TRIBYTE

MEET 48

SECURITY AUDIT REPORT

SEPTEMBER 2025

Every Byte Builds Immutable Trust

No. 202509120001

Contents

Summary

Executive Summary

Tribyte performed a comprehensive security audit of the Meet48 Token Unlock contracts throughout September 2025, with the objective of identifying vulnerabilities and ensuring the robustness of the codebase. The audit encompassed both core contracts and their dependencies, delivering a thorough evaluation of the project's security posture.

The assessment leveraged a dual methodology, combining Manual Review and Static Analysis to meticulously examine the contracts. Our team adopted a multifaceted testing strategy, integrating black-box, gray-box, and white-box techniques to simulate real-world attack scenarios and detect potential weaknesses. Black-box testing evaluated the contracts from an external attacker's perspective, gray-box testing probed internal behaviors using specialized scripting tools, and white-box testing featured an in-depth, line-by-line code review by our experts.

Project Summary

Project Name	Meet48 Token Unlock
Language	Solidity
Github Link	https://github.com/meet48/token-unlock
Commit Hash	4f1a841dafe87d7e4cd27136bbdc949944b9a6cb

Vulnerability Summary

Severity Level	Summary
Critical	
High	0
Medium	2 (Resolved: 1 Mitigated: 1 Acknowledged: 0 Declined: 0)
Low	4 (Resolved: 2 Mitigated: 0 Acknowledged: 2 Declined: 0)
Informational	1 (Resolved: 0 Mitigated: 0 Acknowledged: 1 Declined: 0)

Findings

[GV-01] Centralization Risk

Category	General Vulnerability
Severity	Medium
Location	src/tokenUnlock.sol: #L9
Status	Resolved. The protocol states they will grant the ownership permission to multisig wallets (such as Gnosis Safe) during deployment to mitigate single point control risks.

Description

The protocol exhibits significant centralization by vesting critical control in a single owner address. This central point of authority introduces several risks, including:

- The ability to arbitrarily set Periods and Claimers, potentially altering the intended token unlock schedule or beneficiary distribution.
- The capacity to withdraw any amount of ERC20 tokens from the contract at any time, which could jeopardize the funds allocated for users.

This centralized governance model creates a single point of failure and heightens the risk of malicious or erroneous actions, undermining the protocol's security and trustworthiness.

Recommendation

To mitigate these risks, we recommend transitioning the owner address to a more decentralized and secure governance structure using the following tools:

- Multisig Wallet: Implement a 2-of-3 or 3-of-5 Gnosis Safe Wallet to distribute ownership responsibilities among multiple trusted parties, reducing the risk of unilateral control.
- Timelock: Introduce a delay of 24-72 hours for high-impact actions (e.g., withdrawals or period modifications), allowing time for community review and intervention if necessary. These measures will enhance security and promote a more robust governance framework.

[GV-02] Recommend Using Ownable2Step Instead of Ownable

Category	General Vulnerability
Severity	Low
Location	src/tokenUnlock.sol: #L9
Status	Resolved. Contract has been updated to use Ownable2Step to enhance the security of ownership management.

Description

The use of Ownable for ownership management introduces a risk of inadvertently transferring ownership to an uncontrolled address, such as address(0) or an unmanaged externally owned account (EOA) or contract address. This could result in the permanent loss of ownership privileges, rendering the contract's administrative functions inaccessible.

Recommendation

It is recommended to replace Ownable with OpenZeppelin's Ownable2Step. This implementation splits the ownership transfer process into two distinct steps: transferOwnership and acceptOwnership. This two-step approach enhances security by requiring explicit confirmation, thereby mitigating the risk of accidental or unauthorized ownership transfers.

[GV-03] Excessive Permissions in withdraw() Function

Category	General Vulnerability
Severity	Low
Location	src/tokenUnlock.sol: #L119
Status	Acknowledged. The protocol states that this function will be used cautiously to ensure it does not affect tokens already allocated to users.

Description

The withdraw() function, defined as an emergency withdrawal mechanism, allows the owner to extract any amount of any token at any time. This broad authority poses a significant risk, as it enables the owner to withdraw funds arbitrarily, potentially undermining the contract's intended purpose and the trust of users relying on the token unlock mechanism.

Recommendation

It is recommended to restrict the withdraw() function when the token being withdrawn matches the contract's primary token (_token == token). In such cases, the owner should only be permitted to withdraw amounts that exceed the total sum of all amount values defined across the Period structures. This limitation ensures that the owner's withdrawal rights do not interfere with the allocated token amounts intended for user claims, thereby preserving the contract's integrity.

[GV-04] Lack of Duplicate Claimer Check in setClaimers() May Reduce User Claimable Tokens

Category	General Vulnerability
Severity	Medium
Location	src/tokenUnlock.sol: #L73
Status	Mitigated. The protocol states that they will check setClaimers parameters during contract configuration to ensure no duplicated addresses is passed.

Description

The setClaimers() function accepts a _claimers parameter that theoretically may contain duplicate addresses. In such cases, the assignment *periodClaimPercents[j][_claimers[i]] = percents[i]* will overwrite previous values, retaining only the last percent value for a given claimer. This behavior results in the affected claimer being able to claim only the assets corresponding to the final percent, potentially reducing the total token amount they are entitled to claim.

Recommendation

This issue can be addressed through two potential solutions:

1. If duplicates in the _claimers parameter are not permitted, implement a strict validation to enforce uniqueness.
2. If duplicates are allowed in the _claimers parameter, accumulate all percent values for a given claimer in periodClaimPercents[j][_claimers[i]]. For instance, replace *periodClaimPercents[j][_claimers[i]] = percents[i]* with *periodClaimPercents[j][_claimers[i]] += percents[i]* to sum the values.

We recommend adopting the second approach for the following reasons: the first method involves deduplication checks that incur significant gas costs, whereas the second method offers greater compatibility. Additionally, it allows the admin to retain control over whether duplicate addresses are included when calling setClaimers(), providing flexibility without compromising efficiency.

[CC-01] TOTAL_PERCENT Can Be Declared as a Constant

Category	Coding Convention
Severity	Low
Location	src/tokenUnlock.sol: #L22
Status	Resolved. Contract has been updated to declare TOTAL_PERCENT as a constant.

Description

The TOTAL_PERCENT variable is declared as immutable and initialized with a fixed value of 10000 within the constructor. This approach suggests that its value is intended to remain constant throughout the contract's lifecycle.

Recommendation

It is advisable to declare TOTAL_PERCENT as a public constant with the value directly assigned, such as *uint256 public constant TOTAL_PERCENT = 10000;*. This eliminates the need for immutable and constructor initialization, aligning with Solidity best practices for fixed values and improving code readability and gas efficiency.

[CC-02] periodClaimPercents Can Be Simplified to a Single Mapping Layer

Category	Coding Convention
Severity	Low
Location	src/tokenUnlock.sol: #L23
Status	Acknowledged. The protocol states that the double-layer mapping of periodClaimPercents is adopted due to the contract's design requirement to accurately record each period's rewards per address.

Description

The periodClaimPercents variable is declared as a nested mapping, structured as *mapping(uint256 => mapping(address => uint256)) public periodClaimPercents;*, to store the claim percentage for each user per period *(period => user => claim percent)*. However, based on the contract's logic, each user's claim percentage remains consistent across all periods. This redundancy results in unnecessary replication of data for every period. By reducing the storage hierarchy, the gas costs associated with initialization and the claim function execution can be significantly lowered.

Recommendation

It is recommended to simplify periodClaimPercents to a single-layer mapping, such as *mapping(address => uint256) public periodClaimPercents;*. This adjustment aligns with the contract's intended behavior, optimizes storage usage, and reduces gas expenditure during both initialization and claim operations.

[CC-03] Variable Naming Conventions Should Be Consistent

Category	Coding Convention
Severity	Informational
Location	src/tokenUnlock.sol: #L30, #L73, #108, #119
Status	Acknowledged. The protocol states that they understand this recommendation, but will retain the _ prefix to avoid naming conflicts with some variables.

Description

The naming style for function parameters lacks consistency across the contract. For instance, the following functions exhibit mixed conventions, with some parameters prefixed with an underscore (_) while others are not:

```
constructor( address initialOwner,address _token) Ownable(initialOwner) {
    ...
}

function setClaimers(address[] calldata _claimers, uint256[] calldata percents) external onlyOwner onlySetPeriodFinish {
    ...
}

function getClaimable(address _claimer, uint256 time) public view returns (uint256 totalClaimable) {
    ...
}

function withdraw(address _token, uint256 amount) external onlyOwner {
    ...
}
```

This inconsistency can lead to confusion and reduce the overall maintainability of the codebase.

Recommendation

It is recommended to standardize the naming convention by prefixing all function parameters with an underscore (_). Adopting a uniform style, such as _initialOwner, _token, _claimers, _percents, _claimer, _time, _token, and _amount, will enhance code readability and facilitate easier maintenance by the development team.

Appendix

Vulnerability Fix Status

Status	Description
Resolved	The project team has successfully implemented a complete fix to address the vulnerability, eliminating the associated risks. All identified issues have been rectified, and the codebase has been updated to ensure the vulnerability no longer poses a threat.
Mitigated	The project team has taken steps to reduce the impact or likelihood of the vulnerability being exploited, but the issue has not been completely resolved. While the risk has been partially addressed, some potential exposure may still remain, and further action is recommended.
Acknowledged	The project team has reviewed and confirmed the existence of the vulnerability but has chosen not to address or mitigate it at this time. This status indicates awareness of the issue, and the team may accept the associated risks or plan to address it in the future.
Declined	The project team has reviewed the reported vulnerability but determined it does not require action, either because they believe it poses no significant risk to the project or because it falls outside the project's scope or priorities. The issue remains unaddressed, and the associated risks are accepted by the team.

Vulnerability Severity Level

Level	Description
Critical	Critical severity vulnerabilities pose an immediate and severe threat to the project's security, potentially leading to significant loss of funds, unauthorized access, or complete system compromise. These issues must be addressed urgently before deployment or continued operation to ensure the safety of users and the integrity of the project.
High	High severity vulnerabilities can substantially impact the project's functionality, potentially enabling exploitation that results in loss of assets, data breaches, or disruption of critical operations. It is strongly recommended to prioritize and resolve these issues promptly to mitigate risks.
Medium	Medium severity vulnerabilities may affect the project's performance or security under specific conditions, potentially leading to inefficiencies, minor exploits, or degraded user experience. It is advisable to address these issues to enhance the overall robustness of the system.

Low	Low severity vulnerabilities have a minimal impact on the project's security or functionality and are unlikely to be exploited in typical scenarios. However, they may still pose theoretical risks. The project team should evaluate these issues and consider fixing them to improve long-term stability.
Informational	Informational findings do not directly impact the security or functionality of the project but highlight areas for improvement, such as adherence to best practices, code optimization, or architectural enhancements. Addressing these suggestions can lead to better maintainability and alignment with industry standards.

Audit Items

Categories	Audit Items
Coding Convention	Obsolete Code
	Debug Code
	Comments / Dev Notes
	Compiler Versions
	License Identifier
	Require / Revert / Assert Usage
	Contract Size
	Gas Consumption
	Event Emission
	Parameter Check
General Vulnerability	Centralization
	Denial of Service
	Reply Attack
	Reentrancy Attack
	Race Conditions
	Integer Overflow / Underflow
	Arithmetic Accuracy Deviation
	Array Index Out of Bounds

	Receive / Fallback Function
	Payable and msg.value Usage
	tx.origin Authentication
	ERC20 Token Decimals
	ERC20 Safe Transfer
	ERC721 Safe Transfer
	Rebasable Token Support
	Native Token Support
	Storage / Memory Usage
	Function Permissions
External Dependency	Oracle Usage
	External Protocol Interaction
Protocol Design	Economics Design
	Formula Derivation
Contract Design	Factory Contract
	Proxy Usage
	EIP2535 Diamond Pattern Usage
	Upgradability / Pluggability

Disclaimer