SOLIDRATE

https://solidrate.io

SMART CONTRACT AUDIT

LUFFY

date 04 JUNE 2024

Table Of Contents

Contract Details

Contract Address	G4fajHQiM3ic8mn1poLED6coRme88RUFAXq4nEftEiyt
Contract Name	LUFFY
Symbol	LUFFY
Network	Solana Network
Decimals	6
Current Supply	1,000,000,000,000.00
Deployer Address	8Z7CmLBYUHxY4hojDRHboop4QF1mwtSKZQ2UvAv51dGT
Owner Address	TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA

Contract Analysis

Trading Security

Thorough manual examination of the code, including line-by-line analysis and a meticulous review of trading constraints, taxes, and owner privileges. Our assessment ensures a comprehensive understanding of potential security risks.

Trading Constraints

Safety Overview Passed Failed

Analysis	Results
No pause function	Passed
No trading cooldown function	Passed
No blacklist function	Passed
No whitelist function	Passed

Contract Features

Safety Overview | Passed | Failed

Analysis	Results
Fees below 5%	Passed
No max transaction amount feature	Passed
No max wallet feature	Passed
No limited transactions feature	Passed

Token Authority

Contract Checks

Our assessment covers meticulous contract checks to ensure industry standards compliance and a focused review of owner privileges for a secure token governance framework.

Safety Overview Passed Failed

Analysis	Results
Contract Verified	Yes
External Call Risk	No
Proxy Contract	No
Selft Destruct	No
Antibot	No
Antiwhale	No

Owner Privileges

Safety Overview Passed Failed

Analysis	Results
Ownership renounced	Yes
No regain ownership function	No
No hidden mint function	No
Modifiable taxes	No
Modifiable max amount or max transaction	No
No high amount owner token	Yes

Rust Code Security Analysis
Classification Security Analysis

Detected vulnerabilities during our security audit scan of the smart contract. Our assessment includes thorough Rust Registry checks and overflow assessments.

Severity High Medium Low Passed

Id	Name	Description	Result
RUST001	Unsafe code block	Unsafe code blocks can lead to undefined behavior if not used properly.	Passed
RUST002	Unhandled error	Unwrapping a Result or Option without proper error handling can lead to panic.	Passed
RUST003	Unchecked arithmetic	Arithmetic operations that can overflow or underflow without being checked.	Passed
RUST004	Insecure random number generator	Using an insecure random number generator for security-sensitive operations.	Passed
RUST005	Uninitialized memory	Using uninitialized memory can lead to undefined behavior.	Passed
RUST006	Use of mem::transmute	Using mem::transmute can lead to undefined behavior and violate type safety.	Passed
RUST007	Use of std::process::Command	Using std::process::Command without properly sanitizing user input can lead to command injection vulnerabilities.	Passed
RUST008	Use of std::fs::File with unwrap()	Using unwrap() with std::fs::File can lead to panics if the file operation fails.	Passed
RUST009	Deserialization of untrusted data	Deserializing untrusted data without proper validation can lead to security vulnerabilities.	Passed
RUST010	Use of std::net::Tcp Listener with unwrap()	Using unwrap() with std::net::TcpListener can lead to panics if the binding operation fails.	Passed
VULN001	Integer Overflow or Underflow	Performing arithmetic operation without checking for overflow or underflow.	Passed

Severity High Medium Low Passed

Id	Name	Description	Result
VULN002	Loss of Precision	The use of try_round_u64() for rounding up may lead to loss of precision.	Passed
VULN003	Inaccurate Calculation Results	Reliance on saturating arithmetic operations without considering precision loss.	Passed
VULN007	Missing Check for the Permission of Caller	Missing verification of caller permissions before sensitive operations.	Passed
VULN008	Account Signer Check	Ensure the expected signer account has actually signed to prevent unauthorized account modifications.	Passed
VULN009	Account Writable Check	Ensure state accounts are checked as writable to prevent unauthorized modifications.	Passed
VULN010	Account Owner or Program ID Check	Verify the owner of state accounts to prevent fake data injection by malicious programs.	Passed
VULN011	Account Initialized Check	Prevent re-initialization of already initialized accounts.	Passed
VULN017	Signer Authorization - Anchor	Signer check is missing, which could lead to unauthorized execution.	Passed
VULN018	Account Data Matching - Anchor	Missing verification of token ownership or mint authority in SPL Token accounts.	Passed
VULN019	Owner Checks - Anchor	Missing checks on the owner field in the metadata of an Account or on the Account itself.	Passed
VULN020	Type Cosplay - Anchor	Risks of different accounts impersonating each other by sharing identical data structures.	Passed
VULN021	Check Initialize - Anchor	Data should only be initialized once; missing checks can lead to reinitialization.	Passed
VULN022	Arbitrary CPI - Anchor	Unverified target program id in CPI can lead to arbitrary code execution.	Passed

Severity High Medium Low Passed

Id	Name	Description	Result
VULN023	Duplicate Mutable Accounts - Anchor	Passing the same mutable account multiple times may result in unintended data overwriting.	Passed
VULN024	Bump Seed Canonicalization - Anchor	Improper validation of bump seeds can lead to security vulnerabilities.	Passed
VULN025	PDA Sharing - Anchor	Sharing PDA across multiple roles without proper permission separation may lead to unauthorized access.	Passed
VULN026	Closing Accounts - Anchor	Improper closing of accounts may leave them vulnerable to misuse.	Passed
VULN027	Sysvar System Account Not Checked	Sysvar system account is accessed without verifying its legitimacy, exposing the contract to potential manipulation or attacks.	Passed
VULN028	PDA Account Misuse Without Proper Verification	The PDA account is utilized without validating the caller's and beneficiary's accounts, allowing unauthorized actions.	Passed
VULN029	Unchecked Account Deserialization	Failing to check if an account is of the expected type before deserializing can lead to incorrect assumptions about state.	Passed
VULN031	CPI to Unauthorized Programs	Invoking unauthorized or risky external programs can expose the contract to vulnerabilities present in those programs.	Passed
Rust-Solana001	Misuse of Unsafe Code	Unsafe blocks may lead to undefined behavior and memory safety violations if not used carefully.	Passed
Rust-Solana002	Improper Error Handling	Overuse of unwrap() or expect() can lead to panics.	Passed
Rust-Solana003	Overuse of Panics for Control Flow	Using panics for control flow makes code hard to follow and can lead to unexpected termination.	Passed
Rust-Solana004	Concurrency Issues and Data Races	Improper handling of threads and synchronization can lead to data races, deadlocks, and other concurrency issues.	Passed
Rust-Solana005	Potential Memory Leaks	Cyclic references or improper use of smart pointers can lead to memory leaks.	Passed

Severity High Medium Low Passed

Id	Name	Description	Result
Rust-Solana006	Potential DoS Vulnerabilities	Allocations based on untrusted input sizes can lead to DoS via memory exhaustion.	Passed
Rust-Solana007	Missing Boundary Checks	Accessing arrays or vectors without boundary checks can lead to panics or buffer overflows.	Passed
Rust-Solana008	Unnecessary Cloning of Large Data Structures	Cloning large data structures can lead to performance issues due to excessive memory use.	Passed
Rust-Solana009	Blocking I/O in Asynchronous Code	Performing blocking I/O operations in async contexts can lead to thread starvation and reduced scalability.	Passed
Rust-Solana010	Misuse of Arc<Mutex>	Incorrect use of Arc<Mutex> can lead to deadlocks or inefficient locking mechanisms.	Passed
Rust-Solana011	Improper Implementation of Drop Trait	Incorrect custom implementations of the Drop trait can lead to resource leaks or panic safety issues.	Passed
Rust-Solana012	Usage of mem::uninitialized and mem::zeroed	Using mem::uninitialized or mem::zeroed can lead to undefined behavior if the type has any non-zero or complex initialization requirements.	Passed

Community Trust

Our commitment to community trust extends to a thorough evaluation of social media channels, ensuring active engagement and responsiveness. Investor relations and partnerships are also scrutinized for credibility and transparency.

Social Media Presence

Website Health	Good	
Twitter	High Activity	+50 Followers
Telegram	High Activity	+50 Members
	Active Community	
Youtube	Not Available	
Discord	Not Available	

Listings and Partnerships

Listings	Not Available
Partnerships	Not Available

Audit Summary

Throughout the audit process, our team of auditors and smart contract developers conducted a cautious evaluation to identify security-related issues, code quality, and adherence to specification and best practices.

The smart contract does not contain high severity issues.

Security Score : 90%

Audit Methodology

Our smart contract audit methodology is designed to provide a thorough and comprehensive analysis of any contract code. We believe that a collaborative and multi-faceted approach is the best way to ensure the reliability and accuracy of our results.

Our audits are performed by experienced blockchain developers and security experts who work independently to provide a more comprehensive and accurate assessment. The auditors begin by reading the contract code to understand its structure and purpose, and examining the functional and technical requirements as well as other relevant documents provided by the client.

Next, automated tools are used in a controlled environment to search security vulnerabilities and best practices. These tools can help us identify potential vulnerabilities, such as Smart Contract Weakness vulnerabilities, and provide a more complete picture of the contract's functionality.

Data flow diagrams are generated to visualise all possible states and interactions with other contracts. This allows the auditors to track changes in data and funds flow and identify any potential issues or risks.

Line-by-line review of the contract code is conducted to check for hidden malicious code or other security risks. This meticulous and thorough review is essential for protecting our clients' interests and ensuring the success of their projects.

Once the audit is complete, we provide a comprehensive report on our findings and recommendations. Our methodology is standardised to ensure consistent and reliable results, and we are committed to providing our clients with the information they need to improve the security and functionality of their contracts. By conducting a thorough and comprehensive audit, we can help our clients protect their interests and ensure the success of their projects.

Disclaimer

This audit report is intended to provide a comprehensive analysis of the contract code and its potential vulnerabilities. The findings and recommendations in this report are based on our best efforts and knowledge at the time of the audit, but they are not guaranteed to be complete or accurate.

All smart contract audits performed by us are provided for informational purposes only, and should not be considered legal or financial advice. We do not guarantee the security or functionality of any smart contracts that we audit, and we shall not be liable for any losses or damages arising from the use of any audited smart contracts.

We recommend that our clients carefully review the report and consider its findings and recommendations, but they should not rely on it as the sole basis for making any decisions regarding the contract. We encourage our clients to conduct their own independent analysis and seek additional advice as needed. This report is provided for informational purposes only, and it does not constitute legal, financial, or other professional advice. It does not create any warranties, representations, or guarantees, and it does not establish any legal or contractual obligations.

By accepting this report, our clients acknowledge and agree to the terms of this disclaimer. We appreciate the opportunity to conduct the audit and provide our findings, and we are committed to assisting our clients in any way we can.