Rocko Flash Refinance Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Hans

March 28, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Rocko is a protocol which aims to simplify DeFi lending user experience by:

- collating a multitude of DeFi loan offers
- allowing users to easily choose the best or most attractive loan offers

The new "Flash Refinance" component aims to make refinancing of existing loans easy and seamless for users by handling all third-party protocol interactions using the new `RockoFlashRefinance.sol` smart contract. Instead of users previously having to interact with multiple different third-party protocol-specific smart contracts, users can now complete the entire refinance in one transaction by interacting with Rocko's new smart contract. The result is instant refinancing, lower gas costs and a smoother, frictionless user experience.

The `RockoFlashRefinance` smart contract features:

- minimal admin privileges; admins can only set a fee which is capped <= 1%, pause the contract and change the fee address. Users can use this contract safely in the knowledge the admin has no control over funds in transit
- immutable; the contract is not upgradeable so the admin can't change its code once deployed, giving users greater trust that the admin can't upgrade the implementation to execute new code they weren't expecting
- no storage of funds; after every refinance transaction is complete, there are no funds remaining in this contract. Additionally there are no funds belonging to this contract stored in third-party lending protocols this contract interacts with. While this contract facilitates the movement of tokens from one third-party protocol to another, users retain full custody over their funds
- instant refinance integration with 3 protocols: Aave, Compound & Morpho

5 Audit Scope

The only contract in scope is: `packages/hardhat/contracts/RockoFlashRefinance.sol`

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Rocko Flash Refinance smart contracts provided by Rocko. In this period, a total of 19 issues were found.

The findings consist of 2 Low severity issues with the remainder being informational and gas optimizations.

Of the 2 Lows:

- 7.1.1 resulted in slightly less protocol fees being collected
- 7.1.2 resulted in refinance always reverting for popular non-standard ERC20 debt tokens such as USDT (Tether)

Protocol Invariants

We identified a number of useful invariants however we were not able to break any of them! These invariants have been implemented into our fuzz testing suite:

1) Refinancing clears the user's existing position such that no debt or collateral remain in the third-party protocol being exited
2) Refinancing opens a new position in a different third-party protocol with the same collateral and increased debt to cover the protocol fees, within a specific tolerance for rounding
3) No tokens remain in the protocol after each transaction has completed
4) The protocol itself does not accrue any collateral or debt balances on any of the third-party protocols it interacts with
5) If any approvals remain after refinance is complete, no other actor (including the protocol owner) can use those approvals to harm users in any way
6) The admin can't seize or otherwise take custody of user tokens; users maintain full custody of their tokens
7) Refinancing either totally completes or totally reverts in the one transaction; there is no half-way state which can be reached

Fuzz Testing

As part of the audit we made heavy use of fork fuzz testing by writing a test suite which:

- forks mainnet
- fuzzes refinancing from every possible protocol combination
- validates all relevant user and protocol state including that the above invariants hold
- achieves 94.92% line coverage, 96.93% statement coverage and 100% function coverage

Our fuzz testing suite uncovered:

- an interesting edge-case bug in Aave not related to this protocol which results in Aave at times leaking small amounts of both collateral and debt token value to users. This has been reported to Aave who acknowledged the issue but believe it is not further exploitable and plan to fix it at a later date
- tolerance thresholds due to the above bug and other third-party protocol roundings which result in slightly different amounts of debt tokens once the refinance has been complete; these tolerances have been built into the fuzz testing suite such that every token is accounted for

The fuzz testing suite has been supplied to the client as an additional deliverable for inclusion into their source code repository.

Summary

Project Name	Rocko Flash Refinance
Repository	onchain
Commit	911c3a6ef976...
Audit Timeline	Mar 17th - Mar 21th 2025
Methods	Manual Review, Fork Fuzz Testing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	8
Gas Optimizations	9
Total Issues	19

Summary of Findings

[L-1] Protocol fee should round up in favor of the protocol	Resolved
[L-2] Refinancing reverts for `USDT` debt token	Resolved
[I-1] Error messages hardcode `USDC` but other debt tokens may be used	Resolved
[I-2] Events missing indexed parameters	Resolved
[I-3] Unnecessary event emission when configuration values do not change	Resolved
[I-4] Inconsistent implementation approach for retrieving collateral balance from Morpho	Resolved
[I-5] Insufficient data length validation in `onMorphoFlashLoan`	Resolved
[I-6] In `_withdrawAaveCollateral` fetch `aTokenAddress` from Aave instead of receiving as input in `refinance` as passing it to morpho and back again	Resolved
[I-7] Provide a way for users to revoke all approvals	Resolved
[I-8] Consider allowing update to `AAVE_DATA_PROVIDER`	Acknowledged
[G-1] Use `ReentrancyGuardTransient` instead of `ReentrancyGuard` or more gas-efficient `nonReentrant` modifiers	Resolved
[G-2] Remove obsolete check in `updateFee`	Resolved
[G-3] Use `msg.sender` instead of `owner()` inside `onlyOwner` functions	Resolved
[G-4] Prevent repetitive hashing of identical strings	Resolved
[G-5] Don't initialize to default values	Resolved

[G-6] Use named return variables to eliminate redundant local variables and `return` statements	Resolved
[G-7] Remove redundant `onBehalfOf` variables	Resolved
[G-8] Remove redundant `morphoMarketId` validation checks in `_closeLoanMorphoWithShares` and `_openLoanPosition`	Resolved
[G-9] Redundant collateral balance check in `_openLoanMorpho`	Resolved

7 Findings

7.1 Low Risk

7.1.1 Protocol fee should round up in favor of the protocol

Description: Protocol fee should round up in favor of the protocol in `onMorphoFlashLoan`:

```
uint256 rockoFeeBP = ROCKO_FEE_BP;
if (rockoFeeBP > 0) {
    unchecked {
        feeAmount = (flashBorrowAmount * rockoFeeBP) / BASIS_POINTS_DIVISOR;
        borrowAmountWithFee += feeAmount;
    }
}
```

Consider using OpenZeppelin `Math::mulDiv` with the rounding parameter or Solady `FixedPointMathLib::fullMulDivUp`.

Another benefit of using these libraries is that intermediate overflow from the multiplication of `flashBorrowAmount * rockoFeeBP` is avoided.

Rocko: Fixed in commit a59ba0e.

Cyfrin: Verified.

7.1.2 Refinancing reverts for `USDT` debt token

Description: Refinancing reverts for `USDT` debt token due to the way protocol uses standard `IERC20::approve` and `transfer` functions.

Impact: Refinancing is bricked for `USDT` debt tokens. Marked as Low severity as officially only `USDC` is supported at this time. Note the implementation of `USDT` is different across chains; the protocol "as-is" would work with `USDT` on Base but not on Ethereum mainnet.

Proof of Concept: As part of the audit we have provided a fork fuzz testing suite; run this command: `forge test --fork-url ETH_RPC_URL --fork-block-number 22000000 --match-test test_FuzzRefinance_AaveToCompound_DepWeth_BorUsdt -vvv`

Recommended Mitigation: Replace all uses of `IERC20::approve` with `SafeERC20::forceApprove` and `IERC20::transfer` with `SafeERC20::safeTransfer` at L738, then re-run the PoC test and it now passes.

Ideally for added safety to prevent front-running of changes to existing approvals, use `SafeERC20::safeIncreaseAllowance` and `safeDecreaseAllowance` where suitable (for example in `_revokeTokenSpendApprovals` when the previous allowance amount is known could instead use `safeDecreaseAllowance`).

Rocko: Fixed in commit 751e906.

Cyfrin: Verified.

7.2 Informational

7.2.1 Error messages hardcode `USDC` but other debt tokens may be used

Description: Error messages hardcode `USDC` but other token may be used, eg:

```
function _closeLoanPositionAndReturnCollateralBalance(
    // @audit debt token can be other tokens apart from USDC but error
    // message hardcodes USDC
    require(
        debtBalance <= IERC20(debtTokenAddress).balanceOf(FLASH_LOAN_CONTRACT),
        "Insufficient USDC available in the flash contract"
    );
```

This code in `onMorphoFlashLoan` also assumes the debt token will be USDC:

```
uint256 usdcBalance = IERC20(ctx.debtTokenAddress).balanceOf(FLASH_LOAN_CONTRACT);
bool feeAmountAvailable = usdcBalance >= feeAmount;
```

Rocko: Fixed in commit ec9f5be.

Cyfrin: Verified.

7.2.2 Events missing indexed parameters

Description: Events in Solidity can have up to three indexed parameters, which are stored as topics in the event log. Indexed parameters allow for efficient filtering and searching of events by off-chain services. Without indexed parameters, it becomes more difficult and resource-intensive for applications to filter for specific events.

There are instances of events missing indexed parameters that could be improved.

```
    event LogRefinanceLoanCall(
        string logType,
        address rockoWallet,
        string from,
        string to,
        uint256 debtBalance,
        address debtTokenAddress,
        address collateralTokenAddress,
        address aCollateralTokenAddress,
        Id morphoMarketId
    );
    event LogFlashLoanCallback(
        string logType,
        address rockoWallet,
        string from,
        string to,
        address debtTokenAddress,
        address collateralTokenAddress,
        address aCollateralTokenAddress,
        uint256 flashBorrowAmount,
        bytes data,
        Id morphoMarketId
    );
```

Recommended Mitigation: Add the `indexed` keyword to important parameters in the event that would commonly be used for filtering, such as `rockoWallet`, `debtTokenAddress`, and `collateralTokenAddress`.

Rocko: Fixed in commit f5c9c80.

Cyfrin: Verified.

7.2.3 Unnecessary event emission when configuration values do not change

Description: `RockoFlashRefinance::updateFee` updates the `ROCKO_FEE_BP` variable and emits a `FeeUpdated` event regardless of whether the new fee value is different from the current one. This leads to unnecessary event emissions when the owner calls the function with the same fee value that is already set. The function `pauseContract` can be improved similarly too.

Rocko: Fixed in commit 99a73dc.

Cyfrin: Verified.

7.2.4 Inconsistent implementation approach for retrieving collateral balance from Morpho

Description: `RockoFlashRefinance::_collateralBalanceOfMorpho` uses direct storage slot access to retrieve a user's collateral balance from Morpho, while similar functionality for debt retrieval is implemented using `MorphoLib`. This inconsistency in the implementation approach makes the code less readable and maintainable.

```
    function _collateralBalanceOfMorpho(
        Id morphoMarketId,
        address rockoWallet
    ) private view returns (uint256 totalCollateralAssets) {//@audit-issue use MorphoLib::collateral
    ↪  instead
        bytes32[] memory slots = new bytes32[](1);
        slots[0] = MorphoStorageLib.positionBorrowSharesAndCollateralSlot(morphoMarketId, rockoWallet);
        bytes32[] memory values = MORPHO.extSloads(slots);
        totalCollateralAssets = uint256(values[0] >> 128);
    }

    function _getMorphoDebtAndShares(Id marketId, address rockoWallet) private returns (uint256 debt,
    ↪  uint256 shares) {
        MarketParams memory marketParams = MORPHO.idToMarketParams(marketId);
        MORPHO.accrueInterest(marketParams);

        uint256 totalBorrowAssets = MORPHO.totalBorrowAssets(marketId);
        uint256 totalBorrowShares = MORPHO.totalBorrowShares(marketId);
        shares = MORPHO.borrowShares(marketId, rockoWallet);
        debt = shares.toAssetsUp(totalBorrowAssets, totalBorrowShares);
    }
```

Recommended Mitigation: Refactor `_collateralBalanceOfMorpho` to use `MorphoLib::collateral` for consistency with other parts of the codebase:

```
function _collateralBalanceOfMorpho(
    Id morphoMarketId,
    address rockoWallet
) private view returns (uint256 totalCollateralAssets) {
-    bytes32[] memory slots = new bytes32[](1);
-    slots[0] = MorphoStorageLib.positionBorrowSharesAndCollateralSlot(morphoMarketId, rockoWallet);
-    bytes32[] memory values = MORPHO.extSloads(slots);
-    totalCollateralAssets = uint256(values[0] >> 128);
+    totalCollateralAssets = MorphoLib.collateral(MORPHO, morphoMarketId, rockoWallet);
}
```

Rocko: Fixed in commit 5ef86b4.

Cyfrin: Verified.

7.2.5 Insufficient data length validation in `onMorphoFlashLoan`

Description: `RockoFlashRefinance::onMorphoFlashLoan` performs a basic check on the length of the `data` parameter, requiring it to be at least 20 bytes. However, this check is insufficient as the actual data being sent is much

larger, containing multiple addresses, strings, and an Id parameter. The minimum expected data length should be at least 256 bytes plus additional bytes for dynamic string data.

Recommended Mitigation:

```
-         require(data.length >= 20, "Invalid data");
+         require(data.length >= 256, "Invalid data");
```

Rocko: Fixed in commit 1da67d7.

Cyfrin: Verified.

7.2.6 In `_withdrawAaveCollateral` fetch `aTokenAddress` from Aave instead of receiving as input in `refinance` as passing it to morpho and back again

Description: Aave's `aTokenAddress` is only required when withdrawing collateral in `_withdrawAaveCollateral`, but currently it is:

- passed in as input to `refinance`
- has some validation performed on it
- encoded along with other data and sent to `Morpho::flashLoan`
- then Morpho passes it back when calling `onMorphoFlashLoan`
- where it is decoded again and passed around some more

Instead of all this, simply use Aave's API function `IPool::getReserveData` to get the correct `aTokenAddress` inside `_withdrawAaveCollateral` where it is required:

```
    function _withdrawAaveCollateral(
        address collateralAddress,
        uint256 collateralBalance,
        address rockoWallet
    ) private {
        DataTypes.ReserveData memory reserveData = AAVE.getReserveData(collateralAddress);

        // Rocko Wallet needs to send aToken here after debt is paid off
        // Be sure that Rocko Wallet has approved this contract to spend aTokens for >
        ↪  `collateralBalance` tokens
        _pullTokensFromCallerWallet(reserveData.aTokenAddress, rockoWallet, collateralBalance);

        // function withdraw(address asset, uint256 amount, address to)
        AAVE.withdraw(collateralAddress, collateralBalance, FLASH_LOAN_CONTRACT);
    }
```

Fetching this parameter via Aave's API removes unnecessary code/validations also decreases the attack surface.

Rocko: Fixed in commit d793f96.

Cyfrin: Verified.

7.2.7 Provide a way for users to revoke all approvals

Description: `RockoFlashRefinance` is designed to move existing loan positions from one lending protocol to another. On behalf of the user the contract must be able to:

- close the loan from the previous lending provider
- open a new loan on the new lending provider

For Aave, users must allow the refinance contract to spend the AToken to close the position and to spend the VariableDebtToken to open a new position.

For Compound (Comet), users must allow the refinance contract by calling the allow function.

For Morpho, users must authorize the refinance protocol by calling the setAuthorization function.

The protocol team provided their frontend source related to these approvals and there were only "approving" support, not revoking. It is recommended to provide an easy way for users to revoke all these approvals.

Rocko: Users revokes will be included in the batch transaction when called from the Rocko app.

7.2.8 Consider allowing update to `AAVE_DATA_PROVIDER`

Description: `RockoFlashRefinance::AAVE_DATA_PROVIDER` immutably stores the address of `AaveProtocolDataProvider`. However `AaveProtocolDataProvider` is not upgradeable and the "current" one on mainnet was deployed 43 days ago to address 0x497a1994c46d4f6C864904A9f1fac6328Cb7C8a6.

Hence consider whether `AAVE_DATA_PROVIDER` should not be `immutable` and an `onlyOwner` function should exist to allow updating it in the future.

Since `RockoFlashRefinance` has no relevant internal state it can just be re-deloyed. The trade-off is having `immutable` `AAVE_DATA_PROVIDER` means user transactions involving it cost slightly less gas but the contract needs to be re-deployed to update it.

Rocko: Acknowledged; prefer the current setup for lower user gas costs.

7.3 Gas Optimization

7.3.1 Use `ReentrancyGuardTransient` instead of `ReentrancyGuard` or more gas-efficient `nonReentrant` modifiers

Description: Use `ReentrancyGuardTransient` instead of `ReentrancyGuard` for more gas-efficient `nonReentrant` modifiers. The OpenZeppelin version would need to be bumped to 5.1.

Rocko: Fixed in commit 675f4b2.

Cyfrin: Verified.

7.3.2 Remove obsolete check in `updateFee`

Description: Remove obsolete check in `updateFee`:

```
-        require(newFee >= 0, "Fee must not be negative");
```

This check is obsolete since `newFee` is declared as `uint256` therefore cannot be negative.

Rocko: Fixed in commit 2c50838.

Cyfrin: Verified.

7.3.3 Use `msg.sender` instead of `owner()` inside `onlyOwner` functions

Description: Using `msg.sender` instead of `owner()` inside `onlyOwner` functions is more efficient as it eliminates reading from storage. It is also safe since the `onlyOwner` modifier ensures that `msg.sender` is the owner:

```
757:         IERC20(tokenAddress).safeTransfer(owner(), amount);
766:         (bool success, ) = owner().call{ value: amount }("");
```

Rocko: Fixed in commit 751e906.

Cyfrin: Verified.

7.3.4 Prevent repetitive hashing of identical strings

Description: `RockoFlashRefinance::_compareStrings` is often called with the same values resulting in duplicate unnecessary work. A simple and more efficient way to prevent this is by first performing the conversion using `_parseProtocol` for both `from`/`to` inputs then simply comparing the enums as needed in functions like `refinance` and `_revokeTokenSpendApprovals`.

If string comparisons are required:

- hard-code the hash result as `bytes32` constants for common expected strings such as "aave", "morpho", "compound" and using these hard-coded constants inside `_parseProtocol` and other functions
- in functions such as `RockoFlashRefinance::refinance`, cache the hash of the `from`/`to` inputs in local `bytes32` variables and use the cached hashes and the hard-coded constants for the comparisons

One simple way to achieve this is by:

- defining a function to return the hash of a string:

```
function _hashString(string calldata input) private pure returns (bytes32 output) {
    output = keccak256(bytes(input));
}
```

- changing `_compareStrings` to take two `bytes32` as input:

```
    function _compareStrings(bytes32 a, bytes32 b) private pure returns (bool) {
        return a == b;
    }
```

Consider OpenZeppelin's string equality implementation as well.

Rocko: Fixed in commit a59ba0e.

Cyfrin: Verified.

7.3.5 Don't initialize to default values

Description: Don't initialize to default values as Solidity already does this:

```
78:         ROCKO_FEE_BP = 0;
597:         uint256 debtBalance = 0;
598:         uint256 morphoDebtShares = 0;
```

Rocko: Fixed in commit 751e906.

Cyfrin: Verified.

7.3.6 Use named return variables to eliminate redundant local variables and `return` statements

Description: Use named return variables to eliminate redundant local variables and `return` statements:

```
// _closeLoanPositionAndReturnCollateralBalance L457
-    ) private returns (uint256) {
+    ) private returns (uint256 collateralBalance) {

// L464
-        uint256 collateralBalance;

// L480
-        return collateralBalance;
```

Same idea can be applied to `_collateralBalanceOfAave`, `_getDebtBalanceOfAave`.

Rocko: Fixed in commit 751e906.

Cyfrin: Verified.

7.3.7 Remove redundant `onBehalfOf` variables

Description: Remove redundant `onBehalfOf` variables:

```
    function _supplyToAave(address collateralAddress, uint256 collateralBalance, address rockoWallet)
    ↪  private {
-       address onBehalfOf = rockoWallet;
-       AAVE.supply(collateralAddress, collateralBalance, onBehalfOf, AAVE_REFERRAL_CODE);
+       AAVE.supply(collateralAddress, collateralBalance, rockoWallet, AAVE_REFERRAL_CODE);
    }
    function _borrowFromAave(address rockoWallet, address token, uint256 borrowAmount) private {
-       address onBehalfOf = rockoWallet;
-       AAVE.borrow(token, borrowAmount, AAVE_INTERESTE_RATE_MODE, AAVE_REFERRAL_CODE, onBehalfOf);
+       AAVE.borrow(token, borrowAmount, AAVE_INTERESTE_RATE_MODE, AAVE_REFERRAL_CODE, rockoWallet);
    }
```

Rocko: Fixed in commit 751e906.

Cyfrin: Verified.

7.3.8 Remove redundant `morphoMarketId` validation checks in `_closeLoanMorphoWithShares` and `_openLoanPosition`

Description: `RockoFlashRefinance::_closeLoanMorphoWithShares` and `_openLoanPosition` contain redundant validation `morphoMarketId`. The reasons why this validation is redundant:

- `RockoFlashRefinance::refinance` already validates the input `morphoMarketId`, encodes it into the `data` payload then calls `Morpho::flashLoan` with the `data` payload:

```
if (_compareStrings(to, "morpho") || _compareStrings(from, "morpho")) {
    require(_isValidId(morphoMarketId), "Morpho Market ID required for Morpho refinance");
}

bytes memory data = abi.encode(
    rockoWallet,
    from,
    to,
    debtTokenAddress,
    collateralTokenAddress,
    aCollateralTokenAddress,
    morphoMarketId,
    morphoDebtShares
);

MORPHO.flashLoan(debtTokenAddress, debtBalance, data);
```

- `Morpho::flashLoan` always passes the unmodified `data` payload to `RockoFlashRefinance::onMorphoFlashLoan`:

```
function flashLoan(address token, uint256 assets, bytes calldata data) external {
    require(assets != 0, ErrorsLib.ZERO_ASSETS);

    emit EventsLib.FlashLoan(msg.sender, token, assets);

    IERC20(token).safeTransfer(msg.sender, assets);

    // @audit passing unmodified `data` payload to `onMorphoFlashLoan`
    IMorphoFlashLoanCallback(msg.sender).onMorphoFlashLoan(assets, data);

    IERC20(token).safeTransferFrom(msg.sender, address(this), assets);
}
```

*`RockoFlashRefinance::onMorphoFlashLoan` decodes the unmodified `data` payload and calls `_closeLoanMorphoWithShares` and `_openLoanPosition` using the decoded `morphoMarketId` which has already been validated in `RockoFlashRefinance::refinance`.

Recommended Mitigation: Remove the redundant `morphoMarketId` validation checks at:

```
325: require(_isValidId(morphoMarketId), "Invalid Morpho Market ID");

503: require(_isValidId(morphoMarketId), "Morpho Market ID required for Morpho refinance");
```

Rocko: Fixed in commit 5a9aa7d.

Cyfrin: Verified.

7.3.9 Redundant collateral balance check in `_openLoanMorpho`

Description: `RockoFlashRefinance::_openLoanMorpho` contains a redundant check for collateral balance availability. The function verifies that the flash loan contract has sufficient collateral balance, but this check is already performed in the calling `_openLoanPosition` function.

Recommended Mitigation:

```
    function _openLoanMorpho(
        Id morphoMarketId,
        uint256 borrowAmount,
        address collateralAddress,
        uint256 collateralBalance,
        address rockoWallet
    ) private {
        _checkAllowanceAndApproveContract(address(MORPHO), collateralAddress, collateralBalance);
        MarketParams memory marketParams = MORPHO.idToMarketParams(morphoMarketId);
-       uint256 flashLoanContractBalance = IERC20(collateralAddress).balanceOf(FLASH_LOAN_CONTRACT);
-       // emit LogBalance("Flash Loan Contract Balance", flashLoanContractBalance);
-       require(
-           flashLoanContractBalance >= collateralBalance,
-           "Insufficient collateral available in the flash contract"
-       );
```

Rocko: Fixed in commit a8efb43.

Cyfrin: Verified.