




AUDIT REPORT

December 2025

For

CrossFX 

Table of Content

Executive Summary	03
Number of Security Issues per Severity	04
Summary of Issues	05
Checked Vulnerabilities	06
Techniques and Methods	08
Types of Severity	10
Types of Issues	11
Severity Matrix	12
 Informational Issues	13
1. Use Fixed Solidity Version	13
Functional Tests	14
Automated Tests	14
Trust and Assumption	15
Threat Model	16
Closing Summary & Disclaimer	17



Executive Summary

Project Name	Supesa
Protocol Type	Stablecoin
Overview	<p>SupesaStablecoin is a Transparent Proxy–upgradeable ERC20 stablecoin implementing role-based minting, burning, pausing, and per-account freezing using OpenZeppelin upgradeable modules.</p> <p>The contract enforces freeze and pause restrictions across transfers, approvals, and permit, includes emergency ERC20 recovery, and is designed for controlled, compliance-oriented token operations.</p>
Audit Scope	The scope of this Audit was to analyze the Supesa Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/harshala-supesa/smart-contract
Contracts in Scope	supesaStablecoin.sol
Branch	develop
Commit Hash	471b0429ec3611efac1c664a4740af4159bb0cf8
Language	Solidity
Blockchain	Polygon
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	17th December 2025 to 18th December 2025
Updated Code Received	19th December 2025
Review 2	19th December 2025
Fixed In	592393ebe1cfe9f908ed73a2284fddb3ce7764ef

Verify the Authenticity of Report on QuillAudits Leaderboard:

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



Number of Issues per Severity



Critical	0(0.0%)
High	0(0.0%)
Medium	0(0.0%)
Low	0(0.0%)
Informational	1 (100%)

		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	0	1



Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Use Fixed Solidity Version	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.



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		
		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.



Informational Issues

Use Fixed Solidity Version

Resolved

Path

supesaStablecoin.sol

Description

Using a caret (^) in the pragma statement allows compilation with any compatible minor or patch version greater than or equal to 0.8.28.

While generally safe, different compiler versions may:

- Introduce new optimizations
- Change edge-case behaviors
- Modify opcode generation

This can affect reproducibility and audit assumptions.

Impact

Low

Likelihood

Low

Recommendation

Use a fixed Solidity version to ensure deterministic builds and consistent behavior across deployments.



Functional Tests

Some of the tests performed are mentioned below:

- ✓ `initialize()` executes only once and securely assigns all roles, preventing re-initialization or implementation takeover.
- ✓ Minting increases total supply only when called by `MINTER_ROLE`, and always respects pause and freeze constraints.
- ✓ Burning reduces total supply only for `BURNER_ROLE` and cannot be executed while paused or frozen.
- ✓ Global pause immediately halts all transfers, minting, and burning without breaking accounting invariants.
- ✓ Frozen accounts are fully prevented from sending, receiving, approving, or permitting tokens across all ERC20 paths.
- ✓ Transfers consistently revert when either sender or receiver is frozen, enforcing bilateral freeze safety.
- ✓ Allowances cannot be granted by frozen accounts, preventing indirect value extraction.
- ✓ ERC20 recovery cannot drain native token supply and is protected against reentrancy and over-withdrawal.
- ✓ Upgrade safety guarantees prevent storage corruption, re-initialization, or role escalation post-deployment.

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.



Trust and Assumption

The protocol architecture intentionally incorporates privileged on-chain roles with broad administrative authority, including token minting and burning, global pausing, account freezing, and role management. These capabilities inherently introduce a centralized control model and require users to place trust in the entities operating these roles.

This audit does not evaluate the operational practices or decision-making behavior of privileged role holders.

Any risks arising from centralized administrative control or misuse of privileged roles or their decisions are explicitly outside the scope of this audit and are assumed and accepted by the protocol team.

Threat Model

Contract	Function	Threats
SupesaStable coin	transfer()	Transfers may revert unexpectedly if the recipient becomes frozen between transaction signing and execution.
	transferFrom()	Allowance may exist but transfer can still fail due to freeze or pause, causing UX inconsistency for integrators.
	approve()	Frozen accounts cannot update allowances, potentially leaving stale approvals that appear valid off-chain.
	permit()	Signatures created before an account is frozen become unusable, which may surprise users relying on off-chain signing.
	burn()	Users holding BURNER_ROLE cannot burn tokens while frozen or paused, potentially blocking emergency burn flows.
	isFrozen()	Freeze state may change after off-chain checks but before transaction execution, causing race-condition reverts.



Closing Summary

In this report, we have considered the security of Supesa. We performed our audit according to the procedure described above.

No critical issues in Supesa, just 1 issue of Informational severity was found. The Supesa Team resolved the issue mentioned.

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 seven years of expertise, we've secured over 1400 projects globally, averting over \$3 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

50+

Chains Supported

1400+

Projects Secured

Follow Our Journey



AUDIT REPORT

December 2025

For

CrossFX 

 **QuillAudits**

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillaudits.com