Alchemix V3 POC

Security Review

Cantina Managed review by:

Rikard Hjort, Lead Security Researcher
Kankodu, Security Researcher

May 2, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Critical Risk . . . 4
  - 3.1.1 Synthetic tokens may be untransmutable . . . 4
- 3.2 High Risk . . . 5
  - 3.2.1 Lack of yield token validation can lead to loss of funds . . . 5
  - 3.2.2 Using `timeToTransmute` instead of `transmutationTime` affects already created redemptions . . . 5
- 3.3 Medium Risk . . . 6
  - 3.3.1 There should be only one Alchemist and yield token per Transmuter . . . 6
  - 3.3.2 Missing `_baseURI()` override causes empty NFT metadata . . . 7
  - 3.3.3 Changing Transmuters for an Alchemist would harm creditors . . . 7
  - 3.3.4 No Incentive to Liquidate Bad Debt Due to Zero Fees . . . 7
- 3.4 Low Risk . . . 8
  - 3.4.1 Suggested improvements to event indexing . . . 8
  - 3.4.2 Transmuter admin cannot be updated . . . 8
  - 3.4.3 Incorrect index reassignment in `removeAlchemist` prevents future removals . . . 8
  - 3.4.4 Unused and unsafe code in `EulerUSDCAdapter` . . . 9
  - 3.4.5 Not being able to lower `depositCap` makes it hard to prevent new deposits . . . 9
  - 3.4.6 Graph not properly updated for small positions at the boundary of maturation . . . 10
  - 3.4.7 `MintAllowances` Persist After NFT Transfer . . . 10
  - 3.4.8 `globalCollateralizationRatio` Can Be Manipulated . . . 10
  - 3.4.9 `EulerAdapter.price` Assumes 6-decimals . . . 11
  - 3.4.10 Include `chainId` and `transmuter` Address in ERC1155 URI for Multi-Chain Support . . . 11
  - 3.4.11 Improve ERC1155 Token Transferability and Fungibility . . . 11
- 3.5 Gas Optimization . . . 12
  - 3.5.1 Unnecessary check for unset Alchemist Position NFT . . . 12
  - 3.5.2 Unnecessary if-clause . . . 13
  - 3.5.3 Unnecessary `address(0)` check . . . 13
  - 3.5.4 Unnecessary ternary `if` . . . 13
  - 3.5.5 Check for unreasonably large redemptions can be skipped . . . 13
  - 3.5.6 Unnecessary check for validity on just created account . . . 14
  - 3.5.7 Unnecessary Existence Check . . . 14
- 3.6 Informational . . . 14
  - 3.6.1 Incorrect `recipientId` in `AlchemistV3.Deposit` event . . . 14
  - 3.6.2 Use custom errors in `require` statements (Solidity `0.8.27`) . . . 15
  - 3.6.3 Trust assumptions about the yield tokens . . . 15
  - 3.6.4 Bad import breaks compilation . . . 15
  - 3.6.5 The word "guardian" is misspelled throughout the codebase . . . 16
  - 3.6.6 Usage of `1e18` instead of `FIXED_POINT_SCALAR` . . . 16
  - 3.6.7 How to create new NFT positions in the Alchemist is not documented in the NatSpec 16
  - 3.6.8 `AlchemistPositionNFT` Contract Cannot Be Switched Smoothly After Deposits . . . 16
  - 3.6.9 Missing `TransmutationTimeSet` Event . . . 17
  - 3.6.10 Improve Code Readability and Consistency . . . 17
  - 3.6.11 Known issue: the weight system for earmarking does not properly handle decreases in the total supply of the debt token . . . 17

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Alchemix provides immediate access to loans based on the future yield of deposited collateral. The interest generated by the collateral automatically services the debt over time. Alchemix loans feature self-repaying mechanisms without interest charges or liquidation risks.

From Feb 26th to Mar 13th the Cantina team conducted a review of v3-poc on commit hash 90afd9f9. The team identified a total of **36** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	2	2	0
Medium Risk	4	4	0
Low Risk	11	10	1
Gas Optimizations	7	7	0
Informational	11	9	2
Total	**36**	**33**	**3**

3 Findings

3.1 Critical Risk

3.1.1 Synthetic tokens may be untransmutable

Severity: Critical Risk

Context: Transmuter.sol#L183-L184

Description: When creating a claim towards an Alchemist, the Transmuter uses the `totalDebt` for the Alchemist as a cap. Presumably, this is to ensure, in the multi-Alchemist scenario, that users don't overburden one Alchemist with paying back more debt than that Alchemist has issued, essentially sanctioning other Alchemists.

```
if (totalLocked + syntheticDepositAmount > IAlchemistV3(alchemist).totalDebt())
    revert DepositCapReached();
```

However, this logic is flawed, since `totalDebt` tracks how leveraged the Alchemist is, not how many synthetic tokens are outstanding from it (and thus need to be accounted for).

Specifically, `repay()` will reduce the `totalDebt` and send yield tokens to the Transmuter, hence making sure that the synthetic tokens it has issued are fully redeemable. But the `totalDebt` check will prevent redemptions in the Transmuter from being created if the syntehtic tokens have been "paid off", due to the failure to account for the yield tokens sent to the Transmuter.

Proof of Concept: In `AlchemistV3.t.sol`:

```
function testLoanRepayAndRedemption() external {
    address user = address(0xbeef);
    uint256 ownedAmount = 100e18;
    uint256 depositAmount = ownedAmount / 2;

    // Deposit yield tokens to get a position
    vm.startPrank(user);
    SafeERC20.safeApprove(address(fakeYieldToken), address(alchemist), ownedAmount);
    alchemist.deposit(depositAmount, user, 0);
    uint256 nftId = AlchemistNFTHelper.getFirstTokenId(address(0xbeef), address(alchemistNFT));

    // Mint synthetic tokens
    uint256 mintAmount = alchemist.getMaxBorrowable(nftId) / 2;
    alchemist.mint(nftId, mintAmount, user);

    // Repay the loan with yield tokens
    uint256 repayAmount = alchemist.convertDebtTokensToYield(mintAmount);
    alchemist.repay(repayAmount, nftId);

    // Create redemption in Transmuter
    SafeERC20.safeApprove(address(alToken), address(transmuterLogic), mintAmount);

    transmuterLogic.createRedemption(mintAmount); // [FAIL. Reason: DepositCapReached()]

    vm.stopPrank();
}
```

This test imagines a scenario with a single user. They deposit yield tokens, takes out a loan, then repays their loan with their synthetic tokens. In this scenario, they then try to redeem them through the Transmuter. In reality, they may have sold the tokens in the open market, and some other user is looking to redeem them to e.g. close an arbitrage. However, the redemption cannot be created because the Alchemist has no outstanding debt and `totalDebt == 0`.

Recommendation: If the protocol is designed to have only one Alchemist-Transmuter pair per syntetic token, then the check can be removed: all synthetic tokens should be redeemable through the one Transmuter.

If the intention is to have multiple Transmuters, even just during migration from V2, then it is better keep track of the total outsanding tokens issued by the Alchemist in question, separately from `totalDebt`, and use that for the above check.

Alchemix: Fixed in commit bcde3827.

Cantina Managed: Fix verified.

3.2 High Risk

3.2.1 Lack of yield token validation can lead to loss of funds

Severity: High Risk

Context: Transmuter.sol#L170

Description: The `createRedemption` function in `Transmuter` does not validate whether the `yieldToken` provided by the user is actually associated with the specified `alchemist`. Additionally, during redemption, there is no such validation either.

`ITransumter.createRedemption` is specified to take `underlying` as an input parameter. The assumption that all underlying assets are worth the same and should convert at a 1:1 exchange rate might be correct. However, the actual implementation allows users to provide a `yieldToken` instead of `underlying`, and different `yieldTokens` do not have the same value.

An attacker can exploit this by:

1. Creating a redemption request using an `alchemist` associated with a lower-value `yieldToken` but inputting a more valuable `yieldToken` in the function arguments.
2. During redemption, calculations are performed based on the cheaper `yieldToken`, but the attacker ultimately receives the more valuable token, leading to a loss for the protocol.

Additionally, the provided `yieldToken` can be an attacker-controlled ERC20 token, potentially allowing re-entrancy or manipulation via `balanceOf` calls.

Recommendation: Ensure that the `yieldToken` provided by the user is explicitly associated with the specified `alchemist`.

Alchemix: Fixed in commit c2d24165.

Cantina Managed: Fix verified.

3.2.2 Using `timeToTransmute` instead of `transmutationTime` affects already created redemptions

Severity: High Risk

Context: Transmuter.sol#L212

Description: The following line is used to calculate the amount of debt left to be transmuted in a position:

```
uint256 blocksLeft = position.maturationBlock > block.number ? position.maturationBlock - block.number: 0;
uint256 amountNottransmuted = blocksLeft > 0 ? position.amount * blocksLeft / timeToTransmute : 0;
uint256 amountTransmuted = position.amount - amountNottransmuted;
```

The global variable `timeToTransmute` determines the duration that new redemptions must wait before completely transmuting. According to the team, the Transmuter is designed so that the total time required for maturity is established when a redemption is created.

By using the variable `timeToTransmute` here, the maturation time for an existing position is affected when the global variable is updated, which is not intended.

Furthermore, as the correct maturation times are used for updating and querying the staking graph, the earmarking accounting would become incorrect, and either too little or too much would be earmarked as `timeToTransmute` changed.

Proof of Concept: `timeToTransmute` is set to 200,000 blocks, roughly 28 days.

- User A creates a redemption for 10,000 alUSD at block T. The maturation block is set to T + 200,000.
- A week and 50,000 blocks later, due to increasing yield and an unstable alUSD peg, `timeToTransmute` is halved to 100,000 blocks.
- User A decides to exit because they need liqudity. Their position has matured for a week, so their remaining position should be 7,500 alUSD, and they should receive the rest (valued at 2,500 USD) in yield tokens.

- Due to the above calculation, however, their remaining position will be calculated to:

```
uint256 blocksLeft = T + 200_000 > T + 50_000 ? T + 200_000 - (T + 50_000) : 0;     // 150_000
uint256 amountNottransmuted = 150_000 > 0 ? 10_000 * 150_000 / 100_000 : 0;  // 15_000
uint256 amountTransmuted = 10_000 - 15_000;  // Underflow!
```

- Because `timeToTransmute` halved, User A's remaining debt immediately doubled to 15,000, which is higher than their original deposit. As a result of transmutation times in general decreasing, theirs, and all existing positions, had their maturation time increased. Used A would now have to wait another week until they even start transmuting.
- Another week and 50,000 blocks go by. It's been a total of 100,000 blocks since Users A and B created their positions. At this time, `timeToTransmute` is increased to 400,000 blocks.
- At this point, User A decides to try to exit again. Their position should be half matured, and they should have approximately 5,000 debt tokens left to transmute and receive yield tokens worth 5,000 USD, minus fees.
- However, they get the following calculation:

```
uint256 blocksLeft = T + 200_000 > T + 100_000 ? T + 200_000 - (T + 100_000) : 0;     // 100_000
uint256 amountNottransmuted = 100_000 > 0 ? 10_000 * 100_000 / 400_000 : 0;  // 2_500
uint256 amountTransmuted = 7_500 - 2_500;  //
```

- User A thus is considered 3/4ths matured. As a result of maturation times generally increasing, theirs went down significantly.

Recommendation: Further down in the code, there is the following line:

```
uint256 transmutationTime = position.maturationBlock - position.startBlock;
```

Move that line up to before the problematic line and replace `timeToTransmute` in the denominator with `transmutationTime`.

Alchemix: Fixed in commit 8e68032c.

Cantina Managed: Fix verified.

3.3 Medium Risk

3.3.1 There should be only one Alchemist and yield token per Transmuter

Severity: Medium Risk

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

Description: After discussions with the team, we concluded that the correct design is to make Transmuters and Alchemists one-to-one. In Alchemix v2, having multiple underlying tokens was handled by having different Alchemists for each underlying yield token. In v3, this diversity of collateral is handled at the vault level, so that the yield token the Alchemist deals with can consist of any basket of underlying tokens the controlling DAO authorizes, through the managed Euler Eearn vault. Hence the need for multiple Alchemists per synthetic token (and hence per Transmuter) goes away.

The current design would not be able to correctly handle several Alchemists, since each Transmuter has only one staking graph, which the Alchemist(s) query to determine how much to earmark. If there were several Alchemists they would earmark extra funds for each other.

There is also a related issue with several alchemists, reported with the title "Incorrect index reassignment in `removeAlchemist` prevents future removals".

Recommendation: Redesign the Transmuter to use only one Alchemist and one yield token. Remove the `alchemist` and `yieldToken` parameters from `createRedemption()`. Also remove these fields from the `StakingPosition` struct.

Alchemix: Fixed in commit c2d24165.

Cantina Managed: Fix verified.

3.3.2 Missing `_baseURI()` override causes empty NFT metadata

Severity: Medium Risk

Context: AlchemistV3Position.sol#L13

Description: The `_baseURI()` function in AlchemistPosition is not overridden, resulting in empty NFT images and metadata in wallets and portfolio sites. These platforms rely on `tokenURI` to fetch NFT metadata, so without a valid URI, NFTs will appear blank.

Recommendation: Override `_baseURI()` and provide at least a placeholder URI like `nftMetadata.alchemix.fi/` that returns a logo. Additionally, consider including useful metadata such as the NFT's value in underlying terms. For a fully on-chain solution, you can take inspiration from UniswapV3 and SablierV2.

Alchemix: Fixed in commit 6e6c5881.

Cantina Managed: Fix verified.

3.3.3 Changing Transmuters for an Alchemist would harm creditors

Severity: Medium Risk

Context: AlchemistV3.sol#L235-L239, Transmuter.sol#L117

Description: The Alchemist allows swapping out Transmuters through the `setTransmuter()` function. However, sunsetting a Transmuter without simultaneously sunsetting (all) its Alchemist(s) would be very difficult to do in a way that does not harm creditors with debt tokens in the Transmuter.

The Transmuter relies on its privileged access to it's Alchemist(s) in order to redeem yield tokens to pay back creditors. In the event that it loses this privilege, redemptions will fail for any position that the Transmuter cannot cover. The value in the Transmuter would likely be drained by new or old creditors entering and exiting positions until all yield tokens have been siphoned off.

Recommendation: Only sunset a Transmuter, which is in some way buggy and absolutely needs swapping out. Make it possible to lower the deposit cap arbitrarily, even to 0, to block new deposits (currently, it cannot be lowered below the current `totalLocked` in the Transmuter). Make the Transmuter creditors whole with emergency funds, if available.

Alchemix: Fixed in commit 6e8b7780.

Cantina Managed: Fix verified.

3.3.4 No Incentive to Liquidate Bad Debt Due to Zero Fees

Severity: Medium Risk

Context: AlchemistV3.sol#L632-L637

Description: The current calculation results in **zero fees** for liquidating accounts with bad debt. This removes any incentive for liquidators to act, meaning that positions where debt exceeds collateral will not be cleared. Over time, these bad debt positions will **continue to deteriorate**, increasing protocol risk.

Recommendation:

- Remove the conditional logic that prevents fees from being paid when debt exceeds collateral. Always pay feeInDebt.
- Track when fees exceed remainingCollateral and replenish the transmuter from a DAO security fund when necessary.

Alchemix: Fixed in commit e258c62e.

Cantina Managed: Fix verified.

3.4 Low Risk

3.4.1 Suggested improvements to event indexing

Severity: Low Risk

Context: IAlchemistV3.sol#L455-L486

Description: The current event indexing seems inconsistent from a usability perspective and makes it hard to monitor a position via events.

Recommendation:

- On all the events that affect single positions, add indexation on the token ID affected.
- Remove the indexation on `BatchLiquidated`. The topic would be based on the keccak hash of the full array of liquidated accounts, so it would only be possible to match the full array of liquidated accounts and not search by any one account.
- Think about utilizing a `Liquidated` event, even for batch liquidations, to enable tracking of all events impacting a position.
- Remove the `Liquidate` event, which is unused.

Alchemix: Fixed in commit bcd5510e.

Cantina Managed: Fix verified.

3.4.2 Transmuter admin cannot be updated

Severity: Low Risk

Context: Transmuter.sol#L92

Description: The Transmuter admin is set once and cannot be changed, which looks like an oversight. The contract deployer has to be the admin forever which limits governance flexibility and could lead to issues if admin privileges need to be transferred.

Recommendation: Implement `setPendingAdmin` and `acceptAdmin` methods, similar to AlchemistV3, to allow controlled admin updates.

Alchemix: Fixed in commit c2d24165.

Cantina Managed: Fix verified.

3.4.3 Incorrect index reassignment in `removeAlchemist` prevents future removals

Severity: Low Risk

Context: Transmuter.sol#L106

Description: The `Transmuter` contract keeps track of allowed alchemists using an array and a mapping (`alchemist address => index`). When an alchemist is removed, the last element in the array is moved to its position before calling `pop`. However, the index of the last alchemist is **not updated** in the mapping.

As a result, this last alchemist's recorded index becomes greater than the array size, leading to an out-of-bounds error when attempting to remove it later.

Proof of Concept:

```
function testRemoveAlchemistBug() public {
    address alchemistA = address(0xA);
    address alchemistB = address(0xB);

    transmuter.addAlchemist(alchemistA);
    transmuter.addAlchemist(alchemistB);
    (uint256 alchemistBIndexBefore,) = transmuter.alchemistEntries(alchemistB);

    (uint256 alchemistAIndex,) = transmuter.alchemistEntries(alchemistA);

    transmuter.removeAlchemist(alchemistA);

    assertEq(transmuter.alchemists(alchemistAIndex), alchemistB); //this is expected that the last alchemist is
    ↪  moved to the removed alchemist's index
    (uint256 alchemistBIndex,) = transmuter.alchemistEntries(alchemistB);

    assertEq(alchemistBIndex, alchemistBIndexBefore); //alchemistB's index should not be the same as before. It
    ↪  should be replaced with removed alchemist's index

    //This is the impact of the bug. Now, alchemistB cannot be removed because it panics with array
    ↪  out-of-bounds access
    vm.expectRevert();
    transmuter.removeAlchemist(alchemistB);
}
```

Recommendation: When swapping the last element into the removed alchemist's position, update its index in the mapping.

Alchemix: Fixed in commit c2d24165.

Cantina Managed: Fix verified.

3.4.4 Unused and unsafe code in `EulerUSDCAdapter`

Severity: Low Risk

Context: EulerUSDCAdapter.sol#L26-L47

Description: The `wrap` and `unwrap` functions are unused. Furthermore, the code as it is is unsafe because `wrap()` double-sends tokens with both `deposit()` and `safeTransfer()`. `unwrap()` behaves similarly. Furthermore the USDC adapter uses a hard-coded 6 decimals to compute price, and thus the code should not be copied for another type of adapter without first modifying the decimals.

Recommendation: Remove the `unwrap()` and `wrap()` functions.

Alchemix: Fixed in commit f9ad9785.

Cantina Managed: Fix verified.

3.4.5 Not being able to lower `depositCap` makes it hard to prevent new deposits

Severity: Low Risk

Context: Transmuter.sol#L117, Transmuter.sol#L177

Description: The deposit cap cannot be lowered below what the current amount of `totalLocked` is.

```
function setDepositCap(uint256 cap) external onlyAdmin {
    _checkArgument(cap >= totalLocked);
    // ...
}
```

This seems overly restrictive.The only other use of the deposit cap is at line 177, which would not be buggy if you removed this restriction. If you want to lower the deposit cap (e.g. because you're switching out transmuters and would prefer to not allow more deposits) you should be able to.

Recommendation: Remove the line in question.

Alchemix: Fixed in commit 6e8b7780.

Cantina Managed: Fix verified.

3.4.6 Graph not properly updated for small positions at the boundary of maturation

Severity: Low Risk

Context: Transmuter.sol#L232

Description: The following line updates the staking graph with withdrawals at the end of `claimRedemption()`:

```
if (amountNottransmuted > 0) _updateStakingGraph(-position.amount.toInt256() * BLOCK_SCALING_FACTOR /
↪  transmutationTime.toInt256(), blocksLeft);
```

However, there is no limit to how small a redemption position can be. So there is no guarantee that `amountNottransmuted` will be 0 only when the position is fully matured.

Proof of Concept: User A creates many small redemptions of 100 tokens (with 18 decimals, for e.g. alUSD this is 10^-16 USD worth of tokens). If `timeToTransmute` is 200,000 blocks (~28 days), then the calculation proceeds as follows after waiting for 198,001 blocks:

```
uint256 amountNottransmuted = blocksLeft > 0 ? position.amount * blocksLeft / timeToTransmute : 0; // 100 *
↪  1_999 / 200_000 = 0
```

Now, if there are many redemptions maturing in the same block, for example, if User A created 100 of these redemptions, then the staking graph will still contain a diff of 100 tokens at the remaining blocks that will not be updated.

Recommendation: Use `blocksLeft` instead of `amountNottransmuted`:

```
if (blocksLeft > 0) _updateStakingGraph(-position.amount.toInt256() * BLOCK_SCALING_FACTOR /
↪  transmutationTime.toInt256(), blocksLeft);
```

Alchemix: Fixed in commit 571a7748.

Cantina Managed: Fix verified.

3.4.7 `MintAllowances` Persist After NFT Transfer

Severity: Low Risk

Context: AlchemistV3.sol#L400

Description: Currently, `MintAllowances` are attached to token IDs.

- Owner A of token ID x grants an allowance of a certain amount to address P.
- Owner A transfers the NFT (ID x) to owner B.

Here, Address P still retains permission to mint debt tokens using ID x, even after the transfer.

Transferring NFTs does not reset the allowances. This could be problematic if someone assumes the usual behavior of NFTs, where allowances are typically reset upon ownership transfer.

Recommendation: Reset `MintAllowances` on NFT transfer.

Alchemix: Fixed in commit bc12b2d6.

Cantina Managed: Fix verified.

3.4.8 `globalCollateralizationRatio` Can Be Manipulated

Severity: Low Risk

Context: AlchemistV3.sol#L629

Description: If `depositCap` is set to a very high value, users can manipulate the `globalCollateralizationRatio` by strategically depositing and withdrawing yieldTokens:

- To lower `globalCollateralizationRatio`.
  - Deposit a large amount of `yieldTokens`.
  - Perform a liquidation.

    - Withdraw the `yieldTokens`.
- (This benefits users liquidating their own positions).
- To increase `globalCollateralizationRatio`.
    - Deposit a large amount of `yieldTokens`.
    - Mint the maximum allowed `alAssets`.
    - Perform liquidation.
    - Repay the `alAssets` and withdraw the `yieldTokens`.
- (This benefits liquidators, maximizing fees).

Recommendation: Impose reasonable `depositCap` limits to prevent excessive deposits.

Alchemix: We acknowledge this risk and the DAO will monitor.

Cantina Managed: Acknowledged.

3.4.9 `EulerAdapter.price` Assumes 6-decimals

Severity: Low Risk

Context: EulerUSDCAdapter.sol#L22-L23

Description: The ´EulerAdapter.price´ function hardcodes 1e6 when calling the underlying vault. While this works for USDC (6 decimals), it fails for tokens with different decimal places. If deployed for other tokens, the price calculation will be incorrect.

Recommendation: Instead of using a fixed `1e6`, dynamically determine the appropriate value based on the token's decimals:

```
function price() external view returns (uint256) {
    uint8 decimals = TokenUtils.expectDecimals(token);
    return IERC4626(token).convertToAssets(10**decimals);
}
```

Ensure this update is made before deploying this adapter for non-6-decimal tokens to prevent incorrect price calculations.

Alchemix: Fixed in commit fe3ea647.

Cantina Managed: Fix verified.

3.4.10 Include `chainId` and `transmuter` Address in ERC1155 URI for Multi-Chain Support

Severity: Low Risk

Context: Transmuter.sol#L86

Description: Currently, the URI does not include chainId or transmuter address, which can cause issues when deploying multiple contracts across different chains. The server that returns ERC1155 metadata won't be able to determine which chain's or which transmuter's metadata to return for the same token ID, leading to incorrect or missing data.

Recommendation: Append chainId and transmuter address to the URI to ensure each metadata request is correctly mapped to the appropriate chain and transmuter.

Alchemix: Fixed in commit 17b95d1d.

Cantina Managed: Fix verified.

3.4.11 Improve ERC1155 Token Transferability and Fungibility

Severity: Low Risk

Context: Transmuter.sol#L193, Transmuter.sol#L216

Description: Positions are attached to both an address **and** an ID, which restricts the transferability of ERC1155 tokens. Additionally, tracking the amount in `StakingPosition.amount` further limits the fungibility of ERC1155 tokens with the same ID. A user is forced to transfer their entire ERC1155 balance for a specific ID to another wallet; otherwise, they cannot call `claimRedemption`. To resolve this, avoid tracking the amount in `StakingPosition.amount` and instead allow the user to specify the amount as an input when calling `claimRedemption`.

Proof of Concept:

```
function testClaimRedemptionERC115TransferNotAllowedBug() public {
    deal(address(collateralToken), address(transmuter), uint256(type(int256).max) / 1e20);

    vm.prank(address(0xbeef));
    transmuter.createRedemption(address(alchemist), address(collateralToken), 100e18);

    vm.roll(block.number + 5_256_000);

    //The original creator send the ERC1155 tokens to a new wallet
    vm.prank(address(0xbeef));
    transmuter.safeTransferFrom(address(0xbeef), address(0xABC), 1, 100e18, "");

    //this new wallet cannot claim redemption because positions are attached to an address + id. It should only
    ↪  be attached only ID
    vm.startPrank(address(0xABC));
    vm.expectRevert(PositionNotFound.selector);
    transmuter.claimRedemption(1);
    vm.stopPrank();
}
```

Recommendation: There is no need to track ownership of a position—let the ERC1155 token contract handle it. Additionally, do not track the amount in `StakingPosition.amount`. Instead, allow the user to specify the amount when calling `claimRedemption`. This ensures that users can transfer ERC1155 tokens freely without losing value.

Alchemix: Fixed in PR 29

Cantina Managed: Fix verified.

3.5 Gas Optimization

3.5.1 Unnecessary check for unset Alchemist Position NFT

Severity: Gas Optimization

Context: AlchemistV3.sol#L327, AlchemistV3.sol#L352, AlchemistV3.sol#L379, AlchemistV3.sol#L416

Description: The check for if `alchemistPositionNFT != address(0)` is not necessary in the `deposit()`, `withdraw()`, `mint()` and `burn()` functions because they call the NFT contract to check ownership and validity, and those calls will revert if calling to an address with no contract. Since it's a very abnormal state, one that can only happen if the NFT contract has never been set at all, it is wasteful to perform this check every time.

Proof of Concept: In the `AlchemistV3.t.sol` tests, change the `setUp()` to not set an NFT contract. Remove the check in question in `deposit()`. Run the following test:

```
function testDeposit() public {
    uint256 amount = 10;
    vm.startPrank(address(0xbeef));
    SafeERC20.safeApprove(address(fakeYieldToken), address(alchemist), amount + 100e18);
    alchemist.deposit(amount, address(0xbeef), 0);
    vm.stopPrank();
}
```

The test reverts. If the NFT contract is left to be set, the test passes.

Recommendation: Remove the unnecessary check. If you insist on using it for clarity, change `_checkArgument()` to `_checkState()`.

Alchemix: Fixed in commit 7a0b6b34.

Cantina Managed: Fix verified.

3.5.2 Unnecessary if-clause

Severity: Gas Optimization

Context: AlchemistV3.sol#L877-L881

Description: Unnecessary `if`-clause:

```
if (collateralization < minimumCollateralization) {
    return true;
}

return false;
```

Recommendation:

```
return collateralization < minimumCollateralization;
```

Alchemix: Fixed in commit 5ad1cf94.

Cantina Managed: Fix verified.

3.5.3 Unnecessary `address(0)` check

Severity: Gas Optimization

Context: AlchemistV3.sol#L349

Description: There is no need to check that the sender is not the 0-address, since the 0-address cannot be `msg.sender` on Ethereum or any major L2. There is, however, no check that `recipient` is not `address(0)`.

Recommendation: Remove the line:

```
_checkArgument(msg.sender != address(0));
```

If desired the check can be changed to:

```
_checkArgument(recipient != address(0));
```

Alchemix: Fixed in commit 4b8deac2.

Cantina Managed: Fix verified.

3.5.4 Unnecessary ternary `if`

Severity: Gas Optimization

Context: Transmuter.sol#L212

Description: Consider the following line in the Transmuter:

```
uint256 amountNottransmuted = blocksLeft > 0 ? position.amount * blocksLeft / timeToTransmute : 0;
```

If `blocksLeft` is indeed `0`, the calculation `position.amount * blocksLeft / timeToTransmute` gives the result `0`. The branching logic is unnecessary.

Recommendation: Change to:

```
uint256 amountNottransmuted = position.amount * blocksLeft / timeToTransmute;
```

Alchemix: Fixed in commit 45a52137.

Cantina Managed: Fix verified.

3.5.5 Check for unreasonably large redemptions can be skipped

Severity: Gas Optimization

Context: Transmuter.sol#L174-L175

Description: Every time a redemption is created in the Transmuter, the following checks are performed:

```
if ( syntheticDepositAmount > type(int256).max.toUint256())
    revert DepositTooLarge();

if (totalLocked + syntheticDepositAmount > depositCap)
    revert DepositCapReached();
```

The second check should trigger far more often than the first.

Recommendation: A similar result can be achieved by making a check that `depositCap` can never be larger than `type(int256).max.toUint256()`, and thus `totalLocked` can't either. This would prevent similar issues while only resulting in an overflow revert if someone submits a very large number instead of an error.

Alchemix: Fixed in commit 6e8b7780.

Cantina Managed: Fix verified.

3.5.6 Unnecessary check for validity on just created account

Severity: Gas Optimization

Context: AlchemistV3.sol#L331-L335

Description: There is a check for valid accounts performed when depositing:

```
if (tokenId == 0) {
    tokenId = IAlchemistV3Position(alchemistPositionNFT).mint(recipient);
    emit AlchemistV3PositionNFTMinted(recipient, tokenId);
}
_checkForValidAccountId(tokenId);
```

However, if `tokenId == 0` the token will have just been created. This NFT contract is trusted code. `_checkForValidAccount()` makes an external call, and is thus fairly costly.

Recommendation: Put the validity check in an `else` block.

```
if (tokenId == 0) {
    tokenId = IAlchemistV3Position(alchemistPositionNFT).mint(recipient);
    emit AlchemistV3PositionNFTMinted(recipient, tokenId);
} else {
    _checkForValidAccountId(tokenId);
}
```

Alchemix: Fixed in commit 0dc5ea6f.

Cantina Managed: Fix verified.

3.5.7 Unnecessary Existence Check

Severity: Gas Optimization

Context: AlchemistV3.sol#L743-L746

Description: The ERC721 standard specifies that an NFT with `owner == address(0)` is invalid, and `ownerOf` should revert in such cases. Since `AlchemistV3Position` correctly inherits OpenZeppelin's ERC721 implementation, a successful `ownerOf` call guarantees that the owner is not the zero address. Thus, the check `(owner != address(0))` is redundant and can be removed.

Recommendation: Remove the unnecessary check as recommended.

Alchemix: Fixed in commit e44a1988.

Cantina Managed: Fix verified.

3.6 Informational

3.6.1 Incorrect `recipientId` in `AlchemistV3.Deposit` event

Severity: Informational

Context: AlchemistV3.sol#L342

Description: In the `AlchemistV3.Deposit` event, `recipientId` provided by user does not represent the actual recipient in cases where `recipientId = 0`. When the user provides `recipientId = 0`, a new `alchemistPositionNFT` is minted, and the deposit is associated with the newly minted ID.

Recommendation: Modify the event to emit `tokenId` instead, ensuring that the correct recipient ID is reflected in the event.

Alchemix: Fixed in commit d0fee763.

Cantina Managed: Fix verified.

3.6.2 Use custom errors in `require` statements (Solidity `0.8.27`)

Severity: Informational

Context: AlchemistV3Position.sol#L22, AlchemistV3Position.sol#L31, AlchemistV3Position.sol#L42

Description: Solidity 0.8.27 allows the use of custom errors directly in `require` statements without needing the Yul (IR) pipeline. This reduces bytecode size and enables more informative errors with parameters.

Recommendation: Refactor `require` statements to use custom errors for improved efficiency and clarity.

Alchemix: Fixed in commit 5d57a2e8.

Cantina Managed: Fix verified.

3.6.3 Trust assumptions about the yield tokens

Severity: Informational

Context: AlchemistV3.sol#L340, Transmuter.sol#L234-L238, Transmuter.sol#L240-L244

Description: Alchemix v3 should only use yield tokens which revert correctly on failure to transfer, and always transfer the full amount. They must not, for example, issue fees or be able to block transfers to and from certain addresses (unless controlled by the DAO controlling Alchemix), since the transfers of yield tokens do not perform a post-check that the final balance is correct but instead only assumes that a call to `TokenUtils.safeTransferFrom()` will execute correctly or revert.

The synthetic token is assumed to be the Alchemic tokens already deployed and in use in v2, or newly deployed such tokens with the same contract code.

Recommendation: Only use Euler Earn tokens under the DAO's control, as currently planned, or revisit all token transfer locations in the code before adding support for additional yield tokens.

Alchemix: Acknowledged.

Cantina Managed: Acknowledged.

3.6.4 Bad import breaks compilation

Severity: Informational

Context: Transmuter.t.sol#L8

Description: Incorrect capitalization in the import:

```
import {AlEth} from "../external/Aleth.sol";
```

Recommendation: Capitalize the final "E":

```
import {AlEth} from "../external/AlEth.sol";
```

Alchemix: Fixed in commit f003d8dd.

Cantina Managed: Fix verified.

3.6.5 The word "guardian" is misspelled throughout the codebase

Severity: Informational

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

Description: There are many instances of the misspelling "*gaurdian*".

Recommendation: Change everywhere to "guardian".

Alchemix: Fixed in commit 265c5f8d.

Cantina Managed: Fix verified.

3.6.6 Usage of `1e18` instead of `FIXED_POINT_SCALAR`

Severity: Informational

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

Description: There are several places where the literal `1e18` is used instead of the named constant `FIXED_POINT_SCALAR`. This reduces readability and makes it unclear what the multiplication or division is intended to do.

Recommendation: Change all usages of `1e18` to `FIXED_POINT_SCALAR`.

Alchemix: Fixed in commit f98ae50f.

Cantina Managed: Fix verified.

3.6.7 How to create new NFT positions in the Alchemist is not documented in the NatSpec

Severity: Informational

Context: AlchemistV3.sol#L331-L334

Description: The way to create a new position in the Alchemist is to make a deposit with `tokenId` 0.

```
if (tokenId == 0) {
    tokenId = IAlchemistV3Position(alchemistPositionNFT).mint(recipient);
    emit AlchemistV3PositionNFTMinted(recipient, tokenId);
}
```

However this is not prominently documented in the NatSpec, and is only specified in the code.

Recommendation: Add documentation on this behavior.

Alchemix: Fixed in commit e667315a.

Cantina Managed: Fix verified.

3.6.8 `AlchemistPositionNFT` Contract Cannot Be Switched Smoothly After Deposits

Severity: Informational

Context: AlchemistV3.sol#L167

Description: Once multiple deposits have been made, smoothly replacing the AlchemistPositionNFT contract from Alchemist becomes impractical. Existing positions are tied to the current NFT contract, and switching it could lead to inconsistencies or loss of position tracking.

Recommendation: Remove the `setAlchemistPositionNFT` method to prevent unexpected changes that could break existing positions.

Alchemix: Fixed in commit 4168f338.

Cantina Managed: Fix verified.

3.6.9 Missing `TransmutationTimeSet` Event

Severity: Informational

Context: Transmuter.sol#L140-L142

Description: The contract does not emit an event when transmutationTime is set in `setTransmutationTime`.

Recommendation: Add the missing event.

Alchemix: Fixed in commit a766ea8c.

Cantina Managed: Fix verified.

3.6.10 Improve Code Readability and Consistency

Severity: Informational

Context: AlchemistV3Position.sol#L4, AlchemistV3Position.sol#L5, AlchemistV3.sol#L38, AlchemistV3.sol#L325, AlchemistV3.sol#L869, AlchemistV3.sol#L901, IAlchemistV3.sol#L28, StakingGraph.sol#L6

Description: The codebase can be cleaned up and made more readable by addressing several small issues:

1. Resolve remaining TODO comments: IAlchemistV3.sol#L28.
2. Use named imports for clarity: AlchemistV3Position.sol#L4.
3. Use `_checkState` instead of `_checkArgument` for `minimumCollateralization`, as it is a state variable. AlchemistV3.sol#L890.
4. Use named constants like `FIXED_POINT_SCALAR` instead of hardcoded values like `1e18`: AlchemistV3.sol#L890.
5. Fix spelling mistake: `expectedColltaeralForCurrentDebt` `expectedCollateralForCurrentDebt` AlchemistV3.sol#L901.
6. Simplify boolean checks: Replace `_checkState(depositsPaused == false);` with `_checkState(!depositsPaused);` for minor gas savings. AlchemistV3.sol#L325.
7. Correct the comment: `_isUnderCollateralized` returns `true` if the account is undercollateralized and `false` otherwise. It doesn't revert with an `{Undercollateralized}` error. AlchemistV3.sol#L869.
8. Remove unused imports:
   - AlchemistV3Position.sol#L5.
   - StakingGraph.sol#L6.
9. Remove unused state variable: AlchemistV3.sol#L38.
10. Resolve compiler warnings important contracts.

Recommendation: Update the code as recommended.

Alchemix: Fixed in commit 1da6f885.

Cantina Managed: Fix verified.

3.6.11 Known issue: the weight system for earmarking does not properly handle decreases in the total supply of the debt token

Severity: Informational

Context: AlchemistV3.sol#L817, AlchemistV3.sol#L843, Transmuter.sol#L260

Description: The following issue was identified by the Alchemix team shortly before the audit started. The design to fix it is being worked out, and the audit was conducted with this in mind: the earmarking and syncing logic may change in response to how this bug should be addressed.

The weight system used to account for earmarked debt does not properly handle scenarios where the total supply of debt tokens decreases, leading to inaccurate bookkeeping of user debt positions.

In the Alchemist system, users' debt decreases when the Transmuter requests collateral from the Alchemist. When a redemption is claimed from the Transmuter, tokens held in escrow are sent to the position owner, and an equal amount of debt is removed from the system. To correctly account for what portion of earmarked debt belongs to which user, the system uses a global weight that users can sync their accounts against.

These weights represent the ratio of debt that has been earmarked, calculated as:

$$\text{New Weight} = \frac{\text{Newly Earmarked Amount}}{\text{Total Supply of Debt Token}} + \text{Old Weight}$$

The issue occurs because the denominator (Total Supply of Debt Token) decreases when redemptions or burns occur, causing the weight calculation to become inaccurate. This is demonstrated in the `testPositionToFullMaturityTwoTransmuterPositions()` test where:

1. A user mints 100,000 tokens from the Alchemist.
2. A user stakes 50,000 tokens in the Transmuter.
3. After 1/4 of the transmutation period, another user stakes 50,000 tokens.
4. After a full transmutation period, 87,500 tokens are earmarked (50,000 for the first position, 37,500 for the second), giving a weight of 0.875.
5. When the first redemption occurs, the total debt decreases to 50,000.
6. When the final 12,500 tokens are earmarked, the weight is incorrectly incremented by $\frac{12,500}{50,000}$, resulting in a weight of 1.125.

This value suggests the position has more debt tokens earmarked than were minted, which is impossible. The correct increment should be $\frac{12,500}{100,000}$, yielding a weight of 1, representing that all tokens are earmarked.

The fundamental issue is that the denominator for the ratio calculation changes over time due to redemptions, making it impossible to maintain an accurate weight that works for all users across all scenarios.

Alchemix: Acknowledged.

Cantina Managed: Acknowledged.