



# AUDIT REPORT





---

June 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	09
Types of Severity	11
Types of Issues	12
Severity Matrix	13
 <b>High Severity Issues</b>	14
1. Incorrect Allocation Reduction in Vesting Operations	14
2. Missing mint tokens in createAllocation	15
 <b>Medium Severity Issues</b>	16
1. Revoked Schedule's Unvested Tokens Not Transferred from VestingManager to Vault	16
2. Manager Not Removed from Internal Tracking Array	18
 <b>Low Severity Issues</b>	19
1. Storage Collision Risk in Future Upgrades Due to Missing Gaps	19
 <b>Informational Issues</b>	20
1. Spelling Error in Custom Error Name	20
Threat Model	21
Automated Tests	25
Closing Summary & Disclaimer	25



# Executive Summary

<b>Project Name</b>	Toyow
<b>Protocol Type</b>	Token Ecosystem
<b>Project URL</b>	<a href="https://www.toyowtoken.com/">https://www.toyowtoken.com/</a>
<b>Overview</b>	<p>Toyow implements a comprehensive token system for the TTN ecosystem on the Base network with a 3-contract architecture:</p> <ul style="list-style-type: none"><li>• TTNToken - Core ERC20 Token contract</li><li>• TTNTokenVault - Token Treasury &amp; Allocation Manager</li><li>• TTNVestingManager - Vesting, Locking, and Claiming</li></ul>
<b>Audit Scope</b>	The scope of this Audit was to analyze the Toyow Smart Contracts for quality, security, and correctness.
<b>Source Code link</b>	<a href="https://github.com/SoftHorizonSolutions/ttn-token/tree/main/contracts">https://github.com/SoftHorizonSolutions/ttn-token/tree/main/contracts</a>
<b>Branch</b>	main
<b>Contracts in Scope</b>	contracts/TTNToken.sol contracts/TTNTokenVault.sol contracts/TTNVestingManager.sol
<b>Commit Hash</b>	16f3dea919e9f158e236b25e08b89abc9eb8313b
<b>Language</b>	Solidity
<b>Blockchain</b>	Base
<b>Method</b>	Manual Analysis, Functional Testing, Automated Testing
<b>Review 1</b>	28th May 2025 - 4th June 2025
<b>Updated Code Received</b>	6th June 2025
<b>Review 2</b>	6th June 2025
<b>Fixed In</b>	056843565a1280ef819d3efc9aa5bdd61203b450



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

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



# Number of Issues per Severity



Critical	0 (0%)
High	2 (33.3%)
Medium	2 (33.3%)
Low	1 (16.6%)
Informational	1 (16.6%)

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



# Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Incorrect Allocation Reduction in Vesting Operations	High	Resolved
2	Missing mint tokens in createAllocation	High	Resolved
3	Revoked Schedule's Unvested Tokens Not Transferred from VestingManager to Vault	Medium	Resolved
4	Manager Not Removed from Internal Tracking Array	Medium	Resolved
5	Storage Collision Risk in Future Upgrades Due to Missing Gaps	Low	Resolved
6	Spelling Error in Custom Error Name	Informational	Resolved



# Checked Vulnerabilities

✓ Access Management

✓ Arbitrary write to storage

✓ Centralization of control

✓ Ether theft

✓ Improper or missing events

✓ Logical issues and flaws

✓ Arithmetic Computations  
Correctness

✓ Race conditions/front running

✓ SWC Registry

✓ Re-entrancy

✓ Timestamp Dependence

✓ Gas Limit and Loops

✓ Exception Disorder

✓ Gasless Send

✓ Use of tx.origin

✓ Malicious libraries

✓ Compiler version not fixed

✓ Address hardcoded

✓ Divide before multiply

✓ Integer overflow/underflow

✓ ERC's conformance

✓ Dangerous strict equalities

✓ Tautology or contradiction

✓ Return values of low-level calls



✓ Missing Zero Address Validation

✓ Private modifier

✓ Revert/require functions

✓ Multiple Sends

✓ Using suicide

✓ Using delegatecall

✓ Upgradeable safety

✓ Using throw

✓ Using inline assembly

✓ Style guide violation

✓ Unsafe type inference

✓ 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
- Implementation of ERC standards
- 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.

### Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.



# 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.



# 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.



# 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.



# Medium Severity Issues

## Revoked Schedule's Unvested Tokens Not Transferred from VestingManager to Vault

**Resolved**

### Path

contracts/TTNVestingManager.sol

### Function

revokeSchedule

### Description

When a vesting schedule is revoked via `TTNVestingManager.revokeSchedule()`, the calculated `unvestedAmount` of tokens is not actually transferred back from the `VestingManager` (which holds tokens for active schedules) to the `TokenVault`. The function's NatSpec comment even implies these tokens are "returned to the vault," but no such transfer occurs in the code. While the `TokenVault`'s allocation record is marked as revoked, the actual unvested tokens remain stranded within the `TTNVestingManager`.

```
/**
 * @dev Revokes a vesting schedule
 * @param scheduleId ID of the vesting schedule to revoke
 * @return The amount of tokens returned to the vault // @audit implies the transfer
to valut..
 */
function revokeSchedule(uint256 scheduleId) external nonReentrant returns (uint256) {
    // ... role checks and schedule validation ...
    VestingSchedule storage schedule = vestingSchedules[scheduleId];
    uint256 unvestedAmount = schedule.totalAmount - schedule.releasedAmount;
    schedule.revoked = true;

    if (schedule.allocationId > 0) {
        tokenVault.revokeAllocation(schedule.allocationId);
    }
    // @audit no actual token transfer of unvestedAmount happens
    emit ScheduleRevoked(scheduleId, schedule.beneficiary, unvestedAmount);
    return unvestedAmount;
}
```

### Impact

Although the NatSpec suggests unvested tokens are "returned to the vault," they are not. If the `TTNVestingManager` holds the tokens for active schedules, any `unvestedAmount` from a revoked schedule becomes stuck in the `TTNVestingManager` and Over time, this can lead to a significant amount of tokens being permanently stuck.



**Likelihood**

The chance of this issue leading to stranded tokens is HIGH. This will occur every time the `revokeSchedule` function is successfully called for a vesting schedule

**Recommendation**

The `unvestedAmount` from a revoked schedule needs to be actively managed to prevent it from being locked in the `TTNVestingManager`. It is better to ensure these tokens are made available again to the project's main token pool (`TokenVault`)



## Revoked Schedule's Unvested Tokens Not Transferred from VestingManager to Vault

**Resolved**

### Path

contracts/TTNTokenVault.sol

### Function

removeManager

### Description

```
/**
 * @dev Removes an manager
 * @param manager Address of the manager to remove
 */
function removeManager(address manager) external {
    if (
        !hasRole(MANAGER_ROLE, msg.sender) &&
        !hasRole(DEFAULT_ADMIN_ROLE, msg.sender)
    ) revert NotAuthorized();
    if (manager == address(0)) revert InvalidAddress();
    if (manager == msg.sender) revert CannotRemoveSelf();

    _revokeRole(MANAGER_ROLE, manager);
    // @audit missing remove from _managers array
    emit ManagerRemoved(manager);
}
```

### Impact

The getAllManagers function will return an inaccurate list that includes addresses of individuals who are no longer active managers.

### Likelihood

HIGH every time a manager is removed

### Recommendation

Implement logic within the removeManager function to remove the manager





# High Severity Issues

## Incorrect Allocation Reduction in Vesting Operations

**Resolved**

### Path

contracts/TTNVestingManager.sol

### Function

claimVestedTokens, manualUnlock

### Description

When tokens are either claimed by a beneficiary (claimVestedTokens) or unlocked by an admin (manualUnlock), the amount used to reduce the corresponding allocation record in TTNTokenVault.sol is incorrect. Instead of reducing the vault's allocation by the amount of the current claim or current manual unlock, the system uses schedule.releasedAmount. This schedule.releasedAmount variable represents the total cumulative tokens released for that schedule up to that point, including the current transaction.

```
// Reduce the allocated amount in TokenVault if allocationId is set
if (schedule.allocationId > 0) {
    (uint256 allocatedAmount, , bool revoked) =
tokenVault.getAllocationById(schedule.allocationId);
    uint256 vestedAmount = schedule.releasedAmount;
    if (!revoked && allocatedAmount >= vestedAmount) {
        tokenVault.reduceAllocation(schedule.allocationId, vestedAmount); // @audit
        should use releasableAmount instead of vestedAmount
    }
}
```

### Impact

While beneficiaries correctly receive their vested tokens from the VestingManager, the way the TokenVault's allocation records are updated is flawed. The TokenVault ends up thinking its allocations are used up much faster than they really are. This creates a serious accounting mismatch. Down the line, this incorrect bookkeeping in the TokenVault could cause problems, like wrongly blocking future actions.

### Likelihood

HIGH. Any time a vesting schedule linked to a TokenVault allocation has more than one claim or manual unlock transaction

### Recommendation

use releasableAmount instead of vestedAmount

```
if (!revoked && allocatedAmount >= releasableAmount) {
tokenVault.reduceAllocation(schedule.allocationId, releasableAmount);
```



**Missing mint tokens in createAllocation****Resolved****Path**

contracts/TTNTokenVault.sol

**Function**

createAllocation

**Description**

The NatSpec for the createAllocation function incorrectly states that the function “@dev Creates a new allocation and mints tokens”. However, the actual code of createAllocation only creates an internal Allocation record (struct) and emits an event; it does not perform any token minting operation.

**Impact**

The VestingManager expects tokens to be available but they’re never minted.

**Recommendation**

Mint tokens to the TTNVestingManager in createAllocation.



# Low Severity Issues

## Storage Collision Risk in Future Upgrades Due to Missing Gaps

**Resolved**

### Path

TTNToken.sol, TTNTokenVault.sol, and TTNVestingManager.sol

### Description

The contracts utilize the UUPS pattern, which allows their underlying logic to be changed through upgrades. For this upgrade process to be safe regarding state variable storage, it's a standard practice to include storage gaps (e.g., `uint256[50] private __gap;`) at the end of a contract's state variable declarations. These contracts currently do not include such gaps

### Impact

The absence of storage gaps does not affect the functionality or security of the currently deployed version of the contracts. However, it introduces a potential risk if future upgrades are performed that attempt to add new state variables

### Likelihood

Low

### Recommendation

Include a storage gap array at the end of the state variable declarations in TTNToken.sol, TTNTokenVault.sol, and TTNVestingManager.sol



# Informational Issues

## Spelling Error in Custom Error Name

**Resolved****Path**

contracts/TTNVestingManager.sol

**Description**

There is a spelling error in one of the custom error declarations. The error ScheduleRevokeed should likely be ScheduleRevoked.

**Recommendation**

Correct the spelling of the custom error from ScheduleRevokeed to ScheduleRevoked



# Threat Model

Contract	Function	Threats
TTNToken.sol	initialize()	<p>Re-initialization Attack: If initializer protection is weak or bypassed in an upgrade, attacker could re-initialize, potentially re-assigning admin roles or changing token parameters.</p> <p>Initialization with Malicious Parameters: Admin providing incorrect parameters.</p>
TTNToken.sol	mint()	<p>Unauthorized Minting Minting Beyond Max Supply.</p>
	pause() / unpause()	<p>Unauthorized Pause/ Unpause.</p>
	transferTokenAdmin()	<p>Unauthorized Admin Transfer. Transfer to Zero/Self.</p>
	_authorizeUpgrade	<p>Malicious Upgrade Bypassing Authorization Access control</p>
TTNTokenVault.sol	initialize()	<p>Re-initialization Attack: If initializer protection is weak or bypassed in an upgrade, attacker could</p> <p>re-initialize, potentially re-assigning admin roles or changing token parameters.</p> <p>Initialization with Malicious Parameters: Admin providing incorrect parameters.</p>



Contract	Function	Threats
TTNTokenVault.sol	createAllocation()	Unauthorized Allocation Creation Revocation always false in creation Address and amount cannot be zero Missing mint operation
	revokeAllocation()	Cannot reclaim tokens Revoking already revoked allocations Unauthorized access
	executeAirdrop()	Token loss via mismatched arrays Airdrop to zero address Integer overflow Unauthorized access
	pause() / unpause()	Denial of service if abused
	addManager()	Self-assignment or privilege escalation
	removeManager()	Cannot remove self Not removing from _managers array
	isManager() / getAllManagers()	Stale list due to _managers bug
TTNVestingManager.sol	reduceAllocation()	Unauthorized access Reduces to 0 then revokes logic breaks
	initialize()	Zero-address assignment Re-initialization Attack



Contract	Function	Threats
	createVestingSchedule()	<p>Unauthorized users may create vesting schedules.</p> <p>Invalid or malicious input (e.g., cliff &gt; duration, start in past) could lock tokens permanently.</p> <p>Over-allocation can occur if TokenVault allocation check is bypassed.</p>
	computeReleasableAmount()	<p>Incorrect vesting math can lead to under/over distribution.</p> <p>Fails silently if schedule is revoked, possibly hiding important state</p>
	claimVestedTokens()	<p>Reentrancy vector if external token contract becomes malicious (though nonReentrant is in place).</p> <p>Incorrect accounting due to reducing allocation using releasedAmount instead of releasableAmount.</p> <p>Possibility of double-claim if allocation logic is inconsistent.</p>

Contract	Function	Threats
	manualUnlock()	<p>Reduces allocation with total releasedAmount instead of just amount, which may over-deduct from vault.</p> <p>Attackers with admin rights could prematurely drain schedules.</p>
	revokeSchedule()	<p>Could revoke earned tokens if done before user claims vested tokens.</p> <p>Tokens not truly recovered if internal accounting and vault state fall out of sync.</p>





# 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 Toyow. We performed our audit according to the procedure described above.

Issues of High, Medium, Low, and Informational severity were found. The Toyow team resolved all the issues successfully.

# 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 six years of expertise, we've secured over 1000 projects globally, averting over \$30 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

**\$30B+**

Secured in Digital Assets

**1400+**

Projects Secured

Follow Our Journey



# AUDIT REPORT

---

June 2025

For



Canada, India, Singapore, UAE, UK

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

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