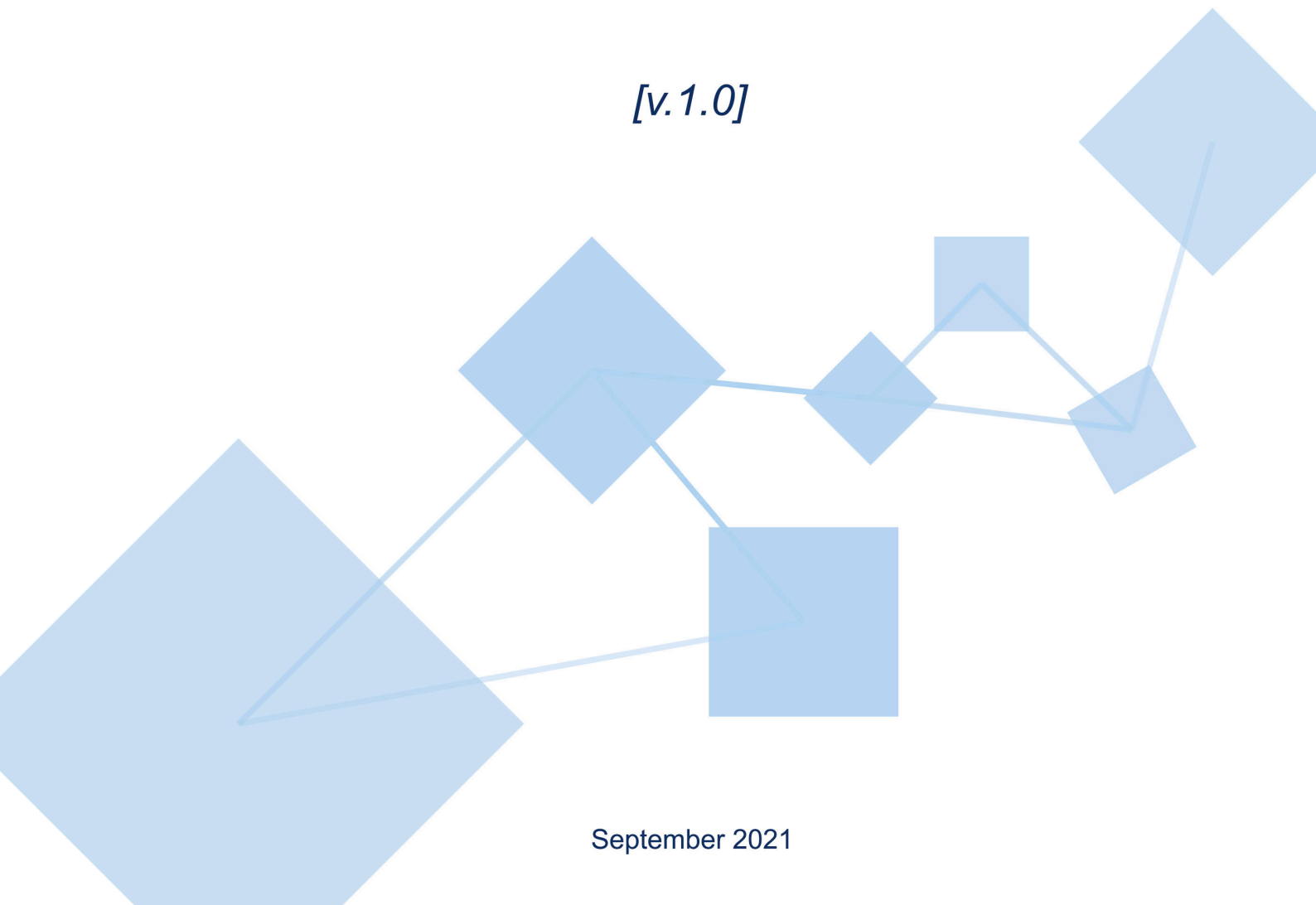




Token Security Audit Report

Prepared for Eagle Vision

[v.1.0]



September 2021

Document Properties

Client	Eagle Vision
Platform	Binance Smart Chain
Language	Solidity
Codebase	0xD4D1290c94cF5530cfB6305Aa9652BE2A24ecb32

Audit Summary

Delivery Date	20.09.2021
Audit Methodology	Static Analysis, Manual Review
Auditor(s)	Erno Patiala
Classification	Public
Version	1.0

Contact Information

Company	CheckPoint
Name	Hanna Järvinen
Telegram	t.me/checkpointreport
E-mail	contact@checkpoint.report

Remark: For more information about this document and its contents, please contact CheckPoint team

Table Of Contents

1 Executive Summary	4
2 Audit Methodology	5
3 Risk Level Classification	8
4 Project Overview	10
4.1 Communication Channels	10
4.2 Smart Contract Details	11
4.3 Contract Function Details	12
4.4 Issues Checking Status	16
4.5 Detailed Findings Information	18
5 Audit Result	21
5.1 Findings Summary	22
6 Disclaimer	23

1 Executive Summary

On 20/09/2021, CheckPoint conducted a full audit for the Eagle Vision to verify the overall security posture including a smart contract review to discover issues and vulnerabilities in the source code. Static Code Analysis, Dynamic Analysis, and Manual Review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem.

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. More information can be found in **Section 5 'Audit Result'**. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.



Eagle Vision Low Risk Level

Communication Channels

Website Content Analysis,
Social Media Listening

Smart Contract Code

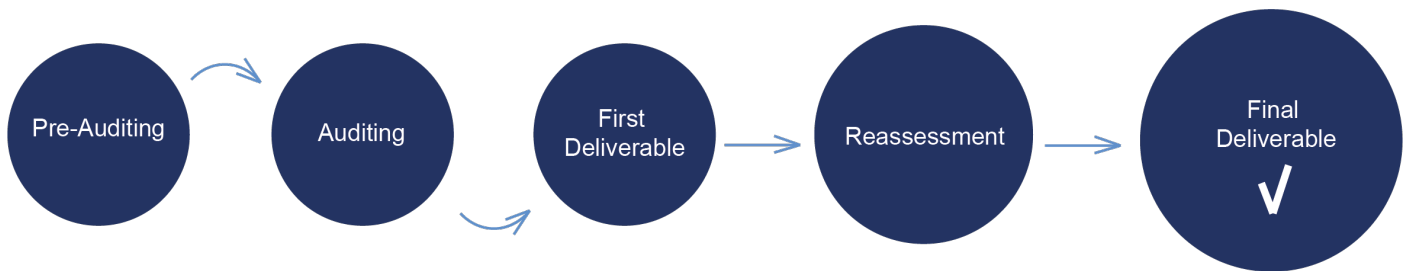
Smart Contract Details, Contract Function Details,
Issues Checking Status, Detailed Findings
Information



**THIS TOKEN PASSES CHECKPOINT'S
SECURITY VERIFICATION STANDART**



2 Audit Methodology



CheckPoint conducts the following procedure to enhance the security level of our clients' tokens:

- **Pre-Auditing**

Planning a comprehensive survey of the token, its ecosystem, possible risks & prospects, getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing.

- **Auditing**

Study of all available information about the token on the Web, inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals.

- **First Deliverable and Consulting**

Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation.

- **Reassessment**

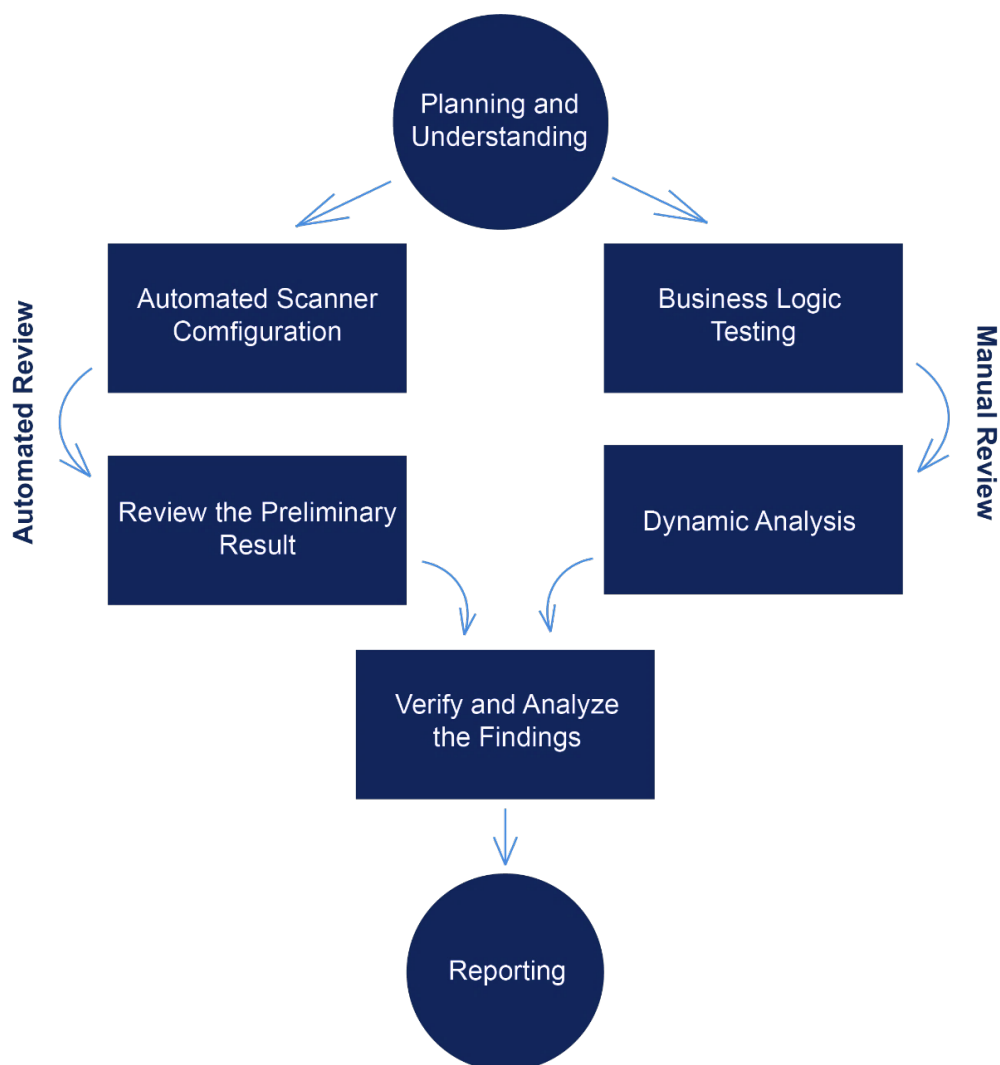
Verifying the status of the issues and whether there are any other complications in the fixes applied.

- **Final Deliverable**

Providing a full report with the detailed status of each issue.

The security audit process of CheckPoint includes three types testing:

1. Examining publicly available information about the token on social networks, including a detailed overview of the official website and analysis of the latest messages and opinions about the token.
2. Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
3. Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.



Remark: Manual and Automated review approaches can be mixed and matched including business logic analysis in terms of malicious doers' perspective

In particular, we perform the audit according to the following procedure:

- **Planning & Understanding**

- determine scope of testing and understand application purpose and workflows;
- identify key risk areas, including technical and business risks;
- determine approach – which sections to review within the resource constraints and review method – automated, manual or mixed.

- **Automated Review**

- adjust automated source code review tools to inspect the code for known unsafe coding patterns;
- verify output of the tool in order to eliminate false positive result, and if necessary, adjust and re-run the code review tool.

- **Manual Review**

- testing for business logic flaws requires thinking in unconventional methods;
- identify unsafe coding behavior via static code analysis.

- **Reporting**

- analyze the root cause of the flaws;
- recommend coding process improvements.

3 Risk Level Classification

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology:

- **Likelihood** represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- **Impact** measures the technical loss and business damage of a successful attack.
- **Severity** demonstrates the overall criticality of the risk and calculated as the product of impact and likelihood values, illustrated in a twodimensional matrix. The shading of the matrix visualizes the different risk levels.

IMPACT	Low	Weakness	Low	Medium
	Medium	Low	Medium	High
	High	Medium	High	Critical
		Low	Medium	High
		LIKELIHOOD		

Remark: Likelihood and Impact are categorized into three levels: H, M, and L, i.e., High, Medium and Low respectively. Severity is determined by likelihood and impact and can be classified into five categories accordingly, i.e., Critical, High, Medium, Low and Weakness

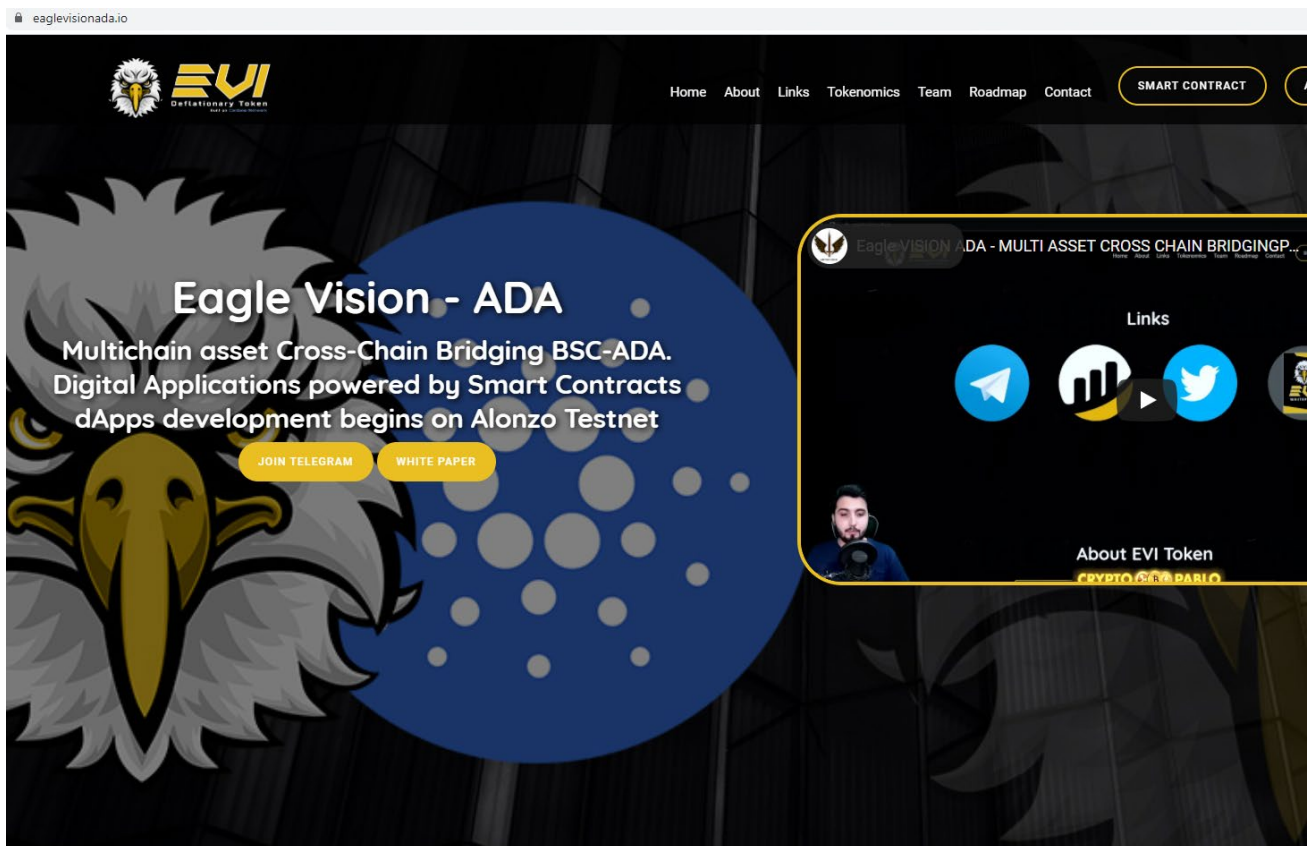
For prioritization of the vulnerabilities, we have adopted the scheme by five distinct levels for risk: Critical, High, Medium, Low, and Weakness. The risk level definitions are presented in table.

LEVEL	DESCRIPTION
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project

4 Project Overview

4.1 Communication Channels

<https://eaglevisionada.io/>



Above the image is an actual snapshot of the current live website of the project.

- | | |
|--------------------------------|---|
| ✓ Mobile Friendly | ✓ 2 Social Media Networks |
| ✓ No JavaScript Errors | ✓ 5000+ Telegram Members |
| ✓ Visionary Roadmap | ✓ 3000+ Twitter Followers |
| ✓ Spell Check | ✓ No Active Voice Chats [RISK] |
| ✓ Valid SSL Certificate | ✓ No Injected Spam and Popus Found |



Remark: This page contains active links

4.2 Smart Contract Details

Contract Name Eagle Vision

Contract Address 0xd4d1290c94cf5530cfb6305aa9652be2a24ecb32

Total Supply 100,000,000

Token Ticker EVI

Decimals 9

Token Holders 1

Transactions Count 1

Top 10 Holders Dominance 100%

Liquidity Fee 0%

Marketing Fee 0%

Burn Fee 0%

Total Fee 0%

Uniswap V2 Pair Contract 0x86482cb39d5a34b25c31d8f9bbf141bcf882d595

Contract Deployer Address 0x6b854f2b0FE987024123034F5995e226D7886a1c

Current Owner Address 0x6b854f2b0FE987024123034F5995e226D7886a1c

4.3 Contract Function Details

\$ = payable function

= non-constant function

[Int] = Internal

[Pub] = Public

[Prv] = Private

[Ext] = External

+ [Int] IERC20

- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] transfer #
- [Ext] allowance
- [Ext] approve #
- [Ext] transferFrom #

+ [Lib] SafeMath

- [Int] add
- [Int] sub
- [Int] sub
- [Int] mul
- [Int] div
- [Int] div
- [Int] mod
- [Int] mod

+ Context

- [Int] _msgSender
- [Int] _msgData

+ [Lib] Address

- [Int] isContract
- [Int] sendValue #
- [Int] functionCall #
- [Int] functionCall #
- [Int] functionCallWithValue #
- [Int] functionCallWithValue #
- [Prv] functionCallWithValue #

+ Ownable (Context)

- [Int] <Constructor> #
- [Pub] owner
- [Pub] renounceOwnership #
 - modifiers: onlyOwner
- [Pub] transferOwnership #
 - modifiers: onlyOwner
- [Pub] geUnlockTime
- [Pub] lock #

- modifiers: onlyOwner
- [Pub] unlock #

+ [Int] IUniswapV2Factory

- [Ext] feeTo
- [Ext] feeToSetter
- [Ext] getPair
- [Ext] allPairs
- [Ext] allPairsLength
- [Ext] createPair #
- [Ext] setFeeTo #
- [Ext] setFeeToSetter #

+ [Int] IUniswapV2Pair

- [Ext] name
- [Ext] symbol
- [Ext] decimals
- [Ext] totalSupply
- [Ext] balanceOf
- [Ext] allowance
- [Ext] approve #
- [Ext] transfer #
- [Ext] transferFrom #
- [Ext] DOMAIN_SEPARATOR
- [Ext] PERMIT_TYPEHASH
- [Ext] nonces
- [Ext] permit #
- [Ext] MINIMUM_LIQUIDITY
- [Ext] factory
- [Ext] token0
- [Ext] token1
- [Ext] getReserves
- [Ext] price0CumulativeLast
- [Ext] price1CumulativeLast
- [Ext] kLast
- [Ext] mint #
- [Ext] burn #
- [Ext] swap #
- [Ext] skim #
- [Ext] sync #
- [Ext] initialize #

+ [Int] IUniswapV2Router01

- [Ext] factory
- [Ext] WETH
- [Ext] addLiquidity #
- [Ext] addLiquidityETH \$
- [Ext] removeLiquidity #
- [Ext] removeLiquidityETH #
- [Ext] removeLiquidityWithPermit #

- [Ext] removeLiquidityETHWithPermit #
 - [Ext] swapExactTokensForTokens #
 - [Ext] swapTokensForExactTokens #
 - [Ext] swapExactETHForTokens \$
 - [Ext] swapTokensForExactETH #
 - [Ext] swapExactTokensForETH #
 - [Ext] swapETHForExactTokens \$
 - [Ext] quote
 - [Ext] getAmountOut
 - [Ext] getAmountIn
 - [Ext] getAmountsOut
 - [Ext] getAmountsIn
- + [Int] IUniswapV2Router02 (IUniswapV2Router01)
- [Ext] removeLiquidityETHSupportingFeeOnTransferTokens #
 - [Ext] removeLiquidityETHWithPermitSupportingFeeOnTransferTokens #
 - [Ext] swapExactTokensForTokensSupportingFeeOnTransferTokens #
 - [Ext] swapExactETHForTokensSupportingFeeOnTransferTokens \$
 - [Ext] swapExactTokensForETHSupportingFeeOnTransferTokens #
- + Eagle Vision (Context, IERC20, Ownable)
- [Pub] <Constructor> #
 - [Pub] name
 - [Pub] symbol
 - [Pub] decimals
 - [Pub] totalSupply
 - [Pub] balanceOf
 - [Pub] transfer #
 - [Pub] allowance
 - [Pub] approve #
 - [Pub] transferFrom #
 - [Pub] increaseAllowance #
 - [Pub] decreaseAllowance #
 - [Pub] isExcludedFromReward
 - [Pub] totalFees
 - [Pub] deliver #
 - [Pub] reflectionFromToken
 - [Pub] tokenFromReflection
 - [Pub] excludeFromReward #
 - modifiers: onlyOwner
 - [Ext] includeInReward #
 - modifiers: onlyOwner
 - [Prv] _transferBothExcluded #
 - [Ext] <Fallback> \$
 - [Prv] _reflectFee #
 - [Prv] _getValues
 - [Prv] _getTValues
 - [Prv] _getRValues
 - [Prv] _getRate
 - [Prv] _getCurrentSupply

- [Prv] _takeLiquidity #
- [Prv] calculateTaxFee
- [Prv] calculateLiquidityFee
- [Prv] removeAllFee #
- [Prv] restoreAllFee #
- [Pub] isExcludedFromFee
- [Prv] _approve #
- [Prv] _transfer #
- [Prv] swapAndLiquify #
 - modifiers: lockTheSwap
- [Pub] excludeFromFee #
 - modifiers: onlyOwner
- [Pub] includeInFee
 - modifiers: onlyOwner
- [Ext] enableAllFees #
 - modifiers: onlyOwner
- [Ext] disableAllFees #
 - modifiers: onlyOwner
- [Ext] setMarketingWallet #
 - modifiers: onlyOwner
- [Ext] setMaxTxPercent #
 - modifiers: onlyOwner
- [Ext] setSwapAndLiquifyEnabled #
 - modifiers: onlyOwner

4.4 Issues Checking Status

CHECKING ITEM	NOTES	RESULT
Arbitrary Jump with Function Type Variable	N / A	PASS
Arithmetic Accuracy Deviation	N / A	PASS
Assert Violation	N / A	PASS
Authorization through tx.origin	N / A	PASS
Business Logic	N / A	HIGH RISK
Code with No Effects	N / A	PASS
Critical Solidity Compiler	N / A	PASS
Delegatecall to Untrusted Callee	N / A	PASS
Design Logic	N / A	PASS
DoS with Block Gas Limit	N / A	LOW RISK
DoS with Failed Call	N / A	PASS
Function Default Visibility	N / A	PASS
Hash Collisions With MVLA	N / A	PASS
Incorrect Constructor Name	N / A	PASS
Incorrect Inheritance Order	N / A	PASS
Integer Overflows and Underflows	N / A	PASS
Lack of Proper Signature Verification	N / A	PASS
Message Call with Hardcoded Gas Amount	N / A	PASS
Missing Protection Against SRA	N / A	PASS
Presence of Unused Variables	N / A	PASS
Reentrancy	N / A	PASS
Requirement Violation	N / A	PASS

CHECKING ITEM	NOTES	RESULT
Right-To-Left-Override Control Character	N / A	PASS
Shadowing State Variables	N / A	PASS
Signature Malleability	N / A	PASS
State Variable Default Visibility	N / A	PASS
Timestamp Dependence	N / A	PASS
Transaction Order Dependence	N / A	PASS
Typographical Error	N / A	PASS
Unencrypted Private Data On-Chain	N / A	PASS
Unexpected Ether balance	N / A	PASS
Uninitialized Storage Pointer	N / A	PASS
Use of Deprecated Solidity Functions	N / A	PASS
Weak Sources of Randomness From CA	N / A	PASS
Write to Arbitrary Storage Location	N / A	PASS

Remark: To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item

4.5 Detailed Findings Information

[RISK] DoS with Block Gas Limit

- The function `includeInReward` uses the loop to find and remove addresses from the `_excluded` list. It could be aborted with out-of-gas exception if there will be a long excluded addresses list. Including an account in the reward again may result in unexpected behavior.

```
function includeInReward(address account) external onlyOwner() {
    require(!_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Recommendation: Consider removing the `includeInReward` function. If this is not desired, consider avoiding it, especially on accounts with a significant balance.

- The function `_getCurrentSupply` uses the loop for evaluating total supply. It also could be aborted with out-of-gas exception if there will be a long excluded addresses list.

```
function _getCurrentSupply() private view returns(uint256, uint256) {
    uint256 rSupply = _rTotal;
    uint256 tSupply = _tTotal;
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
        rSupply = rSupply.sub(_rOwned[_excluded[i]]);
        tSupply = tSupply.sub(_tOwned[_excluded[i]]);
    }
    if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
    return (rSupply, tSupply);
}
```

Recommendation: Consider removing the `_getCurrentSupply` function. If this is not desired, consider avoiding it, especially on accounts with a significant balance.

[RISK] Owner Privileges (in the period when the owner is not renounced)

The contract contains the following privileged functions that are restricted by the onlyOwner.

- The owner of the contract can lock and unlock. Using these functions the owner could retake privileges even after the ownership was renounced.

```
function lock(uint256 time) public virtual onlyOwner {
    _previousOwner = _owner;
    _owner = address(0);
    _lockTime = now + time;
    emit OwnershipTransferred(_owner, address(0));
}

//Unlocks the contract for owner when _LockTime is exceeds
function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock");
    require(now > _lockTime, "Contract is locked until 7 days");
    emit OwnershipTransferred(_owner, _previousOwner);
    _owner = _previousOwner;
}
```

- The owner of the contract can exclude and include accounts from fees and reward distribution.

```
function excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
```

```
function excludeFromReward(address account) public onlyOwner() {
    require(account != 0x05ffF2B0DB69458A0750badebc4f9e13aDd608C7F, 'We can not exclude Pancake router.');
```

```
    require(!_isExcluded[account], "Account is already excluded");
    if(_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

- The owner of the contract can disable and enable fees.

```
//Call this function after finalizing the presale
function enableAllFees() external onlyOwner() {
    _taxFee = 0;
    _previousTaxFee = _taxFee;
    _liquidityFee = 2;
    _previousLiquidityFee = _liquidityFee;
    _MarketingFee = 1;
    _previousMarketingFee = _MarketingFee;
    _burnFee = 2;
    _previousBurnFee = _taxFee;
    inSwapAndLiquify = true;
    emit SwapAndLiquifyEnabledUpdated(true);
}

function disableAllFees() external onlyOwner() {
    _taxFee = 0;
    _previousTaxFee = _taxFee;
    _liquidityFee = 0;
    _previousLiquidityFee = _liquidityFee;
    _burnFee = 0;
    _previousBurnFee = _taxFee;
    _MarketingFee = 0;
    _previousMarketingFee = _MarketingFee;
    inSwapAndLiquify = false;
    emit SwapAndLiquifyEnabledUpdated(false);
}
```

- The owner of the contract can set marketing wallet.

```
function setMarketingWallet(address newWallet) external onlyOwner() {
    MarketingWallet = newWallet;
}
```

- The owner of the contract can set max tx percent.

```
function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent > 10, "Cannot set transaction amount less than 10 percent!");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}
```

- The owner of the contract can set swap and liquidity enabled.

```
function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {
    swapAndLiquifyEnabled = _enabled;
    emit SwapAndLiquifyEnabledUpdated(_enabled);
}
```

5 Audit Result

LEVEL	ISSUES
High	DoS with Block Gas Limit (2)
Low	Owner Privileges (8)

1. The contract utilizes SafeMath libraries along with following the ERC20 standard.
2. There is a 'Marketing fee', a 'Liquidity fee' and a 'Burn Fee' on all transactions for any non-excluded address that participates in a transfer.
3. The owner has the ability to set and update a maximum transaction percent at any time, which will impose a limit to the number of tokens that can be transferred during any given transaction.
4. This maximum transaction amount does not apply to the owner during transactions where the owner is either the sender or the recipient.
5. The owner of the contract can exclude accounts from transfer fees and reward distribution.

6. The owner has the ability to use the 'lock' function in order to temporarily set ownership to address(0). Ownership is restored after the duration of time determined by the owner has passed and they use the unlock function. Ownership can additionally be restored (even if ownership was previously renounced), by using the unlock function a second time.

5.1 Findings Summary



Eagle Vision **Low Risk Level**

✓ **No external vulnerabilities were identified within the smart contract's code**

✓ **We strongly recommend that the team renounces ownership**

✓ **Please ensure trust in the team prior to investing as they have substantial control within the ecosystem**

✓ **We strongly recommend that the contract owners remove errors and re-audit**

6 Disclaimer

CheckPoint team issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these. For the facts that occurred or existed after the issuance, CheckPoint is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to CheckPoint by the information provider till the date of the insurance report. CheckPoint is not responsible for the background and other conditions of the project.

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), CheckPoint suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.