Cat.World

Smart Contract Security Assessment

VERSION 1.0

AUDIT DATES: January 4th to January 5th, 2026

AUDITED BY: vard | milamber

Contents

Introduction 2

- About Algiz 2
- Disclaimer 2
- Risk Classification 3

Executive Summary 4

- About Protocol 4
- Scope and Methodology 4
- Issues Found 5

Findings 5

- Critical 6
- High 6
- Medium 6
- Low 6
  - [L-1] swapBack() can underflow in tokensToSwapForETH - liquidityTokens, causing swap-triggering transfers to revert 6
  - [L-2] setDeveloperAddress() incorrectly overwrites TreasuryAddress 7
- Informational 8
  - [I-1] Inconsistent units across admin setters (tokens vs wei) can cause accidental misconfiguration of limits and swap threshold 8
  - [I-2] Pre-trading transfer gate contains impossible condition 9
  - [I-3] Redundant check for to != address(0) 10

ᛉ Introduction

Algiz conducted an independent review of the smart-contract system. This engagement focused on evaluating the correctness of core protocol logic, identifying vulnerabilities that could lead to loss of funds or protocol manipulation, and assessing resilience under adversarial conditions.

This report summarizes the issues identified during the review, provides severity classifications, and offers actionable recommendations aimed at strengthening the protocol's security posture. The findings presented here reflect the codebase at the specified commit and may not apply to future revisions.

ᛟ About Algiz

Algiz is an independent smart contract security team specializing in helping early-stage protocols launch with confidence. Our team has identified 30+ Critical, High and Medium severity vulnerabilities across DeFi protocols, GameFi projects, and cross-chain systems through competitive audits and security research.

We combine engineering depth with product thinking - reviewing protocols not only for correctness, but for operational risk, upgrade safety, and long-term sustainability. Our founder-to-founder approach means we're a security partner, not just a vendor.

ᚾ Disclaimer

This report documents Algiz's observations based on a time-boxed review of the supplied codebase at the specified commit. The presented conclusions reflect only this snapshot of the system.

While every reasonable effort has been made to identify vulnerabilities, **no security review can guarantee** the complete absence of bugs, exploits, or unexpected behaviors. Smart-contract systems operate in adversarial, permissionless environments, and security must be treated as an ongoing process.

The mitigation recommendations included in this report are provided as guidance based on the information available during the engagement. Because this was a time-boxed review, complex or large-scale remediation efforts may require extended follow-up. In cases where fixes introduce substantial refactoring or modify core logic, we strongly advise conducting an additional full review to ensure the changes do not introduce new risks.

Subsequent security reviews, bug-bounty programs, and continuous on-chain monitoring are strongly recommended to maintain long-term protocol safety.

Risk Classification

Severity Level	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Informational

Impact	Description
High	Leads to a significant material loss of assets in the protocol, significantly harms a group of users, or disrupts a core functionality.
Medium	Leads to a moderate material loss of assets in the protocol, moderately harms a group of users, or affects ancillary functionalities.
Low	Leads to a minor material loss of assets in the protocol or to any unexpected behavior with some of the protocol's functionalities, but does not meet the criteria for higher severity.

Likelihood	Description
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. Vectors that require larger amount of capital to exercise relative to the amount gained or disruption of the protocol.
Low	Has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

ᛉ Executive Summary

Algiz conducted a focused security assessment of Cat.Town's KIBBLE ERC20 token contract deployed on Base. The review identified 2 Low-severity issues and 3 Informational observations, with no Critical, High, or Medium vulnerabilities discovered.

Overall, the codebase follows generally standard patterns for ERC20 tokens with trading controls. The findings are primarily related to operational safety, robustness, and code hygiene, rather than an actively exploitable attacker path.

The [L-1] developer address setter misconfiguration updates `TreasuryAddress` instead of `DeveloperAddress`, which can cause operational misconfiguration and misrouting of treasury distributions. [L-2] swapBack arithmetic robustness shows that `swapBack()` contains a subtraction that can underflow if internal accounting counters diverge from the contract's actual token balance. However, taxes have been set to null and since the contract is not executing swaps anymore, those issues are acknowledged without the need to be fixed.

Informational items include inconsistent unit expectations across admin setters and unreachable or redundant code.

ᚨ About Protocol

Cat Town is a cozy onchain world where users can fish, quest, collect, and discover new surprises as the town grows over time. Users can earn KIBBLE by taking part in village life and use it to shape their journey, unlock new features, and feel more connected to the world and community you're helping bring to life.

Kibble Token is a new ERC-20 token deployed on Base that is an automatic liquidity providing protocol.

ᛗ Scope and Methodology

This security assessment covers the following deployed token contract on Base with a thorough manual review:

KibbleToken	ERC20	`0x64cc19a52f4d631ef5be07947caba14ae00c52eb`

↑ Issues Found

Severity	**Total**	Acknowledged	Resolved
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	2	0
Informational	3	3	0
Total Issues	**5**	**5**	**0**

Y Findings

ID	Title	Severity	Status
L-1	`swapBack()` can underflow in `tokensToSwapForETH - liquidityTokens`, causing swap-triggering transfers to revert	Low	Acknowledged
L-2	`setDeveloperAddress()` incorrectly overwrites the `TreasuryAddress`	Low	Acknowledged
I-1	Inconsistent units across admin setters (tokens vs wei) can cause accidental misconfiguration of limits and swap threshold	Informational	Acknowledged
I-2	Pre-trading transfer gate contains impossible condition	Informational	Acknowledged
I-3	Redundant check for `to != address(0)`	Informational	Acknowledged

ᚦ Critical

No critical vulnerabilities were identified during this assessment.

ᛞ High

No high severity vulnerabilities were identified during this assessment.

ᛗ Medium

No medium severity vulnerabilities were identified during this assessment.

ᛚ Low

The following Low severity vulnerabilities were identified during this assessment:

[L-1] swapBack() can underflow in tokensToSwapForETH - liquidityTokens, causing swap-triggering transfers to revert

IMPACT:	Low	**SEVERITY:**	Low
LIKELIHOOD:	Medium	**STATUS:**	Acknowledged

Target

`KibbleToken`

Description

`swapBack()` performs `swapTokensForEth(tokensToSwapForETH - liquidityTokens)` under the assumption that `tokensToSwapForETH >= liquidityTokens`. This assumption is not guaranteed. If `tokensToSwapForETH < liquidityTokens`, the subtraction underflows and `swapBack()` reverts. Since `swapBack()` is invoked inside `_transfer()`, this can cause subsequent sells / swap-triggering transfers to revert, disrupting trading and operations.

The contract tracks fee allocations using logical counters:

- `tokensForLiquidity`
- `tokensForDeveloper`
- `tokensForTreasury`

These counters are intended to represent how the contract's actual token balance should be distributed during swapback, but they are not inherently tied to the current balance and can diverge over time (e.g., via admin actions or partial processing).

When `swapBack()` runs, it computes:

```
uint256 totalTokensToSwap = tokensForLiquidity + tokensForDeveloper;

uint256 tokensToSwapForETH =
    contractBalance * totalTokensToSwap
    / (totalTokensToSwap + tokensForTreasury);

uint256 liquidityTokens = tokensForLiquidity / 2;

swapTokensForEth(tokensToSwapForETH - liquidityTokens);
```

If `tokensToSwapForETH < liquidityTokens`, then `tokensToSwapForETH - liquidityTokens` underflows and reverts.

Impact

- Once a state is reached where `tokensToSwapForETH < liquidityTokens`, every subsequent transfer that triggers `swapBack()` will revert.
- Since the `swapBack()` trigger is evaluated during `_transfer()`, this typically affects sells (and can also affect wallet-to-wallet transfers depending on trigger conditions), resulting in partial trading disruption until the state is corrected.

Recommendation

- Harden the subtraction to prevent underflow and ensure counters are processed proportionally to the actual balance being handled OR
- Scale `liquidityTokens` to the same processed slice as `contractBalance`

[L-2] setDeveloperAddress() incorrectly overwrites TreasuryAddress

IMPACT:	Low	**SEVERITY:**	Low
LIKELIHOOD:	Medium	**STATUS:**	Acknowledged

Target

`KibbleToken`

Description

`setDeveloperAddress()` incorrectly updates `TreasuryAddress` instead of `DevAddress`. As a result, attempts to rotate the developer wallet will not work, and the treasury destination can be unintentionally overwritten.

```
  function setTreasuryAddress(address _TreasuryAddress) external onlyOwner {
    require(_TreasuryAddress != address(0), "_TreasuryAddress address cannot be 0");
    TreasuryAddress = payable(_TreasuryAddress);
    emit UpdatedTreasuryAddress(_TreasuryAddress);
 }

 function setDeveloperAddress(address _developerAddress) external onlyOwner {
    require(_developerAddress != address(0), "_developerAddress address cannot be 0");
    TreasuryAddress = payable(_developerAddress);    //@audit TreasuryAddress instead of
DeveloperAddress
    emit UpdatedDeveloperAddress(_developerAddress);
 }
```

Impact

- Calling `setDeveloperAddress(newDev)` does not update `DevAddress`, so any ETH distributions intended for the developer wallet will continue going to the old `DevAddress`.
- The call overwrites `TreasuryAddress` instead, which can redirect treasury token distributions to an unintended address.

Recommendation

Update the state of the `DeveloperAddress` instead of `TreasuryAddress`:

```
  function setDeveloperAddress(address _developerAddress) external onlyOwner {
     require(_developerAddress != address(0), "_developerAddress address cannot be 0");
-       TreasuryAddress = payable(_developerAddress);
+       DeveloperAddress = payable(_developerAddress);
     emit UpdatedDeveloperAddress(_developerAddress);
  }
```

Informational

[I-1] Inconsistent units across admin setters (tokens vs wei) can cause accidental misconfiguration of limits and swap threshold

IMPACT: Low **SEVERITY:** Informational

LIKELIHOOD:	Low	**STATUS:**	Acknowledged

Target

`KibbleToken`

Description

The contract's admin setters use different input units:

- `updateMaxBuyAmount` , `updateMaxSellAmount` , `updateMaxWalletAmount` expect whole tokens (human-readable units) and internally multiply by `1e18` .
- `updateSwapTokensAtAmount` expects base units (`wei`) and stores the value directly.

This inconsistency is a configuration footgun: an operator can easily pass the wrong unit and unintentionally disable limits or make `swapBack()` trigger far too often.

Impact

Owner-only, but has misconfiguration risk; No direct attack path.

Recommendation

Adopt a single consistent convention for all amount setters.

[I-2] Pre-trading transfer gate contains impossible condition

IMPACT:	Low	**SEVERITY:**	Informational
LIKELIHOOD:	Low	**STATUS:**	Acknowledged

Target

`KibbleToken`

Description

In `_transfer()` , the pre-trading gate contains a redundant/contradictory require that is impossible to satisfy if reached:

```
    if (limitsInEffect) {
      if (from != owner() && to != owner() && to != address(0) && to != address(0xdead)) {
        if (!tradingActive) {
          require(_isExcludedMaxTransactionAmount[from] || _isExcludedMaxTransactionAmount[to],
  "Trading is not active.");
          require(from == owner(), "Trading is enabled");   //@audit already checked at higher level
        }
        ...
    }
}
```

The second `require(from == owner)` will always revert if the `!tradingActive` branch is reached, because the outer condition already enforces the opposite `from != owner`.

Impact

Dead branch

Recommendation

Remove the redundant/dead code and refactor to align with the correct protocol intent.

[I-3] Redundant check for to != address(0)

IMPACT:	Low	**SEVERITY:**	Informational
LIKELIHOOD:	Low	**STATUS:**	Acknowledged

Target

`KibbleToken`

Description

In `_transfer()`, the initial require checks `to != address(0)`. However in the second if statement this check is again enforced:

```
function _transfer(address from, address to, uint256 amount) internal override {

     require(from != address(0), "ERC20: transfer from the zero address");
     require(to != address(0), "ERC20: transfer to the zero address");   //@audit check
     require(amount > 0, "amount must be greater than 0");

     if (limitsInEffect) {
        if (from != owner() && to != owner() && to != address(0) && to != address(0xdead)) { //@audit to !=
address(0) already checked
     ...
  }
}
```

Second check is redundant.

Impact

No impact. Redundant code.

Recommendation

Remove the redundant/dead code