yAudit UnKat Review

Review Resources:

- None beyond the code repositories

Auditors:

- Invader
- Spalen

Table of Contents

Review Summary

UnKat

UnKat enables the creation of user vaults that connect to protocols running reward campaigns on Merkl. These vaults automatically account for Kat token rewards distributed through campaigns and mint liquid unKat tokens at a 1:1 ratio to the locked Kat tokens received. Once the initial lock period has passed, the locked Kat tokens become claimable through the vault. They are transferred to the UnKat contract, from which a user can then redeem the underlying Kat tokens by burning the corresponding UnKat tokens.

The contracts of the UnKat Repo were reviewed over two days. Two auditors performed the code review between June 16 and June 17, 2025. The review was limited to the latest commit 07b71793b78d3c33673391e42572a409b01ceeb3 for the UnKat repo.

After the initial review, additional changes were made to the contracts. The latest commit reviewed was 454a95ba1b1469c6814df107a9b277662c5a53d1. New changes were reviewed on July 28, 2025, and no new issues were found; only two informational findings were added(2, 3).

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
└── src
    ├── UnKat.sol
    ├── UnKatFactory.sol
    └── UnKatVault.sol
```

After the findings were presented to the UnKat team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, UnKat and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Access control is implemented correctly to limit access to the functions. Each user is the owner of their own UnKatVault. Claiming rewards is public, allowing anyone to claim rewards regardless of the vault owner; no rewards are locked to the vault owner.
Mathematics	Good	Only basic math is used
Complexity	Good	The contracts are simple and easy to understand.
Libraries	Good	The contracts use the OpenZeppelin contracts for main functionalities. The Merkl library is used for compatibility when accessing Merkl rewards.
Decentralization	Average	The admin can limit the deployment of new UnKatVaults and change fees.
Code stability	Good	No additional changes were made to the contracts repository during the review.
Documentation	Good	All contract functions are documented.
Monitoring	Good	All actions are logged using events.
Testing and verification	Average	The tests cover all functions and use fuzzing.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact

    - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Undetermined
    - Findings whose impact could not be fully assessed within the time constraints of the engagement. These issues may range from low to critical severity, and while their exact consequences remain uncertain, they present enough potential risk to warrant attention and remediation.
- Gas savings
    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None.

High Findings

None.

Medium Findings

None.

Low Findings

None.

Undetermined Findings

None.

Gas Saving Findings

1. Gas - Use an interface instead of the contract implementation

Using interfaces instead of contract implementation can save gas.

Technical Details

Contract Distributor is imported as a contract, but it is not deployed. Using only the interface instead of the whole contract implementation can save gas.

Impact

Gas savings.

Recommendation

Create interface `IDistributor` and import it instead of `Distributor` contract here and here.

Developer Response

Acknowledged.

2. Gas - Cache array length

Caching variables can save gas.

Technical Details

Array length `tokens.length` is called 3 times. Calling it only once and saving the value into a local variable can save gas.

Impact

Gas savings

Recommendation

Save `tokens.length` into a local variable.

Developer Response

Acknowledged.

Informational Findings

1. Informational - Call-Blacklist may be inadequate

UnKatVault's multicall employs a blacklist to block direct calls to core contracts (Merkl Distributor, unKat, Kat token, Factory, itself). The addition of the multicall opens up the potential for issues as users can execute arbitrary calls, and the addition of a base set of blacklisted addresses may not be sufficient to stop this.

Technical Details

In UnKatVault.multicall, every entry in the targets array is checked only against a fixed set of forbidden addresses:

```
require(
    targets[i] != address(merklDistributor) &&
    targets[i] != address(unKat) &&
    targets[i] != address(kat) &&
    targets[i] != address(factory) &&
    targets[i] != address(this),
    InvalidTarget()
);
```

While it appears fine at a surface level, it's unknown whether there will be any additional issues or whether simply engaging with an intermediate contract to bypass the restriction will cause any potential problems.

Impact

Informational - While we understand the use case, it generally doesn't seem like a good idea to introduce a multicall in a contract, and a hardcoded list may not be a sufficient guard.

Recommendation

One idea would be to move the blacklist to the factory. You can deploy it with this base list of blacklisted addresses, but also allow the factory owner to set new ones, and potentially even disable the multicall. It's not very decentralized, but we assume trusted owners.

Developer Response

Fixed: https://github.com/UnlockedCash/unKat/commit/2fc14b82fa031461a48e0633f75aedd8ec600da3 and https://github.com/UnlockedCash/unKat/commit/de683e52f0930de327e8ee9e3e6adba5695b7e2b.
Added a whitelist mapping in the factory; we'll just whitelist any contract we integrate on the frontend.

2. Informational - Ops fee can be higher than referral fee

Technical Details

Ops fee is set in the constructor without validation that the value is above the referral fee. `getFees()` will underflow and revert if `opsfee` is lower than `referralfee`.

Impact

Informational.

Recommendation

Add an additional check in the constructor that `opsFee` is above `MAX_REF_FEE`. Follow the same pattern as in the function `setOpsFee()`.

Developer Response

Acknowledged.

3. Informational - Setting referral fee for `address(0)` will cause onClaim to revert

Technical Details

Setting the referral fee for `address(0)` will cause onClaim to revert. Minting to `address(0)` will revert.

Impact

Informational

Recommendation

Validate that referral address is not `address(0)`.

Developer Response

Acknowledged.

Final Remarks

UnKat provides users with the option to get liquid tokens while waiting for their Kat token rewards to be claimable. The protocol is simple and easy to understand. The team has been responsive and helpful during the review process. No issues were found with the code.

The code was later modified and reaudited - the new changes have simplified contracts by utilizing the Merkl Distributor callback function. A user now only needs to specify the UnKat contract address as a receiver of Kat rewards, and UnKat tokens are automatically minted to the user. After the initial lock period, the user can claim their KAT tokens by burning the corresponding UnKat tokens in a fixed 1:1 ratio.