Assure DeFi®

THE VERIFICATION **GOLD STANDARD**

Security Assessment

Shiro Neko

Date: 15/08/2025

Audit Status: FAIL

Audit Edition: Advanced

ASSURE DEFI®

THE VERIFICATION **GOLD STANDARD**

Risk Analysis

Vulnerability summary

Classification	Description
🔴 High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
🟠 Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
🟡 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
🟢 Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Insecure.**

Insecure	Poorly Secured	Secured	Well Secured

Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Shiro Neko contracts followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

Target Code And Revision

Project	Assure
Language	Solidity
Codebase	https://etherscan.io/token/0xb0ac2b5a73da0e67a8e5489ba922b3f8d582e058#code
Audit Methodology	Static, Manual

Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

Category	Item
Code review & Functional Review	• Compiler warnings. • Race conditions and Reentrancy. Cross-function race conditions. • Possible delays in data delivery. • Oracle calls. • Front running. • Timestamp dependence. • Integer Overflow and Underflow. • DoS with Revert. • DoS with block gas limit. • Methods execution permissions. • Economy model. • Private user data leaks. • Malicious Event log. • Scoping and Declarations. • Uninitialized storage pointers. • Arithmetic accuracy. • Design Logic. • Cross-function race conditions. • Safe Zeppelin module. • Fallback function security. • Overpowered functions / Owner privileges

AUDIT OVERVIEW

◆◆◆ **HIGH**

1. External IVerifier kill-switch that can block buys & transfers

Issue: The token calls an external IVerifier contract (via verifier.verify(from, to)) on buys and on most wallet-to-wallet transfers; if the verifier returns false the transfer reverts. The verifier address is injected at deployment and the token contract relies on that external contract to allow/deny transfers. If the team or a third party controls or can upgrade that verifier, they can arbitrarily block buys and prevent normal transfers even after token ownership has been renounced. The verifier contract address is not published so we can't verify if the code is malicious [https://etherscan.io/address/0xCf2eF65268d8863834d6D06e9777Eec3C6509b46#code]. Constructor Arguments:

Arg [0] : name_ (string): Shiro Neko

Arg [1] : symbol_ (string): SHIRO

Arg [2] : _verifier (address): 0xCf2eF65268d8863834d6D06e9777Eec3C6509b46

Arg [3] : _uniswapV2Router (address): 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D

Arg [4] : _uniswapV3Router (address): 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45

Arg [5] : _uniswapUniversalRouter (address): 0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD

Arg [6] : uniswapAddresses (address[]):
0x61fFE014bA17989E743c5F6cB21bF9697530B21e,0xA5644E29708357803b5A882D272c41cC0dF92B34,
0x000000fee13a103A10D593b9AE06b3e05F2E7E1c,0xbc708B192552e19A088b4C4B8772aEeA83bCf760

Recommendation: Remove the external verifier dependency or replace it with an on-chain, deterministic, ownerless verification logic embedded in the token contract.

If a separate verifier contract is absolutely required, ensure it is non-upgradeable and immutable (set to an address with no admin keys, or a contract whose bytecode cannot be altered) and publish proof of immutability.

Implement a fallback that allows transfers if the verifier is not responding (or set the verifier to address(0) after launch).

For future deployments: if using an external gate, make its governance handled by a multisig + timelock and public audits also add a mechanism to permanently disable the gate (for example a one-way disableVerifier() that sets the verifier to address(0) and is guarded by a timelock).

2. Permanent deployer privilege via tx.origin exclusion (bypass of checks)

Issue: The constructor marks tx.origin as excluded from limits/checks (for example when _excludeFromLimits(tx.origin, true)). Because tx.origin is the EOA that originated the transaction, that original deployer address will permanently bypass verifier, cooldown and bot checks for transactions that have that EOA as the origin. This creates a permanent privileged bypass that persists even after ownership renouncement and permits privileged behavior that ordinary users cannot replicate.

Recommendation:

Stop using tx.origin for security logic. Use msg.sender or explicit address parameters with clear semantics. tx.origin is unsafe for access-control and privilege logic.

Remove automatic permanent exclusions in the constructor. If exemption is needed for liquidity provisioning, make it time-limited or revocable.

Add admin-controlled (multisig + timelock) functions to set and revoke exclusions, and ensure exclusions can be revoked permanently (for example revokeAllExclusions()), then execute the revocation before or at renounce.

In this case as is an already-deployed contract: publicize the excluded-address list and encourage proof-of-revocation in future deployments. For this deployment, treat the address as a permanent privileged account and factor that into risk assessments.

3. Unverified / opaque verifier contract (unknown logic & potential admin control)

Issue: As named in [1. External IVerifier kill-switch that can block buys & transfers], the verifier contract used by the token (deployed with the token) is not source-verified / opaque on-chain, so its logic and any admin/upgrade mechanisms are not publicly auditable. An unverified contract can hide upgradeability, backdoors (for example owner-only toggles, DELEGATECALL to change behavior), or malicious logic that can be used as a kill-switch even after the main token owner renounces ownership.

Recommendation:

Publish the verifier source code and verification metadata on Etherscan immediately.

If the verifier exposes an owner/admin, require that owner to renounce ownership on-chain or transfer control to a provable ownerless/immutable contract. Publish the renounce tx.

If the verifier is upgradeable (proxy pattern), publish the implementation and admin/upgrade keys and either renounce upgrades or transfer admin to a public multisig + timelock.

If the project refuses to publish or renounce, treat the verifier as a hostile control point and communicate this risk to the community (prepare monitoring and contingency).

4. External-call-in-transfer increases attack surface for verifier-admin changes (upgradeability risk)

Issue: Even if the verifier currently returns true, if its admin/upgrade key is controlled by an operator the verifier logic can be changed later to perform malicious gating or reentrancy-enabled operations. Because the token performs the external verifier call inside the transfer flow, any future change to the verifier's code or state could be instantly effective and disruptive.

Recommendation:

Audit the verifier for upgradeability (proxy admin, EIP-1967 slots, or custom upgrade functions) and demand that any upgradeability be removed or transferred to a public multisig + timelock.

Add on-chain monitoring for IMPLEMENTATION / UPGRADE / admin transactions targeting the verifier and alert the community on such events.

If proof of immutability cannot be provided, plan operational contingency (holder snapshot and migration plan).

MEDIUM

No medium severity issues were found.

LOW

1. Lack of Event Emissions for Critical State Changes

Issue: Functions like _setBots(), _setRouter(), and _setAutomatedMarketMakerPair() do not emit events, making it hard to track malicious changes.

Recommendation: Emit detailed events for all critical state changes.

INFORMATIONAL

No informational issues were found.

Technical Findings Summary

Findings

Vulnerability Level	Total	Pending	Not Apply	Acknowledged	Partially Fixed	Fixed
🔴 High	4					
🟠 Medium	0					
🟡 Low	1					
🟢 Informational	0					

Assessment Results

Score Results

Review	Score
Global Score	**50/100**
Assure KYC	Not completed yet
Audit Score	50/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit FAIL

Following our comprehensive security audit of the token contract for the Shiro Neko project, the project did not fulfill the necessary criteria required to pass the security audit.

Disclaimer

ASSURE DEFI®
THE VERIFICATION **GOLD STANDARD**