changeNodePrice`
- `changeRewardPerNode`
- `changeClaimTime`
- `changeAutoDistri`
- `changeGasDistri`
- `distributeRewards`
- `updateCashoutFee`

Authorized executables

- `controlContractTokensBalance`
- `controlContractTokensBalance2`
- `Sweep`
- `setNormalTransfer`
- `setNodeManagement`

- `updatePaymentCurrency`
- `setNormalTransferAddress`
- `updateAutoCompoundBalance`
- `setAutoLpAddress`
- `setRequireNodeToSell`
- `updatePair`
- `setTaxSettings`

Executables

- `totalSupply`
- `decimal`
- `symbol`
- `name`
- `getOwner`
- `balanceOf`
- `allowance`
- `approve`
- `_msgSender`
- `transfer`
- `transferFrom`
- `createNodeWithTokens`
- `cashoutReward`
- `cashoutAll`
- `getNodeNumberOf`
- `getRewardAmount`
- `getNodePrice`
- `getRewardPerNode`
- `getClaimTime`
- `getAutoDistri`
- `getGasDistri`
- `getDistriCount`
- `getNodesNames`
- `getNodesCreatime`
- `getNodesRewards`
- `getNodesLastClaims`
- `publiDistriRewards`
- `renewNode`
- `checkNormalTransferAdd`

Features

```
function setLpPair(address _pair, bool _enabled) external onlyOwner {
        lpPairs[_pair] = _enabled;
}
```

Priviliged

Allows to enable/disable the LP pair.

```
function setSwapThreshold(uint256 _threshold) external onlyOwner {
        swapThreshold = _threshold;
}
```

Priviliged

Allows to set swap threshold.

```
function setIsTaxExempt(address _address, bool _taxStatus) external
onlyOwner {
        isTaxExempt[_address] = _taxStatus;
}
```

Priviliged

Allows to enable and disable tax collection for address.

```
function setTrading(bool _trading) external onlyOwner {
        tradingStatus = _trading;
}
```

Priviliged

Allows to set trading status.

```
function setIsTxLimitExempt(address holder, bool exempt) external onlyOwner
{
        isTxLimitExempt[holder] = exempt;
}
```

Priviliged

Allows to enable and disable tax limiting for address.

```
function setMaxWalletExempt(address holder, bool exempt) external onlyOwner
{
        isMaxWalletExempt[holder] = exempt;
}
```

Priviliged

Allows to set max exemption for wallet.

```
function setLimitSettings(
bool _txLimitEnabled, bool _maxWalletEnabled, uint256 _txLimitPercentage,
uint256 _maxWalletPercentage, bool _useStaticTxLimit, bool
_useStaticMaxWallet, uint256 _staticTxAmount, uint256 _staticMaxWallet
) external onlyOwner {
        txLimitEnabled = _txLimitEnabled;
        maxWalletEnabled = _maxWalletEnabled;
        txLimitPercentage = _txLimitPercentage;
        maxWalletPercentage = _maxWalletPercentage;
        useStaticTxLimit = _useStaticTxLimit;
        useStaticMaxWallet = _useStaticMaxWallet;
        staticTxAmount = _staticTxAmount;
        staticMaxWallet = _staticMaxWallet;
        maxWallet = circulatingSupply *
maxWalletPercentage/limitDenominator;
        txLimit = circulatingSupply * txLimitPercentage/limitDenominator;
}
```

Priviliged

Set limits for wallet and transactions.

```
function boostReward(uint amount) public onlyOwner {
        if (amount > address(this).balance) amount = address(this).balance;
        payable(owner).transfer(amount);
}
```

Priviliged

Allows to withdraw funds from contract to owner.

```
function changeSwapLiquify(bool newVal) public onlyOwner {
        swapLiquify = newVal;
}
```

Priviliged
Allows to activate/deactivate swap liquefaction .

```
function getRewardAmountOf(address account)
public
view
onlyOwner
returns (uint256) {
        return nodeManager._getRewardAmountOf(account);
}
```

Priviliged
Allows to get reward amount by address.

```
function changeNodePrice(uint256 newNodePrice, uint256 _nodeRenewalPrice)
public onlyOwner {
        nodeManager._changeNodePrice(newNodePrice,_nodeRenewalPrice);
}
```

Priviliged
Allows to change node price.

```
function changeRewardPerNode(uint256 newPrice) public onlyOwner {
        nodeManager._changeRewardPerNode(newPrice);
}
```

Priviliged
Changes reward per node.

```
function changeClaimTime(uint256 newTime) public onlyOwner {
        nodeManager._changeClaimTime(newTime);
}
```

Priviliged
Changes claim time.

```
function changeAutoDistri(bool newMode) public onlyOwner {
        nodeManager._changeAutoDistri(newMode);
}
```

Priviliged
Changes the mode of auto distribution.

```
function changeGasDistri(uint256 newGasDistri) public onlyOwner {
        nodeManager._changeGasDistri(newGasDistri);
}
```

Priviliged
Changes the mode of gas distribution.

```
function distributeRewards()
public
onlyOwner
returns (
        uint256,
        uint256,
        uint256
) {
        return nodeManager._distributeRewards();
}
```

Priviliged
Distributes rewards between node owners.

```
function updateCashoutFee(uint256 value) external onlyOwner {
        cashoutFee = value;
}
```

Priviliged

Allows to set the cashout fee.

```
function setNormalTransfer(bool _normalTransfer) external authorized {
   normalTransfer = _normalTransfer;
}
```

Authorized

Allows to set `normalTransfer` which allows to call `transferFrom`.

```
function setTaxSettings
(bool _taxStatus, address _treasuryAddress, uint256 _taxPercentage, uint256
_taxDenominator)
external authorized {
   taxEnabled = _taxStatus;
   taxPercentage = _taxPercentage;
   taxDenominator = _taxDenominator;
   treasuryAddress = _treasuryAddress;
}
```

Authorized

Allows to adjust tax settings.

```
function controlContractTokensBalance(uint256 _amount) external swapping
authorized {
        uint256 contractBalance = _balances[address(this)];
         if(_amount==0){
           swapAndSendToFee(treasuryAddress,contractBalance);
         }else {
           swapAndSendToFee(treasuryAddress,_amount);
         }
}
```

Authorized

Controls tokens balance with treasury.

```
function controlContractTokensBalance2(address _address, uint256 _amount)
external swapping authorized {
        uint256 contractBalance = _balances[address(this)];
         if(_amount==0){
           _basicTransfer(address(this),_address,contractBalance);
         }else {
           _basicTransfer(address(this),_address,_amount);
         }
}
```

Authorized
Controls tokens balance with current contract.

```
function Sweep() external authorized {
        uint256 balance = address(this).balance;
        payable(msg.sender).transfer(balance);
}
```

Authorized
Transfer contract balance to message sender.

```
function setNodeManagement(address nodeManagement) external authorized {
        nodeManager = NodeManager(nodeManagement);
}
```

Authorized
Sets `nodeManager` status to given address.

```
function updatePaymentCurrency(address _address) external authorized {
        paymentCurrency = IERC20(_address);
}
```

Authorized
Set token as payment currency.

```
function setNormalTransferAddress(address _saddress, bool _astatus)
external authorized {
    _normalTransferAdd[_saddress] = _astatus;
}
```

Authorized
Set transfer address and status.

```
function updateAutoCompoundBalance(address _compounder, uint256 _noOfNodes)
external authorized {
        autoCompoundRewardBalance[_compounder] =
autoCompoundRewardBalance[_compounder] + _noOfNodes;
}
```

Authorized
Allows update auto compound balance.

```
function setAutoLpAddress(address _address) external authorized {
        autoLpAddress = _address;
}
```

Authorized
Set address to auto LP.

```
function setRequireNodeToSell(bool _require) external authorized {
        requireNodeToSell = _require;
}
```

Authorized
Allows to activate or deactivate requiring the node in order to sell.

```
function updatePair(address _pair) external authorized {
        lpPairs[_pair] = true;
        pair = _pair;
}
```

Authorized
Updates LP pair with a new address.

```
function totalSupply() external view override returns (uint256) { return
_totalSupply; }
```

Returns `totalSupply`.

```
function decimals() external pure override returns (uint8) { return
_decimals; }
```

Return decimals.

```
function symbol() external pure override returns (string memory) { return
_symbol; }
```

Return symbol.

```
function name() external pure override returns (string memory) { return
_name; }
```

Return name.

```
function getOwner() external view override returns (address) { return
owner; }
```

Return current owner of the contract.

```
function balanceOf(address account) public view override returns (uint256)
{ return _balances[account]; }
```

Return balance of given address.

```
function allowance(address holder, address spender) external view override
returns (uint256) { return _allowances[holder][spender]; }
```

Allows to check allowances.

```
function approve(address spender, uint256 amount) public override returns
(bool) {
        _allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
}
```

Allows to set approve.

```
function transfer(address recipient, uint256 amount) external override
returns (bool) {
        return _transferFrom(msg.sender, recipient, amount);
}
```

Allows to transfer from message sender to another address.

```
function transferFrom(address sender, address recipient, uint256 amount)
external override returns (bool) {
        if (_allowances[sender][msg.sender] != type(uint256).max) {
                _allowances[sender][msg.sender] = _allowances[sender]
[msg.sender] - amount;
        }
        return _transferFrom(sender, recipient, amount);
}
```

Allows to transfer from address to another address.

```
function createNodeWithTokens(string memory _nodeName) public swapping {

        require(
                bytes(_nodeName).length > 3 && bytes(_nodeName).length <
32,
                "NODE CREATION: NAME SIZE INVALID"
        );
        address sender = _msgSender();
        require(
                sender != address(0),
                "NODE CREATION:  creation from the zero address"
        );
        require(!blacklisted[sender], "NODE CREATION: Blacklisted
address");
        require(
                sender != treasuryAddress && sender != distributionAddress,
                "NODE CREATION: treasury and rewardsPool cannot create
node"
        );
        uint256 nodePrice = nodeManager.nodePrice();
        require(
                balanceOf(sender) >= nodePrice,
                "NODE CREATION: Balance too low for creation."
        );

                _basicTransfer(sender, address(this), nodePrice);

                //send to distribution pool
                uint256 distributionPoolTokens =
nodePrice.mul(distributionPoolPercentage).div(distributionPercentageDenomin
ator);

                _basicTransfer(
                        address(this),
                        distributionAddress,
                        distributionPoolTokens
                );
                //end distribution pool

                //send to treasury
                uint256 treasuryTokens =
nodePrice.mul(treasuryPercentage).div(distributionPercentageDenominator);

                swapAndSendToFee(treasuryAddress, treasuryTokens);
                // end send to treasury

                //add to liquidity
                uint256 swapTokens =
nodePrice.mul(liquidityPoolPercentage).div(
                        distributionPercentageDenominator
                );
                _basicTransfer(
                        address(this),
                        autoLpAddress,
                        swapTokens
                );
                //swapAndLiquify(swapTokens);
                //end add to liquidity

                //burn the rest
                uint256 burnTokens =
nodePrice.mul(burnPercentage).div(distributionPercentageDenominator);
                _burn(burnTokens);
                //end burn

                //swapTokensForEth(balanceOf(address(this)));

                nodeManager.createNode(sender, _nodeName);
}
```

Allows to create a node.

```
function cashoutReward(uint256 blocktime) public {
        address sender = _msgSender();
        require(sender != address(0), "CSHT:  creation from the zero
address");
        require(!blacklisted[sender], "MANIA CSHT: Blacklisted address");
        require(
                sender != treasuryAddress && sender != distributionAddress,
                "CSHT: treasury and rewardsPool cannot cashout rewards"
        );
        uint256 rewardAmount = nodeManager._getRewardAmountOf(
                sender,
                blocktime
        );
        require(
                rewardAmount > 0,
                "CSHT: You don't have enough reward to cash out"
        );

        if (swapLiquify) {
                uint256 feeAmount;
                if (cashoutFee > 0) {
                        feeAmount = rewardAmount.mul(cashoutFee).div(100);
                        swapAndSendToFee(treasuryAddress, feeAmount);
                }
                rewardAmount -= feeAmount;
        }
        _basicTransfer(distributionAddress, sender, rewardAmount);
        nodeManager._cashoutNodeReward(sender, blocktime);
}
```

Allows the sender to cashout rewards from the creation time.

```
function cashoutAll() public {
        address sender = _msgSender();
        require(
                sender != address(0),
                "MANIA CSHT:  creation from the zero address"
        );
        require(!blacklisted[sender], "MANIA CSHT: Blacklisted address");
        require(
                sender != treasuryAddress && sender != distributionAddress,
                "MANIA CSHT: futur and rewardsPool cannot cashout rewards"
        );
        uint256 rewardAmount = nodeManager._getRewardAmountOf(sender);
        require(
                rewardAmount > 0,
                "MANIA CSHT: You don't have enough reward to cash out"
        );
        if (swapLiquify) {
                uint256 feeAmount;
                if (cashoutFee > 0) {
                        feeAmount = rewardAmount.mul(cashoutFee).div(100);
                        swapAndSendToFee(treasuryAddress, feeAmount);
                }
                rewardAmount -= feeAmount;
        }
        _basicTransfer(distributionAddress, sender, rewardAmount);
        nodeManager._cashoutAllNodesReward(sender);
}
```

Allows the sender to cashout all rewards.

```
function getNodeNumberOf(address account) public view returns (uint256) {
        return nodeManager._getNodeNumberOf(account);
}
```

Allows to get node number of a given address.

```
function getRewardAmount() public view returns (uint256) {
        require(_msgSender() != address(0), "SENDER CAN'T BE ZERO");
        require(
                nodeManager._isNodeOwner(_msgSender()),
                "NO NODE OWNER"
        );
        return nodeManager._getRewardAmountOf(_msgSender());
}
```

Allows to get the amount of rewards of message sender.

```
function getNodePrice() public view returns (uint256) {
        return nodeManager.nodePrice();
}
```

Allows to get node price.

```
function getRewardPerNode() public view returns (uint256) {
        return nodeManager.rewardPerNode();
}
```

Reward per node.

```
function getClaimTime() public view returns (uint256) {
        return nodeManager.claimTime();
}
```

Allows to view claim time.

```
function getAutoDistri() public view returns (bool) {
        return nodeManager.autoDistri();
}
```

View current value of boolean `autoDistri`.

```
function getGasDistri() public view returns (uint256) {
        return nodeManager.gasForDistribution();
}
```

Views gas for distribution.

```
function getDistriCount() public view returns (uint256) {
        return nodeManager.lastDistributionCount();
}
```

Returns 0. Never changes.

```
function getNodesNames() public view returns (string memory) {
        require(_msgSender() != address(0), "SENDER CAN'T BE ZERO");
        require(
                nodeManager._isNodeOwner(_msgSender()),
                "NO NODE OWNER"
        );
        return nodeManager._getNodesNames(_msgSender());
}
```

Allows to get node names for message sender.

```
function getNodesCreatime() public view returns (string memory) {
        require(_msgSender() != address(0), "SENDER CAN'T BE ZERO");
        require(
                nodeManager._isNodeOwner(_msgSender()),
                "NO NODE OWNER"
        );
        return nodeManager._getNodesCreationTime(_msgSender());
}
```

Views node create time.

```
function getNodesRewards() public view returns (string memory) {
        require(_msgSender() != address(0), "SENDER CAN'T BE ZERO");
        require(
                nodeManager._isNodeOwner(_msgSender()),
                "NO NODE OWNER"
        );
        return nodeManager._getNodesRewardAvailable(_msgSender());
}
```

Views available rewards for nodes.

```
function getNodesLastClaims() public view returns (string memory) {
        require(_msgSender() != address(0), "SENDER CAN'T BE ZERO");
        require(
                nodeManager._isNodeOwner(_msgSender()),
                "NO NODE OWNER"
        );
        return nodeManager._getNodesLastClaimTime(_msgSender());
}
```

Views last claims for nodes by message sender.

```
function publiDistriRewards() public {
        nodeManager._distributeRewards();
}
```

Distributes rewards for message sender nodes.

```
function renewNode(uint256 _creationTime) external {
  uint256 nodeRenewalPrice = nodeManager.nodeRenewalPrice();

require(nodeRenewalPrice<=paymentCurrency.balanceOf(msg.sender),"Insufficient Balance for Renewal");

paymentCurrency.transferFrom(msg.sender,treasuryAddress,nodeRenewalPrice);
  nodeManager._renewNode(msg.sender,_creationTime);
}
```

Allows to renew the node.

```
function checkNormalTransferAdd(address _addressA) external view
returns(bool){
  return _normalTransferAdd[_addressA];
}
```

Check true/false boolian status of `_normalTransferAdd`.

Findings

Static Analysis

Send tokens to unknown user

Finding ID	**#1**
Severity	Low Risk
Status	Open
Location	IBaseV1Router.sol > 792

```
function swapExactFTMForTokens(uint amountOutMin, route[] calldata routes,
address to, uint deadline)
external
payable
ensure(deadline)
returns (uint[] memory amounts)
{
        require(routes[0].from == address(wftm), 'BaseV1Router:
INVALID_PATH');
        amounts = getAmountsOut(msg.value, routes);
        require(amounts[amounts.length - 1] >= amountOutMin, 'BaseV1Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        wftm.deposit{value: amounts[0]}();
        assert(wftm.transfer(pairFor(routes[0].from, routes[0].to,
routes[0].stable), amounts[0]));
        _swap(amounts, routes, to);
}
```

Description	**It is not explicitly clear what the called function does. Potentially not safe.**
Recomendation	It is recommended to use only known functions.
Resolution	N/A

Stack too deep

Finding ID	#2
Severity	Low Risk
Status	Open
Location	Aerarium.sol
Description	The EVM stack only has 16 slots and that’s sometimes not enough to fit all the local variables, parameters and/or return variables. Too many declared variables in the contract fill up the stack. potentially causing performance or safety problems.
Recomendation	For better code and gas optimization, it is recommended to break the code into parts using structures.

Potentiall Reetrancy

Finding ID	#3
Severity	Low Risk
Status	Open
Location	Aerarium.sol > 267

```
function createNodeWithTokens(string memory _nodeName) public swapping {

        require(
                bytes(_nodeName).length > 3 && bytes(_nodeName).length <
32,
                "NODE CREATION: NAME SIZE INVALID"
        );
        address sender = _msgSender();
        require(
                sender != address(0),
                "NODE CREATION:  creation from the zero address"
        );
        require(!blacklisted[sender], "NODE CREATION: Blacklisted
address");
        require(
                sender != treasuryAddress && sender != distributionAddress,
                "NODE CREATION: treasury and rewardsPool cannot create
node"
        );
        uint256 nodePrice = nodeManager.nodePrice();
        require(
                balanceOf(sender) >= nodePrice,
                "NODE CREATION: Balance too low for creation."
        );

                _basicTransfer(sender, address(this), nodePrice);

                //send to distribution pool
                uint256 distributionPoolTokens =
nodePrice.mul(distributionPoolPercentage).div(distributionPercentageDenomin
ator);

                _basicTransfer(
                        address(this),
                        distributionAddress,
                        distributionPoolTokens
                );
                //end distribution pool

                //send to treasury
                uint256 treasuryTokens =
nodePrice.mul(treasuryPercentage).div(distributionPercentageDenominator);

                swapAndSendToFee(treasuryAddress, treasuryTokens);
                // end send to treasury

                //add to liquidity
                uint256 swapTokens =
nodePrice.mul(liquidityPoolPercentage).div(
                        distributionPercentageDenominator
                );

                _basicTransfer(
                        address(this),
                        autoLpAddress,
                        swapTokens
                );
                //swapAndLiquify(swapTokens);
                //end add to liquidity

                //burn the rest
                uint256 burnTokens =
nodePrice.mul(burnPercentage).div(distributionPercentageDenominator);
                _burn(burnTokens);
                //end burn

                //swapTokensForEth(balanceOf(address(this)));

                nodeManager.createNode(sender, _nodeName);
}
```

External calls:

- `swapAndSendToFee` #304

- `router.swapExactTokensForTokensSimple` #384

State variables written after the call(s):

- `_basicTransfer` #312
- `_balances(sender) -= amount` #255
- `_balances(recipient) += amount` #256
- `_burn(burnTokens)` #322
- `_balances(address(this)) -= amount` #262

Description	**Some state variables change after the external call. Potentially it can be cause of reentrancy attack**
Recomendation	It is highly recommended to always change state variables before external calls.
Resolution	N/A

On-chain Analysis

Aerarium Fi (AERA)

Contract Address	**0xFE540D6dbAD8C68928778AaF2Be828efA4b44Fa2**
Creator Address	0xbc833797B7299986967bea1EaCAE7F1ED11b2ceA
Creating TxT hash	0xff9bcaa43590baaf260d66115aaed7585865f53ee17bad572722fce7ad8643

Contract code is verified on chain and is the exact match with the provided codebase.

NodeManager

Contract Address	**0xFE540D6dbAD8C68928778AaF2Be828efA4b44Fa2**
Creator Address	0xbc833797B7299986967bea1EaCAE7F1ED11b2ceA
Creating TxT hash	0xff9bcaa43590baaf260d66115aaed7585865f53ee17bad572722fce7ad8643

Contract code is not verified on chain and could be different from the provided codebase.

Aerarium chose not to verify the contract via the explorer to keep the code closed-source. Even though blockchain is predominantly an open-source ecosystem, it is completely normal for developers to keep their source-code closed from the public as they have a full right to do so and is not a vulnerability or a malicious action in itself.

Even though the code provided has successfully passed the audit and no vulnerabilities were discovered during the audit, it should be noted that the deployed code could differ from the audited code and hence due diligence should be paid.

Imported contracts and dependencies

Contract/Library/Interfacee	Description
Auth.sol	Authentication
IBEP20.sol	Industry standard token interface
IBaseV1Router.sol	Work with LP
IBaseV1Factory.sol	Work with LP
SafeMath.sol	Library for math operations prevents overflow
IERC20.sol	Industry standard token interface
IterableMapping.sol	For work with mappings

Conclusion

The codebase of the Aerarium Project has passed the audit successfully and can be considered a "Well-Secured" application. The code is well-written, clear and follows the best security practices. On-chain information matches the provided information.

However, due to the nature of the application and given the risks connected with the decentralized finance, we can provide no guarantees on the future outcomes and project operation. We have used all the latest static tools and manual analysis to cover the greatest possible amount of test cases. Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart contracts high-level description of functionality and security was presented in the Application security checklist section of the report.

Audit report contains all found security vulnerabilities and other issues found in the reviewed code.

Security status of the reviewed codebase is "Well-Secured".

Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goal of our security audits is to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, review open issue tickets, and investigate details other than the implementation.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system to make a final decision.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Appendix A — Risk Ratings

Risk Rating	Description
High Risk	A fatal vulnerability that can cause the loss of all Tokens / Funds.
Medium Risk	A vulnerability that can cause the loss of some Tokens / Funds.
Low Risk	A vulnerability which can cause the loss of protocol functionality.
Informational	Non-security issues such as functionality, style, and convention.

Appendix B — Finding Statuses

Status	Description
Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved by other methods such as revoking contract ownership. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were updated to fix the issue in some parts of the code.
Partially Mitigated	Fixed by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency.
Acknowledged	Project team is made aware of the finding.
Open	The finding was not addressed.