yAudit

November 25, 2025

Prepared for
Resupply

Audited by
Adriro
HHK

Resupply Update Report

Smart Contract Security Assessment

Contents

1 Review Summary

1.1 Protocol Overview

Resupply Finance is a CDP-based lending protocol that allows simple, low-risk, leveraged yield farming while encouraging the use of value-added ecosystem protocols' underlying stables like Curve's crvUSD and Frax's FRAX.

1.2 Audit Scope

This audit covers the changes made to the ResupplyPairCore and ResupplyPairDeployer contracts, along with the new BorrowLimitController.

```
src/
  ├── dao
  │   └── operators
  │   │   └── BorrowLimitController.sol
  └── protocol
      ├── pair
      │   └── ResupplyPairCore.sol
      └── ResupplyPairDeployer.sol
```

1.3 Risk Assessment Framework

1.3.1 Severity Classification

1.4 Key Findings

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	0
Low	0
Informational	4

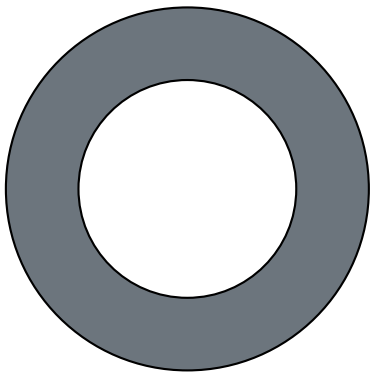

Figure 1: Distribution of security findings by impact level

1.5 Overall Assessment

This update includes post-exchange rate issue fixes, an updated deployer, and a new contract called BorrowLimitController used to slowly increase pairs' borrowing caps over time.

Severity	Description	Potential Impact
Critical	Immediate threat to user funds or protocol integrity	Direct loss of funds, protocol compromise
High	Significant security risk requiring urgent attention	Potential fund loss, major functionality disruption
Medium	Important issue that should be addressed	Limited fund risk, functionality concerns
Low	Minor issue with minimal impact	Best practice violations, minor inefficiencies
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 although their exact consequences remain uncertain, they present a sufficient potential risk to warrant attention and remediation.	Varies based on actual severity
Gas	Findings that can improve the gas efficiency of the contracts.	Reduced transaction costs
Informational	Code quality and best practice recommendations	Improved maintainability and readability

Table 1: severity classification

2 Audit Overview

2.1 Project Information

Protocol Name: Resupply
Repository: https://github.com/resupplyfi/resupply
Commit Hash: 52bf837c24720a3095146a0214a111465bd23c0e
Commit URL:
https://github.com/resupplyfi/resupply/commit/52bf837c24720a3095146a0214a111465bd23c0e

2.2 Audit Team

Adriro, HHK

2.3 Audit Timeline

The audit was conducted from July 14 to 15, 2025.

2.4 Audit Resources

Update overview

3 Resupply Review

3.1 Review Summary

The contracts of the Resupply were reviewed over two days. Two auditors performed the code review between July 14 and July 15, 2025. The repository was under active development during the review, but the review was limited to the latest commit 52bf837c24720a3095146a0214a111465bd23c0e.

3.2 Audited Files

```
src/
  ├── dao
  │   └── operators
  │   │   └── BorrowLimitController.sol
  └── protocol
      ├── pair
      │   └── ResupplyPairCore.sol
      └── ResupplyPairDeployer.sol
```

3.3 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.
- Gas savings
- Findings that can improve the gas efficiency of the contracts.
- Informational
- Findings including recommendations and best practices.

3.4 Critical Findings

None.

3.5 High Findings

None.

3.6 Medium Findings

None.

3.7 Low Findings

None.

3.8 Gas Savings Findings

None.

3.9 Informational Findings

3.9.1 Delegate exchange rate threshold to the Oracle implementation

Instead of hardcoding an arbitrary limit in the pair implementation, it could be more flexible to delegate this logic to the Oracle implementation.

Technical Details

The updated implementation of `_updateExchangeRate()` implements a hardcoded limit to detect Oracle manipulation attempts.

```
571:         // Get the latest exchange rate from the oracle
572:         uint256 priceFromOracle = IOracle(_exchangeRateInfo.oracle).getPrices(
address(collateral));
573:         //all prices should *normally* be within the 1e18 (or 1e15 for curvelend)
range
574:         //reject any prices that are well beyond that
575:         if(priceFromOracle > 1e22){
576:             revert InvalidOraclePrice();
577:         }
```

While the team has acknowledged this threshold as a reasonable limit for the share price, moving this check to the Oracle would improve flexibility, as the implementation can be updated in the pair eventually.
Note that this check is not present in other occurrences in the codebase where `getPrices()` is used, such as `previewRedeem()` in RedemptionHandler or `isSolvent()` in the Utilities contract.

Impact

Informational.

Recommendation

Delegating this logic to the Oracle would allow different strategies, for example:

- Create an Oracle with a configurable limit.
- Define an Oracle for each protocol to adapt to potential pricing particularities (e.g., account for the 1e15 / 1e18 difference in Curve and Frax).
- Implement more advanced strategies, such as a time-weighted Oracle or a dual Oracle.

Developer Response

Updated Core and Basic Vault Oracle here: c6a80f0.

3.9.2 `_migrateState()` could assign the wrong protocol ID

Technical Details

Inside `_migrateState()`, the function iterates through `supportedProtocols` and executes `getBorrowAndCollateralTokens()` to determine the protocol ID used for that pair. It then updates the storage mapping to `_deployer.collateralId()`.
However, suppose two `supportedProtocols` have the same methods to query the `_borrowToken` and `_collateralToken`. In that case, the loop will exit early and use the first matching protocol's ID to get `_deployer.collateralId()` and store it.
Currently, only Frax and Curve are being used, and their methods are different, so this will happen only if new protocols are supported in the future.

Impact

Informational. If the deployer is migrated again with new supported protocols, then the mapping may be incorrectly migrated.

Recommendation

Check that `_deployer.collateralId() > 0` inside the `if(_borrowToken != address(0) && _collateralToken != address(0))` condition.

Developer Response

Agree, this could be an issue if a future supported protocol were added with conflicting selectors. That will not happen before we upgrade the deployer, but nonetheless have re-worked the `_migrateState()` flow a bit: 7445002.

3.9.3 Current deployer is not registered in the ResupplyRegistry

The current deployer contract isn't registered in the main registry, skipping the migration logic in the new deployer.

Technical Details

The new ResupplyPairDeployer contract performs a state migration from the previous deployer during construction.

```
139:         address _previousPairDeployer = IResupplyRegistry(registry).getAddress("
DEPLOYER");
140:         if(_previousPairDeployer != address(0)) {
141:             _migrateState(_previousPairDeployer);
142:         }
```

Currently, the registry (`0x10101010E0C3171D894B71B3400668aF311e7D94`) doesn't contain the "DEPLOYER" key, causing the migration logic to be skipped.

Impact

Informational.

Recommendation

Ensure the current deployer contract is registered to enable the migration mechanism when the new deployer contract is created.

Developer Response

Pair deployer registry check updated to look for `PAIR_DEPLOYER` key: 7445002.
`PAIR_DEPLOYER` key set on registry via this transaction:
0xe644dc0e44ec6d4f51ccf9d8966d63f0bcc3edfe26ab93e499d87169d63331f7.

3.9.4 Missing burn amount checks in `updateSupportedProtocol()`

Technical Details

The sanity checks over `amountToBurn` and `minShareBurnAmount` applied in `addSupportedProtocol()` are missing while the config is updated in `updateSupportedProtocol()`.

Impact

Informational.

Recommendation

Add the sanity checks to `updateSupportedProtocol()`.

Developer Response

Fixed in f6da20f.

3.10 Final Remarks

Overall, the protocol demonstrates good security practices and code quality. The team has been responsive to feedback and has implemented most recommendations. We recommend addressing the identified issues before mainnet deployment.