TRIBYTE

WAND PROTOCOL

SECURITY AUDIT REPORT

FEB 2025

Every Byte Builds Immutable Trust

No. 202502050001

Contents

Summary

Executive Summary

Tribyte conducted a security audit of the Bribe Vault contracts for Wand Finance throughout February 2025. The audit aimed to identify vulnerabilities and ensure the robustness of the codebase, focusing on both the core contracts and their dependencies.

The assessment utilized a dual approach of Manual Review and Static Analysis to thoroughly evaluate the contracts. Our team employed a comprehensive testing strategy, incorporating black-box, gray-box, and white-box methodologies, to simulate real-world attack scenarios and uncover potential weaknesses. Black-box testing assessed the contracts from an external attacker’ s perspective, gray-box testing analyzed internal behaviors using scripting tools, and white-box testing involved a detailed line-by-line code review.

The audit identified twelve findings, ranging from Medium to Informational severity, including issues such as ineffective data persistence (GV-02), mutable token metadata (GV-03), and potential gas exhaustion (CC-01). All findings have been resolved, reflecting Wand Finance’ s commitment to security. We recommend ongoing testing against diverse attack vectors, enhancing code documentation, and ensuring transparency in privileged operations to maintain a high security standard.

Project Summary

Project Name	Wand Finance
Language	Solidity
Github Link	https://github.com/wandfi/bribe-vault-contract-public
Commit Hash	34b9eb182765592010a92d37df34e33a1ae19cc2

Vulnerability Summary

Severity Level	Summary
Critical	

High		
Medium	Resolved: 1	Mitigated: 1
Low	Resolved: 7	
Informational	Resolved: 3	

Findings

[GV-01] Centralization Risk

Category	General Vulnerability
Severity	Medium
Location	contracts/Protocol.sol contracts/settings/ProtocolOwner.sol
Status	Mitigated

Description

The whole set of contracts is controlled by an owner account, and the owner account has the ability to alter key aspects of the protocol's functionality, like updating protocol parameters, registering new vaults to Protocol contract, updating name or symbol of PToken, etc. If the owner account is comprised, team will completely lose control of the protocol.

Recommendation

It’ s recommended to use multi-signature wallet, timelock and DAO, to improve decentralization and transparency.

Alleviation

The project states that they will use https://safe.global/ wallet as the protocol owner account.

[GV-02] Ineffective Data Persistence Due to Memory-Type Struct Declaration

Category	General Vulnerability
Severity	Medium
Location	contracts/vault/Vault.sol: #L241
Status	Resolved

Description

The currentEpoch variable, defined as a memory-type struct at line 241 in the Vault.sol contract, does not persist modifications to its internal attributes on the blockchain. This design choice renders any updates to the struct transient, as they are discarded after execution and do not alter the contract's state. Such behavior could precipitate serious issues, including inaccurate balance tracking or flawed state-dependent logic, compromising the contract's reliability in scenarios where persistent data accuracy is essential.

Recommendation

We recommend reassessing the use of the memory modifier for the currentEpoch struct. If the goal is to ensure that updates to its attributes are permanently recorded on the blockchain, replacing memory with the storage modifier is advised. This adjustment will guarantee that changes are preserved in the contract’ s state, enhancing data integrity and consistency. Furthermore, a comprehensive review of the contract is suggested to identify and address any similar misapplications of memory-type variables that could affect state persistence.

[GV-03] Mutable Token Name and Symbol Post-Deployment

Category	General Vulnerability
Severity	Low
Location	contracts/tokens/PToken.sol: #L125~#131
Status	Resolved

Description

The setName() and setSymbol() functions, located at lines 125 – 131 in PToken.sol, permit the token's name and symbol to be modified after deployment. Although access to these functions is restricted to the contract owner, allowing such changes introduces risks of user confusion or misrepresentation of the token's identity. In decentralized ecosystems, token names and symbols are typically expected to be immutable, as they serve as critical identifiers for users, exchanges, and applications. Alterations post-deployment could erode trust or disrupt systems relying on consistent token metadata.

Recommendation

To enhance trust and align with standard practices, we recommend ensuring the token's name and symbol are immutable after deployment. This can be achieved by removing the setName() and setSymbol() functions or setting the name and symbol as immutable variables during contract initialization. Additionally, consider documenting the immutability of these attributes in the contract's interface and public documentation to reinforce user confidence and clarify expectations for downstream systems.

[GV-04] Inappropriate Access Control on Read-Only Functions

Category	General Vulnerability
Severity	Low
Location	contracts/vaults/RedeemPool.sol: #L75, #L80, #L91, #L95
Status	Resolved

Description

The functions at lines 75, 80, 91, and 95 in RedeemPool.sol, marked as view or pure, are unnecessarily restricted by the onlyBeforeSettlement modifier. These read-only functions, which do not alter blockchain state, are limited to specific addresses, contrary to their intended transparency. Such restrictions are ineffective, as private state variables can be accessed off-chain via blockchain inspection tools, rendering the access control redundant. This design choice deviates from best practices and may obscure contract data unnecessarily, potentially reducing user trust and interoperability.

Recommendation

We recommend removing the onlyBeforeSettlement modifier from view and pure functions to align with best practices for read-only operations. Making these getter functions public will enhance transparency and accessibility without compromising security, as they do not modify state. If sensitive data is involved, consider reevaluating its storage and management, such as using encryption or off-chain solutions, since access restrictions on read-only functions do not provide meaningful protection. Additionally, document the purpose and visibility of these functions to clarify their intended use.

[GV-05] Rounding Bias in Favor of Users in Redemption Calculations

Category	General Vulnerability
Severity	Low
Location	contracts/tokens/PToken.sol: #L158; contracts/vaults/RedeemPool.sol: #L123
Status	Resolved

Description

The withdrawRedeem() function in RedeemPool.sol (line 123) utilizes getRedeemingSharesByBalance(), which invokes _convertToShares() and rounds down the resulting sharesAmount. This amount is then subtracted from _userRedeemingShares, effectively rounding in favor of the user. Similarly, in PToken.sol (line 158), the burn() function calls getSharesByBalance(), which also rounds down and omits a decimals offset, exposing the contract to potential inflation attacks. Such rounding biases can lead to minor financial losses for the protocol and undermine its economic integrity over time.

Recommendation

To mitigate financial risk and align with best practices, we recommend modifying the rounding logic to favor the protocol in both RedeemPool.getRedeemingSharesByBalance() and PToken.burn(). Specifically, adjust _convertToShares() and getSharesByBalance() to round up share calculations where applicable. Additionally, incorporate a decimals offset in PToken.sol to prevent inflation attacks. Thoroughly test these changes to ensure accurate share accounting and consider documenting the rounding policy to maintain transparency with users.

[GV-06] Use of Deprecated OpenZeppelin Counters Library

Category	General Vulnerability
Severity	Low
Location	contracts/vaults/Vault.sol: #L29
Status	Resolved

Description

The Vault.sol contract at line 29 imports and utilizes OpenZeppelin's Counters.sol library, which has been officially deprecated by OpenZeppelin (reference: GitHub Issue #4233). The deprecation indicates that the library is no longer actively maintained and may pose future compatibility or security risks. Continued reliance on deprecated third-party dependencies could lead to technical debt or vulnerabilities if unaddressed, potentially affecting the contract’s long-term reliability.

Recommendation

We recommend removing the dependency on the deprecated Counters.sol library and replacing its functionality with native Solidity alternatives, such as simple uint256 counters with explicit increment operations (e.g., counter++). Ensure that replacement logic includes safeguards against overflow, leveraging Solidity’s built-in arithmetic checks (post version 0.8.0) or custom checks if using an earlier version. After implementing changes, thoroughly test the contract to verify counter behavior and update documentation to reflect the removal of the deprecated library.

[GV-07] Unlimited Token Approval in Constructor Poses Security Risk

Category	General Vulnerability
Severity	Low
Location	contracts/vaults/ERC4626BribeVault.sol: #L26
Status	Resolved

Description

The constructor of ERC4626BribeVault.sol at line 26 executes IERC20(assetToken).approve(address(stakingPool), type(uint256).max), granting unlimited token approval to the stakingPool contract. This practice is considered insecure, as it exposes the vault to significant risks if the stakingPool contract is compromised, upgraded maliciously, or behaves unexpectedly. Unlimited approvals increase the attack surface, potentially allowing an adversary to drain all approved tokens without further user interaction, undermining the contract’ s security posture.

Recommendation

We recommend replacing the unlimited approval with precise, transaction-specific approvals to minimize risk. Before transferring tokens, call IERC20(assetToken).approve(address(stakingPool), amount) with the exact amount required for the operation, ideally within the same transaction as the transfer (e.g., using TokensTransfer.transferTokens). This approach limits exposure to the minimum necessary and aligns with secure token management practices. Additionally, consider implementing a mechanism to revoke or reduce approvals after transfers and document the approval strategy to enhance transparency and user trust.

[CC-01] Potential Gas Exhaustion from Unbounded Iteration Over _bribeTokens

Category	Coding Convention
Severity	Low
Location	contracts/bribes/AdhocBribesPool.sol: #L91, #L156 contracts/bribes/StakingBribesPool.sol: #L62, #L109
Status	Resolved

Description

The updateAllBribes modifier, applied in AdhocBribesPool.sol (lines 91, 156) and StakingBribesPool.sol (lines 62, 109), iterates over the _bribeTokens array to invoke _updateBribes for each token before executing the function logic. Additionally, the getBribes function, which also uses this modifier, performs a second iteration over _bribeTokens within the same transaction. When _bribeTokens contains a large number of entries, these consecutive loops may exceed the transaction gas limit, causing execution to fail. This design restricts the contract' s scalability and usability, particularly in high-gas environments, and could render critical functions impractical.

Recommendation

To enhance scalability and prevent gas exhaustion, we recommend introducing a cap on the number of _bribeTokens processed in a single transaction within the updateAllBribes modifier and getBribes function. If the token count exceeds this limit, the contract should revert with a descriptive error, prompting users to process tokens in smaller batches. Alternatively, consider implementing an iterative update mechanism that allows partial processing of _bribeTokens across multiple transactions. Document the imposed limit and any batch-processing requirements to ensure clarity for users and developers interacting with the contract.

[CC-02] Lack of Zero Address Validation for Input Parameters

Category	Coding Convention
Severity	Low
Location	contracts/bribes/AdhocBribesPool.sol: #L38; contracts/bribes/StakingBribesPool.sol: #L34 contracts/settings/ProtocolSettings.sol: #L31
Status	Resolved

Description

The functions at lines 38 in AdhocBribesPool.sol, 34 in StakingBribesPool.sol, and 31 in ProtocolSettings.sol accept address parameters without validating that they are not the zero address (address(0)). This omission could lead to unintended behavior, such as assigning critical roles or assets to an invalid address, resulting in errors, loss of functionality, or unexpected contract state. Implementing zero address checks is a standard coding practice to enhance robustness and prevent such issues in smart contract execution.

Recommendation

We recommend adding explicit validation to ensure that all address inputs in the affected functions are not address(0). This can be achieved by incorporating a check, such as require(inputAddress != address(0), "Invalid address"), before processing the address. This validation will safeguard against erroneous inputs, improve contract reliability, and align with best practices for secure smart contract development. Additionally, consider documenting the expected behavior for address inputs to guide users and developer.

[CC-03] Redundant Code Statements in Contract Implementation

Category	Coding Convention
Severity	Informational
Location	contracts/settings/ProtocolOwner.sol: #L15
Status	Resolved

Description

The statement at line 15 in ProtocolOwner.sol serves no functional purpose within the contract and appears to be a remnant of test code or deprecated functionality. Such redundant code introduces unnecessary complexity, increases gas costs marginally, and may confuse future developers or auditors reviewing the codebase. Removing these statements will streamline the contract, enhancing its clarity and maintainability without impacting its intended behavior.

Recommendation

We recommend removing the redundant statement at line 15 in ProtocolOwner.sol to optimize the contract for production deployment. Prior to removal, verify that the statement has no unintended side effects by conducting thorough testing. Additionally, consider performing a comprehensive code review to identify and eliminate any other residual or obsolete components, ensuring the contract remains concise, efficient, and aligned with its functional requirements.

[CC-04] Inclusion of Development Debugging Console in Contract Code

Category	Coding Convention
Severity	Informational
Location	contracts/settings/ProtocolSettings.sol : #L4
Status	Resolved

Description

The ProtocolSettings.sol contract at line 4 includes references to the Hardhat/Foundry console contract, which is designed solely for debugging during development and testing. Although these references may be commented out and non-functional, their presence in the codebase detracts from its cleanliness and professionalism. Retaining debugging artifacts, such as console.log statements, in production-ready contracts is considered poor practice, as it increases code clutter and may confuse auditors or developers reviewing the final implementation.

Recommendation

We strongly recommend removing all references to the console contract and any associated console.log statements from ProtocolSettings.sol prior to deployment on public or production networks. This cleanup will enhance the contract’ s readability, maintainability, and alignment with production standards. Before removal, ensure that debugging information is adequately captured through alternative means, such as events or off-chain logging, during development. A thorough review of the codebase is also advised to eliminate any other development-specific artifacts.

[CC-05] Redundant Use of SafeMath Library in Solidity >=0.8.0

Category	Coding Convention
Severity	Informational
Location	contracts/libs/VaultCalculator.sol: #L8
Status	Resolved

Description

The VaultCalculator.sol contract at line 8 imports and utilizes the SafeMath library for arithmetic operations. Since Solidity version 0.8.0 and above includes built-in checked arithmetic operations that automatically revert on overflow or underflow, the use of SafeMath is unnecessary. Retaining this library introduces redundant code, increasing contract complexity and deployment gas costs without providing additional security benefits, which detracts from best practices in modern Solidity development.

Recommendation

We recommend removing the SafeMath library from VaultCalculator.sol and replacing its operations with standard arithmetic operators (+, -, *, /). This simplification will reduce code complexity, lower deployment gas costs, and align with Solidity’ s native safety features in version 0.8.0 and later. Prior to removal, verify that the contract’ s arithmetic operations do not rely on specific SafeMath behaviors (e.g., custom error messages). Additionally, ensure the project’ s Solidity version is explicitly set to at least 0.8.0 in the compiler configuration to leverage built-in checks.

Appendix

Vulnerability Fix Status

Status	Description
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

TRIBYTE

CONTACT US

https://x.com/TriByteLabs

tribytelabs@gmail.com

https://github.com/tribyteio