QuillAudits

AUDIT REPORT

April , 2025

For

CAR

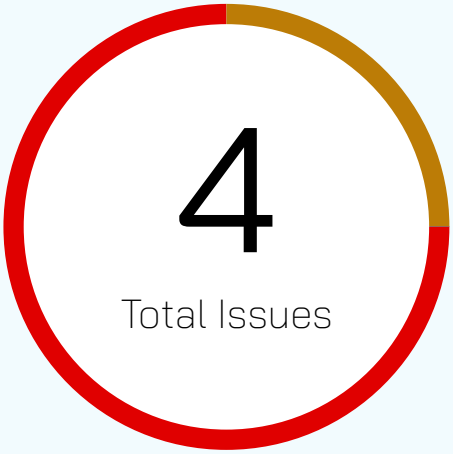
Table of Content

Executive Summary

Project name	Car Coin
Project URL	https://carmemecoin.com/
Overview	Sale smart contract manages land sales, allowing users to buy land with tokens. It verifies purchases using authority signatures, records ownership, and sets expiration dates based on rental periods. SPL Token library (imported by the main contract) provides comprehensive functionality for token operations on Solana, including transfers, minting, burning, approvals, and account data retrieval. It defines structures and methods for interacting with Solana's token program, enabling secure token transactions within the land sale ecosystem.
Audit Scope	The scope of this Audit was to analyze the Car Coin Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/codemelt-dev/car-sale/tree/main/solidity
Contracts in Scope	solidity/sale.sol solidity/spl_token.sol
Branch	Main
Commit Hash	2bff488beab122a1dd395b4a19d3725737ca7ee6
Language	Solang
Blockchain	Solana

Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	29th April 2025
Updated Code Received	25th April 2025
Review 2	30th April 2025
Fixed In	https://drive.google.com/file/d/1xrZCdyOv7Guk-LLeue6_42xn-h12w-U6O/view?usp=drive_link

Number of Issues per Severity



High	3 (75.00%)
Medium	1 (25.00%)
Low	0 (0.00%)
Informational	0 (0.00%)

Severity

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

Checked Vulnerabilities

- ☑ Signer authorization
- ☑ Account data matching
- ☑ Sysvar address checking
- ☑ Owner checks
- ☑ Type cosplay
- ☑ Initialization
- ☑ Arbitrary CPI
- ☑ Duplicate mutable accounts
- ☑ Bump seed canonicalization
- ☑ PDA Sharing
- ☑ Incorrect closing accounts
- ☑ Missing rent exemption checks
- ☑ Arithmetic overflows/underflows
- ☑ Numerical precision errors
- ☑ Solana account confusions
- ☑ Casting truncation
- ☑ Insufficient SPL token account verification
- ☑ Signed invocation of unverified programs

Techniques and Methods

Throughout the audit of Solana Programs, 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
- 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 analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.

Techniques and Methods

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 analysed, and 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 behaviour of Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

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.

High Severity Issues

Signature Bypass

Resolved

Path

Sale.sol

Function

buy

Description

This contract handles land sales where users can buy land using an off-chain signed message. The buy function is supposed to verify that the message was signed by a trusted authority before allowing the purchase. However, the check is broken: it uses
assert(isSignatureValid || !isSignatureValid);,
which is always true. This means anyone can call buy with a fake or random message and still purchase land. The purpose of using off-chain signatures for permission is completely defeated.

Recommendation

Replace the check with assert(isSignatureValid);

Authority Address is Set to Zero

Resolved

Path

Sale.sol

Function

buy

Description

This contract is designed so that land can only be purchased using a message signed by a trusted authority (like an admin or backend service). To enforce this, it defines an AUTHORITY address that's supposed to represent the signer. But instead of setting it to a real address, the contract sets AUTHORITY = address(0), which is just an empty placeholder.
address constant private AUTHORITY = address(0);
On top of that, the line that checks if the signer of the message actually matches this AUTHORITY is commented out. This means the contract does not validate who signed the message — anyone can pass in a fake message with any signature, and the contract won't stop it.

Recommendation

Replace the placeholder with the actual authority's address — the signer you trust to approve purchases. Then uncomment the line assert(tx.accounts.authority.key == AUTHORITY); to make sure only approved signers can authorize land sales. Without this, the whole permission system is meaningless.

Expired Land Cannot Be Re-Purchased

Acknowledged

Path

Sale.sol

Function

buy

Description

The contract allows users to buy land for a specific rental period. Once purchased, the land's ownership is stored along with an expiresAt timestamp. However, there's no logic that resets or frees the land after the expiration time. The check
assert(land.owner == address(0));
prevents anyone from buying land that has already been purchased — even if the rental period has ended.
This means once a land is bought, it becomes permanently locked in the contract, even after its expiry. No one — not even the original buyer — can renew or re-purchase the same land. This breaks the idea of land rental, making the expiration timestamp pointless and reducing the contract's usability long-term.

Recommendation

Add logic to check if the land has expired by comparing block.timestamp > land.expiresAt. If it's expired, allow it to be re-purchased by treating it as available.

Medium Severity Issues

No Replay Protection on Message Timestamp

Resolved

Path

Sale.sol

Function

Buy

Description

The land sale contract relies on off-chain signatures to authorize purchases. Each signed message includes a timestamp, but the contract currently does not enforce any strict freshness checks. The timestamp is only weakly verified with a placeholder condition:
assert(block.timestamp > timestamp);
This means any previously signed message can be reused indefinitely, as long as the current time is after the message's timestamp. This opens the door for replay attacks, where a buyer can submit the same message multiple times.
This issue becomes critical when combined with the earlier fix we suggested—to allow land to be re-purchased after expiration by checking if block.timestamp > land.expiresAt. While this fix enables proper land reuse, it also means that without replay protection, a buyer can simply reuse an old signature with a previously agreed-upon price and expiration date. This allows them to bypass any updated pricing or terms, undermining the purpose of having dynamic pricing or rental durations.

Recommendation

Validate the freshness of the timestamp using a tighter condition like:
assert(block.timestamp > timestamp && block.timestamp - timestamp < 5 minutes);

Functional Tests

Some of the tests performed are mentioned below:

- Buy Functionality
- Signature verification
- State Updates.

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 Car Coin. We performed our audit according to the procedure described above.

Issues of high and medium severity were found. Car coin team fixed almost all and acknowledged one

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 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

April , 2025

For

CAR

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com