



# AUDIT REPORT



---

August 2025

For



# Table of Content

Executive Summary	03
Number of Security Issues per Severity	05
Summary of Issues	06
Checked Vulnerabilities	07
Techniques and Methods	08
Types of Severity	10
Types of Issues	11
Severity Matrix	12
 <b>Low Severity Issues</b>	13
1. Centralization Risk	13
2. Missing Event Emissions in Deposit and Withdraw Functions	14
 <b>Informational Issues</b>	15
3. Redundant Transfer Reference Generation	15
4. Defined Mint Event not Used	16
Functional Tests	17
Automated Tests	17
Closing Summary & Disclaimer	18



# Executive Summary

<b>Project Name</b>	Kanalabs
<b>Protocol Type</b>	Token
<b>Project URL</b>	<a href="https://dev.kanalabs.io">https://dev.kanalabs.io</a>
<b>Overview</b>	<p>KANA is fungible token on the Aptos chain with a maximum supply of 30 billion tokens (30_000_000_000_000_000 with 8 decimals). The contract features admin controls including token burning (restricted to designated burn vaults), pause/unpause functionality, and administrative transfer mechanisms with a two-step verification process. All token transfers are initially disabled by default ( set_untransferable ) and require custom deposit/withdraw functions that check for pause status, suggesting controlled distribution rather than free trading. The contract implements dispatchable fungible asset patterns with override functions for deposits and withdrawals that enforce pause state validation. Admin functions include metadata updates, burn vault management, and emergency pause controls, indicating this is designed for a centrally managed token with strict operational oversight.</p>
<b>Audit Scope</b>	The scope of this Audit was to analyze the KANA Smart Contracts for quality, security, and correctness.
<b>Source Code link</b>	<a href="https://github.com/kanalabs/kana-token-contracts/blob/main/token/sources/Kana.move">https://github.com/kanalabs/kana-token-contracts/blob/main/token/sources/Kana.move</a>
<b>Branch</b>	Main
<b>Commit Hash</b>	7f687c30dd63cf55469fa44d3acedbf442632c2e
<b>Contracts in Scope</b>	Kana.move
<b>Language</b>	Move
<b>Blockchain</b>	Ethereum
<b>Method</b>	Manual Analysis, Functional Testing, Automated Testing
<b>Review 1</b>	13th August 2025 - 15th August 2025



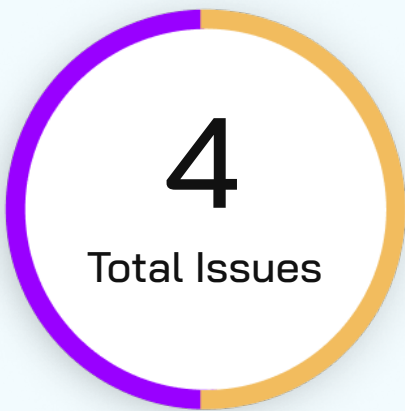
<b>Updated Code Received</b>	21st August 2025
<b>Review 2</b>	21st August 2025
<b>Fixed In</b>	5c35d5a51576dbc600ba88a42b2d8812bf8757af

**Verify the Authenticity of Report on QuillAudits Leaderboard:**

<https://www.quillaudits.com/leaderboard>



# Number of Issues per Severity



Critical	0 (0.0%)
High	0 (0.0%)
Medium	0 (0.0%)
Low	2 (50.0%)
Informational	2 (50.0%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	0	2	1
	Partially Resolved	0	0	0	0	0
	Resolved	0	0	0	0	1



# Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Centralization Risk	Low	Acknowledged
2	Missing Event Emissions in Deposit and Withdraw Functions	Low	Acknowledged
3	Redundant Transfer Reference Generation	Informational	Resolved
4	Defined Mint Event not Used	Informational	Acknowledged



# Checked Vulnerabilities

✓ Transaction-ordering dependence

✓ Timestamp dependence

✓ Denial of service / logical oversights

✓ Timestamp dependence

✓ Access control

✓ Code clones, functionality duplication

✓ Witness Type

✓ Access Management

✓ Integer overflow/underflow by bit operations

✓ Number of rounding errors

✓ Business logic contradicting the specification

✓ Number of rounding errors

✓ Gas usage

✓ Unchecked CALL Return Values

✓ Centralization of power

✓ Implicit visibility level

# Techniques and Methods

**Throughout the audit of smart contracts, care was taken to ensure:**

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

**The following techniques, methods, and tools were used to review all the smart contracts:**

## Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

## Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.





### Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.



# Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

## ■ **Critical: Immediate and Catastrophic Impact**

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

## ■ **High (H): Significant Risk of Major Loss or Compromise**

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

## ■ **Medium (M): Potential for Moderate Harm Under Specific Circumstances**

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

## ■ **Low (L): Minor Imperfections with Limited Repercussions**

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

## ■ **Informational (I): Opportunities for Improvement, Not Immediate Risks**

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.



# Types of Issues

## Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

## Resolved

These are the issues identified in the initial audit and have been successfully fixed.

## Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

## Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

# Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

## Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

## Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.



# Low Severity Issues

## Centralization Risk

**Acknowledged**

### Path

Kana.move

### Description

In the kana module, there is an issue of excessive centralization in admin permissions, including the ability to execute critical operations such as `set_pause()`. However, the process of configuring these crucial protocol parameters lacks specific timing or well-defined ranges. This absence of constraints could potentially lead to suboptimal user experiences and introduce risks associated with centralization.

### Recommendation

It is recommended to mitigate module centralization risks by employing the following methods:

- Foster community governance and participation to ensure decision-making power is distributed among the system's participants.
- Implement robust security measures to protect against potential attacks or malicious actions, such as multi-signature.

### Kana Labs Team's Comment

This contract is owned by and administered by multi-sig wallets.



## Missing Event Emissions in Deposit and Withdraw Functions

**Acknowledged**

### Path

Kana.move

### Function Name

`deposit()`, `withdraw()`

### Description

There are no custom events emitted in the deposit and withdraw functions, and this could lead to a visibility issue.

- Harder to track KANA-specific transfer patterns
- No easy way to audit who's moving tokens when
- Analytics/monitoring tools have less visibility
- Inconsistent with the contract's pattern (it emits events for burn, admin changes, etc.)

### Recommendation

Add custom events for deposit and withdraw operations and Emit these events in the respective functions after successful operations.



# Informational Issues

## Redundant Transfer Reference Generation

**Resolved**

### Path

Kana.move

### Path

`init_module()`

### Description

The initialization function generates two `TransferRef` objects from the same `constructor_ref` - one stored in a local variable and another in the `KanaManagement` struct.

### Code Location

```
1 let transfer_ref = fungible_asset::generate_transfer_ref(constructor_ref); // First generation
2 // ...
3 KanaManagement {
4     transfer_ref: fungible_asset::generate_transfer_ref(constructor_ref), // Second generation
5     // ...
```

- Code clarity issues
- Unnecessary resource usage
- Double `transfer_ref` generation

### Recommendation

Refactor to generate the management struct first, then borrow from it:

```
1
2 // Create management first
3 let management = KanaManagement {
4     transfer_ref: fungible_asset::generate_transfer_ref(constructor_ref),
5     // ... other fields
6 };
7
8 // Borrow from management for initial deposit
9 let tokens = fungible_asset::mint(&mint_ref, KANA_MAX_SUPPLY);
10 fungible_asset::deposit_with_ref(&management.transfer_ref, primary_store, tokens);
11
12 // Then move to storage
13 move_to(metadata_object_signer, management);
```



## Defined Mint Event not Used

**Acknowledged**

### Path

Kana.move

### Description

The contract defines a **Mint** event structure that is never emitted anywhere in the codebase, creating dead code.

### Code Location



```
1
2 // Events
3 #[event]
4 // Triggered when new tokens are minted
5 struct Mint has drop, store {
6     to: address,
7     amount: u64
8 }
9
```

- Dead code in contract
- Misleading documentation
- Potential confusion about contract capabilities

### Recommendation (Either)

1. Remove the unused **Mint** event definition if no future minting is planned
2. Emit the event during initial token minting in `init_module()`



# Functional Tests

Some of the tests performed are mentioned below:

- ✓ Pause/unpause
- ✓ Access control enforcement
- ✓ Event emission correctness

# Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.



# Closing Summary

In this report, we have considered the security of Kana.move. We performed our audit according to the procedure described above.

Issues of Low and Informational severity were found. Kana Labs team resolved one and acknowledged the remaining issues.

# Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.



# About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers.

With seven years of expertise, we've secured over 1400 projects globally, averting over \$3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**7+**

Years of Expertise

**1M+**

Lines of Code Audited

**50+**

Chains Supported

**1400+**

Projects Secured

Follow Our Journey



# AUDIT REPORT

---

August 2025

For



Canada, India, Singapore, UAE, UK

[www.quillaudits.com](http://www.quillaudits.com)

[audits@quillaudits.com](mailto:audits@quillaudits.com)