



Cyberscope

# Audit Report

# **PinnacleToken**

June 2024

Network     BASE

Address     0x73ae332e0c312f94ba9052b527fbcc226cf8519e

Audited by   © cyberscope

# Analysis

● Critical   ● Medium   ● Minor / Informative   ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

# Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PAMAR	Pair Address Max Amount Restriction	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

# Table of Contents

<b>Analysis</b>	<b>1</b>
<b>Diagnostics</b>	<b>2</b>
<b>Table of Contents</b>	<b>4</b>
<b>Review</b>	<b>6</b>
Audit Updates	6
Source Files	6
<b>Findings Breakdown</b>	<b>7</b>
ST - Stops Transactions	8
Description	8
Recommendation	9
ELFM - Exceeds Fees Limit	10
Description	10
Recommendation	11
AOI - Arithmetic Operations Inconsistency	12
Description	12
Recommendation	12
DDP - Decimal Division Precision	13
Description	13
Recommendation	14
MEM - Misleading Error Messages	15
Description	15
Recommendation	15
MEE - Missing Events Emission	16
Description	16
Recommendation	16
PAMAR - Pair Address Max Amount Restriction	17
Description	17
Recommendation	17
PLPI - Potential Liquidity Provision Inadequacy	18
Description	18
Recommendation	19
PVC - Price Volatility Concern	20
Description	20
Recommendation	20
RRS - Redundant Require Statement	21
Description	21
Recommendation	21
RSML - Redundant SafeMath Library	22
Description	22

Recommendation	22
RSW - Redundant Storage Writes	23
Description	23
Recommendation	23
OCTD - Transfers Contract's Tokens	24
Description	24
Recommendation	24
L04 - Conformance to Solidity Naming Conventions	25
Description	25
Recommendation	26
L05 - Unused State Variable	27
Description	27
Recommendation	27
L07 - Missing Events Arithmetic	28
Description	28
Recommendation	28
L09 - Dead Code Elimination	29
Description	29
Recommendation	30
L13 - Divide before Multiply Operation	31
Description	31
Recommendation	31
L15 - Local Scope Variable Shadowing	32
Description	32
Recommendation	32
L16 - Validate Variable Setters	33
Description	33
Recommendation	33
L20 - Succeeded Transfer Check	34
Description	34
Recommendation	34
<b>Functions Analysis</b>	<b>35</b>
<b>Inheritance Graph</b>	<b>40</b>
<b>Flow Graph</b>	<b>41</b>
<b>Summary</b>	<b>42</b>
<b>Disclaimer</b>	<b>43</b>
<b>About Cyberscope</b>	<b>44</b>

## Review

Contract Name	PinnacleToken
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Testing Deploy	<a href="https://testnet.bscscan.com/address/0x7213b05702297bb45a20fc679c6606cdd38b6267">https://testnet.bscscan.com/address/0x7213b05702297bb45a20fc679c6606cdd38b6267</a>
Explorer	<a href="https://basescan.org/address/0x73ae332e0c312f94ba9052b527fbcc226cf8519e">https://basescan.org/address/0x73ae332e0c312f94ba9052b527fbcc226cf8519e</a>
Address	0x73ae332e0c312f94ba9052b527fbcc226cf8519e
Network	BASE
Symbol	PNC
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Yes

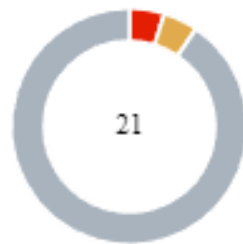
## Audit Updates

Initial Audit	14 Jun 2024
---------------	-------------

## Source Files

Filename	SHA256
contracts/PinnacleToken.sol	6b042a5318dda45d26f5e7ae3f07c3d4283d42449931dd4d9cb7873c4ac90a76

## Findings Breakdown



Critical	1
Medium	1
Minor / Informative	19

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	1	0	0	0
Minor / Informative	19	0	0	0



## ST - Stops Transactions

Criticality	Critical
Location	contracts/PinnacleToken.sol#L607,700
Status	Unresolved

### Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (!isTrading) {
    require(!_isExcludedFromFees[sender] ||
_isExcludedFromFees[recipient], "Trading is not active.");
}

function swapTrading() external onlyOwner {
    isTrading = true;
    swapEnabled = true;
    taxTill = block.number + 10;
}
```

## Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

### Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

### Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

## ELFM - Exceeds Fees Limit

Criticality	Medium
Location	contracts/PinnacleToken.sol#L607,744
Status	Unresolved

### Description

The contract owner has the authority to increase over the allowed limit of 25%. Specifically, The `swapTrading` function can be called by the owner an unlimited number of times, each time resetting the `taxTill` variable to the current block number plus ten. This functionality, combined with the `_transfer` method, means that any transfers made within the next 10 blocks after the owner calls `swapTrading` will incur a 99% fee. This high fee structure, although temporary, can be repeatedly triggered by the owner, creating a situation where normal users are subjected to very high fees.

```
function swapTrading() external onlyOwner {
    isTrading = true;
    swapEnabled = true;
    taxTill = block.number + 10;
}

if (takeFee) {
    uint256 fees = 0;
    if (block.number < taxTill) {
        fees = amount.mul(99).div(100);
        tokensForMarketing += (fees * 94) / 99;
        tokensForDev += (fees * 5) / 99;
    }
}
```

## Recommendation

The contract could embody a check that limits the frequency with which the `swapTrading` function can be called. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

### Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

### Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

## AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L582,824...
Status	Unresolved

### Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, \*, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
_fees.buyTotalFees = _fees.buyMarketingFee +  
_fees.buyLiquidityFee + _fees.buyDevFee;  
  
uint256 ethForMarketing =  
newBalance.mul(tokensForMarketing).div(toSwap);
```

### Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

## DDP - Decimal Division Precision

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L747,751,758
<b>Status</b>	Unresolved

### Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
fees = amount.mul(99).div(100);
tokensForMarketing += (fees * 94) / 99;
tokensForDev += (fees * 5) / 99;

fees = amount.mul(_fees.sellTotalFees).div(100);
tokensForLiquidity += fees * _fees.sellLiquidityFee /
_feess.sellTotalFees;
tokensForMarketing += fees * _fees.sellMarketingFee /
_feess.sellTotalFees;
tokensForDev += fees * _fees.sellDevFee / _fees.sellTotalFees;

fees = amount.mul(_fees.buyTotalFees).div(100);
tokensForLiquidity += fees * _fees.buyLiquidityFee /
_feess.buyTotalFees;
tokensForMarketing += fees * _fees.buyMarketingFee /
_feess.buyTotalFees;
tokensForDev += fees * _fees.buyDevFee / _fees.buyTotalFees;
```

## Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

## MEM - Misleading Error Messages

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L645,646
<b>Status</b>	Unresolved

### Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_fees.buyTotalFees <= 10, "Must keep fees at 30% or less");  
require(_fees.sellTotalFees <= 10, "Must keep fees at 30% or less");
```

### Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.



## MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L635,650,653,656,661,673
Status	Unresolved

### Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function excludeFromFees(address account, bool excluded) public
onlyOwner {
    _isExcludedFromFees[account] = excluded;
}
function excludeFromWalletLimit(address account, bool excluded)
public onlyOwner {
    _isExcludedMaxWalletAmount[account] = excluded;
}
function excludeFromMaxTransaction(address updAds, bool isEx)
public onlyOwner {
    _isExcludedMaxTransactionAmount[updAds] = isEx;
}

...
```

### Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

## PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L653,710
Status	Unresolved

### Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
if (!_isExcludedMaxWalletAmount[recipient]) {
    require(amount + balanceOf(recipient) <= maxWalletAmount, "Max
wallet exceeded");
}

function excludeFromWalletLimit(address account, bool excluded) public
onlyOwner {
    _isExcludedMaxWalletAmount[account] = excluded;
}
```

### Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

## PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L775
Status	Unresolved

### Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
function swapTokensForEth(uint256 tAmount) private {  
  
    // generate the uniswap pair path of token -> weth  
    address[] memory path = new address[] (2);  
    path[0] = address(this);  
    path[1] = router.WETH();  
  
    _approve(address(this), address(router), tAmount);  
  
    // make the swap  
    router.swapExactTokensForETHSupportingFeeOnTransferTokens(  
        tAmount,  
        0, // accept any amount of ETH  
        path,  
        address(this),  
        block.timestamp  
    );  
  
}
```

## Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

## PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L616
Status	Unresolved

### Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `thresholdSwapAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function updateThresholdSwapAmount(uint256 newAmount) external
onlyOwner returns(bool) {
    thresholdSwapAmount = newAmount;
    return true;
}
```

### Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

## RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L186
Status	Unresolved

### Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
    return c;
}
```

### Recommendation

It is recommended to remove the `require` statement from the `add` function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

## RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol
Status	Unresolved

### Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

### Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

## RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L635,650,653,656,661,673
Status	Unresolved

### Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function excludeFromFees(address account, bool excluded) public
onlyOwner {
    _isExcludedFromFees[account] = excluded;
}

function excludeFromWalletLimit(address account, bool excluded)
public onlyOwner {
    _isExcludedMaxWalletAmount[account] = excluded;
}

function excludeFromMaxTransaction(address updAds, bool isEx)
public onlyOwner {
    _isExcludedMaxTransactionAmount[updAds] = isEx;
}

...
```

### Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.



## OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L669
Status	Unresolved

### Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueERC20` function.

```
function rescueERC20(address tokenAdd, uint256 amount) external  
onlyOwner {  
    IERC20(tokenAdd).transfer(owner(), amount);  
}
```

### Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

## L04 - Conformance to Solidity Naming Conventions

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L347,516,536,537,635,673
<b>Status</b>	Unresolved

### Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX\_VALUE, ERROR\_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);

Fees public _fees = Fees({
    buyTotalFees: 0,
    buyMarketingFee: 0,
    buyDevFee:0,
    buyLiquidityFee: 0,

    sellTotalFees: 0,
    sellMarketingFee: 0,
    sellDevFee:0,
    sellLiquidityFee: 0
})
mapping(address => bool) public _isExcludedMaxTransactionAmount
mapping(address => bool) public _isExcludedMaxWalletAmount

...
```

## Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

## L05 - Unused State Variable

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L282
<b>Status</b>	Unresolved

### Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

### Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

## L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L617,625,632
Status	Unresolved

### Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
thresholdSwapAmount = newAmount
maxBuyAmount = (totalSupply() * newMaxBuy) / 1000
maxWalletAmount = (totalSupply() * newPercentage) / 1000
```

### Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

## L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L324,330,337
Status	Unresolved

### Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function abs(int256 a) internal pure returns(int256) {
    require(a != MIN_INT256);
    return a < 0 ? -a : a;
}

function toUint256Safe(int256 a) internal pure returns(uint256)
{
    ...
}

function toInt256Safe(uint256 a) internal pure returns(int256)
{
    int256 b = int256(a);
    require(b >= 0);
    return b;
}
```

## Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

## L13 - Divide before Multiply Operation

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L747,748,749,751,752,753,754,758,759,760,761
<b>Status</b>	Unresolved

### Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of precision.

```
fees = amount.mul(_fees.buyTotalFees).div(100)
tokensForMarketing += fees * _fees.buyMarketingFee /
_feess.buyTotalFees
```

### Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.



## L15 - Local Scope Variable Shadowing

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L572
<b>Status</b>	Unresolved

### Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 totalSupply = 1e8 * 1e18
```

### Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

## L16 - Validate Variable Setters

<b>Criticality</b>	Minor / Informative
<b>Location</b>	contracts/PinnacleToken.sol#L674,675
<b>Status</b>	Unresolved

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
marketingWallet = _marketingWallet  
devWallet = _devWallet
```

### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

## L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/PinnacleToken.sol#L670
Status	Unresolved

### Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(tokenAdd).transfer(owner(), amount)
```

### Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](#).

## Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>IUniswapV2Factory</b>	Interface			
	createPair	External	✓	-
<b>IERC20</b>	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>IERC20Metadata</b>	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
<b>Context</b>	Implementation			
	_msgSender	Internal		

ERC20	Implementation	Context, IERC20, IERC20Meta data		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		

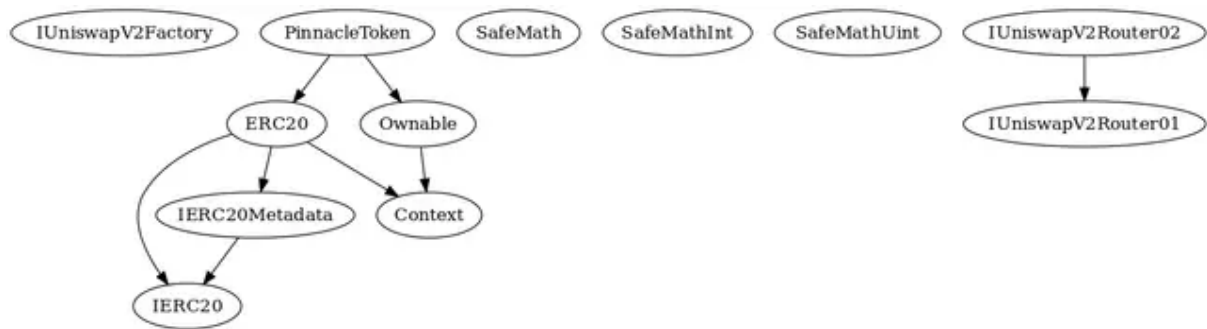
	div	Internal		
	div	Internal		
<b>Ownable</b>	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
<b>SafeMathInt</b>	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toInt256Safe	Internal		
<b>SafeMathUint</b>	Library			
	toInt256Safe	Internal		
<b>IUniswapV2Router01</b>	Interface			
	factory	External		-
	WETH	External		-

	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
<b>IUniswapV2Router02</b>	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

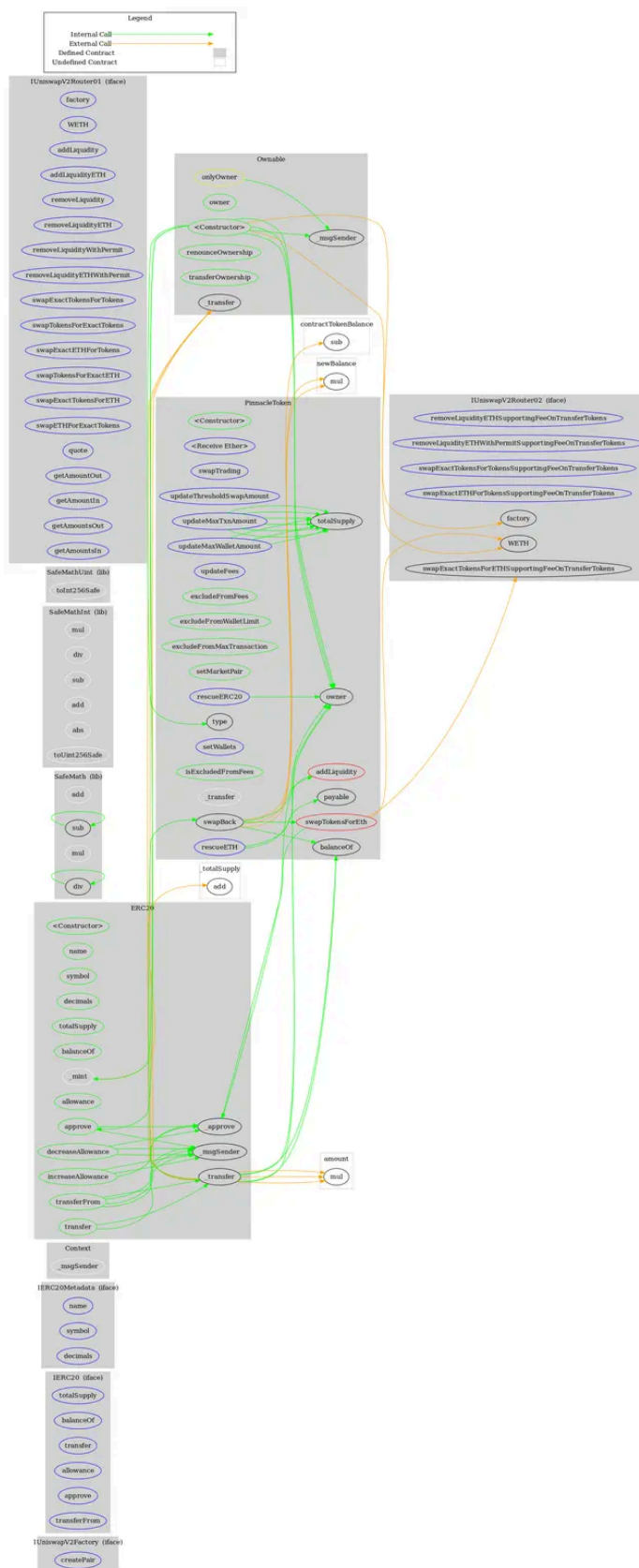
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
<b>PinnacleToken</b>	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	swapTrading	External	✓	onlyOwner
	updateThresholdSwapAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	updateFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeFromWalletLimit	Public	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	setMarketPair	Public	✓	onlyOwner
	rescueETH	External	✓	onlyOwner
	rescueERC20	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	



## Inheritance Graph



# Flow Graph



## Summary

PinnacleToken contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 10% fees.

## Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

# About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



**The Cyberscope team**

<https://www.cyberscope.io>