QuillAudits

AUDIT REPORT

June 2025

For

CAR

Table of Content

Executive Summary

Project Name	Car staking
Protocol Type	staking
Project URL	https://carmemecoin.com/
Overview	Staking smart contract in Rust using the Anchor framework. The contract allows users to stake SPL tokens and earn rewards at a configurable annual percentage rate, with a mandatory withdrawal delay period for security. Key features include admin functions for managing reward rates and adding reward funds, user staking/unstaking with accumulated rewards, and a two-step withdrawal process (request → wait → withdraw). The contract uses precise mathematical calculations with a scaling factor to handle reward distributions fairly among all stakers. It includes comprehensive event logging and multiple view functions to check current rewards, unallocated reward pools, and reward runway duration.
Audit Scope	The scope of this Audit was to analyze the car staking Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/alexpipun/spl-staking-locked/blob/single-token/programs/spl-staking-locked/src/lib.rs
Branch	single-token
Contracts in Scope	lib.rs
Commit Hash	9f06c5a9a01967194eea7a495af6f9c65b930e96
Language	Rust
Blockchain	Solana
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	23th June 2025 - 30th June 2025
Updated Code Received	30th June 2025

Review 2	1st July 2025
Fixed In	a0d9cfb7c69356787c73d0357206641034ccfd70

Verify the Authenticity of Report on QuillAudits Leaderboard:

https://www.quillaudits.com/leaderboard

Number of Issues per Severity

2
Total Issues

🟥	Critical	0 (0%)
🟥	High	0 (0%)
🟧	Medium	2 (100%)
🟨	Low	0 (0%)
🟪	Informational	0 (0%)

Severity

Issues

	🟥 Critical	🟥 High	🟧 Medium	🟨 Low	🟪 Informational
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Partially Resolved	0	0	0	0	0
Resolved	0	0	2	0	0

Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Unbounded Initial Withdrawal Delay Configuration	Medium	Resolved
2	Unsafe Reward Forfeiture Function Without Necessity Validation	Medium	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.

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

Likelihood	🟥 High	🟧 Medium	🟨 Low
🟥 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.

Medium Severity Issues

Unbounded Initial Withdrawal Delay Configuration

Resolved

Path

lib.rs

Function

initialize

Description

The staking contract's initialize function accepts withdrawal_delay_days without any upper bound validation, allowing the administrator to set arbitrarily long withdrawal delays. While the configure_withdrawal_delay function enforces a 31-day maximum, this protection is absent during initialization. The delay is converted to seconds as (withdrawal_delay_days * SECONDS_PER_DAY) as u32, potentially allowing delays of years or even decades to be configured at deployment.

Impact

A malicious or compromised administrator could initialize the contract with an excessive withdrawal delay, effectively locking user funds indefinitely. Users who stake tokens would be unable to withdraw their assets for the configured period, which could extend far beyond reasonable timeframes.

Recommendation

Apply the same 31-day validation check in the initialize function that exists in configure_withdrawal_delay.

Unsafe Reward Forfeiture Function Without Necessity Validation

Resolved

Path

lib.rs

Function

initialize

Description

The staking contract provides two withdrawal methods: withdraw (which requires sufficient reward tokens in the protocol) and withdraw_and_forfeit_rewards (which allows users to exit when rewards are unavailable).

While withdraw_and_forfeit_rewards serves as an emergency exit when the protocol lacks reward tokens, it unconditionally forfeits all user rewards regardless of protocol solvency. The function executes identically whether the protocol has zero rewards or sufficient tokens to pay the user's full claim, making it possible for users to accidentally forfeit earnings when normal withdrawal would have succeeded.

Impact

Users can permanently lose earned rewards through accidental or unnecessary use of the forfeiture function. This creates an error-prone interface where misclicks, UI confusion, or misunderstanding leads to irrecoverable financial loss.

Recommendation

Implement automatic reward payment in withdraw_and_forfeit_rewards when possible, only forfeiting the unavailable portion..

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 VPOP. We performed our audit according to the procedure described above.

Issues of Critical , High , Medium and Low severity were found. Vpop team acknowledged three and resolved the rest of 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 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

CAR

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com