Wakehacker Report

Contract address	0x45e57907058c707a068100de358ba4535b18e2f3
Chain ID	56
Report version	1.0
Last edit	November 18, 2025

"Static analysis or stay rekt"

Document Revisions

1.0	Wakehacker Report - 0x45e57907058c707a068100de358ba4535b18e2f3 on chain 56	18.11.2025

Contents

Overview

This report was generated using Wakehacker, an automated vulnerability analysis tool. Wakehacker utilizes Wake with additional detectors to perform comprehensive AI and static analysis.

To identify potential vulnerabilities and issues in smart contracts Wake framework utilizes:

- Code structure and patterns
- Control flow graph
- Data flow graph
- Common vulnerability patterns
- Contract interactions

The findings presented in this report are based on automated analysis optimized for precision, aiming for a low false-positive rate. The detection is not optimized for recall—it doesn't target finding all issues (which come at the cost of a high false-positive rate). This code review should be complemented with additional manual code review for a complete security assessment.

Disclaimer

The best effort has been put into finding known vulnerabilities in the system, however automated findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Finding Classification

Each finding is classified by two independent ratings: *Impact* and *Confidence*.

Impact

Measuring the potential consequences of the issue on the system.

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue represents a potential security concern in the code structure or logic that could become problematic with code modifications.
- **Info** - The issue relates to code quality practices that may affect security. Examples include insufficient logging for critical operations or inconsistent error handling patterns.

Confidence

Indicating the probability that the identified issue is a valid security concern.

- **High** - The analysis has identified a pattern that strongly indicates the presence of the issue.
- **Medium** - Evidence suggests the issue exists, but manual verification is recommended.
- **Low** - Potential indicators of the issue have been detected, but there is a significant possibility of false positives.

Executive Summary

Revision 1.0

Report Description

The audit of the MEFAI token contract (0x45e57907058c707a068100de358ba4535b18e2f3) on BSC Chain identified a total of 12 detections across 2 unique detector categories. The analysis revealed 2 critical detections that require immediate attention, along with 3 high impact findings that pose significant risks to the protocol's security and functionality. With 11 high confidence detections, the majority of identified issues are substantiated and warrant thorough review by the development team. The critical findings represent serious vulnerabilities that could potentially compromise the contract's integrity or user funds, making them priority items for remediation before any production deployment or continued operation.

Findings

Summary by Impact

0	1	3	5	3	12
High	Medium	Low	Warning	Info	Total

Complete List

Finding Title	Impact	Reported	Status
M1: Race condition vulnerability in approve function	Medium	1.0	Reported
L1: Missing event emission for fee exclusion changes	Low	1.0	Reported
L2: Non-standard ERC20 behavior blocking zero transfers	Low	1.0	Reported
L3: Potential precision loss in fee calculation	Low	1.0	Reported
W1: Centralized initial supply distribution	Warning	1.0	Reported
W2: Fee-on-Transfer Breaking ERC20 Transfer Amount Invariant	Warning	1.0	Reported
W3: Centralization risk with non-renounced ownership	Warning	1.0	Reported
W4: MEV Sandwich Attack Amplification Through Selective Fee Exclusion	Warning	1.0	Reported

W5: Permanent loss of fee management capability	Warning	1.0	Reported
I1: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I2: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I3: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported

M1: Race condition vulnerability in approve function

Impact	Medium	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken inherits the standard ERC20 approve function without implementing any protection against the well-known approval race condition vulnerability. When a user attempts to change an existing non-zero allowance to a different non-zero value, an attacker can exploit the time window between transaction submission and confirmation to double-spend allowances.

Listing 1. Code snippet from contracts/Source.sol

```
489 function approve(address spender, uint256 value) public virtual returns (
    bool) {
490     address owner = _msgSender();
491     _approve(owner, spender, value);  // Direct overwrite of allowance
492     return true;
493 }
```

The vulnerability occurs because:

- the approve function directly overwrites the existing allowance;
- no atomic comparison or validation of the previous value;
- no implementation of safer patterns like increaseAllowance/decreaseAllowance;
- front-runners can observe pending transactions and exploit the race condition; and

- the attack is particularly damaging with high-value allowances.

Go back to Findings Summary

L1: Missing event emission for fee exclusion changes

Impact	Low	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The excludeFromFee function modifies critical protocol state by changing fee exclusion status for addresses, but fails to emit any event. This creates a transparency gap where off-chain services, block explorers, and users cannot track these important state changes.

Listing 2. Code snippet from contracts/Source.sol

```
844 function excludeFromFee(address account, bool excluded) external onlyOwner {
845     _isExcludedFromFee[account] = excluded;  // No event emitted
846 }
```

The lack of event emission causes:

- inability to track fee exclusion history on-chain;
- no transparency for when addresses gain or lose fee privileges;
- difficulty auditing owner actions and potential abuse;
- wallets and DEXs cannot detect fee status changes; and
- no notification mechanism for affected addresses.

Go back to Findings Summary

L2: Non-standard ERC20 behavior blocking zero transfers

Impact	Low	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken explicitly blocks zero-amount transfers, deviating from the ERC20 standard. The standard ERC20 specification allows and expects zero-amount transfers to be valid operations. Many DeFi protocols rely on this behavior for various purposes.

Listing 3. Code snippet from contracts/Source.sol

```
825 function _validateTransfer(address sender, address recipient, uint256 amount)
    private pure {
826     require(sender != address(0), "Transfer from zero address");
827     require(recipient != address(0), "Transfer to zero address");
828     require(amount != 0, "Transfer amount must be positive");  // Blocks zero
    transfers
829 }
```

This non-standard behavior causes:

- integration failures with protocols expecting standard ERC20 behavior;
- breaking of state update mechanisms that use zero transfers;
- incompatibility with yield aggregators and lending protocols;
- potential for locked funds in smart contract integrations; and
- failed transactions where zero transfers are used for signaling.

Go back to Findings Summary

L3: Potential precision loss in fee calculation

Impact	Low	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The fee calculation uses integer division which results in precision loss for small transfer amounts. Transfers of less than 100 tokens will have a fee of 0, allowing users to bypass the fee mechanism with small transfers.

Listing 4. Code snippet from contracts/Source.sol

```
834 if (takeFee) {
835     uint256 feeAmount = amount * FEE_PERCENT / 100;  // 0 fee for amounts <
    100
836     uint256 transferAmount = amount - feeAmount;
837     _transfer(sender, recipient, transferAmount);
838     _transfer(sender, teamWallet, feeAmount);
839 }
```

Go back to Findings Summary

W1: Centralized initial supply distribution

Impact	Warning	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken contract mints the entire token supply of 1 billion tokens to the deployer address in a single transaction during deployment. This creates a highly centralized token distribution where one address controls 100% of the circulating supply.

Listing 5. Code snippet from contracts/Source.sol

```
795 constructor(address initialTeamWallet) ERC20("META FINANCIAL AI", "MEFAI")
    Ownable(msg.sender) {
796     require(initialTeamWallet != address(0), "Team wallet cannot be zero");
797     teamWallet = initialTeamWallet;
798     _mint(msg.sender, MAX_SUPPLY);  // Entire supply to deployer
799
800     _isExcludedFromFee[msg.sender] = true;  // Deployer excluded from fees
801     _isExcludedFromFee[initialTeamWallet] = true;
802 }
```

The centralized distribution is exacerbated by:

- the deployer is excluded from the 1% transfer fee, giving them an unfair trading advantage;
- no vesting schedule or distribution mechanism is implemented;
- no multi-sig or time-lock controls are enforced on the deployer's tokens; and
- the entire supply is liquid immediately upon deployment.

> Regarding the distribution of the 1 billion MEFAI supply, this was entirely released to the market during the presale. Therefore, such a centralized

> distribution is no longer the case, and it is now organically distributed to more than 10,000 hodlers.
>
> — MEFAI Team Response

Go back to Findings Summary

W2: Fee-on-Transfer Breaking ERC20 Transfer Amount Invariant

Impact	Warning	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken implements a fee-on-transfer mechanism that violates the fundamental ERC20 standard invariant. When `transferFrom(sender, recipient, amount)` is called, the ERC20 standard requires that the recipient receives exactly `amount` tokens. However, this implementation applies a 1% fee, causing the recipient to receive only 99% of the specified amount.

Listing 6. Code snippet from contracts/Source.sol showing the fee mechanism

```
831 function _transferWithFee(address sender, address recipient, uint256 amount)
    private {
832     bool takeFee = !(_isExcludedFromFee[sender] || _isExcludedFromFee
    [recipient]);
833
834     if (takeFee) {
835         uint256 feeAmount = amount * FEE_PERCENT / 100;
836         uint256 transferAmount = amount - feeAmount;
837         _transfer(sender, recipient, transferAmount);  // Only 99% to
    recipient
838         _transfer(sender, teamWallet, feeAmount);       // 1% to teamWallet
839     } else {
840         _transfer(sender, recipient, amount);
841     }
842 }
```

This violation has severe consequences:

- DEX routers will fail when expecting exact output amounts;
- lending protocols will calculate incorrect collateral ratios;
- escrow contracts will have accounting mismatches;
- multi-step transactions will compound the fee losses; and
- smart contract integrations will break due to unexpected balance changes.

Go back to Findings Summary

W3: Centralization risk with non-renounced ownership

Impact	Warning	Confidence	Medium
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken maintains centralized control over fee exclusions through the owner-controlled excludeFromFee function. Despite potential documentation or comments suggesting ownership renunciation, the code does not enforce this critical security step, allowing the owner to maintain indefinite control over fee collection mechanisms.

Listing 7. Code snippet from contracts/Source.sol

```
844 function excludeFromFee(address account, bool excluded) external onlyOwner {
845     _isExcludedFromFee[account] = excluded;  // Unrestricted owner control
846 }
```

This centralization creates multiple risks:

- owner can arbitrarily grant or revoke fee exemptions;
- no time-lock or governance controls on exclusion changes;
- no transparency requirements (no events emitted);
- creates trust dependency on owner behavior; and
- enables selective manipulation of fee collection.

Note: This issue overlaps significantly with the MEV sandwich attack amplification vulnerability, representing the broader centralization risk of the same underlying mechanism.

> It is correct that there is a 1% fee. As can be verified, only the initial contract creator and the 1% fee wallet are exempt. Other exempt wallets were specified as comments within the contract code, and this is how exchanges were made exempt from the fee. Since the contract ownership has been renounced, there is no longer a chance for anyone to add a new wallet to the whitelist.
>
> Additionally, some other points are directly related to contract ownership. In MEFAI's case, after the presale distribution (the Pinksale wallet address was also exempted from the fee), the ownership was renounced. Because it doesn't operate like a bridge or protocol and no longer has an owner, these aspects cannot be changed, and therefore, there is no longer any attack risk.
>
> — MEFAI Team Response

Go back to Findings Summary

W4: MEV Sandwich Attack Amplification Through Selective Fee Exclusion

Impact	Warning	Confidence	Low
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken allows the owner to arbitrarily grant fee immunity to any address through the `excludeFromFee` function. This creates a significant MEV amplification vector where sandwich attackers can gain a 1% profit advantage over regular traders, making their attacks more profitable and damaging to users.

Listing 8. Code snippet from contracts/Source.sol

```
844 function excludeFromFee(address account, bool excluded) external onlyOwner {
845     _isExcludedFromFee[account] = excluded;  // No restrictions or validation
846 }
```

Listing 9. Fee mechanism giving excluded addresses advantage

```
831 function _transferWithFee(address sender, address recipient, uint256 amount)
    private {
832     bool takeFee = !(_isExcludedFromFee[sender] || _isExcludedFromFee
    [recipient]);
833
834     if (takeFee) {
835         uint256 feeAmount = amount * FEE_PERCENT / 100;  // 1% fee for
    regular users
836         uint256 transferAmount = amount - feeAmount;
837         _transfer(sender, recipient, transferAmount);
838         _transfer(sender, teamWallet, feeAmount);
839     } else {
840         _transfer(sender, recipient, amount);  // No fee for excluded
    addresses
841     }
```

```
842 }
```

The vulnerability enables:

- MEV bots to execute sandwich attacks with 1% higher profit margins;
- owner to monetize by selling fee exclusions to MEV operators;
- creation of a two-tier system where privileged addresses trade fee-free;
- amplified losses for regular users who become sandwich attack victims; and
- unfair market dynamics favoring colluding parties.

> After the presale distribution, the ownership was renounced. Since the contract no longer has an owner and doesn't operate like a bridge or protocol, these aspects cannot be evolved or changed.
>
> — MEFAI Team Response

Go back to Findings Summary

W5: Permanent loss of fee management capability

Impact	Warning	Confidence	Low
Target	./contracts/Source.sol	Detection	Wake AI

Description

The CertifiedSecureToken implements a renounceOwnership function that permanently transfers ownership to the zero address. Once executed, this action is irreversible and prevents any future modifications to the fee exclusion list, potentially causing integration issues with exchanges and DeFi protocols.

Listing 10. Code snippet from contracts/Source.sol

```
821 function renounceOwnership() public override onlyOwner {
822     _transferOwnership(address(0));  // Permanent transfer to zero address
823 }
```

Listing 11. Fee exclusion function requiring owner privileges

```
844 function excludeFromFee(address account, bool excluded) external onlyOwner {
845     _isExcludedFromFee[account] = excluded;
846 }
```

The permanent loss of ownership creates several risks:

- new exchange listings cannot be added to fee exclusion list;
- existing exclusions cannot be modified if circumstances change;
- protocol cannot adapt to changing market conditions;
- integration issues with future DeFi protocols requiring fee-free transfers; and
- no recovery mechanism if critical addresses need exclusion.

> After the presale distribution, the ownership was renounced. Since the contract no longer has an owner and doesn't operate like a bridge or protocol, these aspects cannot be evolved or changed.
>
> — MEFAI Team Response

Go back to Findings Summary

I1: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	Medium
Target	Source	Detection	Unused public functionality

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 12. Function is declared as public but could be external since it has no internal references in contracts/Source.sol (L806-L810)

```
804 }
805
806 function transfer(address recipient, uint256 amount) public override returns
    (bool) {
807     _validateTransfer(_msgSender(), recipient, amount);
808     _transferWithFee(_msgSender(), recipient, amount);
809     return true;
```

Go back to Findings Summary

I2: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	Medium
Target	Source	Detection	Unused public functionality

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 13. Function is declared as public but could be external since it has no internal references in contracts/Source.sol (L812-L819)

```
809     return true;
810 }
811
812 function transferFrom(address sender, address recipient, uint256 amount)
    public override returns (bool) {
813     _validateTransfer(sender, recipient, amount);
814     _transferWithFee(sender, recipient, amount);
815     uint256 currentAllowance = allowance(sender, _msgSender());
```

Go back to Findings Summary

I3: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	Medium
Target	Source	Detection	Unused public functionality

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 14. Function is declared as public but could be external since it has no internal references in contracts/Source.sol (L821-L823)

```
818     return true;
819 }
820
821 function renounceOwnership() public override onlyOwner {
822     _transferOwnership(address(0));
823 }
```

Go back to Findings Summary

”Static analysis or stay rekt”

https://wakehacker.ai