CANTINA

Aragon Katana
Security Review

Cantina Managed review by:

Rikard Hjort, Lead Security Researcher
High Byte, Security Researcher

November 4, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Aragon gives organizations the tools to build, govern, and accrue value effectively onchain.

From Oct 12th to Oct 15th the Cantina team conducted a review of katana-governance on commit hash 5048116b. The team identified a total of **12** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	0	0	0
Gas Optimizations	2	2	0
Informational	8	6	2
Total	**12**	**10**	**2**

3 Findings

3.1 High Risk

3.1.1 Auto-compound strategy can have its rewards stolen

Severity: High Risk

Context: AragonMerklAutoCompoundStrategy.sol#L93

Description: In order to have the `Swapper` perform claims for it, `AragonMerklAutoCompoundStrategy` sets `Swapper` as its operator. That is necessary for the call to pass the following check, in the `Distributor` contract from Merkl:

```
if (msg.sender != user && tx.origin != user && operators[user][msg.sender] == 0)
    revert Errors.NotWhitelisted();
```

However, this violates the principle that the `Swapper` should not be trusted or hold approvals between transactions, and it allows other users to steal the auto-compounder's rewards by using the arbitrary execution privileges of the `Swapper`.

Any attacker with a valid claim can pass such a claim to `claimAndSwap()`, thereby succeeding in the call on line 70 of `Swapper`:

```
rewardDistributor.claim(users, _claim.tokens, _claim.amounts, _claim.proofs);
```

By clearing that, the attacker can pass any `_actions` they prefer to the `Executor`, and perform arbitrary executions. This includes calling the `Distributor` again, and claiming rewards on behalf of the auto compounder strategy. The next action can then send those tokens to the attacker.

Proof of Concept: User Alice has a claim on the distributor. She calls the `Swapper` with a valid claim for herself, with correct `token` and `amount` parameters. This can be a claim she has already made; it's only necessary that it exists in the Merkle root of the distributor.

In the `_actions` she has encoded the following transaction:

```
rewardDistributor.claim([autoCompounder], [rewardToken], [amountAwarded], [merkleProof], [alice] /* recipient
↪  */, [""]);
```

If there are multiple rewards available, she can populate the arrays with more copies of the auto-compounder address and the correct tokens, amounts and merkle proofs to claim them, as well as more transfer actions.

She can also do this for any other user that has made the `Swapper` an operator. However, only if those users have set the `Swapper` as `recipient` in the `Distributor` can she actually steal funds. Otherwise they will just be paid out. This might still be disruptive if the recipients do not expect to receive funds out-of-band, and can cause the funds to be stuck.

Recommendation: First, remove the `toggleOperator()` call in the initializer of `AragonMerklAutoCompound-Strategy`.

Then, there are a few ways to deal with this issue, while keeping the functionality.

One is for `AragonMerklAutoCompoundStrategy` to not call the `Swapper` but instead inherit from the contract, or `delegatecall` to it, so as to execute it in its own context.

Other solutions, which can be implemented together, are:

1. Call `toggleOperator()` with the swapper at the beginning of `claimAndCompound()`, and again before exiting. This guarantees the `Swapper` does not have privileges outside of the scope of that transaction.
2. Change the `Swapper` to check that it is never calling the `Distributor` contract in the `_actions`.

Aragon: Fixed in PR 21.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 `AragonMerklAutoCompoundStrategy` does not set `Swapper` as its recipient for claims

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The idea behind the auto compounder and the swapper is that the auto compounder should call the swapper, which.

1. Claims rewards from the distributor on behalf of the auto compounder,
2. Performs some set of swaps,
3. Returns the full amount of tokens to the auto compounder.

The auto compounder would then deposit these token by locking them in the escrow. However, the Merkl `Distributor` contains the following lines:

```
if (msg.sender != user || recipient == address(0)) {
    address userSetRecipient = claimRecipient[user][token];
    if (userSetRecipient == address(0)) recipient = user;
    else recipient = userSetRecipient;
}
```

Since the swapper sets the auto compounder as the `user` and `msg.sender` will always be the swapper, this `if` block will always be entered. If the auto compound has not set the swapper as a recipient, the recipient will be `user`, i.e. the auto compounder itself. Thus the tokens will never go to the swapper, which will not have a chance to swap them. Instead they will end up as balances in the auto compounder.

Furthermore, the swapper, inspecting its balance, would find 0 tokens have been gained:

```
tokenAmountGained = escrowToken.balanceOf(address(this));

// [...]

return (tokenAmountGained, lock.tokenId);
```

This returned value, `tokenAmountGained`, is used by the auto compounder (as `claimedAmount`) to determine how much should be deposited in escrow.

```
(uint256 claimedAmount,) = ISwapper(swapper).claimAndSwap(claimTokens, _actions, 0);

// If claimedAmount is greater than 0, autocompound received some amounts on `token`.
// Donate to vault to increase totalAssets without minting shares.
// This increases the value of all existing shares proportionally.
if (claimedAmount > 0) {
    _deposit(claimedAmount);
}
```

This will be 0. There is no other direct way to trigger deposits on the auto compounder. Thus, the tokens will be stuck, until an appropriate upgrade of the implementation contract is made. This disrupts the functioning of the strategy and costs some amount of lost gauge voting power.

Recommendation: Set the swapper address as a claims recipient for all relevant tokens, if known, in the initializer. Alternatively, set and unset the claim recipient before every call to the swapper:

```
function claimAndCompound(
    address[] calldata _tokens,
    uint256[] calldata _amounts,
    bytes32[][] calldata _proofs,
    Action[] calldata _actions
)
    public
    virtual
    auth(AUTOCOMPOUND_STRATEGY_CLAIM_COMPOUND_ROLE)
    returns (uint256)
{
    // Grant swapper temporary permission to claim on behalf of this contract.
    _toggleSwapperOperator();

    for (uint256 i; i < _tokens.length; ++i)  {
```

```
        IRewardsDistributor(rewardsDistributor).setClaimRecipient(swapper, _tokens[i]); // <-- add this
    }

    // which tokens to claim for with their proofs and amounts.
    ISwapper.Claim memory claimTokens = ISwapper.Claim(_tokens, _amounts, _proofs);

    (uint256 claimedAmount,) = ISwapper(swapper).claimAndSwap(claimTokens, _actions, 0);

    // If claimedAmount is greater than 0, autocompound received some amounts on `token`.
    // Donate to vault to increase totalAssets without minting shares.
    // This increases the value of all existing shares proportionally.
    if (claimedAmount > 0) {
        _deposit(claimedAmount);
    }

    // Revoke swapper's permission.
    for (uint256 i; i < _tokens.length; ++i) {
        IRewardsDistributor(rewardsDistributor).setClaimRecipient(address(0), _tokens[i]); // <-- add this
    }
    _toggleSwapperOperator();

    return claimedAmount;
}
```

Aragon: Fixed in PR 25.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 `safeTransferFrom()` unecessary to known contracts

Severity: Gas Optimization

Context: AvKATVault.sol#L150, AvKATVault.sol#L242

Description: The `AvKATVault` contract transfers NFTs representing positions. At some places it uses `transferFrom()`, at others `safeTransferFrom()`. `safeTransferFrom()` makes sense to unknown or untrusted addresses, as long as reentrancy is prevented. For known contracts, especially `this`, it is a waste of gas, if it is known and checked ahead of time that the contract is capable of receiving NFTs and handling them correctly.

Recommendation: Use `transferFrom()` when transferring to the strategy or the vault itself. Ensure that any new strategy can properly handle the NFTs. This responsibility falls on the DAO who are responsible for the upgrades.

Aragon: Fixed in PR 20.

Cantina Managed: Fix verified.

3.3.2 Avoid redundant storage writes

Severity: Gas Optimization

Context: AvKATVault.sol#L340-L344

Summary: In `AvKATVault`, during `_setStrategy()`, the variable `strategy` is potentially written to twice.

Recommendation: Instead of writing twice, you can rewrite as:

```
strategy = _strategy != 0 ? _strategy : defaultStrategy
```

which is one less storage write.

Aragon: Fixed in PR 20.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Swapper contract should never hold tokens or allowances between transactions

Severity: Informational

Context: Swapper.sol#L64, Swapper.sol#L70-L75

Description: The `Swapper` contract allows any user with a claim on the `rewardDistributor` to execute arbitrary code in the context of the `Swapper` contract. After the `Swapper` performs a `claim` on `rewardDistributor` (which will fail if `msg.sender` does not pass a valid claim for their address), it performs a `delegatecall` on an `Executor` contract, which will perform the actions passed to it as external calls with arbitrary data.

This may include sending tokens, of any kind, including base tokens. It is thus unsafe to trust the `Swapper` to hold any privileged position in the system, or to let it hold any balances.

The `Swapper` only serves as a utility for other contracts to briefly take control over for the duration of a transaction.

Aragon: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 Check on executor is insufficent

Severity: Informational

Context: Swapper.sol#L32-L34

Description: In the constructor, `Swapper` checks that the `_executor` parameter is non-zero. This is to guarantee that any call to it fails properly when passing actions for it to perform, e.g. if the contract does not have an `IExecutor.execute` function signature. If the address does not have a contract deployed at it, then `delegatecall` will succeed.

This is a good check but could be greatly improved by checking for code at the given address. There are many ways an incorrect and unpopulated address could accidentally be supplied, e.g. in a multichain scenario, passing the address of a deployed executor on one chain that is not present on the current chain, or by single-character typos.

Recommendation: Check `_executor.code.length > 0` instead. If the executor is already deployed on-chain, consider hard-coding its address as a constant.

Aragon: Fixed in PR 20.

Cantina Managed: Fix verified.

3.4.3 Low granularity for choosing percentage KAT to lock

Severity: Informational

Context: Swapper.sol#L53

Description: When claiming through the `Swapper`, the user (or strategy) can choose a percentage to lock up in escrow. This amount can be chosen as whole percentage points. This approach does not offer very granular control, especially around the edges. Choosing between 4% and 5% (or 96% and 95%) for example is a step of 25%.

Recommendation: Consider using basis points (1/10 000, or 1/100%) as your percentage setting. This is industry standard.

Aragon: Fixed in PR 20.

Cantina Managed: Fix verified.

3.4.4 Ensure no approvals linger unnecessarily

Severity: Informational

Context: AvKATVault.sol#L301-L304

Description: After `deposit()` into `AvKATVault`, lingering approvals may remain if not all funds are spent.

Recommendation: As a safeguard you could call `approve(0)` after the call to `deposit()`.

Aragon: Acknowledged.

Cantina Managed: Acknowledged.

3.4.5 Dangerous `delegatecall` can be replaced with inheritance

Severity: Informational

Context: Swapper.sol#L73-L75

Description: `delegatecall` without strict input sanitization is potentially very dangerous. For context, the contract in question is in Executor.sol, and while it could be harmless, since setting the target executor is done dynamically it can be anything in deployment.

Recommendation: instead of referencing an external contract, the executor's logic can be either inherited in this contract or even simply implement the `execute` logic in-place and simply do an internal call to `execute` and save even more gas and complexity.

Aragon: Fixed in PR 21.

Cantina Managed: Fix verified.

3.4.6 Call `_pause()` before other initializer settings

Severity: Informational

Context: AvKATVault.sol#L101

Description: `_pause()` is being called after operations which may potentially reenter the contract, anything above it such as `escrow.token()` may potentially reenter.

Recommendation: This edge case scenario can be prevented by calling `_pause()` as early as possible. It is simply a safeguard for good measure.

Aragon: Fixed in PR 20.

Cantina Managed: Fix verified.

3.4.7 KAT tokens stuck in the swapper cannot be rescued

Severity: Informational

Context: Swapper.sol#L83

Description: The `Swapper` checks its balance of KAT tokens before and after claiming rewards and performing swaps. If the balance is lower after the swaps, the transaction fails due to an underflow:

```
rewardDistributor.claim(users, _claim.tokens, _claim.amounts, _claim.proofs);
```

So it's not, for example, possible to send KAT tokens to the `Swapper` ahead of callind `claimAndSwap()` in order to compound them. It is also not possible for someone to ”rescue” or at least compound any stuck tokens.

Recommendation: Remove the before and after check on KAT tokens and simply treat the balance after the calls as the amount to compound and transfer out.

Aragon: Fixed in PR 21.

Cantina Managed: Fix verified.

3.4.8 Actions cannot send native tokens in `claimAndSwap()` because it is not `payable`

Severity: Informational

Context: Swapper.sol#L43-L50, Swapper.sol#L73-L75

Description: The `_action` array passed to `claimAndSwap()` and then used by the `Executor` to perform calls contains a `value` for each action, representing the native token to be passed to each external call. However, since `claimAndSwap()` is not `payable` it is not possible to send the tokens along directly with the call. This seems like an oversight.

Recommendation: Make `claimAndSwap()` payable. If there is no use for ever transferring native tokens in `claimAndSwap()`, then having to pass a `value` for every action is a waste of gas.

Aragon: Fixed in PR 21.

Cantina Managed: Fix verified.