QuillAudits

Audit Report, February, 2024

For

ROCKWALLET

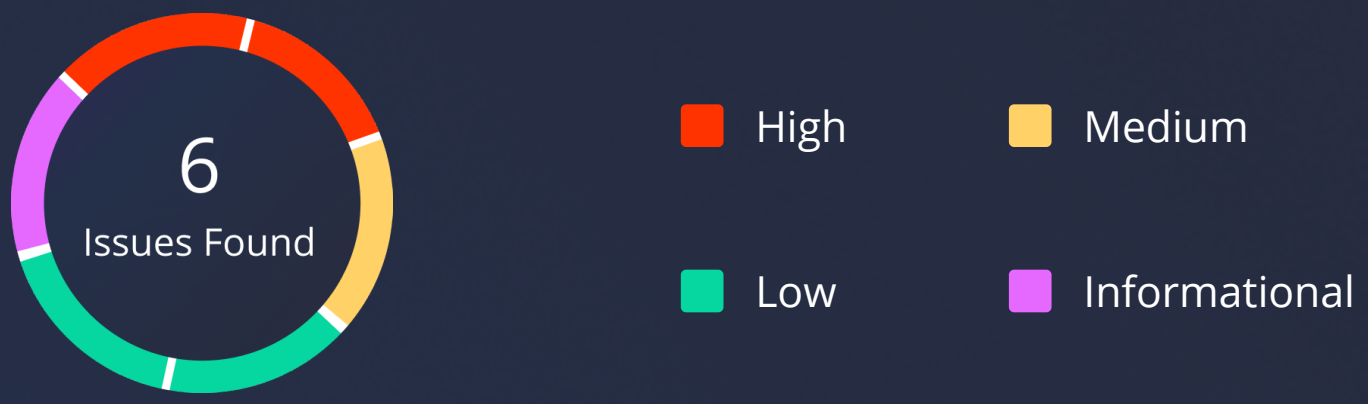
Table of Content

Table of Content

Executive Summary

Project Name	RockWallet
Project URL	*https://www.rockwallet.com/*
Overview	A token contract which can be minted and burned by role owners. Specific roles can also blacklist and freeze accounts.
Audit Scope	*https://mumbai.polygonscan.com/address/0xa30bADadC2d0f871fbA4f210D41c970187C55803#code* *https://mumbai.polygonscan.com/address/0xca699297be1855fbcc8f9a6bd6424970bae22651#code* *https://mumbai.polygonscan.com/address/0xcD24113B54e6dd9c126d473899A5AF6e87a4e255#code*
Contracts in Scope	RWT.sol SigningLibrary.sol
Commit Hash	NA
Language	Solidity
Blockchain	Polygon
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	12 January 2024 - 30 January 2024
Updated Code Received	8 February 2024
Review 2	9 February 2024 - 12 February 2024
Fixed In	*https://bitbucket.org/monkhub/rockwalletblockchain/commits/de9ec4f9ed6d06dc32d360f56a9c291bae41ff50*

Number of Security Issues per Severity



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

Checked Vulnerabilities

- Access Management
- Arbitrary write to storage
- Centralization of control
- Ether theft
- Improper or missing events
- Logical issues and flaws
- Arithmetic 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

Checked Vulnerabilities

- 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, Truffle, Solhint, Mythril, Slither, Solidity Static Analysis.

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

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

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.

A. Contract - RWT.sol

High Severity Issues

A.1 Signature replay attack by a colluded role owner

Description

The SignatureVerifier lib checks if the signature is signed by the given signer but the signature is not invalidated by the RWT token contract once used. In case a role owner's account gets compromised or a role owner is colluded, the signatures executed can be replayed which can result in burning, whitelisting, freezing of user funds etc.

Remediation

It is recommended to use a nonce to identify each transaction and signature which can be expired once executed on chain.

Status

Resolved

A.2 Signature replay attack on a different chain

Description

The data signed by signers include addresses and an amount which does not indicate where it is executed, hence if the same contract is deployed on a different chain, a colluded role owner can execute all the signatures again.

Remediation

It is recommended to use add a chain identifier like chain id which can be verified in the signature verification.

Status

Resolved

Medium Severity Issues

A.3 Invalid Signer check can be exploited by 2 colluded signers

Description

In functions replaceValidator(), mintBurnPauseUnpause() and blacklisterFreezerOps(), it is expected to have signatures from at least 3 unique signers to execute an operation.

```
        }

        if (i > 0)
            if (signers[i] == signers[i - 1]) revert invalidSigner();
        if (
            !SigningLibrary.verify(
```

Here, the check is insufficient and can be exploited if the array has the same address as the first and last element when the total signers required are 3. This check won't verify if they are different, and hence, it can be easily bypassed by 2 different signers.

Remediation

It is recommended to check if the first and last addresses are different; this is valid till requiredSignatures are 3.

Status

Resolved

Low Severity Issues

A.4 Blacklist and freeze operations are ambiguous.

Description

In the function blacklisterFreezerOps(), while blacklisting or freezing an account, it checks for both cases.

```
    } else if (fType == functionType.blacklist) {
        if ((blacklisted[_address]) || (frozen[_address])) revert BLorF();
        blacklisted[_address] = true;
        emit AccountBlacklisted(_address);
    } else if (fType == functionType.freeze) {
        if ((blacklisted[_address]) || (frozen[_address])) revert BLorF();
        frozen[_address] = true;
        emit AccountFrozen(_address);
```

But in case of _beforeTokenTransfer(), there is no check for _to address in case it is frozen.

```
function _beforeTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    if (!isBlacklisterFreezer[msg.sender]) {
        if (blacklisted[from]) revert blacklistedAddress();
        if (frozen[from]) revert frozenAddress();
    }
    if (blacklisted[to]) revert blacklistedAddress();
    if (paused()) revert tokenPaused();
    super._beforeTokenTransfer(from, to, amount);
}
```

Remediation

Use OpenZeppelin's Ownable2Step instead of Ownable.

```
    } else if (fType == functionType.blacklist) {
        if (blacklisted[_address]) revert Blacklisted();
        blacklisted[_address] = true;
        emit AccountBlacklisted(_address);
    } else if (fType == functionType.freeze) {
        if (frozen[_address]) revert Frozen();
        frozen[_address] = true;
        emit AccountFrozen(_address);
```

Status

Acknowledged

A.5 Check-effect-interaction pattern

Description

There are multiple instances in the contract where check-effect-interaction pattern is violated:

1. The function mintBurnPauseUnpause() is updating the states before signature verification.
2. The functions changeRedeemer() and changeRescuer() emit event before updating the states,

Remediation

It is recommended to first perform all the checks and then update the states before emitting events from the contract.

Reference: *https://docs.soliditylang.org/en/v0.8.24/security-considerations.html#use-the-checks-effects-interactions-pattern*

Status

Acknowledged

Informational Issues

A.6 Not proper naming conventions

Description

It is recommended to follow style guide for solidity for easy readability of smart contracts, more can be referred from here: *https://docs.soliditylang.org/en/v0.8.24/style-guide.html#naming-conventions*

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

- [PASS] test token transfer
- [PASS] test token transfer revert when frozen
- [PASS] test token transfer revert when blacklisted
- [PASS] test token transfer revert when paused
- [PASS] test token transfer when unpaused
- [PASS] test replacing validator for minter role
- [PASS] test replacing validator for burner role
- [PASS] test replacing validator for blacklister or freezer role
- [PASS] test revert when replacing validator with valid signers but invalid msg.sender
- [PASS] test minting tokens with valid signers
- [PASS] test revert minting tokens with invalid signers
- [PASS] test revert minting tokens with valid signers but invalid msg.sender
- [PASS] test revert when blacklisting with invalid signers
- [PASS] test revert when freezing with invalid signers
- [PASS] test revert when blacklisting tokens with valid signers but invalid msg.sender
- [PASS] test revert when freezing tokens with valid signers but invalid msg.sender
- [PASS] test sending transaction to proxy address
- [PASS] test revert when calling admin functions on proxy from non admin account

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

Some issues of High, Low and informational severity were found. Some suggestions, gas optimisations and best practices are also provided in order to improve the code quality and security posture.

In The End, the RWT Team resolved high and Medium issues and Acknowledged other low and informational Issues.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in **RockWallet** smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of **RockWallet** smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of **RockWallet** to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

850+
Audits Completed

$30B
Secured

800K
Lines of Code Audited

Follow Our Journey

Audit Report
February, 2024

For

ROCKWALLET

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com