CHAINPROOF

Audit Report

Token SLT

May 2025

Network BSC

Address 0x28ba6Ef35f27f1eD3873cc788698d2DDb28D2a82

Audited by © ChainProof

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	L09	Dead Code Elimination	Unresolved

Table of Contents

Risk Classification

The criticality of findings in ChainProof's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	SLT
Compiler Version	v0.5.16+commit.9c3226ce
Optimization	200 runs
Explorer	https://bscscan.com/address/0x28ba6ef35f27f1ed3873cc788698d2ddb28d2a82

Address	0x28ba6ef35f27f1ed3873cc788698d2ddb28d2a82
Network	BSC
Symbol	SLT
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Yes

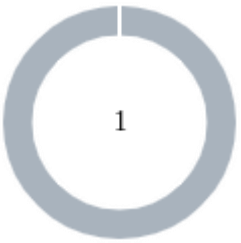
Audit Updates

Initial Audit	27 May 2025

Source Files

Filename	**SHA256**
SLT.sol	58154ae30ff3c1af21e50a2281a55c208a90e7e1a045ff63d6ff14e91ce846d0

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	1

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	1	0	0	0

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	SLT.sol#L460,476
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal {
require(account != address(0), "BEP20: burn from the zero address");
    _balances[account] = _balances[account].sub(amount, "BEP20: burn
amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
emit Transfer(account, address(0), amount);
  }  function _burnFrom(address account, uint256 amount)
internal {
    _burn(account, amount);
    _approve(account, _msgSender(),
_allowances[account][_msgSender()].sub(amount, "BEP20: burn amount
exceeds allowance"));
  }
```

Recommendation

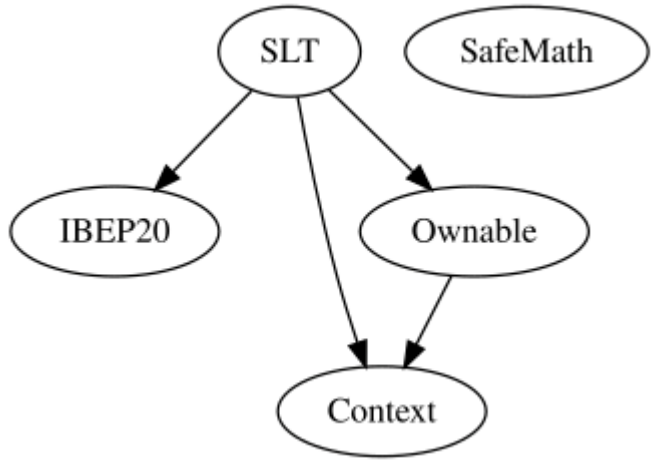
To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

Functions Analysis

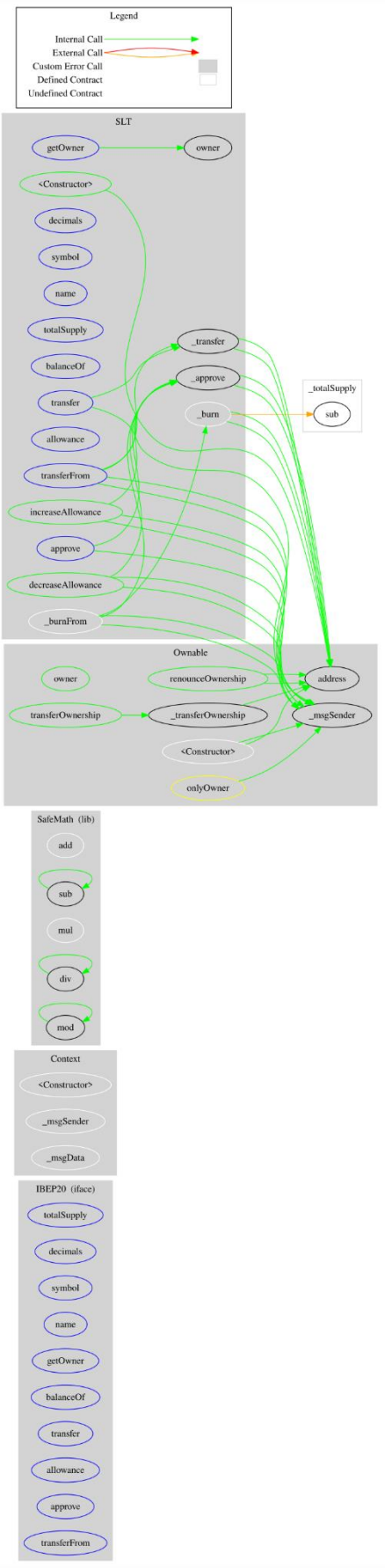
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SLT	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	getOwner	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_burnFrom	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
SLT
getOwner
owner
<Constructor>
decimals
symbol
name
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
approve
decreaseAllowance
_burnFrom
_transfer
_approve
_burn
_totalSupply
sub
Ownable
owner
renounceOwnership
address
transferOwnership
_transferOwnership
_msgSender
<Constructor>
onlyOwner
SafeMath (lib)
add
sub
mul
div
mod
Context
<Constructor>
_msgSender
_msgData
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

SLT contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. SLT is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without ChainProof’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts ChainProof to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk ChainProof’s position is that each company and individual are responsible for their own due diligence and continuous security ChainProof’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by ChainProof are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About ChainProof

Chainproof is an Audit & KYC firm for Blockchain Projects, aimed at securing the Blockchain and the assets at risk. Chainproof is fueled by Industry grade experienced Blockchain Developers from all around the globe. From finding vulnerabilities, potential scams, malicious code mitigation, improper implementation of the token which can lead to loss of user's fund, you name it and we cover and secure them all.

Security testing and risk mitigation is given the highest priority at ChainProof. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.
ChainProof is aiming to make crypto discoverable and efficient globally. We associate with extremely robust testing and code review, leaving no room for any security risks because, when it comes to user's funds, we need to leave no stone unturned. Cheers!

The ChainProof team

ChainProof.dev