



AUDIT REPORT



June , 2025

For



MONWU

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
Severity Matrix	12
Types of Issues	13
 Low Severity Issues	14
1. Premium Investor cannot be removed	14
2. Use ownable2step instead	15
3. Centralization Concern	16
 Informational Issues	17
4. Emit even for critical functions	17
5. Missing functions for investors to view their claimable tokens	18
6. Unused state variable	19
7. Typo in state variable	20
Functional Tests	21
Threat Model	22
Automated Tests	29
Closing Summary & Disclaimer	29



Executive Summary

Project name MONWU Smart Contract

Protocol Type Token

Project URL <https://monwu.com/>

Overview

The MONWU protocol is a token distribution system built around an ERC20 token with a 1 billion token maximum supply. The protocol implements a sophisticated vesting mechanism where tokens are allocated across multiple investor tiers and purposes, including founders, various investor classes (Elite, Platinum, Premium), creators, community members, public sale, liquidity, rewards, marketing, and charity. Most allocations follow a 5-year vesting schedule with a 3-year cliff period followed by 2 years of gradual token releases in approximately 6-month intervals. The token includes a burn mechanism that allows holders to permanently reduce the total supply while attempting to maintain a minimum of 500 million tokens in circulation. The system is managed through six separate smart contracts - the main MONWU token contract that handles minting and burning, and five vesting contracts that manage the time-locked distribution of tokens to different participant categories. After 10 years, any unclaimed tokens can be recovered, and throughout the vesting period, contract owners maintain administrative control to add, remove, or modify investor allocations.

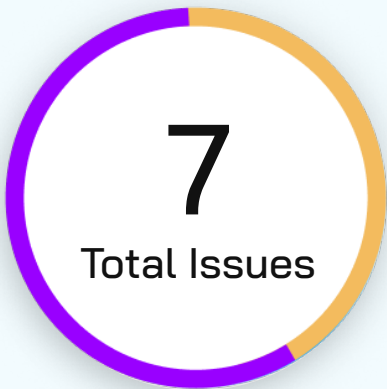
Audit Scope	The scope of this Audit was to analyze the MONWU Smart Contract Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/BlackH3art/monwu-token/tree/main
Contracts in Scope	MonwuCharity.sol MonwuElite150.sol MonwuFounding15.sol MonwuMarketingDevelopment.sol MonwuPlatinum1500.sol MonwuPremiumVesting.sol TokenMonwu.sol
Branch	main
Commit Hash	547b64eb0fcfc8db3231e7b38c3dcf3deb47f002
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th June 2025 - 17th June 2025
Updated Code Received	19th June 2025
Review 2	20th June 2025
Fixed In	80274678b96351fe9c5d92d0fc507b22d59146ba

Verify the Authenticity of Report on QuillAudits Leaderboard:

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



Number of Issues per Severity



Critical	0 (0%)
High	0 (0%)
Medium	0 (0%)
Low	3 (43%)
Informational	4 (57%)

		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	0	0	3	4



Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Premium Investor cannot be removed	Low	Resolved
2	Use ownable2step instead	Low	Resolved
3	Centralization Concern	Low	Resolved
4	Emit even for critical functions	Informational	Resolved
5	Missing functions for investors to view their claimable tokens	Informational	Resolved
6	Unused state variable	Informational	Resolved
7	Typo in state variable	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

<div>Open</div> <p>Security vulnerabilities identified that must be resolved and are currently unresolved.</p>	<div>Resolved</div> <p>Security vulnerabilities identified that must be resolved and are currently unresolved.</p>
<div>Acknowledged</div> <p>Vulnerabilities which have been acknowledged but are yet to be resolved.</p>	<div>Partially Resolved</div> <p>Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.</p>

Low Severity Issues

Premium Investor cannot be removed

Resolved

Path

MONWUPremiumVesting.sol

Description

MONWUPremiumVesting contract lacks remove and edit functions that exist in all other vesting contracts. Once added, premium investors cannot be removed or have their allocations adjusted, even if added by mistake.

Recommendation

Consider adding a functionality to remove premium investors



Use ownable2step instead

Resolved

Description

Contracts use basic Ownable which allows single-step ownership transfer. Should use Ownable2Step to prevent accidental transfer to wrong address. Current implementation could permanently lock owner functions if transferred to incorrect address.

Recommendation

Consider using Ownable2Step.sol instead.



Centralization Concern

Resolved

Description

The current implementation of contract poses centralization risk where a single owner address has complete control over investor allocations, can remove investors, edit allocations after vesting starts, and claim all leftover tokens. No multi-sig or time-lock mechanisms.

The contract has a wide attack surface provided that the owner address is compromised.

Recommendation

Ensure that the owner is a multisig wallet



Informational Severity Issues

Emit even for critical functions

Resolved

Path

-

Function

handleLeftovers()

Description

handleLeftovers() transfers potentially large amounts of tokens but emits no event. Makes it difficult to track 10-year token movements on-chain. Should emit an event with the recipient and amount.

Recommendation

Consider adding an event for this function.



Missing functions for investors to view their claimable tokens

Resolved

Path

-

Function

-

Description

There is no public view function like `getReleasableAmount()` for investors to check available tokens before attempting withdrawal.
As a result, users must waste gas on failed withdrawal attempts to discover the available amount.



Unused state variable

Resolved

Path

MONWU.sol

Function

burn()

Description

The burned state variable is incremented but never read or used in any logic. The burn protection check uses `cap` instead.

Recommendation

Either use the variable for protection logic or remove it to save gas.



Typo in state variable

Resolved

Path

-

Function

-

Description

Inconsistent naming: elit150Intialized missing 'e' (should be elite150Intialized).

Recommendation

Consider fixing the typo



Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should revert when burn would reduce cap below 500M
- ✓ Should include vested but unclaimed tokens in handleLeftovers after 10 years
- ✓ Should create vesting schedule with 3-year cliff
- ✓ Should release 100% of allocation after vesting end
- ✓ Should transfer remaining tokens via handleLeftovers

Threat Model

Contract	Function	Threats
Vesting Contract	addInvestor() / addEliteInvestor() / addFounder() / addPlatinumInvestor() / addPremiumInvestor()	<p>Inputs</p> <ul style="list-style-type: none"> investor (address) - The investor address to add <ul style="list-style-type: none"> Control: Fully controlled by owner Constraints: Must not already exist in mapping Impact: Determines who can claim tokens allocation (uint256) - Token amount to allocate <ul style="list-style-type: none"> Control: Fully controlled by owner Constraints: None (can be 0 or any value) Impact: Determines vesting amount <p>Internal State Dependencies</p> <ul style="list-style-type: none"> addressToInvestor[investor] <ul style="list-style-type: none"> Control: Modified by add/remove/edit functions Constraints: investor field must be address(0) to add Impact: Stores vesting schedule and allocation block.timestamp <ul style="list-style-type: none"> Control: None (external dependency) Constraints: None Impact: Determines token precision MONWUToken.decimals()



Contract	Function	Threats
	<p>removeInvestor() / removeEliteInvestor() / removeFounder() / removePlatinumInvestor()</p>	<ul style="list-style-type: none"> • Control: None (external dependency) • Constraints: None • Impact: Determines token precision <p>Branches and Code Coverage</p> <p>Intended branches</p> <ul style="list-style-type: none"> • Should add new investor with allocation • Should calculate cliff end (3 years from now) • Should calculate vesting end (5 years from now) • Should emit InvestorAdded event • Should scale allocation by token decimals <p>Negative behavior</p> <ul style="list-style-type: none"> • Should not allow adding existing investor • Currently allows address(0) as investor (critical bug) Currently allows 0 allocation (gas waste) • No maximum allocation check • Only callable by owner (centralization) • For Premium: requires token transfer approval <p>Inputs</p> <ul style="list-style-type: none"> • investor (address) - The investor to remove <ul style="list-style-type: none"> • Control: Fully controlled by owner • Constraints: Must exist in mapping • Impact: Removes access to tokens



Contract	Function	Threats
		<p>Internal State Dependencies</p> <ul style="list-style-type: none"> • addressToInvestor[investor] <ul style="list-style-type: none"> • Control: Modified by this function • Constraints: investor field must not be address(0) • Impact: Zeros address and allocation only <p>Branches and Code Coverage</p> <p>Intended branches</p> <ul style="list-style-type: none"> • Should remove investor access • Should emit InvestorRemoved event <p>Negative behavior</p> <ul style="list-style-type: none"> • Missing from MONWUPremiumVesting • (inconsistency bug) Doesn't clear all struct fields • (gas inefficiency) Tokens remain locked in • contract (design flaw) Can remove after partial • withdrawal (accounting issue) Only callable by owner (centralization) <p>Inputs</p> <ul style="list-style-type: none"> • investor (address) - The investor to edit <ul style="list-style-type: none"> • Control: Fully controlled by owner • Constraints: Must exist in mapping • Impact: Identifies which allocation to modify
	editInvestor() / editEliteInvestor() / editFounder() / editPlatinumInvestor()	



Contract	Function	Threats
		<ul style="list-style-type: none"> • newAllocation (uint256) - New token amount • Control: Fully controlled by owner • Constraints: None Impact: Changes vesting amount <p>Internal State Dependencies</p> <ul style="list-style-type: none"> • addressToInvestor[investor].allocation • Control: Modified by this function • Constraints: None • Impact: Changes total claimable amount <p>Branches and Code Coverage</p> <p>Intended branches</p> <p>Should update allocation Should emit InvestorEdited event</p> <p>Negative behavior</p> <ul style="list-style-type: none"> • Missing from MONWUPremiumVesting (inconsistency bug) Can set allocation below already withdrawn amount (critical bug) No recalculation of vesting schedule (logic bug) Allows retroactive vesting if allocation increased (exploit) Only callable by owner (centralization)

Contract	Function	Threats
	investorRelease() / founderRelease()	<p>Inputs</p> <ul style="list-style-type: none"> • amount (uint256) - Amount to withdraw • Control: Fully controlled by caller • Constraints: Must not exceed releasable amount • amount Impact: Determines tokens transferred <p>Internal State Dependencies</p> <ul style="list-style-type: none"> • addressToInvestor[msg.sender] • Control: Released field modified by this function • Constraints: Must be valid investor • Impact: Tracks withdrawal progress • block.timestamp • Control: None (blockchain controlled) • Constraints: Must be past cliff end • Impact: Determines releasable amount • MONWUToken.balanceOf(address(this)) • Control: None (depends on token balance) • Constraints: Must have sufficient balance • Impact: Limits withdrawal capability

Contract	Function	Threats
	handleLeftovers()	<p>Branches and Code Coverage</p> <ul style="list-style-type: none">• Intended branches<ul style="list-style-type: none">• Should calculate correct releasable amount• Should transfer tokens to investor• Should update released amount• Should emit ReleaseTokens event• Should handle precision loss at vesting end <p>Negative behavior</p> <ul style="list-style-type: none">• Integer division causes precision loss (minor bug)• Misleading vesting schedule (4 periods + remainder)• No view function to check releasable amount• Can't withdraw during cliff period <p>Inputs</p> <ul style="list-style-type: none">• No external inputs <p>Internal State Dependencies</p> <ul style="list-style-type: none">• block.timestamp<ul style="list-style-type: none">• Control: None (blockchain controlled)• Constraints: Must be > burnDeadline (10 years)• Impact: Enables function execution

Contract	Function	Threats
		<ul style="list-style-type: none">• <code>MONWUToken.balanceOf(address(this))</code><ul style="list-style-type: none">• Control: None• Constraints: Must be > 0• Impact: Determines transfer amount• <code>MONWUToken.owner()</code><ul style="list-style-type: none">• Control: None (external dependency)• Constraints: None• Impact: Determines recipient (bug - wrong recipient) <p>Branches and Code Coverage</p> <p>Intended branches</p> <ul style="list-style-type: none">• Should transfer remaining tokens after 10 years• Should only work after deadline <p>Negative behavior</p> <ul style="list-style-type: none">• Sends to token owner, not contract owner (critical bug)• Takes legitimate vested tokens too (design flaw)• No event emitted (transparency issue)• Only callable by owner (expected)

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 MONWU Smart Contract. We performed our audit according to the procedure described above.

Issues of Low and Informational severity were found. The MONWU team successfully resolved all the 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 pro-gram to enhance resilience against emerging threats.

Stay proactive. Stay secure.



About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem



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



MONWU



QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com