Wakehacker Report

Contract address	0xcb21311d3b91b5324f6c11b4f5a656fcacbff122
Chain ID	1
Report version	1.0
Last edit	October 23, 2025

"Static analysis or stay rekt"

Document Revisions

1.0	Wakehacker Report - 0xcb21311d3b91b5324f6c11b4f5a656fcacbff122 on chain 1	23.10.2025

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

The security audit of Quantix Capital's smart contract revealed 4 total detections from 2 unique detectors, all with high confidence levels. While no high-impact vulnerabilities were identified, one warning-level detection was found. The contract, deployed on Ethereum mainnet, demonstrates relatively low detection counts overall, suggesting a generally sound implementation. However, the presence of warning-level findings indicates areas that could benefit from further review and potential improvements.

Findings

Summary by Impact

0	1	1	1	1	4
High	Medium	Low	Warning	Info	Total

Complete List

Finding Title	Impact	Reported	Status
M1: Direct approval to non-zero value exposes users to approval race condition	Medium	1.0	Reported
L1: Lack of increaseAllowance and decreaseAllowance functions	Low	1.0	Reported
W1: Economic centralization enables sophisticated market manipulation and death spiral attacks	Warning	1.0	Reported
I1: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported

M1: Direct approval to non-zero value exposes users to approval race condition

Impact	Medium	Confidence	High
Target	./src/contracts/ERC20/qai.sol	Detection	Wake AI

Description

The QuantixAIERC20 contract inherits the standard ERC20 `approve` function from OpenZeppelin, which contains a well-known race condition vulnerability. When a user changes an existing non-zero approval to a different non-zero value, a malicious spender can front-run the transaction to spend both the old and new allowance amounts.

The vulnerability occurs because the `approve` function directly sets the new allowance value without checking the current state. This creates a window of opportunity where:

- the spender sees a pending approval change in the mempool;
- the spender front-runs with a `transferFrom` using the old allowance;
- the approval change executes, setting the new allowance; and
- the spender can now spend the new allowance as well.

Listing 1. Code snippet from @openzeppelin/contracts/token/ERC20/ERC20.sol

```
132 function approve(address spender, uint256 value) public virtual returns (
    bool) {
133     address owner = _msgSender();
134     _approve(owner, spender, value);
135     return true;
136 }
```

This is a fundamental issue with the ERC20 standard that affects many tokens. The severity depends on user behavior and whether they change non-zero approvals directly.

Go back to Findings Summary

L1: Lack of increaseAllowance and decreaseAllowance functions

Impact	Low	Confidence	High
Target	./src/contracts/ERC20/qai.sol	Detection	Wake AI

Description

The QuantixAIERC20 contract does not expose the safer `increaseAllowance` and `decreaseAllowance` functions that help mitigate approval race conditions. These functions are available in modern OpenZeppelin ERC20 implementations but are not included in this minimal contract.

The absence of these functions forces users to rely solely on the vulnerable `approve` function, which is susceptible to front-running attacks when changing non-zero allowances. The `increaseAllowance` and `decreaseAllowance` functions modify allowances relative to current values rather than setting absolute values, eliminating the race condition vulnerability.

Listing 2. Code snippet from src/contracts/ERC20/qai.sol

```
27 contract QuantixAIERC20 is ERC20, Ownable {
28     constructor() ERC20("QuantixAI", "QAI") Ownable(msg.sender) {
29         _mint(msg.sender, 10000000 * 10 ** 18);
30     }
31
32     function renounceOwnership() public virtual override onlyOwner {
33         super.renounceOwnership();
34     }
35 }
```

The contract only inherits the basic ERC20 implementation without any safety enhancements. This compounds the approval race condition vulnerability by not

providing users with safer alternatives.

Go back to Findings Summary

W1: Economic centralization enables sophisticated market manipulation and death spiral attacks

Impact	Warning	Confidence	High
Target	./src/contracts/ERC20/qai.sol	Detection	Wake AI

Description

The QuantixAIERC20 contract mints 100% of the total token supply (10 million tokens) directly to the deployer address in the constructor, creating extreme economic centralization. This design choice introduces severe trust assumptions and enables multiple attack vectors.

The complete concentration of tokens in a single address allows the owner to:

- execute market manipulation through artificial scarcity by withholding tokens;
- perform liquidity drainage attacks by providing and suddenly removing liquidity;
- manipulate prices through wash trading between controlled addresses;
- execute rug pulls by dumping the entire supply at once; and
- create death spiral conditions where fear of dumps causes cascading sell pressure.

Listing 3. Code snippet from src/contracts/ERC20/qai.sol

```
28 constructor() ERC20("QuantixAI", "QAI") Ownable(msg.sender) {
29     _mint(msg.sender, 10000000 * 10 ** 18);
30 }
```

The contract lacks any protective mechanisms such as:

- token vesting schedules;
- lock-up periods;
- multi-signature requirements;
- fair launch distribution; and
- time-locked releases.

Team Response

> Centralized token control during the initial phase is an intentional design choice aligned with our business model and operational safeguards. Maintaining deployer control over the full token supply allows QuantixAI to manage liquidity programs, market stabilization, vesting schedules, and exchange allocations in a secure and transparent manner prior to decentralization. This ensures structured distribution, prevents premature market distortions, and supports long-term sustainability as governance is progressively transitioned to the community.
>
> — QuantixAI Team

Go back to Findings Summary

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

Impact	Info	Confidence	High
Target	qai	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 4. Function is declared as public but could be external since it has no internal references in src/contracts/ERC20/qai.sol (L32-L34)

```
29          _mint(msg.sender, 10000000 * 10 ** 18);
30      }
31
32      function renounceOwnership() public virtual override onlyOwner {
33          super.renounceOwnership();
34      }
35  }
```

Go back to Findings Summary

”Static analysis or stay rekt”

https://wakehacker.ai